PyPCAPKit

Release 0.15.5

Jarry Shaw

Oct 11, 2020

CONTENTS

The *PyPCAPKit* project is an open source Python program focus on PCAP parsing and analysis, which works as a stream PCAP file extractor. With support of `DictDumper`, it shall support multiple output report formats.

Important: The whole project supports **Python 3.4** or later.

CHAPTER

ONE

STREAM PCAP FILE EXTRACTOR

pcapkit is an independent open source library, using only DictDumper as its formatted output dumper.

> There is a project called `jspcapy` works on *pcapkit*, which is a command line tool for PCAP extraction.

Unlike popular PCAP file extractors, such as Scapy, DPKT, PyShark, and etc, *pcapkit* uses streaming strategy to read input files. That is to read frame by frame, decrease occupation on memory, as well as enhance efficiency in some way.

1.1 Library Foundation

pcapkit.foundation is a collection of fundations for *pcapkit*, including PCAP file extraction tool `Extrator`, application layer protocol analyser `Analysis`, and TCP flow tracer `TraceFlow`.

1.1.1 Analyser for Application Layer

pcapkit.foundation.analysis works as a header quarter to analyse and match application layer protocol. Then, call corresponding modules and functions to extract the attributes.

pcapkit.foundation.analysis.**_analyse**(*protocol*, *file*, *length=None*, ***, *seekset=0*)

Analyse packet.

Parameters

- **protocol** (`Protocol`) – target protocol class
- **file** (*io.BytesIO*) – source data stream
- **length** (*Optional[int]*) – packet length

Keyword Arguments **seekset** (*int*) – original file offset

Returns If the packet is parsed successfully, returns the parsed packet; otherwise returns `None`.

Return type Optional[*Protocol*]

pcapkit.foundation.analysis.**analyse**(*file*, *length=None*, ***, *termination=False*)

Analyse application layer packets.

Parameters

- **file** (*io.BytesIO*) – source data stream
- **length** (*Optional[int]*) – packet length

Keyword Arguments **termination** (*bool*) – If terminate parsing application layer protocol.

Returns Parsed application layer protocol.

Return type *Protocol*

Notes

Currently, the analysis processes in following order:

1. *FTP*
2. *HTTP/1.**
3. *HTTP/2*

and *Raw* as the fallback result.

See also:

The analysis processes order is defined by *ANALYSE_PROTO*.

pcapkit.foundation.analysis.**register**(*module*, *class_*, *, *index=None*)
Register a new protocol class.

Parameters

- **module** (*str*) – module name
- **class** (*str*) – class name

Keyword Arguments **index** (*Optional[int]*) – Index of the protocol class when inserted to *ANALYSE_PROTO*.

Notes

The full qualified class name of the new protocol class should be as `{module}.{class_}`.

---

pcapkit.foundation.analysis.**ANALYSE_PROTO = [('pcapkit.protocols.application.ftp', 'FTP'),**
List of protocols supported by the analyser.

1.1.2 Extractor for PCAP Files

pcapkit.foundation.extraction contains *Extractor* only, which synthesises file I/O and protocol analysis, coordinates information exchange in all network layers, extracts parametres from a PCAP file.

Todo: Implement engine support for pypcap & pycapfile.

---

class pcapkit.foundation.extraction.**Extractor**(*fin=None*, *fout=None*, *format=None*, *auto=True*, *extension=True*, *store=True*, *files=False*, *nofile=False*, *verbose=False*, *engine=None*, *layer=None*, *protocol=None*, *ip=False*, *ipv4=False*, *ipv6=False*, *tcp=False*, *strict=True*, *trace=False*, *trace_fout=None*, *trace_format=None*, *trace_byteorder='little'*, *trace_nanosecond=False*)

Bases: object

Extractor for PCAP files.

For supported engines, please refer to corresponding driver method for more information:

- Default drivers:
  - Global header: *record_header()*
  - Packet frames: *record_frames()*
- DPKT driver: *_run_dpkt()*
- Scapy driver: *_run_scapy()*
- PyShark driver: *_run_pyshark()*
- Multiprocessing driver:
  - Pipeline model: *_run_pipeline()*
  - Server model: *_run_server()*

_ifnm: **str**
Input file name.

_ofnm: **str**
Output file name.

_fext: **str**
Output file extension.

_flag_a: **bool**
Auto extraction flag (as the `auto` parameter).

_flag_d: **bool**
Data storing flag (as the `store` parameter).

_flag_e: **bool**
EOF flag.

_flag_f: **bool**
Split output into files flag (as the `files` parameter).

_flag_m: **bool**
Multiprocessing engine flag.

_flag_q: **bool**
No output flag (as the `nofile` parameter).

_flag_t: **bool**
TCP flow tracing flag (as the `trace` parameter).

_flag_v: **Union[bool, Callable[[*pcapkit.foundation.extraction.Extractor*, *pcapkit.proto***
A `bool` value or a function takes the `Extract` instance and current parsed frame (depends on the engine selected) as parameters to print verbose output information (as the `verbose` parameter).

_vfunc: **Union[NotImplemented, Callable[[*pcapkit.foundation.extraction.Extractor*, *pcap***
If the `verbose` parameter is a callable, then it will be assigned as *self._vfunc*; otherwise, it keeps `NotImplemented` as a placeholder and has specific function for each engine.

_frnum: **int**
Current frame number.

_frame: **List[*pcapkit.protocols.pcap.frame.Frame*]**
Frame records storage.

_proto: *pcapkit.corekit.protochain.ProtoChain*
Current frame's protocol chain.

_reasm: **List[Optiona[***pcapkit.reassembly.ipv4.IPv4_Reassembly***], Optiona[***pcapkit.reasse**
Reassembly buffers.

_trace: **Optional[***pcapkit.foundation.traceflow.TraceFlow***]**
TCP flow tracer.

_ipv4: **bool**
IPv4 reassembly flag (as the `ipv4` and/or `ip` flag).

_ipv6: **bool**
IPv6 reassembly flag (as the `ipv6` and/or `ip` flag).

_tcp: **bool**
TCP reassembly flag (as the `tcp` parameter).

_exptl: **str**
Extract til protocol flag (as the `protocol` parameter).

_exlyr: **str**
Extract til layer flag (as the `layer` parameter).

_exeng: **str**
Extration engine (as the `engine` parameter).

_ifile: **io.BufferedReader**
Source PCAP file (opened in binary mode).

_ofile: **Optional[Union[dictdumper.dumper.Dumper, Type[dictdumper.dumper.Dumper]]]**
Output dumper. If *self._flag_f* is `True`, it is the `Dumper` object, otherwise it is an initialised `Dumper` instance.

Note: We customised the `object_hook()` method to provide generic support of `enum.Enum`, `ipaddress.IPv4Address`, `ipaddress.IPv6Address` and *Info*.

See also:

When the output format is unsupported, we uses *NotImplementedIO* as a fallback solution.

_gbhdr: *pcapkit.protocols.pcap.header.Header*
Parsed PCAP global header instance.

_vinfo: *pcapkit.corekit.version.VersionInfo*
The version info of the PCAP file (as the *self._gbhdr.version* property).

_dlink: *pcapkit.const.reg.linktype.LinkType*
Protocol type of data link layer (as the *self._gbhdr.protocol* property).

_nnsec: **bool**
Nanosecond PCAP file flag (as the *self._gbhdr.nanosecond* property).

_type: **str**
Output format (as the `self._ofile.kind` property).

_expkg: **types.ModuleType**
Extraction engine module.

_extmp: **Iterator[Any]**
Temporary storage for frame parsing iterator.

_mpprc: **List[multiprocessing.Process]**
List of active child processes.

_mpfdp: **DefaultDict[multiprocessing.Queue]**
File pointer (offset) queue for each frame.

_mpmng: **multiprocessing.sharedctypes.multiprocessing.Manager**
Multiprocessing manager context.

_mpkit: **multiprocessing.managers.SyncManager.Namespace**
Multiprocessing utility namespace.

_mpkit.**counter:** **int**
Number of active workers.

_mpkit.**pool:** **int**
Number of prepared workers.

_mpkit.**current:** **int**
Current processing frame number.

_mpkit.**eof:** **bool**
EOF flag.

_mpkit.**frames:** **Dict[int, *pcapkit.protocols.pcap.frame.Frame*]**
Frame storage.

_mpkit.**trace:** **Optional[*pcapkit.foundation.traceflow.TraceFlow*]**
TCP flow tracer.

_mpkit.**reassembly:** **List[Optiona[*pcapkit.reassembly.ipv4.IPv4_Reassembly*], Optiona[*pca*
Reassembly buffers.

_mpsrv: **multiprocessing.Proccess**
Server process for frame analysis and processing.

_mpbuf: **Union[multiprocessing.managers.SyncManager.dict, Dict[int, *pcapkit.protocols.*
Multiprocessing buffer for parsed PCAP frames.

_mpfrm: **Union[multiprocessing.managers.SyncManager.list, List[*pcapkit.protocols.pcap.*
Multiprocessing storage for proccessed PCAP frames.

_mprsm: **Union[multiprocessing.managers.SyncManager.list, List[Optiona[*pcapkit.reassem*
Multiprocessing storage for reassembly buffers.

__call__()
Works as a simple wrapper for the iteration protocol.

Raises ***IterableError*** – If *self._flag_a* is True, as iteration is not applicable.

__enter__()
Uses *Extractor* as a context manager.

__exit__(*exc_type*, *exc_value*, *traceback*)
Close the input file when exits.

__init__(*fin=None*, *fout=None*, *format=None*, *auto=True*, *extension=True*, *store=True*, *files=False*, *nofile=False*, *verbose=False*, *engine=None*, *layer=None*, *protocol=None*, *ip=False*, *ipv4=False*, *ipv6=False*, *tcp=False*, *strict=True*, *trace=False*, *trace_fout=None*, *trace_format=None*, *trace_byteorder='little'*, *trace_nanosecond=False*)
Initialise PCAP Reader.

Parameters

- **fin** (*Optiona[str]*) – file name to be read; if file not exist, raise FileNotFound

- **fout** (*Optiona[str]*) – file name to be written
- **format** (*Optional[Literal['plist', 'json', 'tree']]*) – file format of output
- **auto** (*bool*) – if automatically run till EOF
- **extension** (*bool*) – if check and append extensions to output file
- **store** (*bool*) – if store extracted packet info
- **files** (*bool*) – if split each frame into different files
- **nofile** (*bool*) – if no output file is to be dumped
- **(Union[bool** (*verbose*) – pcapkit.protocol.pcap.frame.Frame]]]): a `bool` value or a function takes the `Extract` instance and current parsed frame (depends on engine selected) as parameters to print verbose output information
- **Callable[[pcapkit.foundation.extraction.Extractor** – pcapkit.protocol.pcap.frame.Frame]]]): a `bool` value or a function takes the `Extract` instance and current parsed frame (depends on engine selected) as parameters to print verbose output information

:param [pcapkit.protocol.pcap.frame.Frame]]]): a `bool` value or a function takes the `Extract`] instance and current parsed frame (depends on engine selected) as parameters to print verbose output information

Parameters

- **engine** (*Optional[Literal['default', 'pcapkit', 'dpkt', 'scapy', 'pyshark', 'server', 'pipeline']]*) – extraction engine to be used
- **layer** (*Optional[Literal['Link', 'Internet', 'Transport', 'Application']]*) – extract til which layer
- **protocol** (*Optional[Union[str, Tuple[str], Type[Protocol]]]*) – extract til which protocol
- **ip** (*bool*) – if record data for IPv4 & IPv6 reassembly
- **ipv4** (*bool*) – if perform IPv4 reassembly
- **ipv6** (*bool*) – if perform IPv6 reassembly
- **tcp** (*bool*) – if perform TCP reassembly
- **strict** (*bool*) – if set strict flag for reassembly
- **trace** (*bool*) – if trace TCP traffic flows
- **trace_fout** (*Optional[str]*) – path name for flow tracer if necessary
- **trace_format** (*Optional[Literal['plist', 'json', 'tree', 'pcap']]*) – output file format of flow tracer
- **trace_byteorder** (*Literal['little', 'big']*) – output file byte order
- **trace_nanosecond** (*bool*) – output nanosecond-resolution file flag

Warns **FormatWarning** – Warns under following circumstances:

- If using PCAP output for TCP flow tracing while the extraction engine is PyShark.
- If output file format is not supported.

__iter__()
Iterate and parse PCAP frame.

Raises ***IterableError*** – If *self._flag_a* is True, as such operation is not applicable.

__next__()
Iterate and parse next PCAP frame.

It will call *_read_frame()* to parse next PCAP frame internally, until the EOF reached; then it calls *_cleanup()* for the aftermath.

_aftermathmp()
Aftermath for multiprocessing.

The method will *join* all child processes forked/spawned as in *self._mpprc*, and will *join* *self._mpsrv* server process if using multiprocessing server engine.

For multiprocessing server engine, it will

- assign *self._mpfrm* to *self._frame*
- assign *self._mprsm* to *self._reasm*
- copy *self._mpkit.trace* to *self._trace*

For multiprocessing pipeline engine, it will

- restore *self._frame* from *self._mpkit.frames*
- copy *self._mpkit.reassembly* to *self._reasm*
- copy *self._mpkit.trace* to *self._trace*

After restoring attributes, it will *shutdown* multiprocessing manager context *self._mpmng*, delete all multiprocessing attributes (i.e. starts with *_mp*), and deduct the frame number *self._frnum* by 2 (*hacking solution*).

Notes

If *self._flag_e* is already set as True, do nothing.

Raises ***UnsupportedCall*** – If *self._flag_m* is False, as such operation is not applicable.

_cleanup()
Cleanup after extraction & analysis.

The method clears the *self._expkg* and *self._extmp* attributes, sets *self._flag_e* as True and closes the input file.

_default_read_frame(*, *frame=None*, *mpkit=None*)
Read frames with default engine.

This method performs following operations:

- extract frames and each layer of packets;
- make *Info* object out of frame properties;
- write to output file with corresponding dumper;
- reassemble IP and/or TCP datagram;
- trace TCP flows if any;

- record frame *Info* object to frame storage.

  Keyword Arguments

  - **frame** (*Optional[pcapkit.protocols.pcap.frame.Frame]*) – The fallback `frame` data (for multiprocessing engines).
  - **mpkit** (*multiprocessing.managers.SyncManager.Namespace*) – The multiprocess data kit.

  Returns Parsed frame instance.

  Return type Optional[*pcapkit.protocols.pcap.frame.Frame*]

_dpkt_read_frame()
Read frames with DPKT engine.

Returns Parsed frame instance.

Return type dpkt.dpkt.Packet

See also:

Please refer to *_default_read_frame()* for more operational information.

_pipeline_read_frame(*, *mpfdp*, *mpkit*)
Extract frame with multiprocessing pipeline engine.

The method calls `Frame` to parse the PCAP frame data. Should `EOFError` raised, it will toggle *self._mpkit.eof* as `True`. Finally, it will decendant `self.mpkit.counter` by `1` and closes the input source file (as the child process exits).

For the parsed `Frame` instance, the instant will first wait until `self.mpkit.current` is the same as *self._frnum*, i.e. it's now time to process the parsed frame as in a linear sequential order.

It will proceed by calling *_default_read_frame()*, whilst temporarily assigning `self.mpkit.trace` to *self._trace* and `self.mpkit.reassembly` to *self._reasm* then put back.

Keyword Arguments

- **mpfdp** (*multiprocessing.Queue*) – `Queue` for multiprocessing file pointer (offset).
- **mpkit** (*multiprocessing.managers.SyncManager.Namespace*) – `Namespace` instance as *self._mpkit*.

Raises **EOFError** – If *self._flag_e* is `True`, as the parsing had finished.

_pyshark_read_frame()
Read frames with PyShark engine.

Returns Parsed frame instance.

Return type pyshark.packet.packet.Packet

Notes

This method inserts *packet2dict()* to the parsed frame instance as `packet2dict()` method.

See also:

Please refer to *_default_read_frame()* for more operational information.

_read_frame()
Headquarters for frame reader.

This method is a dispatcher for parsing frames.

- For Scapy engine, calls *_scapy_read_frame()*.
- For DPKT engine, calls *_dpkt_read_frame()*.
- For PyShark engine, calls *_pyshark_read_frame()*.
- For default (PyPCAPKit) engine, calls *_default_read_frame()*.

Returns The parsed frame instance.

_run_dpkt(*dpkt*)
Call dpkt.pcap.Reader to extract PCAP files.

This method assigns *self._expkg* as dpkt and *self._extmp* as an iterator from dpkt.pcap.Reader.

Parameters **dpkt** (*types.ModuleType*) – The dpkt module.

Warns **AttributeWarning** – If *self._exlyr* and/or *self._exptl* is provided as the DPKT engine currently does not support such operations.

_run_pipeline(*multiprocessing*)
Use pipeline multiprocessing to extract PCAP files.

Notes

The basic concept of multiprocessing pipeline engine is that we parse the PCAP file as a pipeline. Each frame per worker. Once the length of a frame is known, i.e. the PCAP frame header is parsed, then we can start a new working and start parsing the next frame concurrently.

However, as the datagram reassembly and TCP flow tracing require linear sequential processing, we still need to *wait* for the completion of analysis on previous frames before proceeding on such operations.

This method assigns *self._expkg* as multiprocessing, creates a file pointer storage as *self._mpfdp*, manager context as *self._mpmng* and namespace as *self._mpkit*.

In the namespace, we initiate number of (on duty) workers as counter, pool of (ready) workers as pool, current frame number as current, EOF flag as eof, frame storage as frames, TCP flow tracer *self._trace* as trace and the reassembly buffers *self._reasm* as reassembly.

After initial setup, the method calls *record_header()* to parse the PCAP global header and *put* the file offset to *self._mpfdp* as the start of first frame. Then it starts the parsing of each PCAP frame.

During this phrase, it's a while clause until *self._mpkit.eof* is set as True then it calls *_update_eof()* and breaks. In the while clause, it maintains a multiprocessing.pool.Pool like worker pool. It checks the *self._mpkit.pool* for available workers and *self._mpkit.counter* for active workers.

When starts a new worker, it first update the input file offset to the file offset as specified in *self._mpfdp*. Then creates a child process running *_pipeline_read_frame()* with keyword arguments mpkit as *self._mpkit* and mpfdp as corresponding Queue from *self._mpfdp*. Later, it decendants the *self._mpkit.pool* and increments the *self._mpkit.counter*, both by 1. The child process will be appended to *self._mpprc*.

When the number of active workers is greater than or equal to CPU_CNT, it waits and *join* the leading child processes in *self._mpprc* then removes their reference.

Parameters **multiprocessing** (*types.ModuleType*) – The multiprocessing module.

Warns **AttributeWarning** – If *self._flag_q* is False, as multiprocessing engines do not support output.

Raises ***UnsupportedCall*** – If *self._flag_m* is False, as such operation is not applicable.

_run_pyshark(*pyshark*)

Call pyshark.FileCapture to extract PCAP files.

This method assigns *self._expkg* as pyshark and *self._extmp* as an iterator from pyshark.FileCapture.

Parameters **pyshark** (*types.ModuleType*) – The pyshark module.

Warns **AttributeWarning** – Warns under following circumstances:

- if *self._exlyr* and/or *self._exptl* is provided as the PyShark engine currently does not support such operations.
- if reassembly is enabled, as the PyShark engine currently does not support such operation.

_run_scapy(*scapy_all*)

Call scapy.all.sniff() to extract PCAP files.

This method assigns *self._expkg* as scapy.all and *self._extmp* as an iterator from scapy.all.sniff().

Parameters **scapy_all** (*types.ModuleType*) – The scapy.all module.

Warns **AttributeWarning** – If *self._exlyr* and/or *self._exptl* is provided as the Scapy engine currently does not support such operations.

_run_server(*multiprocessing*)

Use server multiprocessing to extract PCAP files.

Notes

The basic concept of multiprocessing server engine is that we further separate the logic of PCAP frame parsing and analysis/processing, comparing to the multiprocessing pipeline engine (c.f. *_run_pipeline()*).

We starts a *server* process to perform the datagram reassembly and TCP flow tracing, etc. of all parsed PCAP frames, whilst parsing each PCAP frame in the same manner as in multiprocessing pipeline engine, i.e. each frame per worker.

This method assigns *self._expkg* as multiprocessing, creates a file pointer storage as *self._mpfdp*, manager context as *self._mpmng* and namespace as *self._mpkit*. We will also maintain the active process list *self._mpprc* as in *_run_pipeline()*.

It will also creates a dict as *self._mpbuf*, frame buffer (temporary storage) for the server process to obtain the parsed frames; a list as *self._mpfrm*, eventual frame storage; and another list as *self._mprsm*, storing the reassembly buffers *self._reasm* before the server process exits.

In the namespace, we initiate number of (on duty) workers as counter, pool of (ready) workers as pool, current frame number as current, EOF flag as eof, frame storage as frames, and trace for storing TCP flow tracer *self._trace* before the server process exits.

After initial setup, the method calls *record_header()* to parse the PCAP global header and *put* the file offset to *self._mpfdp* as the start of first frame. It will then starts the server process *self._mpsrv* from *_server_analyse_frame()*. Finally, it starts the parsing of each PCAP frame.

During this phrase, it's a `while` clause until *self._mpkit.eof* is set as `True` then it calls *_update_eof()* and breaks. In the `while` clause, it maintains a `multiprocessing.pool.Pool` like worker pool. It checks the *self._mpkit.pool* for available workers and *self._mpkit.counter* for active workers.

When starts a new worker, it first update the input file offset to the file offset as specified in *self._mpfdp*. Then creates a child process running *_server_extract_frame()* with keyword arguments `mpkit` as *self._mpkit*, *mpbuf* as *self._mpbuf* and `mpfdp` as corresponding `Queue` from *self._mpfdp*. Later, it decendants the *self._mpkit.pool* and increments the *self._mpkit.counter*, both by `1`. The child process will be appended to *self._mpprc*.

When the number of active workers is greater than or equal to `CPU_CNT`, it waits and *join* the leading child processes in *self._mpprc* then removes their reference.

Parameters **multiprocessing** (*types.ModuleType*) – The `multiprocessing` module.

Warns **AttributeWarning** – If *self._flag_q* is `False`, as multiprocessing engines do not support output.

Raises ***UnsupportedCall*** – If *self._flag_m* is `False`, as such operation is not applicable.

_scapy_read_frame()

Read frames with Scapy engine.

Returns Parsed frame instance.

Return type scapy.packet.Packet

See also:

Please refer to *_default_read_frame()* for more operational information.

_server_analyse_frame(*, *mpkit*, *mpfrm*, *mprsm*, *mpbuf*)

Analyse frame using multiprocessing server engine.

This method starts a `while` clause. For each round, it will *pop* the frame *self._frnum* from `mpbuf` then calls *_default_read_frame()* to perform datagram reassembly and TCP flow tracing, etc.

Once the frame popped is `EOFError`, i.e. the frame parsing had finished, it breaks from the clause and updates `mpfrm` with *self._frame*, `mprsm` with *self._reasm*, and `mpkit.trace` with *self._trace*.

Keyword Arguments

- **mpkit** (*multiprocessing.managers.SyncManager.Namespace*) – `Namespace` instance as *_mpkit*.
- **mpfrm** (*multiprocessing.managers.SyncManager.list*) – Frame storage.
- **mprsm** (*multiprocessing.managers.SyncManager.list*) – Reassembly buffers.
- **mpbuf** (*multiprocessing.managers.SyncManager.dict*) – Frame buffer (temporary storage) for the server process *self._mpsrv* to obtain the parsed frames.

_server_extract_frame(*, *mpfdp*, *mpkit*, *mpbuf*)

Extract frame using multiprocessing server engine.

The method calls `Frame` to parse the PCAP frame data. The parsed frame will be saved to `mpbuf` under the corresponding frame number *`self._frnum`*.

Should `EOFError` raised, it will toggle *`self._mpkit.eof`* as `True`, and save `EOFError` object to `mpbuf` under the corresponding frame number *`self._frnum`*.

Finally, it will decendant `self.mpkit.counter` by `1` and closes the input source file (as the child process exits).

Parameters

- **mpfdp** (*multiprocessing.Queue*) – `Queue` for multiprocessing file pointer (offset).
- **mpkit** (*multiprocessing.managers.SyncManager.Namespace*) – `Namespace` instance as *`_mpkit`*.
- **mpbuf** (*multiprocessing.managers.SyncManager.dict*) – Frame buffer (temporary storage) for the server process *`self._mpsrv`* to obtain the parsed frames.

Raises **EOFError** – If *`self._flag_e`* is `True`, as the parsing had finished.

_update_eof()
Update EOF flag.

This method calls *`_aftermathmp()`* to cleanup multiproccessing stuff, closes the input file and toggle *`self._flag_e`* as `True`.

check()
Check layer and protocol thresholds.

Warns

- **LayerWarning** – If *`self._exlyr`* is not recognised.
- **ProtocolWarning** – If *`self._exptl`* is not recognised.

See also:

- List of available layers: `LAYER_LIST`
- List of available protocols: `PROTO_LIST`

static import_test(*engine*, *, *name=None*)
Test import for extractcion engine.

Parameters **engine** (*str*) – Extraction engine module name.

Keyword Arguments **name** (*Optional[str]*) – Extraction engine display name.

Warns **EngineWarning** – If the engine module is not installed.

Returns If succeeded, returns `True` and the module; otherwise, returns `False` and `None`.

Return type Tuple[bool, Optional[ModuleType]]

classmethod make_name(*fin*, *fout*, *fmt*, *extension*, *, *files=False*, *nofile=False*)
Generate input and output filenames.

The method will perform following processing:

1. sanitise `fin` as the input PCAP filename; `in.pcap` as default value and append `.pcap` extension if needed and `extension` is `True`; as well as test if the file exists;
2. if `nofile` is `True`, skips following processing;

3. if `fmt` provided, then it presumes corresponding output file extension;
4. if `fout` not provided, it presumes the output file name based on the presumptive file extension; the stem of the output file name is set as `out`; should the file extension is not available, then it raises *`FormatError`*;
5. if `fout` provided, it presumes corresponding output format if needed; should the presumption cannot be made, then it raises *`FormatError`*;
6. it will also append corresponding file extension to the output file name if needed and `extension` is `True`.

Parameters

- **fin** (*Optional[str]*) – Input filename.
- **fout** (*Optional[str]*) – Output filename.
- **fmt** (*str*) – Output file format.
- **extension** (*bool*) – If append `.pcap` file extension to the input filename if `fin` does not have such file extension; if check and append extensions to output file.

Keyword Arguments

- **files** (*bool*) – If split each frame into different files.
- **nofile** (*bool*) – If no output file is to be dumped.

Returns

Generated input and output filenames:

0. input filename
1. output filename / directory name
2. output format
3. output file extension (without `.`)
4. if split each frame into different files

Return type Tuple[str, str, str, str, bool]

Raises

- ***FileNotFound*** – If input file does not exists.
- ***FormatError*** – If output format not provided and cannot be presumpted.

record_frames()

Read packet frames.

The method calls *`_read_frame()`* to parse each frame from the input PCAP file; and calls *`_cleanup()`* upon complision.

Notes

Under non-auto mode, i.e. *`self._flag_a`* is `False`, the method performs no action.

---

record_header()

Read global header.

The method will parse the PCAP global header and save the parsed result as *self._gbhdr*. Information such as PCAP version, data link layer protocol type, nanosecond flag and byteorder will also be save the current *Extractor* instance.

If TCP flow tracing is enabled, the nanosecond flag and byteorder will be used for the output PCAP file of the traced TCP flows.

For output, the method will dump the parsed PCAP global header under the name of `Global Header`.

run()
Start extraction.

We uses *import_test()* to check if a certain engine is available or not. For supported engines, each engine has different driver method:

- Default drivers:
  - Global header: *record_header()*
  - Packet frames: *record_frames()*
- DPKT driver: *_run_dpkt()*
- Scapy driver: *_run_scapy()*
- PyShark driver: *_run_pyshark()*
- Multiprocessing driver:
  - Pipeline model: *_run_pipeline()*
  - Server model: *_run_server()*

Warns **EngineWarning** – If the extraction engine is not available. This is either due to dependency not installed, number of CPUs is not enough, or supplied engine unknown.

property engine
PCAP extraction engine.

Return type str

property format
Format of output file.

Raises ***UnsupportedCall*** – If *self._flag_q* is set as `True`, as output is disabled by initialisation parameter.

Return type str

property frame
Extracted frames.

Raises ***UnsupportedCall*** – If *self._flag_d* is `True`, as storing frame data is disabled.

Return type Tuple[*Info*[*DataType_Frame*]]

property header
Global header.

Raises ***UnsupportedCall*** – If *self._exeng* is `'scapy'` or `'pyshark'`, as such engines does not reserve such information.

Return type *Info*[*DataType_Header*]

property info
Version of input PCAP file.

Raises ***UnsupportedCall*** – If *self._exeng* is `'scapy'` or `'pyshark'`, as such engines does not reserve such information.

Return type *VersionInfo*

property input
Name of input PCAP file.

Return type str

property length
Frame number (of current extracted frame or all).

Return type int

property output
Name of output file.

Raises ***UnsupportedCall*** – If *self._flag_q* is set as `True`, as output is disabled by initialisation parameter.

Return type str

property protocol
Protocol chain of current frame.

Raises ***UnsupportedCall*** – If *self._flag_a* is `True`, as such attribute is not applicable.

Return type *ProtoChain*

property reassembly
Frame record for reassembly.

- `ipv6` – tuple of TCP payload fragment (*IPv4_Reassembly*)
- `ipv4` – tuple of TCP payload fragment (*IPv6_Reassembly*)
- `tcp` – tuple of TCP payload fragment (*TCP_Reassembly*)

Return type *Info*

property trace
Index table for traced flow.

Raises ***UnsupportedCall*** – If *self._flag_t* is `True`, as TCP flow tracing is disabled.

Return type Tuple[*Info*]

pcapkit.foundation.extraction.**CPU_CNT: int**
Number of available CPUs. The value is used as the maximum concurrent workers in multiprocessing engines.

pcapkit.foundation.extraction.**LAYER_LIST = {'Application', 'Internet', 'Link', 'None', 'Tra**
List of layers.

pcapkit.foundation.extraction.**PROTO_LIST = {'ah', 'application', 'arp', 'drarp', 'ethernet**
List of protocols.

1.1.3 Trace TCP Flows

pcapkit.foundation.traceflow is the interface to trace TCP flows from a series of packets and connections.

Note: This was implemented as the demand of my mate @gousaiyang.

Data Structure

trace.packet Data structure for **TCP flow tracing** (*dump()*) is as following:

```
tract_dict = dict(
    protocol=data_link,                     # data link type from global header
    index=frame.info.number,                # frame number
    frame=frame.info,                       # extracted frame info
    syn=tcp.flags.syn,                      # TCP synchronise (SYN) flag
    fin=tcp.flags.fin,                      # TCP finish (FIN) flag
    src=ip.src,                             # source IP
    dst=ip.dst,                             # destination IP
    srcport=tcp.srcport,                    # TCP source port
    dstport=tcp.dstport,                    # TCP destination port
    timestamp=frame.info.time_epoch,        # frame timestamp
)
```

trace.buffer Data structure for internal buffering when performing reassembly algorithms (*_buffer*) is as following:

```
(dict) buffer --> memory buffer for reassembly
 |--> (tuple) BUFID : (dict)
 |       |--> ip.src      |
 |       |--> ip.dst      |
 |       |--> tcp.srcport |
 |       |--> tcp.dstport |
 |                         |--> 'fpout' : (dictdumper.dumper.Dumper) output dumper␣
↪object
 |                         |--> 'index': (list) list of frame index
 |                         |              |--> (int) frame index
 |                         |--> 'label': (str) flow label generated from ``BUFID``
 |--> (tuple) BUFID ...
```

trace.index Data structure for **TCP flow tracing** (element from *index tuple*) is as following:

```
(tuple) index
 |--> (Info) data
 |     |--> 'fpout' : (Optional[str]) output filename if exists
 |     |--> 'index': (tuple) tuple of frame index
 |     |              |--> (int) frame index
 |     |--> 'label': (str) flow label generated from ``BUFID``
 |--> (Info) data ...
```

Implementation

class pcapkit.foundation.traceflow.**TraceFlow**(*fout=None*, *format=None*, *byteorder='little'*, *nanosecond=False*)

Bases: object

Trace TCP flows.

__call__(*packet*)
Dump frame to output files.

Parameters **packet** (*Dict[str, Any]*) – a flow packet (*trace.packet*)

__init__(*fout=None*, *format=None*, *byteorder='little'*, *nanosecond=False*)
Initialise instance.

Parameters

- **fout** (*Optional[str]*) – output path
- **format** (*Optional[str]*) – output format
- **byteorder** (*str*) – output file byte order
- **nanosecond** (*bool*) – output nanosecond-resolution file flag

dump(*packet*)
Dump frame to output files.

Parameters **packet** (*Dict[str, Any]*) – a flow packet (*trace.packet*)

static make_fout(*fout='./tmp'*, *fmt='pcap'*)
Make root path for output.

Positional arguments: fout (str): root path for output fmt (str): output format

Returns dumper of specified format and file extension of output file

Return type Tuple[Type[dictdumper.dumper.Dumper], str]

Warns

- **FormatWarning** – If `fmt` is not supported.
- **FileWarning** – If `fout` exists and `fmt` is `None`.

Raises ***FileExists*** – If `fout` exists and `fmt` is **NOT** `None`.

submit()
Submit traced TCP flows.

Returns traced TCP flow (*trace.buffer*)

Return type Tuple[*Info*]

trace(*packet*, *, *check=True*, *output=False*)
Trace packets.

Parameters **packet** (*Dict[str, Any]*) – a flow packet (*trace.packet*)

Keyword Arguments

- **check** (*bool*) – flag if run validations
- **output** (*bool*) – flag if has formatted dumper

Returns If `output` is `True`, returns the initiated `Dumper` object, which will dump data to the output file named after the flow label; otherwise, returns the flow label itself.

Return type Union[dictdumper.dumper.Dumper, str]

Notes

The flow label is formatted as following:

```
f'{packet.src}_{packet.srcport}-{packet.dst}_{info.dstport}-{packet.timestamp}
↪'
```

_buffer
Buffer field (*trace.buffer*).

Type dict

_endian
Output file byte order.

Type Literal['little', 'big']

_fdpext
Output file extension.

Type str

_foutio
Dumper class.

Type Type[dictdumper.dumper.Dumper]

_fproot
Output root path.

Type str

_newflg
New packet flag.

Type bool

_nnsecd
Output nanosecond-resolution file flag.

Type bool

_stream
Stream index (*trace.index*).

Type list

property index
Index table for traced flow.

Return type Tuple[*Info*]

1.2 User Interface

pcapkit.interface defines several user-oriented interfaces, variables, and etc. These interfaces are designed to help and simplify the usage of *pcapkit*.

1.2.1 PCAP Extration

pcapkit.interface.**extract**(*fin=None*, *fout=None*, *format=None*, *auto=True*, *extension=True*, *store=True*, *files=False*, *nofile=False*, *verbose=False*, *engine=None*, *layer=None*, *protocol=None*, *ip=False*, *ipv4=False*, *ipv6=False*, *tcp=False*, *strict=True*, *trace=False*, *trace_fout=None*, *trace_format=None*, *trace_byteorder='little'*, *trace_nanosecond=False*)

Extract a PCAP file.

Parameters

- **fin** (*Optiona[str]*) – file name to be read; if file not exist, raise `FileNotFound`
- **fout** (*Optiona[str]*) – file name to be written
- **format** (*Optional[Literal['plist', 'json', 'tree']]*) – file format of output
- **auto** (*bool*) – if automatically run till EOF
- **extension** (*bool*) – if check and append extensions to output file
- **store** (*bool*) – if store extracted packet info
- **files** (*bool*) – if split each frame into different files
- **nofile** (*bool*) – if no output file is to be dumped
- **verbose** (*bool*) – if print verbose output information
- **engine** (*Optional[Literal['default', 'pcapkit', 'dpkt', 'scapy', 'pyshark', 'server', 'pipeline']]*) – extraction engine to be used
- **layer** (*Optional[Literal['Link', 'Internet', 'Transport', 'Application']]*) – extract til which layer
- **protocol** (*Optional[Union[str, Tuple[str], Type[Protocol]]]*) – extract til which protocol
- **ip** (*bool*) – if record data for IPv4 & IPv6 reassembly
- **ipv4** (*bool*) – if perform IPv4 reassembly
- **ipv6** (*bool*) – if perform IPv6 reassembly
- **tcp** (*bool*) – if perform TCP reassembly
- **strict** (*bool*) – if set strict flag for reassembly
- **trace** (*bool*) – if trace TCP traffic flows
- **trace_fout** (*Optional[str]*) – path name for flow tracer if necessary
- **trace_format** (*Optional[Literal['plist', 'json', 'tree', 'pcap']]*) – output file format of flow tracer
- **trace_byteorder** (*Literal['little', 'big']*) – output file byte order

- **trace_nanosecond** (*bool*) – output nanosecond-resolution file flag

Returns Extractor – an *Extractor* object

1.2.2 Application Layer Analysis

pcapkit.interface.**analyse**(*file*, *length=None*)
Analyse application layer packets.

Parameters

- **file** (*Union[bytes, io.BytesIO]*) – packet to be analysed
- **length** (*Optional[int]*) – length of the analysing packet

Returns an `Analysis` object

Return type Analysis

1.2.3 Payload Reassembly

pcapkit.interface.**reassemble**(*protocol*, *strict=False*)
Reassemble fragmented datagrams.

Parameters

- **protocol** (*Union[str, Type[Protocol]]*) –
- **strict** (*bool*) – if return all datagrams (including those not implemented) when submit

Returns a *Reassembly* object of corresponding protocol

Return type Union[*IPv4_Reassembly*, *IPv6_Reassembly*, *TCP_Reassembly*]

Raises ***FormatError*** – If `protocol` is **NOT** any of IPv4, IPv6 or TCP.

1.2.4 TCP Flow Tracing

pcapkit.interface.**trace**(*fout=None*, *format=None*, *byteorder='little'*, *nanosecond=False*)
Trace TCP flows.

Parameters

- **fout** (*str*) – output path
- **format** (*Optional[str]*) – output format
- **byteorder** (*str*) – output file byte order
- **nanosecond** (*bool*) – output nanosecond-resolution file flag

Returns a *TraceFlow* object

Return type *TraceFlow*

1.2.5 Output File Formats

pcapkit.interface.**TREE = 'tree'**

pcapkit.interface.**JSON = 'json'**

pcapkit.interface.**PLIST = 'plist'**

pcapkit.interface.**PCAP = 'pcap'**

1.2.6 Layer Thresholds

pcapkit.interface.**RAW = 'None'**

pcapkit.interface.**LINK = 'Link'**

pcapkit.interface.**INET = 'Internet'**

pcapkit.interface.**TRANS = 'Transport'**

pcapkit.interface.**APP = 'Application'**

1.2.7 Extration Engines

pcapkit.interface.**DPKT = 'dpkt'**

pcapkit.interface.**Scapy = 'scapy'**

pcapkit.interface.**PCAPKit = 'default'**

pcapkit.interface.**PyShark = 'pyshark'**

pcapkit.interface.**MPServer = 'server'**

pcapkit.interface.**MPPipeline = 'pipeline'**

1.3 Protocol Family

pcapkit.protocols is collection of all protocol families, with detailed implementation and methods.

1.3.1 PCAP File Headers

pcapkit.protocols.pcap contains header descriptions for PCAP files, including global header (*Header*) and frame header (*Frame*).

Global Header

pcapkit.protocols.pcap.header contains `Header` only, which implements extractor for global headers*[0] of PCAP, whose structure is described as below:

[0] https://wiki.wireshark.org/Development/LibpcapFileFormat#Global_Header

```
typedef struct pcap_hdr_s {
    guint32 magic_number;   /* magic number */
    guint16 version_major;  /* major version number */
    guint16 version_minor;  /* minor version number */
    gint32  thiszone;       /* GMT to local correction */
    guint32 sigfigs;        /* accuracy of timestamps */
    guint32 snaplen;        /* max length of captured packets, in octets */
    guint32 network;        /* data link type */
} pcap_hdr_t;
```

class pcapkit.protocols.pcap.header.**Header**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.protocol.Protocol*

PCAP file global header extractor.

classmethod __index__()

Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__len__()

Total length of corresponding protocol.

Return type Literal[24]

__length_hint__()

Return an estimated length for the object.

Return type Literal[24]

__post_init__(*file=None*, *length=None*, ***kwargs*)

Post initialisation hook.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

_decode_next_layer(**args*, ***kwargs*)

Decode next layer protocol.

Parameters ***args** – arbitrary positional arguments

Keyword Arguments ****kwargs** – arbitrary keyword arguments

Raises ***UnsupportedCall*** – This protocol doesn't support *_decode_next_layer()*.

_import_next_layer(**args*, ***kwargs*)

Import next layer extractor.

Parameters ***args** – arbitrary positional arguments

Keyword Arguments ****kwargs** – arbitrary keyword arguments

Raises ***UnsupportedCall*** – This protocol doesn't support *_import_next_layer()*.

_make_magic(***kwargs*)

Generate magic number.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Magic number and little-endian flag.

Return type Tuple[bytes, bool]

_read_protos(*size*)
Read next layer protocol type.

Parameters **size** (*int*) –

Returns link layer protocol enumeration

Return type *pcapkit.const.reg.linktype.LinkType*

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments

- **byteorder** (*str*) – header byte order
- **lilendian** (*bool*) – little-endian flag
- **bigendian** (*bool*) – big-endian flag
- **nanosecond** (*bool*) – nanosecond-resolution file flag (default: `False`)
- **version** (*Tuple[int, int]*) – version information (default: `(2, 4)`)
- **version_major** (*int*) – major version number (default: `2`)
- **version_minor** (*int*) – minor version number (default: `4`)
- **thiszone** (*int*) – GMT to local correction (default: `0`)
- **sigfigs** (*int*) – accuracy of timestamps (default: `0`)
- **snaplen** (*int*) – max length of captured packets, in octets (default: `262_144`)
- **network** (*Union[pcapkit.const.reg.linktype.LinkType, enum.IntEnum, str, int]*) – data link type (default: *DLT_NULL*)
- **network_default** (*int*) – default value for unknown data link type
- **network_namespace** (*Union[pcapkit.const.reg.linktype.LinkType, enum.IntEnum, Dict[str, int], Dict[int, str]]*) – data link type namespace (default: *LinkType*)
- **network_reversed** (*bool*) – if namespace is `str -> int` pairs (default: `False`)
- ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read global header of PCAP file.

Notes

PCAP file has **four** different valid magic numbers.

- `d4 c3 b2 a1` – Little-endian microsecond-timestamp PCAP file.
- `a1 b2 c3 d4` – Big-endian microsecond-timestamp PCAP file.

- `4d 3c b2 a1` – Little-endian nanosecond-timestamp PCAP file.
- `a1 b2 3c 4d` – Big-endian nano-timestamp PCAP file.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_Header*

Raises ***FileError*** – If the magic number is invalid.

property byteorder
Header byte order.

Return type Literal['big', 'little']

property length
Header length of corresponding protocol.

Return type Literal[24]

property name
Name of corresponding protocol.

Return type Literal['Global Header']

property nanosecond
Nanosecond-resolution flag.

Return type bool

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – This protocol doesn't support *payload*.

property protochain
Protocol chain of current instance.

Raises ***UnsupportedCall*** – This protocol doesn't support *protochain*.

property protocol
Data link type.

Return type *pcapkit.const.reg.linktype.LinkType*

property version
Version infomation of input PCAP file.

Return type *pcapkit.corekit.version.VersionInfo*

pcapkit.protocols.pcap.header.**_MAGIC_NUM = {('big', False): b'\xa1\xb2\xc3\xd4', ('big', '**
Mapping of PCAP file magic numbers.

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.pcap.header.**DataType_Header**

Bases TypedDict

PCAP global header.

magic_number: ***DataType_MagicNumber***
magic number

version_major: **int**
major version number

version_minor: **int**
minor version number

thiszone: **int**
GMT to local correction

sigfigs: **int**
accuracy of timestamps

snaplen: **int**
max length of captured packets, in octets

network: ***pcapkit.const.reg.linktype.LinkType***
data link type

class pcapkit.protocols.pcap.header.**DataType_MagicNumber**

Bases TypedDict

PCAP magic number.

data: **bytes**
original magic number

byteorder: **str**
byte order (`big` / `little`)

nanosecond: **bool**
nanosecond-timestamp support

Frame Header*[0]

**pcapkit.protocols.pcap.frame* contains *Frame* only, which implements extractor for frame headers of PCAP, whose structure is described as below:

```
typedef struct pcaprec_hdr_s {
    guint32 ts_sec;     /* timestamp seconds */
    guint32 ts_usec;    /* timestamp microseconds */
    guint32 incl_len;   /* number of octets of packet saved in file */
    guint32 orig_len;   /* actual length of packet */
} pcaprec_hdr_t;
```

[0] https://wiki.wireshark.org/Development/LibpcapFileFormat#Record_.28Packet.29_Header

class pcapkit.protocols.pcap.frame.**Frame**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.protocol.Protocol*

Per packet frame header extractor.

__proto__: **DefaultDict[int, Tuple[str, str]]**
Protocol index mapping for decoding next layer, c.f. *self._decode_next_layer* & *self._import_next_layer*. The values should be a tuple representing the module name and class name.

Code	Module	Class
1	*pcapkit.protocols.link.ethernet*	*Ethernet*
228	pcapkit.protocols.link.internet.ipv4	IPv4
229	pcapkit.protocols.link.internet.ipv6	IPv6

__contains__(*name*)
Returns if name is in self._info or in the frame packet self._protos.

Parameters **name** (*Any*) – name to search

Returns if name exists

Return type bool

__getitem__(*key*)
Subscription (getitem) support.

This method fist checks if key exists in self._info. If so, returns the corresponding value, else calls the original *__getitem__()* method.

Parameters **key** (*Union[str, Protocol, Type[Protocol]]*) – Indexing key.

Returns

- If key exists in self._info, returns the value of the key;
- else returns the sub-packet from the current packet of indexed protocol.

__index__()
Index of the protocol.

Returns If the object is initiated, i.e. self._fnum exists, returns the frame index number of itself; else raises UnsupportedCall.

Return type int

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__length_hint__()
Return an estimated length for the object.

Return type Literal[16]

__post_init__(*file=None*, *length=None*, ***, *num*, *proto*, *nanosecond*, ***kwargs*)
Initialisation.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **num** (*int*) – Frame index number (self._fnum).

- **proto** (pcapkit.const.reg.linktype.LinkType) – Next layer protocol index (self._prot).
- **nanosecond** (*bool*) – Nanosecond-timestamp PCAP flag (self._nsec).
- **mpfdp** (*multiprocessing.Queue*) – Multiprocessing file descriptor queue (self._mpfp).
- **mpkit** (*multiprocessing.Namespace*) – Multiprocessing auxiliaries (self._mpkt).
- ****kwargs** – Arbitrary keyword arguments.

For *multiprocessing* related parameters, please refer to pcapkit.foundation.extration.Extrator for more information.

See also:

For construction argument, please refer to *make()*.

_decode_next_layer(*data*, *length=None*)
Decode next layer protocol.

Positional arguments: data (dict): info buffer length (int): valid (*non-padding*) length

Returns current protocol with packet extracted

Return type dict

_import_next_layer(*proto*, *length*, *error=False*)
Import next layer extractor.

This method currently supports following protocols as registered in *LinkType*:

proto	Protocol
1	*Ethernet*
228	*IPv4*
229	*IPv6*

Parameters

- **proto** (pcapkit.const.reg.linktype.LinkType) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Keyword Arguments **error** (*bool*) – if function called on error

Returns instance of next layer

Return type *pcapkit.protocols.protocol.Protocol*

_make_timestamp(***kwargs*)
Make timestamp.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Second and microsecond/nanosecond value of timestamp.

Return type Tuple[int, int]

index(*name*)
Call *ProtoChain.index*.

Parameters **name** (*Union[str, Protocol, Type[Protocol]]*) – name to be searched

Returns first index of name

Return type int

Raises ***IndexNotFound*** – if name is not present

make(***kwargs*)

Make frame packet data.

Keyword Arguments

- **timestamp** (*float*) – UNIX-Epoch timestamp
- **ts_sec** (*int*) – timestamp seconds
- **ts_usec** (*int*) – timestamp microseconds
- **incl_len** (*int*) – number of octets of packet saved in file
- **orig_len** (*int*) – actual length of packet
- **packet** (*bytes*) – raw packet data (default: b'')
- **nanosecond** (*bool*) – nanosecond-resolution file flag (default: False)
- ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)

Read each block after global header.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_Frame*

Raises **EOFError** – If self._file reaches EOF.

classmethod register(*code*, *module*, *class_*)

Register a new protocol class.

Parameters

- **code** (*int*) – protocol code as in *LinkType*
- **module** (*str*) – module name
- **class** (*str*) – class name

Notes

The full qualified class name of the new protocol class should be as {module}.{class_}.

---

property length

Header length of corresponding protocol.

Return type Literal[16]

property name
Name of corresponding protocol.

Return type str

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.pcap.frame.**DataType_Frame**

Bases TypedDict

PCAP frame header.

frame_info: *DataType_FrameInfo*
PCAP frame information

time: datetime.datetime
timestamp

number: int
frame index number

time_epoch: float
EPOCH timestamp

len: int
captured packet length

cap_len: int
actual packet length

packet: bytes
packet raw data

protocols: *pcapkit.corekit.protochain.ProtoChain*
protocol chain

error: typing.Optional[str]
error message (optional)

class pcapkit.protocols.pcap.frame.**DataType_FrameInfo**

Bases TypedDict

Frame information.

ts_sec: int
timestamp seconds

ts_usec: int
timestamp microseconds/nanoseconds

incl_len: int
number of octets of packet saved in file

orig_len: int
actual length of packet

1.3.2 Link Layer Protocols

pcapkit.protocols.link is collection of all protocols in link layer, with detailed implementation and methods.

ARP/InARP - (Inverse) Address Resolution Protocol

pcapkit.protocols.link.arp contains *ARP* only, which implements extractor for (Inverse) Address Resolution Protocol (ARP/InARP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	arp.htype	Hardware Type
2	16	arp.ptype	Protocol Type
4	32	arp.hlen	Hardware Address Length
5	40	arp.plen	Protocol Address Length
6	48	arp.oper	Operation
8	64	arp.sha	Sender Hardware Address
14	112	arp.spa	Sender Protocol Address
18	144	arp.tha	Target Hardware Address
24	192	arp.tpa	Target Protocol Address

class pcapkit.protocols.link.arp.**ARP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.link.link.Link*

This class implements all protocols in ARP family.

- Address Resolution Protocol (ARP) [**RFC 826**]
- Reverse Address Resolution Protocol (RARP) [**RFC 903**]
- Dynamic Reverse Address Resolution Protocol (DRARP) [**RFC 1931**]
- Inverse Address Resolution Protocol (InARP) [**RFC 2390**]

_acnm: Literal[*ARP*, InARP, RARP, DRARP]
Acronym of corresponding protocol.

The value is based on operation type (*oper*).

_name: Literal[Dynamic Reverse Address Resolution Protocol, Inverse Address Resolutio
Name of current protocol.

The value is based on operation type (*oper*).

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.ethertype.EtherType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[28]

_read_addr_resolve(*length*, *htype*)
Resolve headware address according to protocol.

Parameters

[0] http://en.wikipedia.org/wiki/Address_Resolution_Protocol

- **length** (*int*) – Hardware address length.
- **htype** (*int*) – Hardware type.

Returns Hardware address. If `htype` is `1`, i.e. MAC address, returns `:` seperated *hex* encoded MAC address.

Return type str

_read_proto_resolve(*length*, *ptype*)
Resolve protocol address according to protocol.

Positional arguments: length (int): Protocol address length. ptype (int): Protocol type.

Returns Protocol address. If `ptype` is `0x0800`, i.e. IPv4 adddress, returns an `IPv4Address` object; if `ptype` is `0x86dd`, i.e. IPv6 address, returns an `IPv6Address` object; otherwise, returns a raw `str` representing the protocol address.

Return type Union[ipaddress.IPv4Address, ipaddress.IPv6Address, str]

classmethod id()
Index ID of the protocol.

Returns Index ID of the protocol.

Return type Tuple[Literal['ARP'], Literal['InARP']]

See also:

pcapkit.protocols.protocol.Protocol.__getitem__()

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Address Resolution Protocol [**RFC 826**].

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_ARP*

property alias
Acronym of corresponding protocol.

Return type Literal['ARP', 'InARP', 'RARP', 'DRARP']

property dst
Target hardware & protocol address.

Return type Tuple[str, Union[ipaddress.IPv4Address, ipaddress.IPv6Address, str]]

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Dynamic Reverse Address Resolution Protocol', 'Inverse Address Resolution Protocol', 'Reverse Address Resolution Protocol', 'Address Resolution Protocol']

property src
Sender hardware & protocol address.

Return type Tuple[str, Union[ipaddress.IPv4Address, ipaddress.IPv6Address, str]]

property type
Hardware & protocol type.

Return type Tuple[*pcapkit.const.arp.hardware.Hardware*, *pcapkit.const.reg.ethertype.EtherType*]

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.link.arp.**DataType_ARP**

Bases TypedDict

ARP header [**RFC 826**].

htype: **pcapkit.const.arp.Headware**
hardware type

ptype: **Union[*pcapkit.const.reg.ethertype.EtherType*, str]**
protocol type

hlen: **int**
headware address length

plen: **int**
protocol address length

oper: ***pcapkit.const.arp.operation.Operation***
operation

sha: **str**
sender hardware address

Ethernet Protocol

`pcapkit.protocols.link.ethernet` contains *Ethernet* only, which implements extractor for Ethernet Protocol*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	eth.dst	Destination MAC Address
1	8	eth.src	Source MAC Address
2	16	eth.type	Protocol (Internet Layer)

[0] https://en.wikipedia.org/wiki/Ethernet

class pcapkit.protocols.link.ethernet.**Ethernet**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.link.link.Link*

This class implements Ethernet Protocol.

classmethod __index__()
Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__length_hint__()
Return an estimated length for the object.

Return type Literal[14]

_read_mac_addr()
Read MAC address.

Returns Colon (**:**) seperated *hex* encoded MAC address.

Return type str

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments *****kwargs*** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Ethernet Protocol [**RFC 7042**].

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments *****kwargs*** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_Ethernet*

property dst
Destination mac address.

Return type str

property length
Header length of current protocol.

Return type Literal[14]

property name
Name of current protocol.

Return type Literal['Ethernet Protocol']

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.ethertype.EtherType*

property src
Source mac address.

Return type str

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.link.ethernet.**DataType_Ethernet**

Bases TypedDict

Ethernet header.

dst: **str**
destination MAC address

src: **str**
source MAC address

type: ***pcapkit.const.reg.ethertype.EtherType***
protocol (Internet layer)

L2TP - Layer Two Tunnelling Protocol

`pcapkit.protocols.link.l2tp` contains `L2TP` only, which implements extractor for Layer Two Tunnelling Protocol (L2TP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`l2tp.flags`	Flags and Version Info
0	0	`l2tp.flags.type`	Type (control / data)
0	1	`l2tp.flags.len`	Length
0	2		Reserved (must be zero `x00`)
0	4	`l2tp.flags.seq`	Sequence
0	5		Reserved (must be zero `x00`)
0	6	`l2tp.flags.offset`	Offset
0	7	`l2tp.flags.prio`	Priority
1	8		Reserved (must be zero `x00`)
1	12	`l2tp.ver`	Version (`2`)
2	16	`l2tp.length`	Length (optional by `len`)
4	32	`l2tp.tunnelid`	Tunnel ID
6	48	`l2tp.sessionid`	Session ID
8	64	`l2tp.ns`	Sequence Number (optional by `seq`)
10	80	`l2tp.nr`	Next Sequence Number (optional by `seq`)
12	96	`l2tp.offset`	Offset Size (optional by `offset`)

class pcapkit.protocols.link.l2tp.**L2TP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.link.link.Link*

This class implements Layer Two Tunnelling Protocol.

classmethod **__index__**()
Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

[0] https://en.wikipedia.org/wiki/Layer_2_Tunneling_Protocol

__length_hint__()
Return an estimated length for the object.

Return type Literal[16]

make(***kwargs*)
Make (construct) packet data.

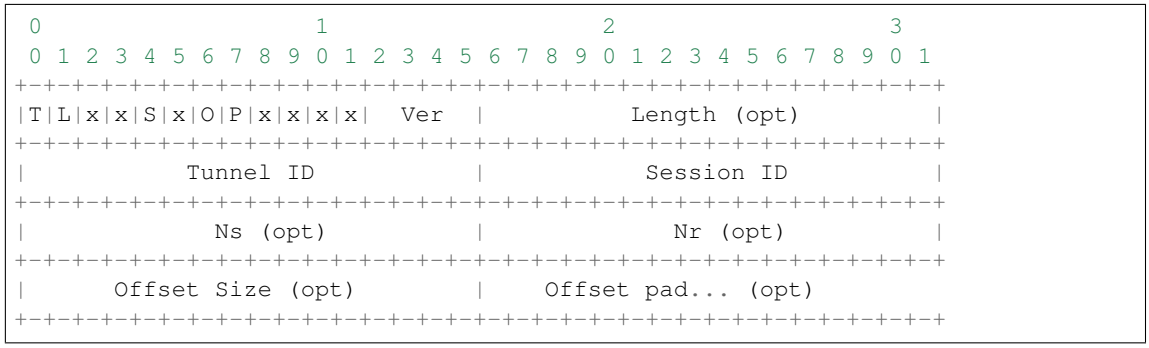
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Layer Two Tunnelling Protocol.

Structure of L2TP header [**RFC 2661**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_L2TP*

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Layer 2 Tunnelling Protocol']

property type
L2TP type.

Return type Literal['Control', 'Data']

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.link.l2tp.**DataType_L2TP**

Bases TypedDict

L2TP header.

flags: **DataTYpe_Flags**
flags & versoion info

version: **Literal[2]**
version (2)

length: **Optional[int]**
length (optional by *len*)

tunnelid: **int**
tunnel ID

sessionid: **int**
session ID

ns: **Optional[int]**
sequence number (optional by *seq*)

nr: **Optional[int]**
next sequence number (optional by *seq*)

offset: **Optional[int]**
offset (optional by *offset*)

class pcapkit.protocols.link.l2tp.**DataType_Flags**

Bases TypedDict

Flags and version info.

type: **Literal[Control, Data]**
type (control / data)

len: **bool**
length

seq: **bool**
sequence

offset: **bool**
offset

prio: **bool**
priority

OSPF - Open Shortest Path First

pcapkit.protocols.link.ospf contains *OSPF* only, which implements extractor for Open Shortest Path First (OSPF)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`ospf.version`	Version Number
0	0	`ospf.type`	Type
0	1	`ospf.len`	Packet Length (header included)
0	2	`ospf.router_id`	Router ID
0	4	`ospf.area_id`	Area ID
0	6	`ospf.chksum`	Checksum
0	7	`ospf.autype`	Authentication Type
1	8	`ospf.auth`	Authentication

class pcapkit.protocols.link.ospf.**OSPF**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.link.link.Link*

This class implements Open Shortest Path First.

classmethod __index__()

Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__length_hint__()

Return an estimated length for the object.

Return type Literal[24]

_read_encrypt_auth()

Read Authentication field when Cryptographic Authentication is employed, i.e. `autype` is `2`.

Structure of Cryptographic Authentication [**RFC 2328**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               0                 |    Key ID     | Auth Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Cryptographic sequence number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **length** (*int*) – packet length

Returns

Parsed packet data.

class Auth(TypedDict): """Cryptographic authentication."""

#: key ID key_id: int #: authentication data length len: int #: cryptographic sequence number seq: int

Return type *DataType_Auth*

_read_id_numbers()

Read router and area IDs.

[0] https://en.wikipedia.org/wiki/Open_Shortest_Path_First

Returns Parsed IDs as an IPv4 address.

Return type IPv4Address

make(***kwargs*)
Make (construct) packet data.

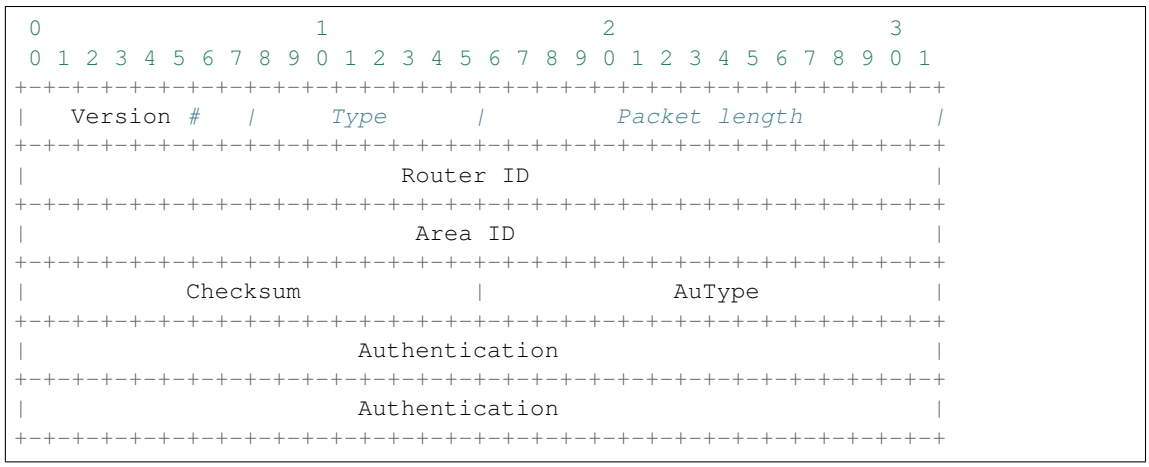
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Open Shortest Path First.

Structure of OSPF header [**RFC 2328**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_OSPF*

property alias
Acronym of current protocol.

Return type str

property length
Header length of current protocol.

Return type Literal[24]

property name
Name of current protocol.

Return type str

property type
OSPF packet type.

Return type *pcapkit.const.ospf.packet.Packet*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.link.ospf.**DataType_OSPF**

Bases TypedDict

OSPF header.

version: **int**
version number

type: ***pcapkit.const.ospf.packet.Packet***
type

len: **int**
packet length (header included)

router_id: **ipaddress.IPv4Address**
router ID

area_id: **ipaddress.IPv4Address**
area ID

chksum: **bytes**
checksum

autype: ***pcapkit.const.ospf.authentication.Authentication***
authentication type

auth: **Union[bytes, *DataType_Auth*]**
authentication

Cryptographic Authentication Information

For cryptographic authentication information as described in **RFC 2328**, its structure is described as below:

Octets	Bits	Name	Description
0	0		Reserved (must be zero `\x00`)
0	0	`ospf.auth.key_id`	Key ID
0	1	`ospf.auth.len`	Authentication Data Length
0	2	`ospf.auth.seq`	Cryptographic Sequence Number

class pcapkit.protocols.link.ospf.**DataType_Auth**

Bases TypedDict

Cryptographic authentication.

key_id: **int**
key ID

len: **int**
authentication data length

seq: **int**
cryptographic sequence number

RARP/DRARP - (Dynamic) Reverse Address Resolution Protocol

pcapkit.protocols.link.rarp contains *RARP* only, which implements extractor for (Dynamic) Reverse Address Resolution Protocol (RARP/DRARP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`rarp.htype`	Hardware Type
2	16	`rarp.ptype`	Protocol Type
4	32	`rarp.hlen`	Hardware Address Length
5	40	`rarp.plen`	Protocol Address Length
6	48	`rarp.oper`	Operation
8	64	`rarp.sha`	Sender Hardware Address
14	112	`rarp.spa`	Sender Protocol Address
18	144	`rarp.tha`	Target Hardware Address
24	192	`rarp.tpa`	Target Protocol Address

class pcapkit.protocols.link.rarp.**RARP**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.link.arp.ARP*

This class implements Reverse Address Resolution Protocol.

classmethod __index__()

Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.ethertype.EtherType*

classmethod id()

Index ID of the protocol.

Returns Index ID of the protocol.

Return type Tuple[Literal['RARP'], Literal['DRARP']]

See also:

pcapkit.protocols.protocol.Protocol.__getitem__()

_acnm = 'RARP'

Acronym of corresponding protocol.

_name = 'Reverse Address Resolution Protocol'

Name of corresponding protocol.

VLAN - 802.1Q Customer VLAN Tag Type

pcapkit.protocols.link.vlan contains *VLAN* only, which implements extractor for 802.1Q Customer VLAN Tag Type*[0], whose structure is described as below:

Octets	Bits	Name	Description
1	0	`vlan.tci`	Tag Control Information
1	0	`vlan.tci.pcp`	Priority Code Point
1	3	`vlan.tci.dei`	Drop Eligible Indicator
1	4	`vlan.tci.vid`	VLAN Identifier
3	24	`vlan.type`	Protocol (Internet Layer)

[0] http://en.wikipedia.org/wiki/Address_Resolution_Protocol

[0] https://en.wikipedia.org/wiki/IEEE_802.1Q

class pcapkit.protocols.link.vlan.**VLAN**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.link.link.Link*

This class implements 802.1Q Customer VLAN Tag Type.

classmethod __index__()
Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__length_hint__()
Return an estimated length for the object.

Return type Literal[4]

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read 802.1Q Customer VLAN Tag Type.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_VLAN*

property alias
Acronym of corresponding protocol.

Return type Literal['802.1Q']

property length
Header length of current protocol.

Return type Literal[4]

property name
Name of current protocol.

Return type Literal['802.1Q Customer VLAN Tag Type']

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.ethertype.EtherType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.link.vlan.**DataType_VLAN**

Bases TypedDict

IEEE 802.1Q customer VLAN tag type [RFC 7042].

tci: *DataType_TCI*
Tag control information.

type: *pcapkit.const.reg.ethertype.EtherType*
Protocol (internet layer).

class pcapkit.protocols.link.vlan.**DataType_TCI**

Bases TypedDict

Tag control information.

pcp: *pcapkit.const.vlan.priority_level.PriorityLevel*
Priority code point.

dei: **bool**
Drop eligible indicator.

vid: **int**
VLAN identifier.

Base Protocol

pcapkit.protocols.link.link contains *Link*, which is a base class for link layer protocols, e.g. ARP/InARP, Ethernet, L2TP, OSPF, RARP/DRARP and etc.

class pcapkit.protocols.link.link.**Link**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.protocol.Protocol*

Abstract base class for link layer protocol family.

__layer__ = 'Link'
Layer of protocol.

__proto__: **DefaultDict[int, Tuple[str, str]]**
Protocol index mapping for decoding next layer, c.f. *self._decode_next_layer* & *self._import_next_layer*. The values should be a tuple representing the module name and class name.

Code	Module	Class
0x0806	*pcapkit.protocols.link.arp*	*ARP*
0x8035	*pcapkit.protocols.link.rarp*	*RARP*
0x8100	*pcapkit.protocols.link.vlan*	*VLAN*
0x0800	*pcapkit.protocols.internet.ipv4*	*IPv4*
0x86DD	*pcapkit.protocols.internet.ipv6*	*IPv6*
0x8137	*pcapkit.protocols.internet.ipx*	*IPX*

_import_next_layer(*proto*, *length=None*)
Import next layer extractor.

This method currently supports following protocols as registered in *EtherType*:

proto	Protocol
0x0806	*ARP*
0x8035	*RARP*
0x8100	*VLAN*
0x0800	*IPv4*
0x86DD	*IPv6*
0x8137	*IPX*

Parameters

- **proto** (*int*) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Returns instance of next layer

Return type *pcapkit.protocols.protocol.Protocol*

_read_protos(*size*)
Read next layer protocol type.

Parameters **size** (*int*) – buffer size

Returns next layer's protocol enumeration

Return type *pcapkit.const.reg.ethertype.EtherType*

classmethod register(*code*, *module*, *class_*)
Register a new protocol class.

Parameters

- **code** (*int*) – protocol code as in *EtherType*
- **module** (*str*) – module name
- **class** (*str*) – class name

Notes

The full qualified class name of the new protocol class should be as `{module}.{class_}`.

---

property layer
Protocol layer.

Return type Literal['Link']

1.3.3 Internet Layer Protocols

pcapkit.protocols.internet is collection of all protocols in internet layer, with detailed implementation and methods.

AH - Authentication Header

pcapkit.protocols.internet.ah contains AH only, which implements extractor for Authentication Header (AH)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	ah.next	Next Header
1	8	ah.length	Payload Length
2	16		Reserved (must be zero)
4	32	sah.spi	Security Parameters Index (SPI)
8	64	sah.seq	Sequence Number Field
12	96	sah.icv	Integrity Check Value (ICV)

class pcapkit.protocols.internet.ah.**AH**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.internet.ipsec.IPsec*

This class implements Authentication Header.

classmethod __index__()

Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()

Return an estimated length for the object.

Return type Literal[20]

__post_init__(*file*, *length=None*, ***, *version=4*, *extension=False*, ***kwargs*)

Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **version** (*Literal[4, 6]*) – IP protocol version.
- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

classmethod id()

Index ID of the protocol.

Returns Index ID of the protocol.

[0] https://en.wikipedia.org/wiki/IPsec

Return type Literal['AH']

make (***kwargs*)
Make (construct) packet data.

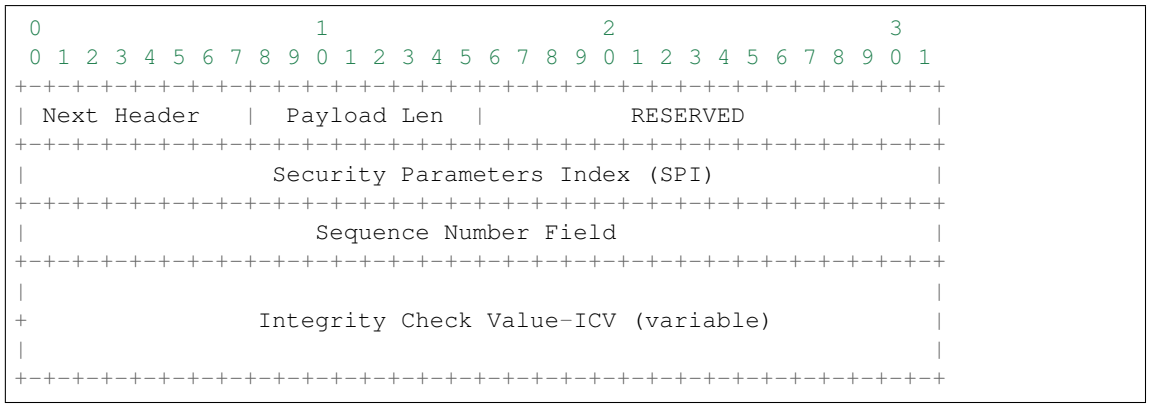
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read (*length=None*, ***, *version=4*, *extension=False*, ***kwargs*)
Read Authentication Header.

Structure of AH header [**RFC 4302**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **version** (*Literal[4, 6]*) – IP protocol version.
- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_AH*

property length
Info dict of current instance.

Return type int

property name
Name of corresponding protocol.

Return type Literal['Authentication Header']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.ah.**DataType_AH**

Bases TypedDict

Authentication header [**RFC 4302**].

next: *pcapkit.const.reg.transtype.TransType*
Next header.

length: *int*
Payload length.

spi: *int*
Security parameters index (SPI).

seq: *int*
Sequence number field.

icv: *int*
Integrity check value (ICV).

HIP - Host Identity Protocol

`pcapkit.protocols.internet.hip` contains `HIP` only, which implements extractor for Host Identity Protocol (HIP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`hip.next`	Next Header
1	8	`hip.length`	Header Length
2	16		Reserved (`\x00`)
2	17	`hip.type`	Packet Type
3	24	`hip.version`	Version
3	28		Reserved
3	31		Reserved (`\x01`)
4	32	`hip.chksum`	Checksum
6	48	`hip.control`	Controls
8	64	`hip.shit`	Sender's Host Identity Tag
24	192	`hip.rhit`	Receiver's Host Identity Tag
40	320	`hip.parameters`	HIP Parameters

class pcapkit.protocols.internet.hip.**HIP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements Host Identity Protocol.

classmethod **__index__**()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

[0] https://en.wikipedia.org/wiki/Host_Identity_Protocol

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[40]

__post_init__(*file*, *length=None*, ***, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

_read_hip_para(*length*, ***, *version*)
Read HIP parameters.

Parameters **length** (*int*) – length of parameters

Keyword Arguments **version** (*Litreal[1, 2]*) – HIP version

Returns extracted HIP parameters

Return type Tuple[Tuple[*pcapkit.const.hip.parameter.Parameter*], *DataType_Parameter*]

Raises ***ProtocolError*** – if packet length threshold check failed

_read_para_ack(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP ACK parameter.

Structure of HIP ACK parameter [**RFC 7401**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Type               |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         peer Update ID 1                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                         peer Update ID n                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (*pcapkit.const.hip.parameter.Parameter*) – parameter type
- **length** (*int*) – remaining packet length

- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

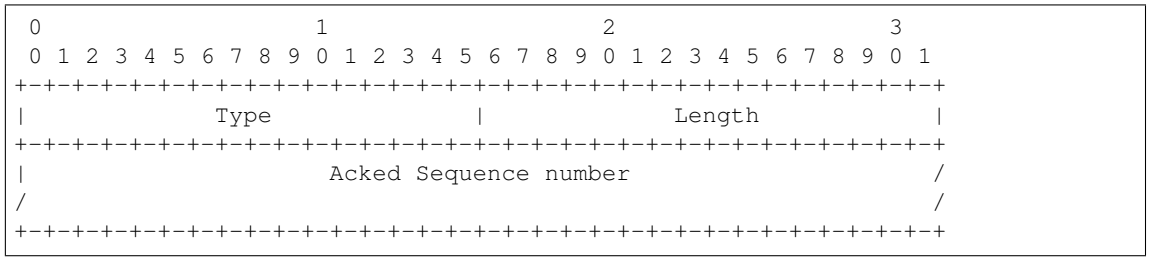
Return type *DataType_Param_ACK*

Raises ***ProtocolError*** – If `clen` is **NOT** `4` modulo.

_read_para_ack_data(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `ACK_DATA` parameter.

Structure of HIP `ACK_DATA` parameter [**RFC 6078**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

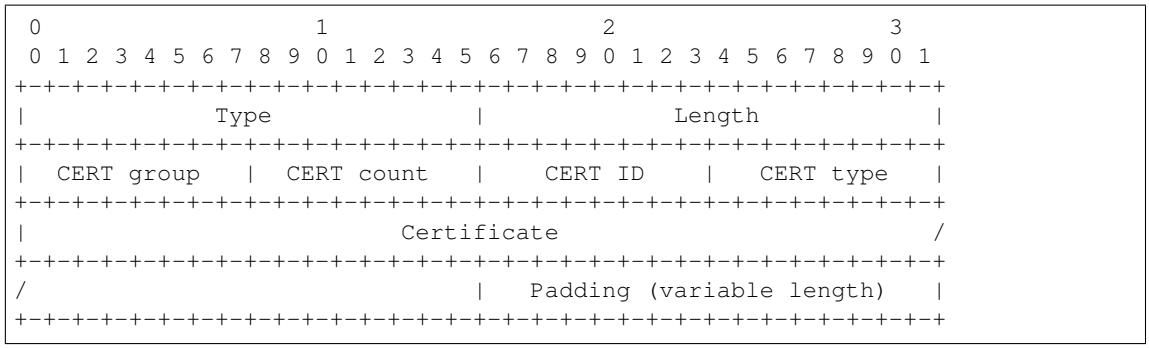
Return type *DataType_Param_ACK_Data*

Raises ***ProtocolError*** – If `clen` is **NOT** `4` modulo.

_read_para_cert(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `CERT` parameter.

Structure of HIP `CERT` parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Cert*

_read_para_dh_group_list(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP DH_GROUP_LIST parameter.

Structure of HIP DH_GROUP_LIST parameter [**RFC 7401**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DH GROUP ID #1| DH GROUP ID #2| DH GROUP ID #3| DH GROUP ID #4|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DH GROUP ID #n|                Padding                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_DH_Group_List*

_read_para_diffie_hellman(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

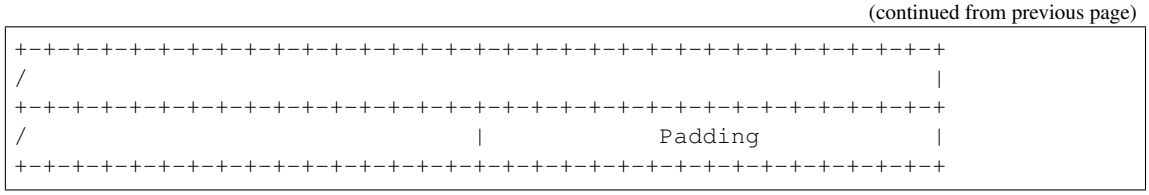
Read HIP DIFFIE_HELLMAN parameter.

Structure of HIP DIFFIE_HELLMAN parameter [**RFC 7401**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Group ID    |      Public Value Length      | Public Value  /
```

(continues on next page)

(continued from previous page)



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

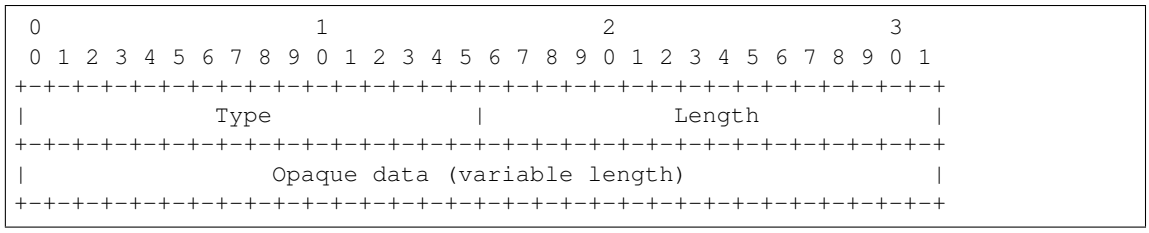
Returns Parsed parameter data.

Return type DataType_Param_Diffie_Hellman

_read_para_echo_request_signed(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP ECHO_REQUEST_SIGNED parameter.

Structure of HIP ECHO_REQUEST_SIGNED parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

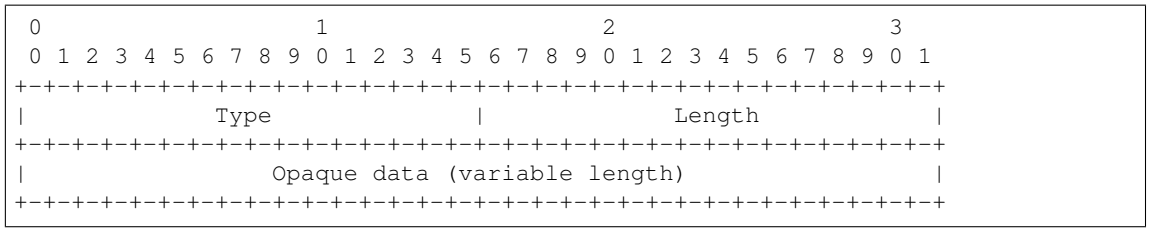
Returns Parsed parameter data.

Return type *DataType_Param_Echo_Request_Signed*

_read_para_echo_request_unsigned(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP ECHO_REQUEST_UNSIGNED parameter.

Structure of HIP ECHO_REQUEST_UNSIGNED parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

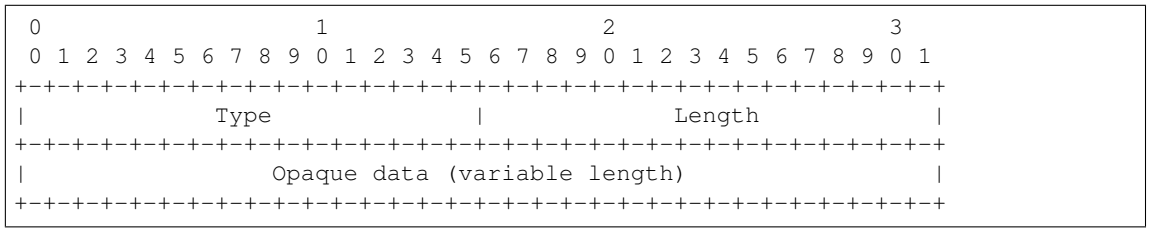
Returns Parsed parameter data.

Return type *DataType_Param_Echo_Request_Unsigned*

_read_para_echo_response_signed(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP ECHO_RESPONSE_SIGNED parameter.

Structure of HIP ECHO_RESPONSE_SIGNED parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

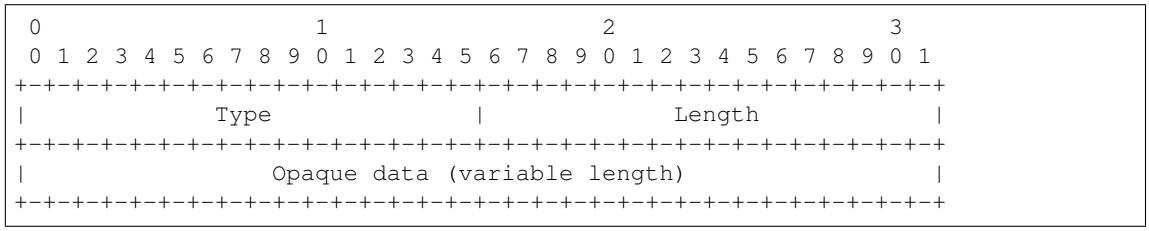
- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Echo_Response_Signed*

_read_para_echo_response_unsigned(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP ECHO_RESPONSE_UNSIGNED parameter.

Structure of HIP ECHO_RESPONSE_UNSIGNED parameter [RFC 7401]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

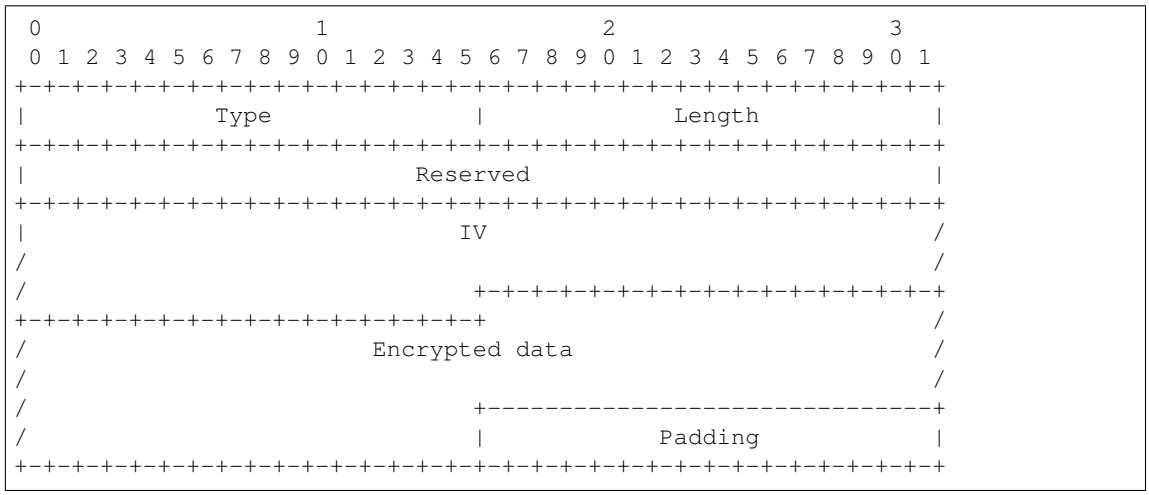
- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Echo_Response_Unsigned*

_read_para_encrypted(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP ENCRYPTED parameter.

Structure of HIP ENCRYPTED parameter [RFC 7401]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

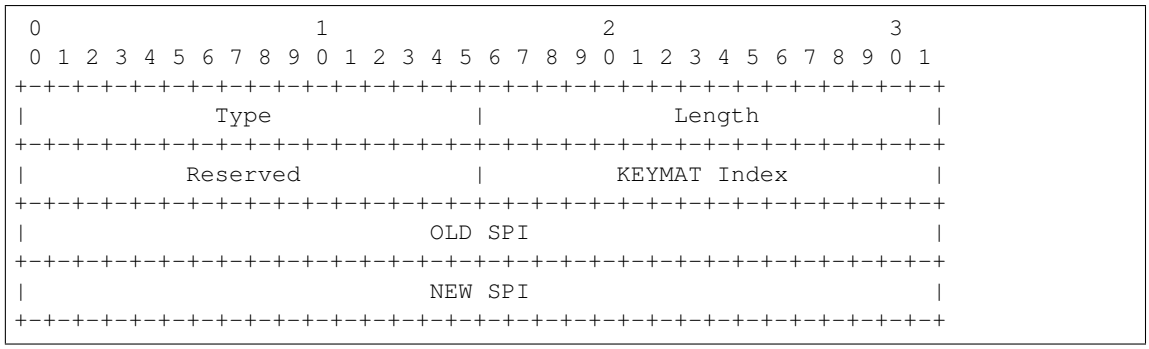
Returns Parsed parameter data.

Return type *DataType_Param_Encrypted*

_read_para_esp_info(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP ESP_INFO parameter.

Structure of HIP ESP_INFO parameter [**RFC 7402**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

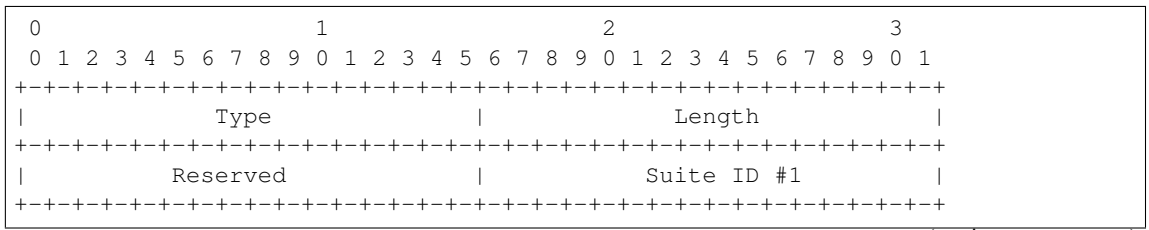
Return type *DataType_Param_ESP_Info*

Raises ***ProtocolError*** – If `clen` is **NOT** `12`.

_read_para_esp_transform(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

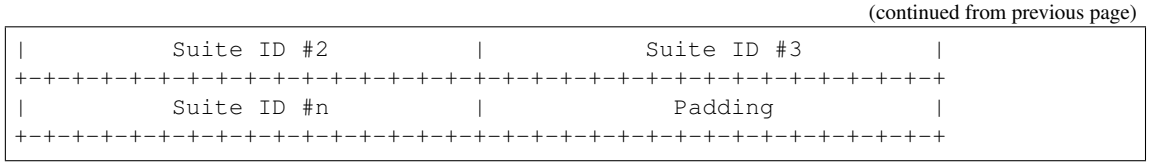
Read HIP ESP_TRANSFORM parameter.

Structure of HIP ESP_TRANSFORM parameter [**RFC 7402**]:



(continues on next page)

(continued from previous page)



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

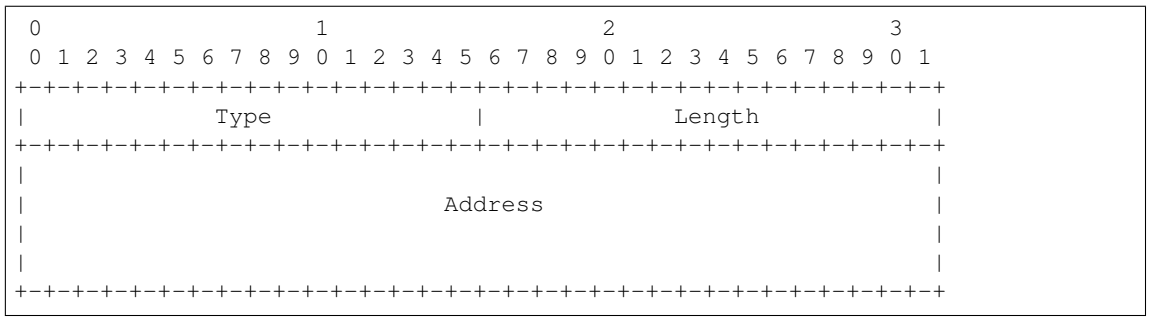
Return type *DataType_Param_Transform_Format_List*

Raises ***ProtocolError*** – If `clen` is **NOT** `2` modulo.

_read_para_from(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `FROM` parameter.

Structure of HIP `FROM` parameter [**RFC 8004**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

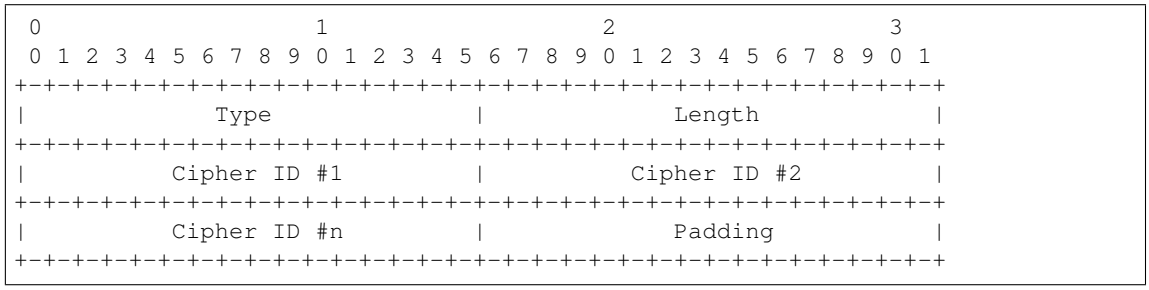
Returns Parsed parameter data.

Return type *DataType_Param_From*

Raises ***ProtocolError*** – If `clen` is **NOT** `16`.

_read_para_hip_cipher(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `HIP_CIPHER` parameter.

Structure of HIP `HIP_CIPHER` parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

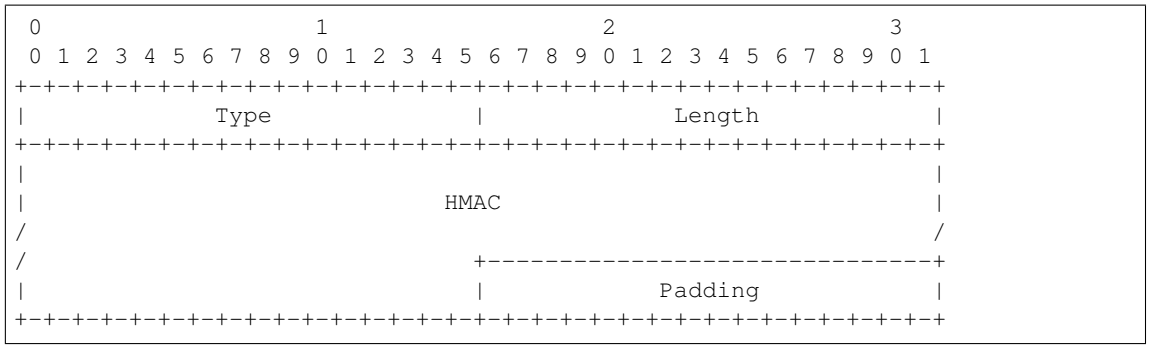
Returns Parsed parameter data.

Return type *DataType_Param_Cipher*

Raises ***ProtocolError*** – If `clen` is **NOT** a `2` modulo.

_read_para_hip_mac(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `HIP_MAC` parameter.

Structure of HIP `HIP_MAC` parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

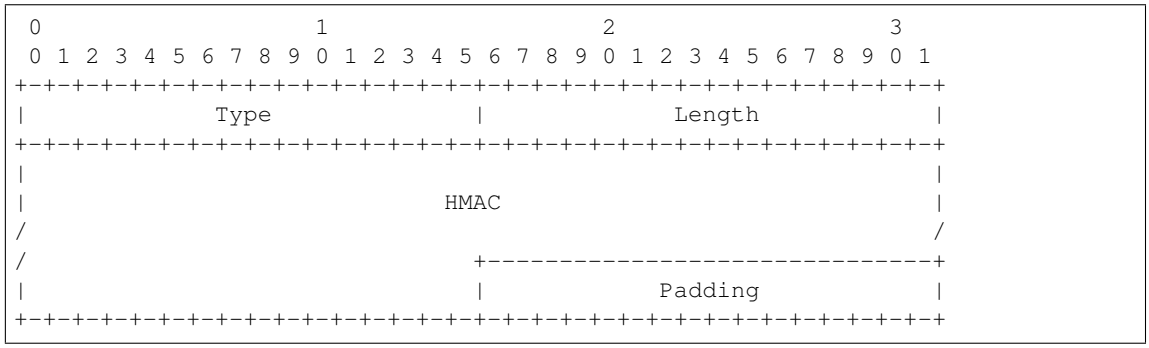
Returns Parsed parameter data.

Return type *DataType_Param_HMAC*

_read_para_hip_mac_2(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP HIP_MAC_2 parameter.

Structure of HIP HIP_MAC_2 parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

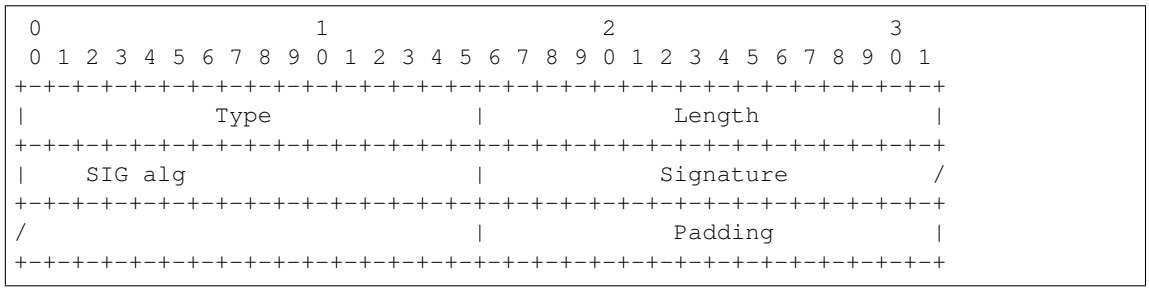
Returns Parsed parameter data.

Return type *DataType_Param_HMAC_2*

_read_para_hip_signature(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP HIP_SIGNATURE parameter.

Structure of HIP HIP_SIGNATURE parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

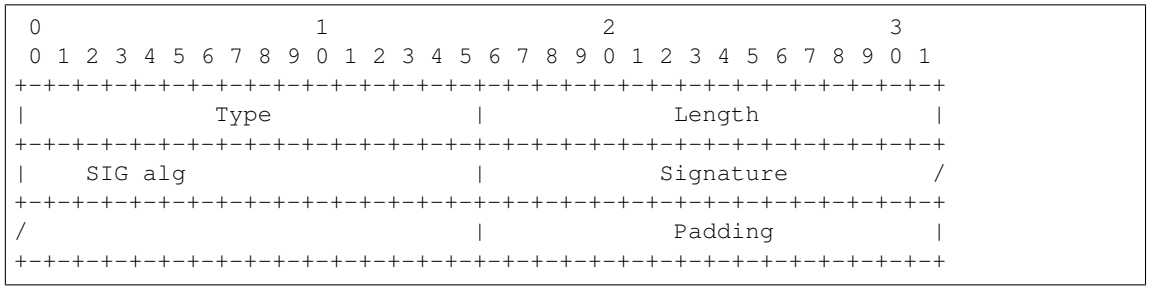
Returns Parsed parameter data.

Return type *DataType_Param_Signature*

_read_para_hip_signature_2(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP HIP_SIGNATURE_2 parameter.

Structure of HIP HIP_SIGNATURE_2 parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

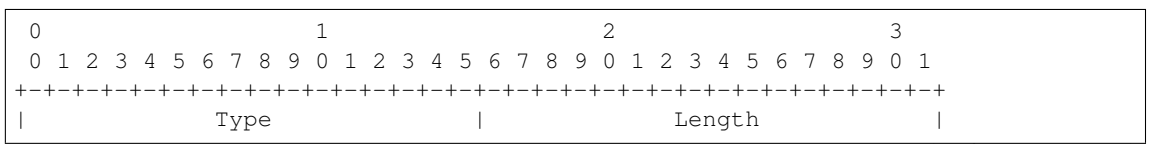
Returns Parsed parameter data.

Return type *DataType_Param_Signature_2*

_read_para_hip_transform(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP HIP_TRANSFORM parameter.

Structure of HIP HIP_TRANSFORM parameter [**RFC 5201**]:



(continues on next page)

(continued from previous page)

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Suite ID #1        |           Suite ID #2           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Suite ID #n        |              Padding            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Transform*

Raises ***ProtocolError*** – The parameter is **ONLY** supported in HIPv1.

_read_para_hip_transport_mode(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `HIP_TRANSPORT_MODE` parameter.

Structure of HIP `HIP_TRANSPORT_MODE` parameter [**RFC 6261**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Type               |              Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Port               |            Mode ID #1           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Mode ID #2            |            Mode ID #3           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Mode ID #n            |              Padding            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

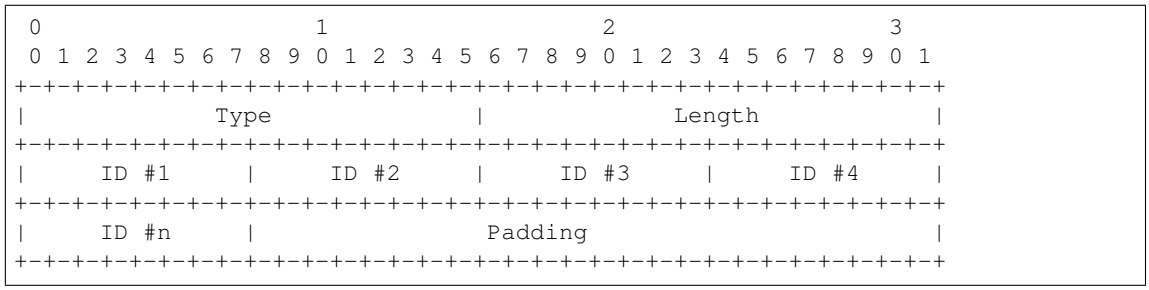
Return type *DataType_Param_Transport_Mode*

Raises ***ProtocolError*** – If `clen` is **NOT** `2` modulo.

_read_para_hit_suite_list(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `HIT_SUITE_LIST` parameter.

Structure of HIP `HIT_SUITE_LIST` parameter [**RFC 7401**]:



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     ID #1     |     ID #2     |     ID #3     |     ID #4     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     ID #n     |                Padding                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

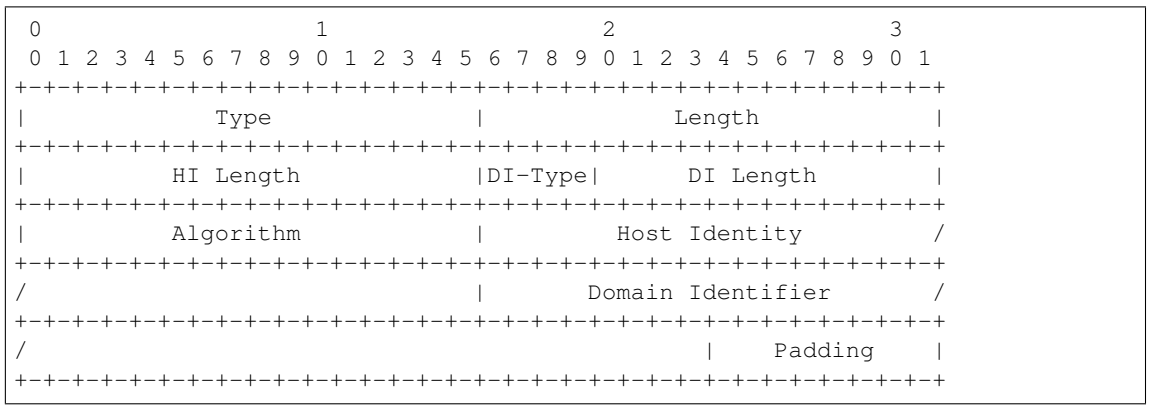
Returns Parsed parameter data.

Return type *DataType_Param_HIT_Suite_List*

_read_para_host_id(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `HOST_ID` parameter.

Structure of HIP `HOST_ID` parameter [**RFC 7401**]:



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          HI Length            |DI-Type|      DI Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Algorithm            |         Host Identity         /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                               |       Domain Identifier       /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                               |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code

- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

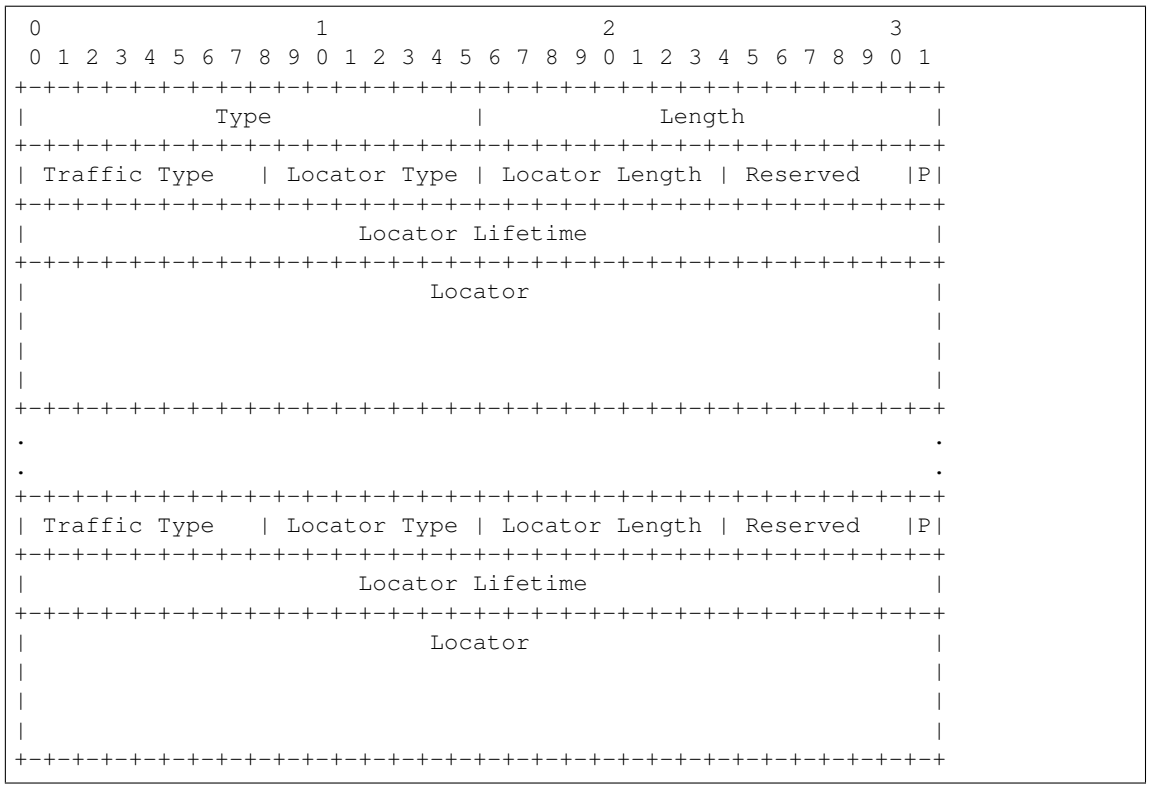
Returns Parsed parameter data.

Return type *DataType_Param_Host_ID*

_read_para_locator_set(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP LOCATOR_SET parameter.

Structure of HIP LOCATOR_SET parameter [**RFC 8046**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

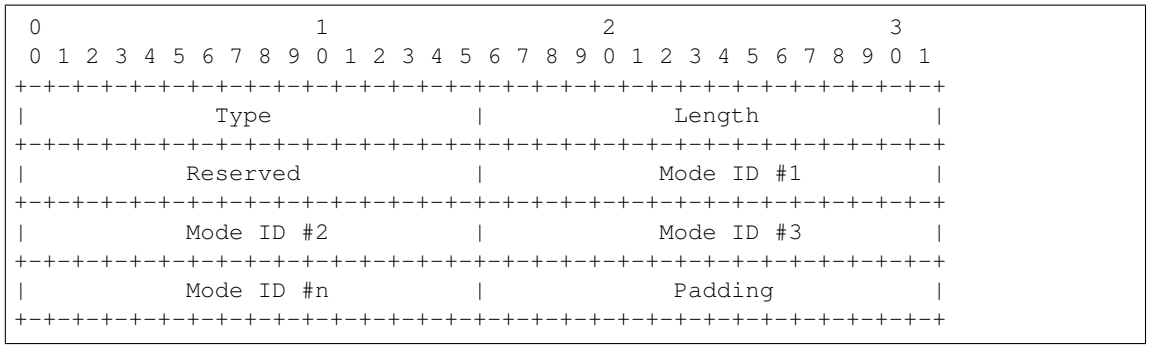
Returns Parsed parameter data.

Return type *DataType_Param_Locator_Set*

Raises ***ProtocolError*** – If locator data is malformed.

_read_para_nat_traversal_mode(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `NAT_TRAVERSAL_MODE` parameter.

Structure of HIP `NAT_TRAVERSAL_MODE` parameter [**RFC 5770**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

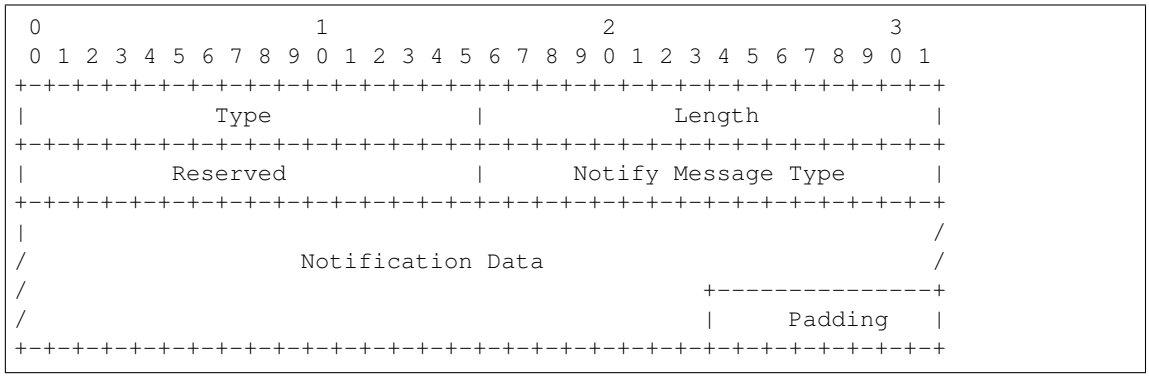
Returns Parsed parameter data.

Return type *DataType_Param_NET_Traversal_Mode*

Raises ***ProtocolError*** – If `clen` is **NOT** a `2` modulo.

_read_para_notification(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `NOTIFICATION` parameter.

Structure of HIP `NOTIFICATION` parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

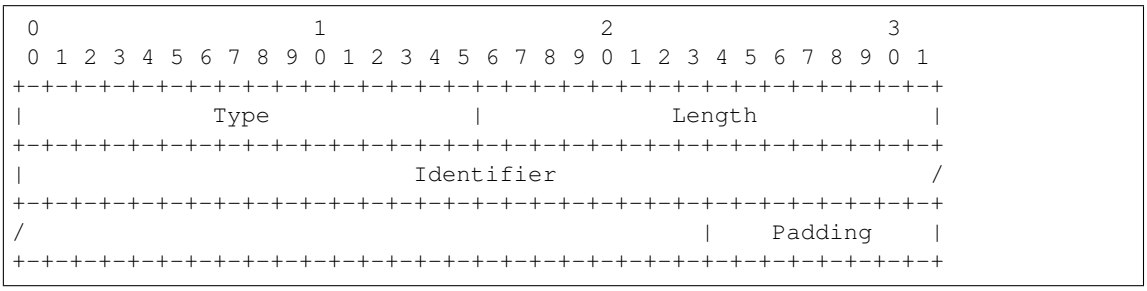
Return type *DataType_Param_Notification*

Raises ***ProtocolError*** – Unregistered notify message type.

_read_para_overlay_id(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP OVERLAY_ID parameter.

Structure of HIP OVERLAY_ID parameter [**RFC 6079**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

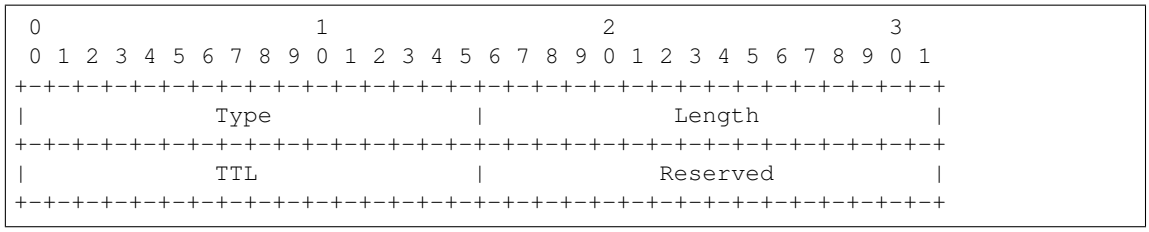
Returns Parsed parameter data.

Return type *DataType_Param_Overlay_ID*

_read_para_overlay_ttl(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP OVERLAY_TTL parameter.

Structure of HIP OVERLAY_TTL parameter [**RFC 6078**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

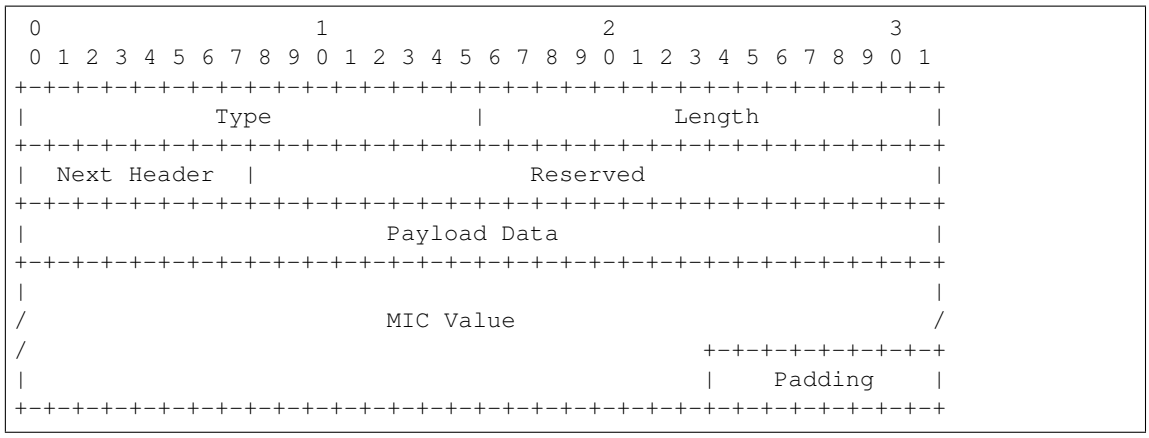
Return type *DataType_Param_Relay_To*

Raises ***ProtocolError*** – If `clen` is **NOT** `4`.

_read_para_payload_mic(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `PAYLOAD_MIC` parameter.

Structure of HIP `PAYLOAD_MIC` parameter [**RFC 6078**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length

- **version** (*Literal[1, 2]*) – HIP protocol version

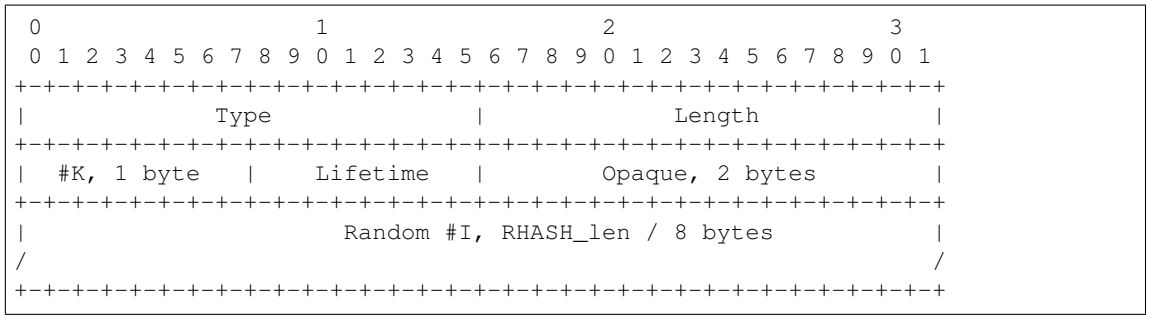
Returns Parsed parameter data.

Return type *DataType_Param_Payload_MIC*

_read_para_puzzle(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP PUZZLE parameter.

Structure of HIP PUZZLE parameter [RFC 5201][RFC 7401]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

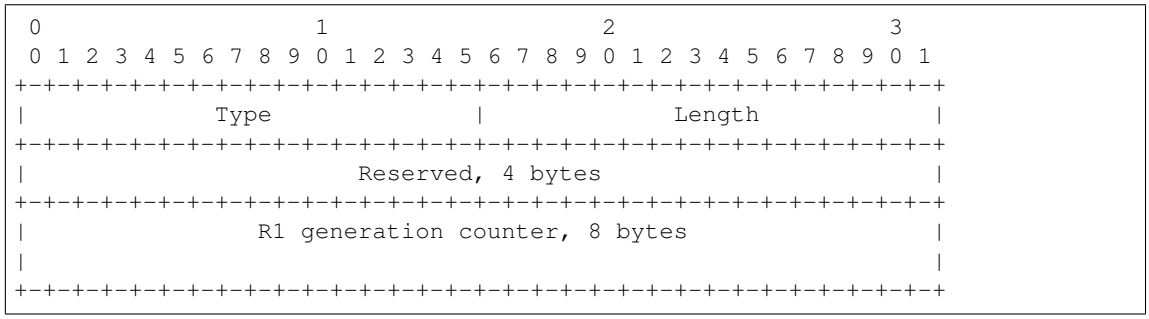
Return type *DataType_Param_Puzzle*

Raises ***ProtocolError*** – The parameter is **ONLY** supported in HIPv1.

_read_para_r1_counter(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP R1_COUNTER parameter.

Structure of HIP R1_COUNTER parameter [RFC 5201][RFC 7401]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_R1_Counter*

Raises ***ProtocolError*** – If `clen` is **NOT** `12` or the parameter is **NOT** used in HIPv1.

_read_para_reg_failed(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `REG_FAILED` parameter.

Structure of HIP `REG_FAILED` parameter [**RFC 8003**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Lifetime    |  Reg Type #1  |  Reg Type #2  |  Reg Type #3  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      ...      |      ...      |  Reg Type #n  |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+    Padding    +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

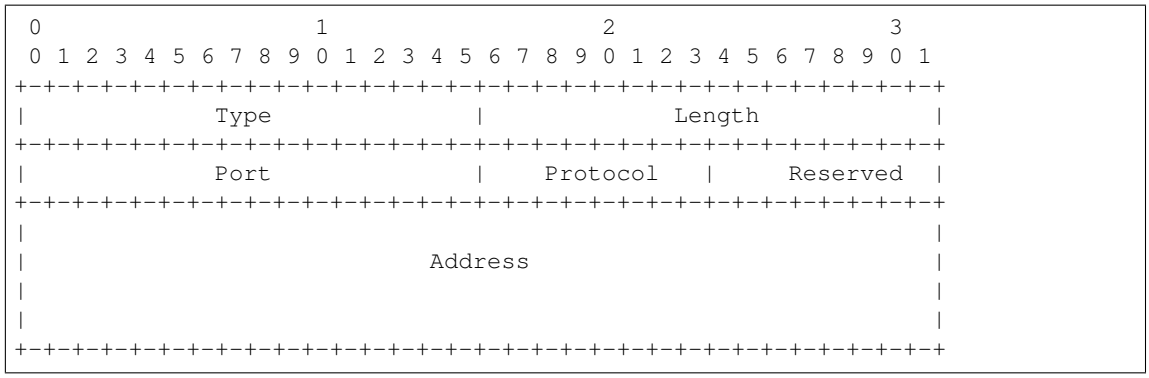
Return type *DataType_Param_Reg_Failed*

Raises ***ProtocolError*** – If the registration type is invalid.

_read_para_reg_from(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `REG_FROM` parameter.

Structure of HIP `REG_FROM` parameter [**RFC 5770**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

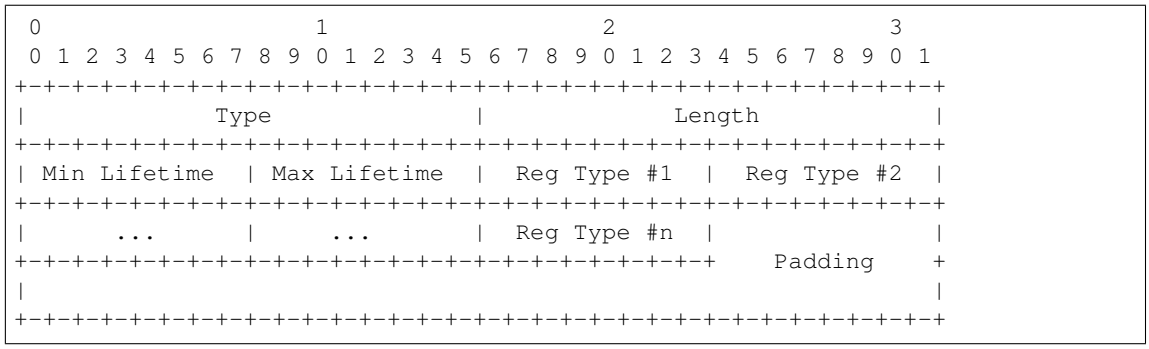
Return type *DataType_Param_Reg_From*

Raises ***ProtocolError*** – If `clen` is **NOT** `20`.

_read_para_reg_info(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `REG_INFO` parameter.

Structure of HIP `REG_INFO` parameter [**RFC 8003**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

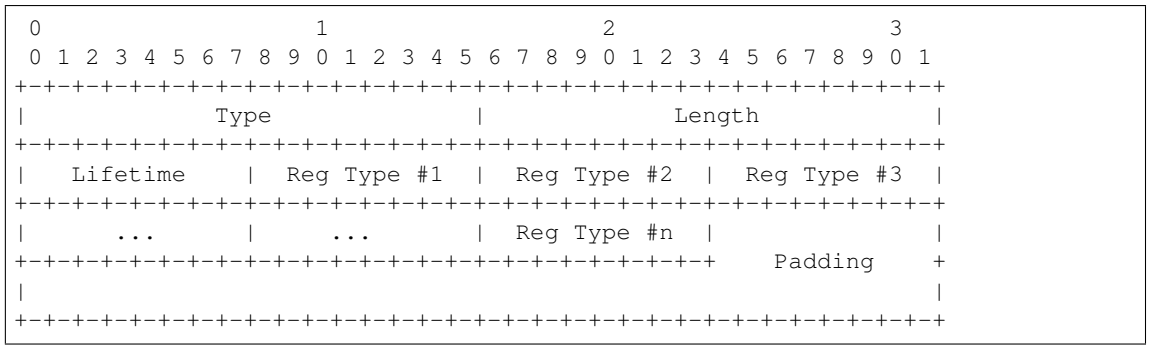
Return type *DataType_Param_Reg_Info*

Raises ***ProtocolError*** – If the registration type is invalid.

_read_para_reg_request(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP REG_REQUEST parameter.

Structure of HIP REG_REQUEST parameter [**RFC 8003**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

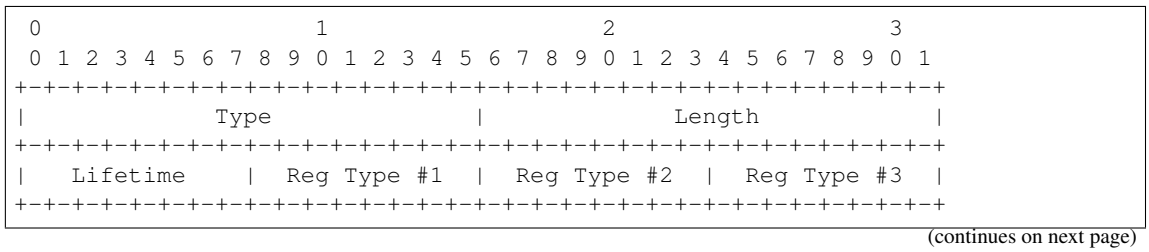
Return type *DataType_Param_Reg_Request*

Raises ***ProtocolError*** – If the registration type is invalid.

_read_para_reg_response(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP REG_RESPONSE parameter.

Structure of HIP REG_RESPONSE parameter [**RFC 8003**]:



(continues on next page)

(continued from previous page)

```
|      ...      |      ...      |  Reg Type #n  |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+    Padding    +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Reg_Response*

Raises ***ProtocolError*** – If the registration type is invalid.

_read_para_relay_from(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP RELAY_FROM parameter.

Structure of HIP RELAY_FROM parameter [**RFC 5770**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Port              |    Protocol   |     Reserved  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                            Address                            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

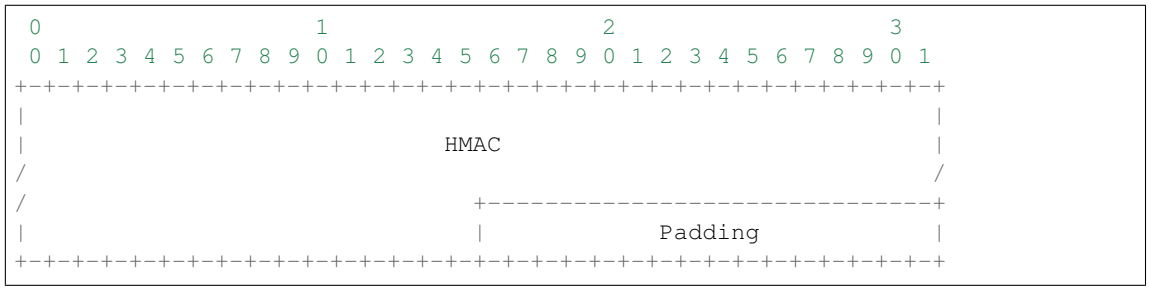
Returns Parsed parameter data.

Return type *DataType_Param_Relay_From*

Raises ***ProtocolError*** – If `clen` is **NOT** `20`.

_read_para_relay_hmac(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `RELAY_HMAC` parameter.

Structure of HIP `RELAY_HMAC` parameter [**RFC 5770**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

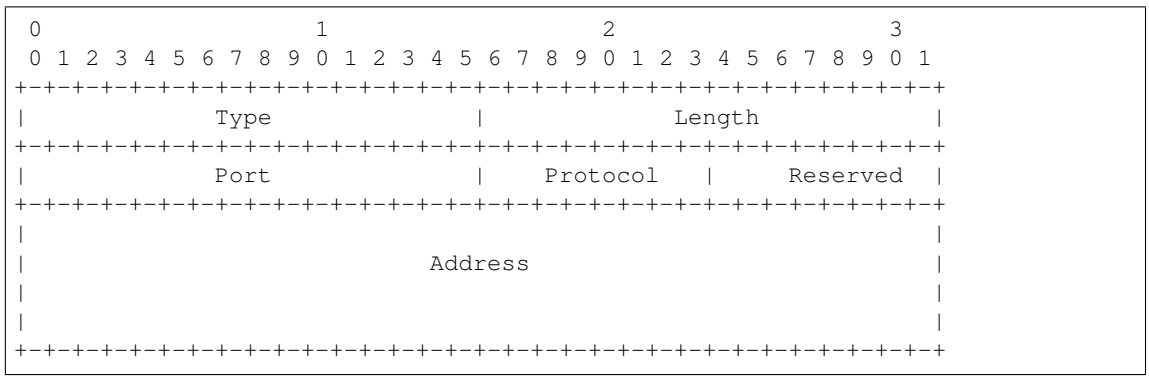
- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Relay_HMAC*

_read_para_relay_to(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `RELAY_TO` parameter.

Structure of HIP `RELAY_TO` parameter [**RFC 5770**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit

- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

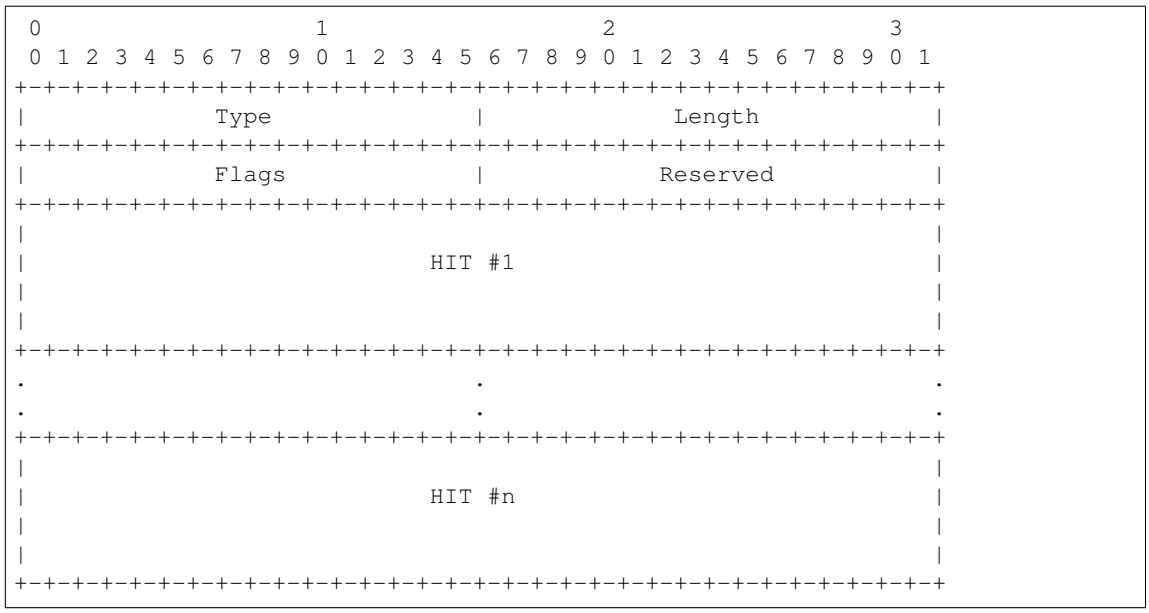
Return type *DataType_Param_Relay_To*

Raises ***ProtocolError*** – If `clen` is **NOT** `20`.

_read_para_route_dst(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `ROUTE_DST` parameter.

Structure of HIP `ROUTE_DST` parameter [**RFC 6028**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

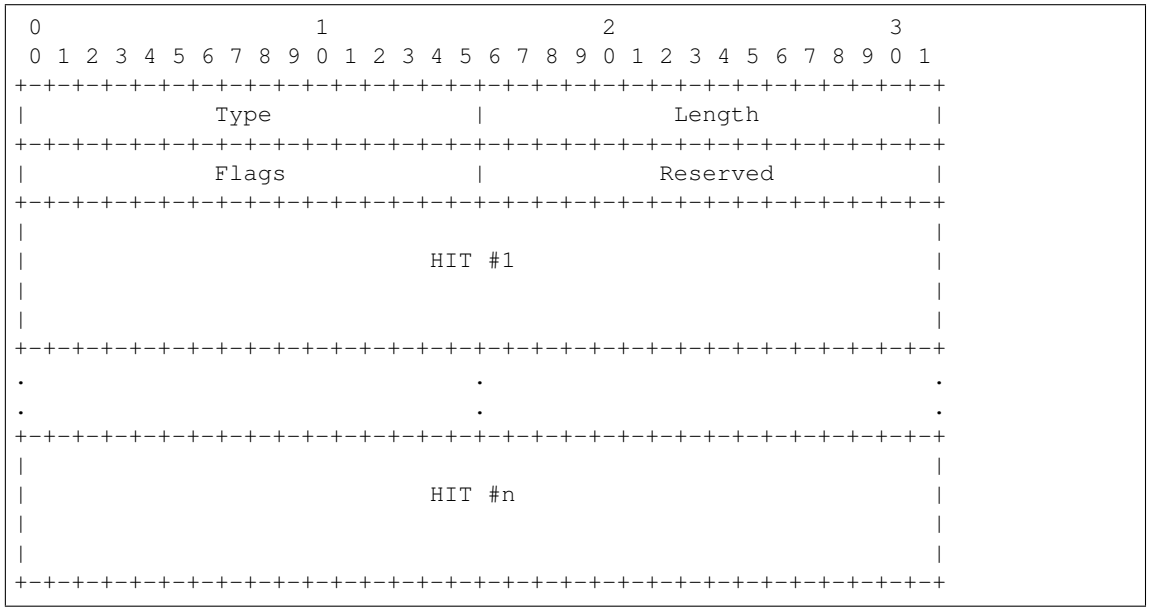
Returns Parsed parameter data.

Return type *DataType_Param_Route_Dst*

Raises ***ProtocolError*** – If the parameter is malformed.

_read_para_route_via(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP ROUTE_VIA parameter.

Structure of HIP ROUTE_VIA parameter [RFC 6028]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

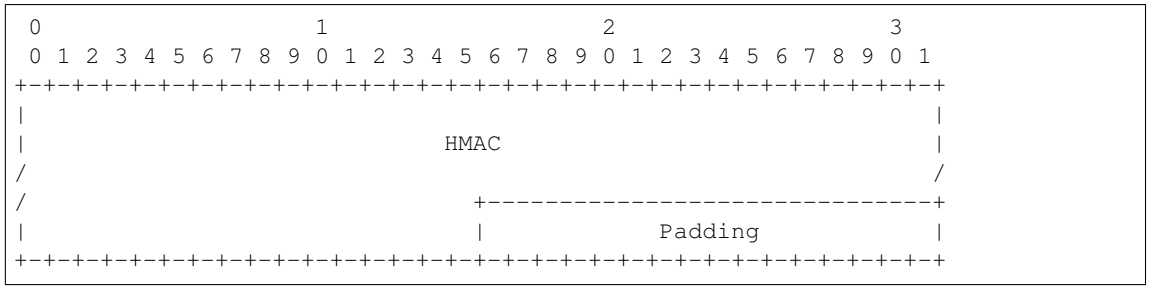
Returns Parsed parameter data.

Return type *DataType_Param_Route_Via*

Raises ***ProtocolError*** – If the parameter is malformed.

_read_para_rvs_hmac(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP RVS_HMAC parameter.

Structure of HIP RVS_HMAC parameter [RFC 8004]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

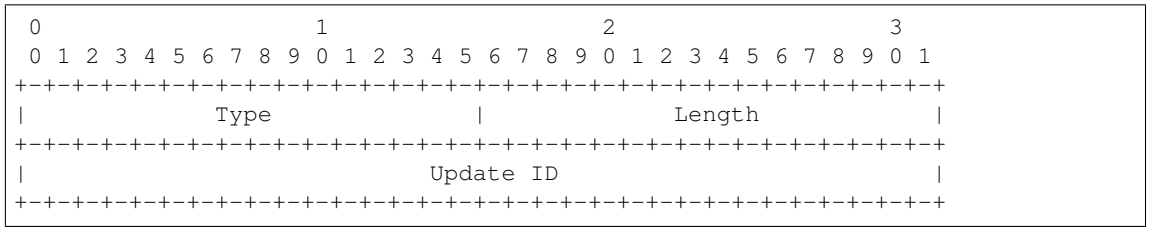
Returns Parsed parameter data.

Return type *DataType_Param_RVS_HMAC*

_read_para_seq(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP SEQ parameter.

Structure of HIP SEQ parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

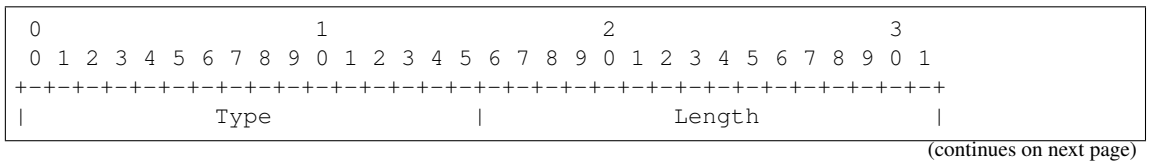
Return type *DataType_Param_SEQ*

Raises ***ProtocolError*** – If clen is **NOT** 4.

_read_para_seq_data(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

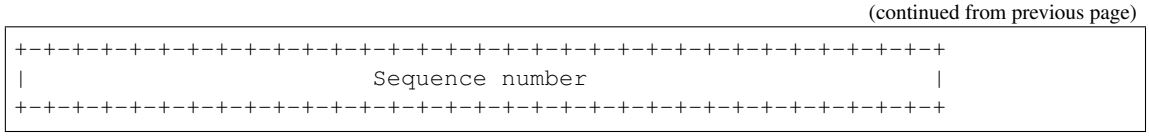
Read HIP SEQ_DATA parameter.

Structure of HIP SEQ_DATA parameter [**RFC 6078**]:



(continues on next page)

(continued from previous page)



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

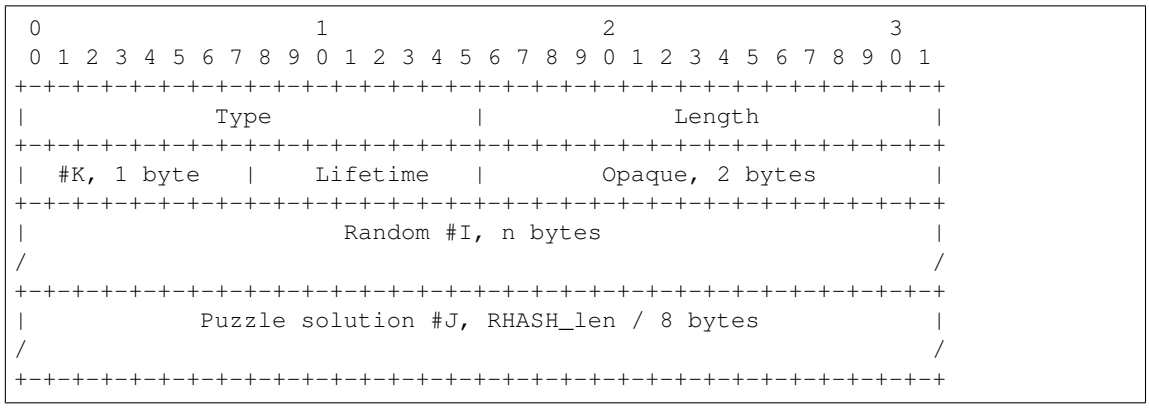
Return type *DataType_Param_SEQ_Data*

Raises ***ProtocolError*** – If `clen` is **NOT** `4`.

_read_para_solution(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `SOLUTION` parameter.

Structure of HIP `SOLUTION` parameter [**RFC 5201**][**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

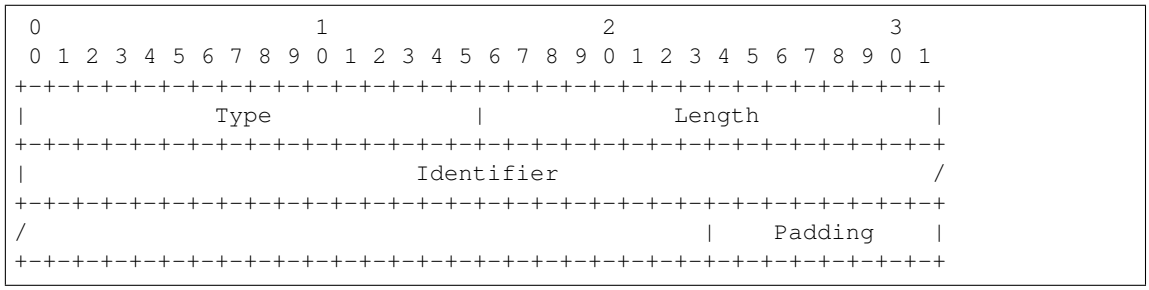
Returns Parsed parameter data.

Return type *DataType_Param_Solution*

Raises ***ProtocolError*** – The parameter is **ONLY** supported in HIPv1.

_read_para_transaction_id(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `TRANSACTION_ID` parameter.

Structure of HIP `TRANSACTION_ID` parameter [**RFC 6078**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

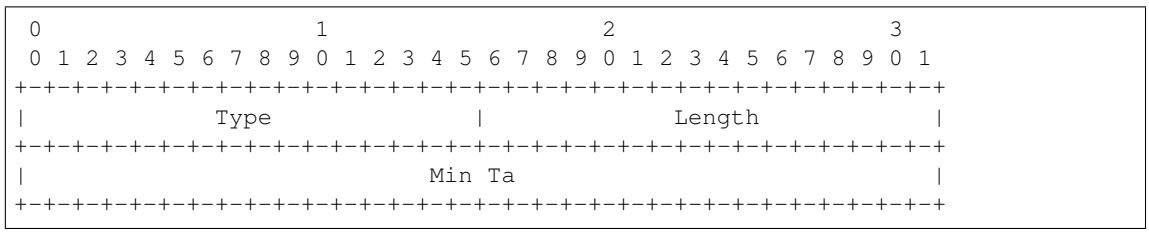
- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Transaction_ID*

_read_para_transaction_pacing(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)
Read HIP `TRANSACTION_PACING` parameter.

Structure of HIP `TRANSACTION_PACING` parameter [**RFC 5770**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type

- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

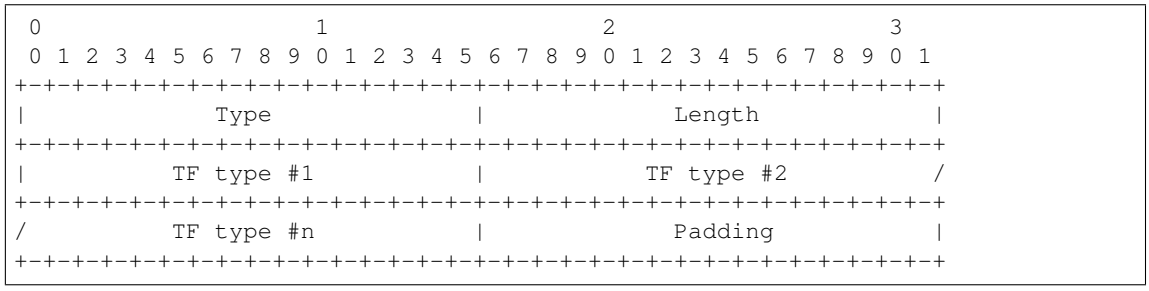
Return type *DataType_Param_Transaction_Pacing*

Raises ***ProtocolError*** – If `clen` is **NOT** `4`.

_read_para_transport_format_list(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP `TRANSPORT_FORMAT_LIST` parameter.

Structure of HIP `TRANSPORT_FORMAT_LIST` parameter [**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (`pcapkit.const.hip.parameter.Parameter`) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

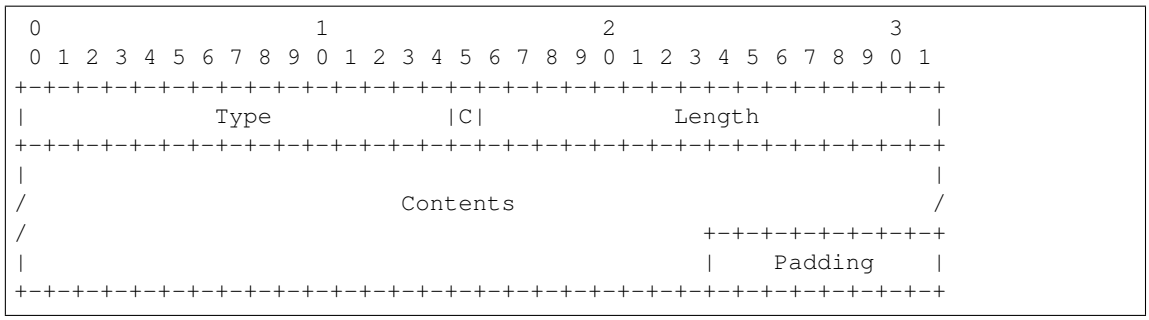
Return type *DataType_Param_Transform_Format_List*

Raises ***ProtocolError*** – If `clen` is **NOT** `2` modulo.

_read_para_unassigned(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP unassigned parameters.

Structure of HIP unassigned parameters [**RFC 5201**][**RFC 7401**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

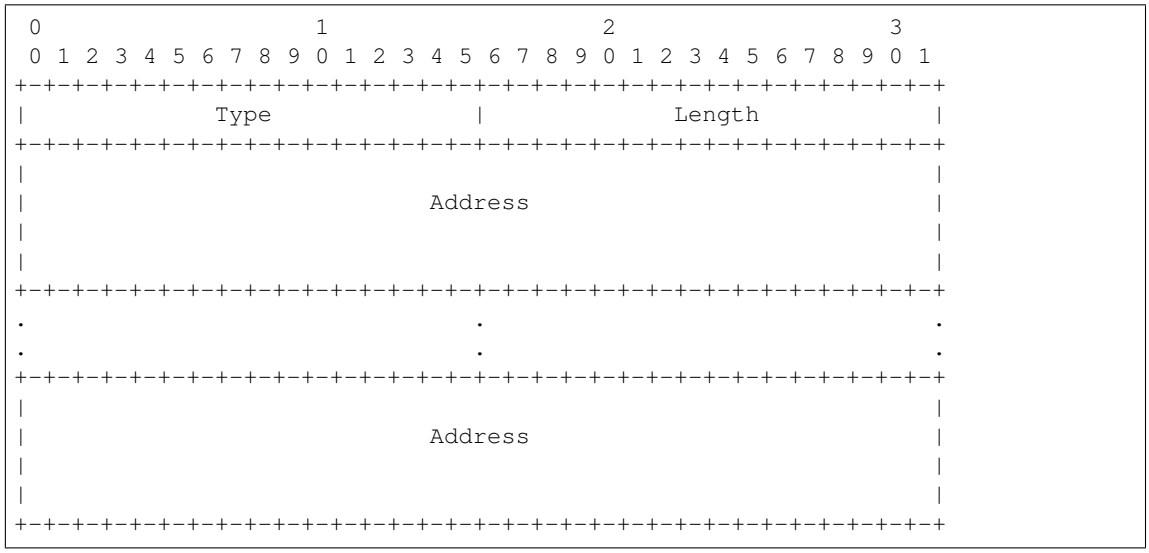
Returns Parsed parameter data.

Return type *DataType_Param_Unassigned*

_read_para_via_rvs(*code*, *cbit*, *clen*, ***, *desc*, *length*, *version*)

Read HIP VIA_RVS parameter.

Structure of HIP VIA_RVS parameter [**RFC 6028**]:



Parameters

- **code** (*int*) – parameter code
- **cbit** (*bool*) – critical bit
- **clen** (*int*) – length of contents

Keyword Arguments

- **desc** (pcapkit.const.hip.parameter.Parameter) – parameter type
- **length** (*int*) – remaining packet length
- **version** (*Literal[1, 2]*) – HIP protocol version

Returns Parsed parameter data.

Return type *DataType_Param_Route_Via*

Raises ***ProtocolError*** – If clen is **NOT** 16 modulo.

make (***kwargs*)
Make (construct) packet data.

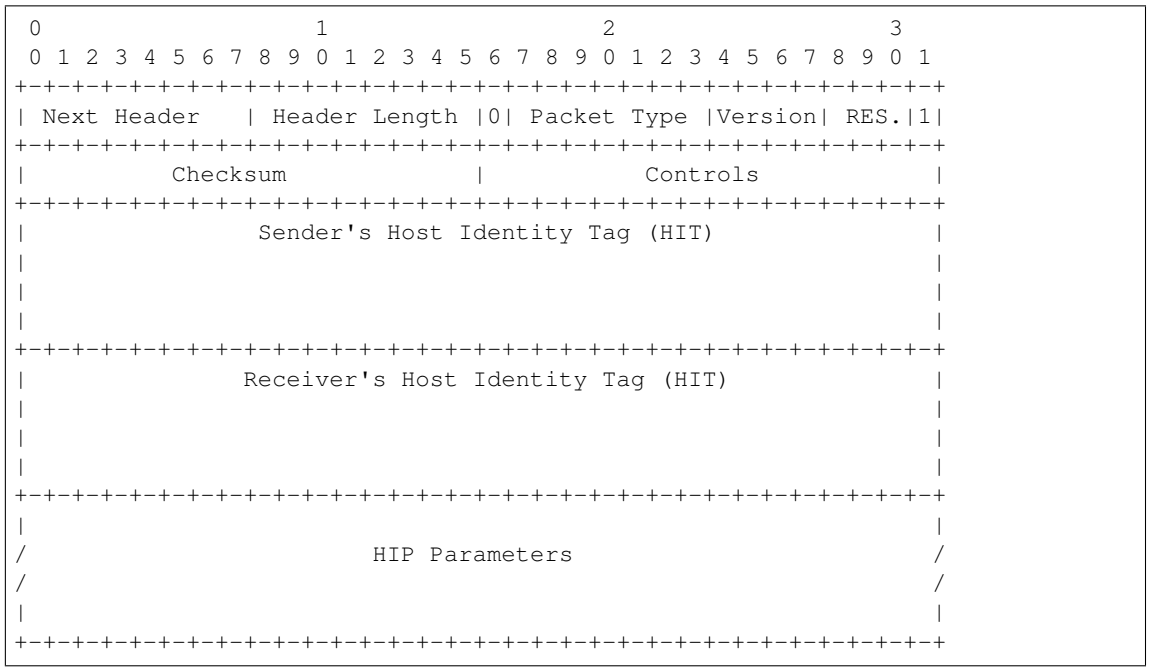
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read (*length=None*, ***, *extension=False*, ***kwargs*)
Read Host Identity Protocol.

Structure of HIP header [RFC 5201][RFC 7401]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_HIP*

Raises ***ProtocolError*** – If the packet is malformed.

property alias
Acronym of corresponding protocol.

Return type str

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Host Identity Protocol', 'Host Identity Protocol Version 2']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.hip.**DataType_HIP**

Bases TypedDict

HIP header [RFC 5201][RFC 7401].

next: ***pcapkit.const.reg.transtype.TransType***
Next header.

length: **int**
Header length.

type: ***pcapkit.const.hip.packet.Packet***
Packet type.

version: **Literal[1, 2]**
Version.

chksum: **bytes**
Checksum.

control: ***DataType_Control***
Controls.

shit: **int**
Sender's host identity tag.

rhit: **int**
Receiver's host identity tag.

parameters: **Optional[Tuple[*pcapkit.const.hip.parameter.Parameter*]]**
HIP parameters.

class pcapkit.protocols.internet.hip.**DataType_Control**

Bases TypedDict

HIP controls.

anonymous: **bool**
Anonymous.

class pcapkit.protocols.internet.hip.**DataType_Parameter**

Bases TypedDict

HIP parameters.

type: *pcapkit.const.hip.parameter.Parameter*
Parameter type.

critical: bool
Critical bit.

length: int
Length of contents.

HIP Unassigned Parameters

For HIP unassigned parameters as described in RFC 5201 and RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	para.type	Parameter Type
1	15	para.critical	Critical Bit
2	16	para.length	Length of Contents
4	32	para.contents	Contents Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Unassigned**

Bases DataType_Parameter

Structure of HIP unassigned parameters [RFC 5201][RFC 7401].

contents: bytes
Contents.

HIP ESP_INFO Parameter

For HIP ESP_INFO parameter as described in RFC 7402, its structure is described as below:

Octets	Bits	Name	Description
0	0	esp_info.type	Parameter Type
1	15	esp_info.critical	Critical Bit
2	16	esp_info.length	Length of Contents
4	32		Reserved
6	48	esp_info.index	KEYMAT Index
8	64	esp_info.old_spi	OLD SPI
12	96	esp_info.new_spi	NEW SPI

class pcapkit.protocols.internet.hip.**DataType_Param_ESP_Info**

Bases DataType_Parameter

Structure of HIP ESP_INFO parameter [RFC 7402].

index: int
KEYMAT index.

old_spi: int
Old SPI.

new_spi: *int*
New SPI.

HIP R1_COUNTER Parameter

For HIP R1_COUNTER parameter as described in RFC 5201 and RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	ri_counter.type	Parameter Type
1	15	ri_counter.critical	Critical Bit
2	16	ri_counter.length	Length of Contents
4	32		Reserved
8	64	ri_counter.count	Generation of Valid Puzzles

class pcapkit.protocols.internet.hip.**DataType_Param_R1_Counter**

Bases DataType_Parameter

Structure of HIP R1_COUNTER parameter [RFC 5201][RFC 7401].

count: *int*
Generation of valid puzzles.

HIP LOCATOR_SET Parameter

For HIP LOCATOR_SET parameter as described in RFC 8046, its structure is described as below:

Octets	Bits	Name	Description
0	0	locator_set.type	Parameter Type
1	15	locator_set.critical	Critical Bit
2	16	locator_set.length	Length of Contents
?	?	...	...
4	32	locator.traffic	Traffic Type
5	40	locator.type	Locator Type
6	48	locator.length	Locator Length
7	56		Reserved
7	63	locator.preferred	Preferred Locator
8	64	locator.lifetime	Locator Lifetime
12	96	locator.object	Locator
?	?	...	...

class pcapkit.protocols.internet.hip.**DataType_Param_Locator_Set**

Bases DataType_Parameter

Structure of HIP LOCATOR_SET parameter [RFC 8046].

locator: **Tuple[*DataType_Locator*]**
Locator set.

class pcapkit.protocols.internet.hip.**DataType_Locator**

Bases TypedDict

Locator.

traffic: *int*
Traffic type.

type: *int*
Locator type.

length: *int*
Locator length.

preferred: *int*
Preferred length.

lifetime: *int*
Locator lifetime.

object: *DataType_Locator_Dict*
Locator.

class pcapkit.protocols.internet.hip.**DataType_Locator_Dict**

Bases TypedDict

Locator type 2.

spi: *int*
SPI.

ip: *ipaddress.IPv4Address*

HIP PUZZLE Parameter

For HIP PUZZLE parameter as described in RFC 5201 and RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	puzzle.type	Parameter Type
1	15	puzzle.critical	Critical Bit
2	16	puzzle.length	Length of Contents
4	32	puzzle.number	Number of Verified Bits
5	40	puzzle.lifetime	Lifetime
6	48	puzzle.opaque	Opaque
8	64	puzzle.random	Random Number

class pcapkit.protocols.internet.hip.**DataType_Param_Puzzle**

Bases DataType_Parameter

Structure of HIP PUZZLE parameter [RFC 5201][RFC 7401].

number: *int*
Number of verified bits.

lifetime: *int*
Lifetime.

opaque: *bytes*
Opaque.

random: *int*
Random number.

HIP SOLUTION Parameter

For HIP SOLUTION parameter as described in RFC 5201 and RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	solution.type	Parameter Type
1	15	solution.critical	Critical Bit
2	16	solution.length	Length of Contents
4	32	solution.number	Number of Verified Bits
5	40	solution.lifetime	Lifetime
6	48	solution.opaque	Opaque
8	64	solution.random	Random Number
?	?	solution.solution	Puzzle Solution

class pcapkit.protocols.internet.hip.**DataType_Param_Solution**

Bases DataType_Parameter

Structure of HIP SOLUTION parameter [RFC 5201][RFC 7401].

number: *number*
Number of verified bits.

lifetime: **int**
Lifetime.

opaque: **bytes**
Opaque.

random: **int**
Random number.

solution: **int**
Puzzle solution.

HIP SEQ Parameter

For HIP SEQ parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	seq.type	Parameter Type
1	15	seq.critical	Critical Bit
2	16	seq.length	Length of Contents
4	32	seq.id	Update ID

class pcapkit.protocols.internet.hip.**DataType_Param_SEQ**

Bases DataType_Parameter

Structure of HIP SEQ parameter [RFC 7401].

id: **int**
Update ID.

HIP ACK Parameter

For HIP ACK parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	ack.type	Parameter Type
1	15	ack.critical	Critical Bit
2	16	ack.length	Length of Contents
4	32	ack.id	Peer Update ID

class pcapkit.protocols.internet.hip.**DataType_Param_ACK**

Bases DataType_Parameter

id: **Tuple[int]**
Array of peer update IDs.

HIP DH_GROUP_LIST Parameter

For HIP DH_GROUP_LIST parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	dh_group_list.type	Parameter Type
1	15	dh_group_list.critical	Critical Bit
2	16	dh_group_list.length	Length of Contents
4	32	dh_group_list.id	DH GROUP ID

class pcapkit.protocols.internet.hip.**DataType_Param_DH_Group_List**

Bases DataType_Parameter

Structure of HIP DH_GROUP_LIST parameter [RFC 7401].

id: **Tuple[*pcapkit.const.hip.group.Group*]**
Array of DH group IDs.

HIP DEFFIE_HELLMAN Parameter

For HIP DEFFIE_HELLMAN parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	diffie_hellman.type	Parameter Type
1	15	diffie_hellman.critical	Critical Bit
2	16	diffie_hellman.length	Length of Contents
4	32	diffie_hellman.id	Group ID
5	40	diffie_hellman.pub_len	Public Value Length
6	48	diffie_hellman.pub_val	Public Value
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Deffie_Hellman**

Bases DataType_Parameter

Structure of HIP DEFFIE_HELLMAN parameter [RFC 7401].

id: *pcapkit.const.hip.group.Group*
Group ID.

pub_len: **int**
Public value length.

pub_val: **bytes**
Public value.

HIP HIP_TRANSFORM Parameter

For HIP HIP_TRANSFORM parameter as described in RFC 5201, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_transform.type	Parameter Type
1	15	hip_transform.critical	Critical Bit
2	16	hip_transform.length	Length of Contents
4	32	hip_transform.id	Group ID
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Transform**

Bases DataType_Parameter

Structure of HIP HIP_TRANSFORM parameter [RFC 5201].

id: **Tuple[*pcapkit.const.hip.suite.Suite*]**
Array of group IDs.

HIP HIP_CIPHER Parameter

For HIP HIP_CIPHER parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_cipher.type	Parameter Type
1	15	hip_cipher.critical	Critical Bit
2	16	hip_cipher.length	Length of Contents
4	32	hip_cipher.id	Cipher ID
?	?	...	...
?	?	•	Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Cipher**

Bases DataType_Parameter

Structure of HIP HIP_CIPHER parameter [RFC 7401].

id: **Tuple[*pcapkit.const.hip.cipher.Cipher*]**
Array of cipher IDs.

HIP NAT_TRAVERSAL_MODE Parameter

For HIP NAT_TRAVERSAL_MODE parameter as described in RFC 5770, its structure is described as below:

Octets	Bits	Name	Description
0	0	nat_traversal_mode.type	Parameter Type
1	15	nat_traversal_mode.critical	Critical Bit
2	16	nat_traversal_mode.length	Length of Contents
4	32		Reserved
6	48	nat_traversal_mode.id	Mode ID
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_NET_Traversal_Mode**

Bases DataType_Parameter

Structure of HIP NAT_TRAVERSAL_MODE parameter [RFC 5770].

id: **Tuple[pcapkit.const.hip.nat_traversal.NETTraversal]**
Array of mode IDs.

HIP TRANSACTION_PACING Parameter

For HIP TRANSACTION_PACING parameter as described in RFC 5770, its structure is described as below:

Octets	Bits	Name	Description
0	0	transaction_pacing.type	Parameter Type
1	15	transaction_pacing.critical	Critical Bit
2	16	transaction_pacing.length	Length of Contents
4	32	transaction_pacing.min_ta	Min Ta

class pcapkit.protocols.internet.hip.**DataType_Param_Transaction_Pacing**

Bases DataType_Parameter

Structure of HIP TRANSACTION_PACING parameter [RFC 5770].

min_ta: **int**
Min Ta.

HIP ENCRYPTED Parameter

For HIP ENCRYPTED parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	encrypted.type	Parameter Type
1	15	encrypted.critical	Critical Bit
2	16	encrypted.length	Length of Contents
4	32		Reserved
8	48	encrypted.iv	Initialization Vector
?	?	encrypted.data	Encrypted data
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Encrypted**

Bases DataType_Parameter

Structure of HIP ENCRYPTED parameter [RFC 7401].

raw: *bytes*
Raw content data.

HIP HOST_ID Parameter

For HIP HOST_ID parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	host_id.type	Parameter Type
1	15	host_id.critical	Critical Bit
2	16	host_id.length	Length of Contents
4	32	host_id.id_len	Host Identity Length
6	48	host_id.di_type	Domain Identifier Type
6	52	host_id.di_len	Domain Identifier Length
8	64	host_id.algorithm	Algorithm
10	80	host_id.host_id	Host Identity
?	?	host_id.domain_id	Domain Identifier
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Host_ID**

Bases DataType_Parameter

Structure of HIP HOST_ID parameter [RFC 7401].

id_len: *int*
Host identity length.

di_type: *pcapkit.const.hip.di_type.DIType*
Domain identifier type.

di_len: *int*
Domain identifier length.

algorithm: *pcapkit.const.hip.hi_algorithm.HIAlgorithm*
Algorithm.

host_id: *Union[bytes, DataType_Host_ID_ECDSA_Curve, DataType_Host_ID_ECDSA_LOW_Curve]*
Host identity.

domain_id: *bytes*
Domain identifier.

class pcapkit.protocols.internet.hip.**DataType_Host_ID_ECDSA_Curve**

Bases TypedDict

Host identity data.

curve: *pcapkit.const.hip.ecdsa_curve.ECDSACurve*
ECDSA curve.

pubkey: *bytes*
Public key.

class pcapkit.protocols.internet.hip.**DataType_Host_ID_ECDSA_LOW_Curve**

Bases TypedDict

Host identity data.

curve: *pcapkit.const.hip.ecdsa_low_curve.ECDSALowCurve*
ECDSA_Low curve.

pubkey: bytes
Public key.

HIP HIT_SUITE_LIST Parameter

For HIP HIT_SUITE_LIST parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	hit_suite_list.type	Parameter Type
1	15	hit_suite_list.critical	Critical Bit
2	16	hit_suite_list.length	Length of Contents
4	32	hit_suite_list.id	HIT Suite ID
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_HIT_Suite_List**

Bases DataType_Parameter

Structure of HIP HIT_SUITE_LIST parameter [RFC 7401].

id: **Tuple[***pcapkit.const.hip.hit_suite.HITSuite***]**
Array of HIT suite IDs.

HIP CERT Parameter

For HIP CERT parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	cert.type	Parameter Type
1	15	cert.critical	Critical Bit
2	16	cert.length	Length of Contents
4	32	cert.group	CERT Group
5	40	cert.count	CERT Count
6	48	cert.id	CERT ID
7	56	cert.cert_type	CERT Type
8	64	cert.certificate	Certificate
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Cert**

Bases DataType_Parameter

Structure of HIP CERT parameter [RFC 7401].

group: *pcapkit.const.hip.group.Group*
CERT group.

count: *int*
CERT count.

id: *int*
CERT ID.

cert_type: *pcapkit.const.hip.certificate.Certificate*

certificate: *bytes*
Certificate.

HIP NOTIFICATION Parameter

For HIP NOTIFICATION parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	notification.type	Parameter Type
1	15	notification.critical	Critical Bit
2	16	notification.length	Length of Contents
4	32		Reserved
6	48	notification.msg_type	Notify Message Type
8	64	notification.data	Notification Data
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Notification**

Bases DataType_Parameter

Structure of HIP NOTIFICATION parameter [RFC 7401].

msg_type: *pcapkit.const.hip.notify_message.NotifyMessage*
Notify message type.

data: *bytes*
Notification data.

HIP ECHO_REQUEST_SIGNED Parameter

For HIP ECHO_REQUEST_SIGNED parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	echo_request_signed.type	Parameter Type
1	15	echo_request_signed.critical	Critical Bit
2	16	echo_request_signed.length	Length of Contents
4	32	echo_request_signed.data	Opaque Data

class pcapkit.protocols.internet.hip.**DataType_Param_Echo_Request_Signed**

Bases DataType_Parameter

Structure of HIP ECHO_REQUEST_SIGNED parameter [RFC 7401].

data: *bytes*
Opaque data.

HIP `REG_INFO` Parameter

For HIP `REG_INFO` parameter as described in **RFC 8003**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`reg_info.type`	Parameter Type
1	15	`reg_info.critical`	Critical Bit
2	16	`reg_info.length`	Length of Contents
4	32	`reg_info.lifetime`	Lifetime
4	32	`reg_info.lifetime.min`	Min Lifetime
5	40	`reg_info.lifetime.max`	Max Lifetime
6	48	`reg_info.reg_type`	Reg Type
?	?	…	…
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Reg_Info**

Bases DataType_Parameter

Structure of HIP `REG_INFO` parameter [**RFC 8003**].

lifetime: *DataType_Lifetime*
Lifetime.

reg_type: **Tuple[***pcapkit.const.hip.registration.Registration***]**
Array of registration type.

class pcapkit.protocols.internet.hip.**DataType_Lifetime**

Bases NamedTuple

Lifetime.

min: **int**
Minimum lifetime.

maz: **int**
Maximum lifetime.

HIP `REG_REQUEST` Parameter

For HIP `REG_REQUEST` parameter as described in **RFC 8003**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`reg_request.type`	Parameter Type
1	15	`reg_request.critical`	Critical Bit
2	16	`reg_request.length`	Length of Contents
4	32	`reg_request.lifetime`	Lifetime
4	32	`reg_request.lifetime.min`	Min Lifetime
5	40	`reg_request.lifetime.max`	Max Lifetime
6	48	`reg_request.reg_type`	Reg Type
?	?	…	…
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Reg_Request**

Bases DataType_Parameter

Structure of HIP `REG_REQUEST` parameter [RFC 8003].

lifetime: *DataType_Lifetime*
Lifetime.

reg_type: **Tuple[***pcapkit.const.hip.registration.Registration***]**
Array of registration type.

HIP `REG_RESPONSE` Parameter

For HIP `REG_RESPONSE` parameter as described in RFC 8003, its structure is described as below:

Octets	Bits	Name	Description
0	0	`reg_response.type`	Parameter Type
1	15	`reg_response.critical`	Critical Bit
2	16	`reg_response.length`	Length of Contents
4	32	`reg_response.lifetime`	Lifetime
4	32	`reg_response.lifetime.min`	Min Lifetime
5	40	`reg_response.lifetime.max`	Max Lifetime
6	48	`reg_response.reg_type`	Reg Type
?	?	…	…
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Reg_Response**

Bases DataType_Parameter

Structure of HIP `REG_RESPONSE` parameter [RFC 8003].

lifetime: *DataType_Lifetime*
Lifetime.

reg_type: **Tuple[***pcapkit.const.hip.registration.Registration***]**
Array of registration type.

HIP `REG_FAILED` Parameter

For HIP `REG_FAILED` parameter as described in RFC 8003, its structure is described as below:

Octets	Bits	Name	Description
0	0	`reg_failed.type`	Parameter Type
1	15	`reg_failed.critical`	Critical Bit
2	16	`reg_failed.length`	Length of Contents
4	32	`reg_failed.lifetime`	Lifetime
4	32	`reg_failed.lifetime.min`	Min Lifetime
5	40	`reg_failed.lifetime.max`	Max Lifetime
6	48	`reg_failed.reg_type`	Reg Type
?	?	…	…
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Reg_Failed**

Bases DataType_Parameter

Structure of HIP REG_FAILED parameter [RFC 8003].

lifetime: *DataType_Lifetime*
Lifetime.

reg_type: **Tuple[***pcapkit.const.hip.registration.Registration***]**
Array of registration type.

HIP REG_FROM Parameter

For HIP REG_FROM parameter as described in RFC 5770, its structure is described as below:

Octets	Bits	Name	Description
0	0	reg_from.type	Parameter Type
1	15	reg_from.critical	Critical Bit
2	16	reg_from.length	Length of Contents
4	32	reg_from.port	Port
6	48	reg_from.protocol	Protocol
7	56		Reserved
8	64	reg_from.ip	Address (IPv6)

class pcapkit.protocols.internet.hip.**DataType_Param_Reg_From**

Bases DataType_Parameter

Structure of HIP REG_FROM parameter [RFC 5770].

port: **int**
Port.

protocol: *pcapkit.const.reg.transtype.TransType*
Protocol.

ip: **ipaddress.IPv6Address**
IPv6 address.

HIP ECHO_RESPONSE_SIGNED Parameter

For HIP ECHO_RESPONSE_SIGNED parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	echo_response_signed.type	Parameter Type
1	15	echo_response_signed.critical	Critical Bit
2	16	echo_response_signed.length	Length of Contents
4	32	echo_response_signed.data	Opaque Data

class pcapkit.protocols.internet.hip.**DataType_Param_Echo_Response_Signed**

Bases DataType_Parameter

Structure of HIP ECHO_RESPONSE_SIGNED parameter [RFC 7401].

data: **bytes**
Opaque data.

HIP TRANSPORT_FORMAT_LIST Parameter

For HIP TRANSPORT_FORMAT_LIST parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	transport_format_list.type	Parameter Type
1	15	transport_format_list.critical	Critical Bit
2	16	transport_format_list.length	Length of Contents
4	32	transport_format_list.tf_type	TF Type
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Transform_Format_List**

Bases DataType_Parameter

Structure of HIP TRANSPORT_FORMAT_LIST parameter [RFC 7401].

tf_type: **Tuple[int]**
Array of TF types.

HIP ESP_TRANSFORM Parameter

For HIP ESP_TRANSFORM parameter as described in RFC 7402, its structure is described as below:

Octets	Bits	Name	Description
0	0	esp_transform.type	Parameter Type
1	15	esp_transform.critical	Critical Bit
2	16	esp_transform.length	Length of Contents
4	32		Reserved
6	48	esp_transform.id	Suite ID
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_ESP_Transform**

Bases DataType_Parameter

Structure of HIP ESP_TRANSFORM parameter [RFC 7402].

id: **Tuple[*pcapkit.const.hip.esp_transform_suite.ESPTransformSuite*]**
Array of suite IDs.

HIP SEQ_DATA Parameter

For HIP SEQ_DATA parameter as described in RFC 6078, its structure is described as below:

Octets	Bits	Name	Description
0	0	seq_data.type	Parameter Type
1	15	seq_data.critical	Critical Bit
2	16	seq_data.length	Length of Contents
4	32	seq_data.seq	Sequence number

class pcapkit.protocols.internet.hip.**DataType_Param_SEQ_Data**

Bases DataType_Parameter

Structure of HIP SEQ_DATA parameter [RFC 6078].

seq: *int*
Sequence number.

HIP ACK_DATA Parameter

For HIP ACK_DATA parameter as described in RFC 6078, its structure is described as below:

Octets	Bits	Name	Description
0	0	ack_data.type	Parameter Type
1	15	ack_data.critical	Critical Bit
2	16	ack_data.length	Length of Contents
4	32	ack_data.ack	Acked Sequence number

class pcapkit.protocols.internet.hip.**DataType_Param_ACK_Data**

Bases DataType_Parameter

Structure of HIP ACK_DATA parameter [RFC 6078].

ack: *Tuple[int]*
Array of ACKed sequence number.

HIP PAYLOAD_MIC Parameter

For HIP PAYLOAD_MIC parameter as described in RFC 6078, its structure is described as below:

Octets	Bits	Name	Description
0	0	payload_mic.type	Parameter Type
1	15	payload_mic.critical	Critical Bit
2	16	payload_mic.length	Length of Contents
4	32	payload_mic.next	Next Header
5	40		Reserved
8	64	payload_mic.data	Payload Data
12	96	payload_mic.value	MIC Value
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Payload_MIC**

Bases DataType_Parameter

Structure of HIP PAYLOAD_MIC parameter [RFC 6078].

next: *pcapkit.const.reg.transtype.TransType*
Next header.

data: *bytes*
Payload data.

value: *bytes*
MIC value.

HIP TRANSACTION_ID Parameter

For HIP TRANSACTION_ID parameter as described in RFC 6078, its structure is described as below:

Octets	Bits	Name	Description
0	0	transaction_id.type	Parameter Type
1	15	transaction_id.critical	Critical Bit
2	16	transaction_id.length	Length of Contents
4	32	transaction_id.id	Identifier

class pcapkit.protocols.internet.hip.**DataType_Param_Transaction_ID**

Bases DataType_Parameter

Structure of HIP TRANSACTION_ID parameter [RFC 6078].

id: *int*
Identifier.

HIP OVERLAY_ID Parameter

For HIP OVERLAY_ID parameter as described in RFC 6079, its structure is described as below:

Octets	Bits	Name	Description
0	0	overlay_id.type	Parameter Type
1	15	overlay_id.critical	Critical Bit
2	16	overlay_id.length	Length of Contents
4	32	overlay_id.id	Identifier

class pcapkit.protocols.internet.hip.**DataType_Param_Overlay_ID**

Bases DataType_Parameter

Structure of HIP OVERLAT_ID parameter [RFC 6079].

id: *int*
Identifier.

HIP ROUTE_DST Parameter

For HIP ROUTE_DST parameter as described in RFC 6079, its structure is described as below:

Octets	Bits	Name	Description
0	0	route_dst.type	Parameter Type
1	15	route_dst.critical	Critical Bit
2	16	route_dst.length	Length of Contents
4	32	route_dst.flags	Flags
4	32	route_dst.flags.symmetric	SYMMETRIC [RFC 6028]
4	33	route_dst.flags.must_follow	MUST_FOLLOW [RFC 6028]
6	48		Reserved
8	64	route_dst.ip	HIT
?	?	...	...

class pcapkit.protocols.internet.hip.**DataType_Param_Route_Dst**

Bases DataType_Parameter

Structure of HIP ROUTE_DST parameter [RFC 6028].

flags: *DataType_Flags*
Flags.

ip: **Tuple[ipaddress.IPv6Address]**
Array of HIT addresses.

class pcapkit.protocols.internet.hip.**DataType_Flags**

Bases TypedDict

Flags.

symmetric: **bool**
SYMMETRIC flag [RFC 6028].

must_follow: **bool**
MUST_FOLLOW flag [RFC 6028].

HIP HIP_TRANSPORT_MODE Parameter

For HIP HIP_TRANSPORT_MODE parameter as described in RFC 6261, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_transport_mode.type	Parameter Type
1	15	hip_transport_mode.critical	Critical Bit
2	16	hip_transport_mode.length	Length of Contents
4	32	hip_transport_mode.port	Port
6	48	hip_transport_mode.id	Mode ID
?	?	...	...
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Transport_Mode**

Bases DataType_Parameter

Structure of HIP HIP_TRANSPORT_MODE parameter [RFC 6261].

port: **int**
Port.

id: **Tuple[*pcapkit.const.hip.transport.Transport*]**
Array of transport mode IDs.

HIP HIP_MAC Parameter

For HIP HIP_MAC parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_mac.type	Parameter Type
1	15	hip_mac.critical	Critical Bit
2	16	hip_mac.length	Length of Contents
4	32	hip_mac.hmac	HMAC
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_HMAC**

Bases DataType_Parameter

Structure of HIP HIP_MAC parameter [RFC 7401].

hmac: **bytes**
HMAC.

HIP HIP_MAC_2 Parameter

For HIP HIP_MAC_2 parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_mac_2.type	Parameter Type
1	15	hip_mac_2.critical	Critical Bit
2	16	hip_mac_2.length	Length of Contents
4	32	hip_mac_2.hmac	HMAC
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_HMAC_2**

Bases DataType_Parameter

Structure of HIP HIP_MAC_2 parameter [RFC 7401].

hmac: **bytes**
HMAC.

HIP HIP_SIGNATURE_2 Parameter

For HIP HIP_SIGNATURE_2 parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	hip_signature_2.type	Parameter Type
1	15	hip_signature_2.critical	Critical Bit
2	16	hip_signature_2.length	Length of Contents
4	32	hip_signature_2.algorithm	SIG Algorithm
6	48	hip_signature_2.signature	Signature
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Signature_2**

Bases DataType_Parameter

Structure of HIP `HIP_SIGNATURE_2` parameter [RFC 7401].

algorithm: *pcapkit.const.hip.hi_algorithm.HIAlgorithm*
SIG algorithm.

signature: bytes
Signature.

HIP `HIP_SIGNATURE` Parameter

For HIP `HIP_SIGNATURE` parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hip_signature.type`	Parameter Type
1	15	`hip_signature.critical`	Critical Bit
2	16	`hip_signature.length`	Length of Contents
4	32	`hip_signature.algorithm`	SIG Algorithm
6	48	`hip_signature.signature`	Signature
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Signature**

Bases DataType_Parameter

Structure of HIP `HIP_SIGNATURE` parameter [RFC 7401].

algorithm: *pcapkit.const.hip.hi_algorithm.HIAlgorithm*
SIG algorithm.

signature: bytes
Signature.

HIP `ECHO_REQUEST_UNSIGNED` Parameter

For HIP `ECHO_REQUEST_UNSIGNED` parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	`echo_request_unsigned.type`	Parameter Type
1	15	`echo_request_unsigned.critical`	Critical Bit
2	16	`echo_request_unsigned.length`	Length of Contents
4	32	`echo_request_unsigned.data`	Opaque Data

class pcapkit.protocols.internet.hip.**DataType_Param_Echo_Request_Unsigned**

Bases DataType_Parameter

Structure of HIP `ECHO_REQUEST_UNSIGNED` parameter [RFC 7401].

data: bytes
Opaque data.

HIP ECHO_RESPONSE_UNSIGNED Parameter

For HIP ECHO_RESPONSE_UNSIGNED parameter as described in RFC 7401, its structure is described as below:

Octets	Bits	Name	Description
0	0	echo_response_unsigned.type	Parameter Type
1	15	echo_response_unsigned.critical	Critical Bit
2	16	echo_response_unsigned.length	Length of Contents
4	32	echo_response_unsigned.data	Opaque Data

class pcapkit.protocols.internet.hip.**DataType_Param_Echo_Response_Unsigned**

Bases DataType_Parameter

Structure of HIP ECHO_RESPONSE_UNSIGNED parameter [RFC 7401].

data: *bytes*
Opaque data.

HIP RELAY_FROM Parameter

For HIP RELAY_FROM parameter as described in RFC 5770, its structure is described as below:

Octets	Bits	Name	Description
0	0	relay_from.type	Parameter Type
1	15	relay_from.critical	Critical Bit
2	16	relay_from.length	Length of Contents
4	32	relay_from.port	Port
6	48	relay_from.protocol	Protocol
7	56		Reserved
8	64	relay_from.ip	Address (IPv6)

class pcapkit.protocols.internet.hip.**DataType_Param_Relay_From**

Bases DataType_Parameter

Structure of HIP RELAY_FROM parameter [RFC 5770].

port: *int*
Port.

protocol: *pcapkit.const.reg.transtype.TransType*
Protocol.

ip: *ipaddress.IPv6Address*
IPv6 address.

HIP `RELAY_TO` Parameter

For HIP `RELAY_TO` parameter as described in **RFC 5770**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`relay_to.type`	Parameter Type
1	15	`relay_to.critical`	Critical Bit
2	16	`relay_to.length`	Length of Contents
4	32	`relay_to.port`	Port
6	48	`relay_to.protocol`	Protocol
7	56		Reserved
8	64	`relay_to.ip`	Address (IPv6)

class pcapkit.protocols.internet.hip.**DataType_Param_Relay_To**

Bases DataType_Parameter

Structure of HIP `RELAY_TO` parameter [**RFC 5770**].

port: **in**
Port.

protocol: *pcapkit.const.reg.transtype.TransType*
Protocol.

ip: **ipaddress.IPv6Address**
IPv6 address.

HIP `OVERLAY_TTL` Parameter

For HIP `OVERLAY_TTL` parameter as described in **RFC 6078**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`overlay_ttl.type`	Parameter Type
1	15	`overlay_ttl.critical`	Critical Bit
2	16	`overlay_ttl.length`	Length of Contents
4	32	`overlay_ttl.ttl`	TTL
6	48		Reserved

class pcapkit.protocols.internet.hip.**DataType_Param_Overlay_TTL**

Bases DataType_Parameter

ttl: **int**
TTL.

HIP ROUTE_VIA Parameter

For HIP ROUTE_VIA parameter as described in RFC 6028, its structure is described as below:

Octets	Bits	Name	Description
0	0	route_via.type	Parameter Type
1	15	route_via.critical	Critical Bit
2	16	route_via.length	Length of Contents
4	32	route_via.flags	Flags
4	32	route_via.flags.symmetric	SYMMETRIC [RFC 6028]
4	33	route_via.flags.must_follow	MUST_FOLLOW [RFC 6028]
6	48	•	Reserved
8	64	route_dst.ip	HIT
?	?	…	…

class pcapkit.protocols.internet.hip.**DataType_Param_Route_Via**

Bases DataType_Parameter

Structure of HIP ROUTE_VIA parameter [RFC 6028].

flags: *DataType_Flags*
Flags.

ip: **Tuple[ipaddress.IPv6Address]**
Array of HITs.

HIP FROM Parameter

For HIP FROM parameter as described in RFC 8004, its structure is described as below:

Octets	Bits	Name	Description
0	0	from.type	Parameter Type
1	15	from.critical	Critical Bit
2	16	from.length	Length of Contents
4	32	from.ip	Address

class pcapkit.protocols.internet.hip.**DataType_Param_From**

Bases DataType_Parameter

Structure of HIP FROM parameter [RFC 8004].

ip: **ipaddress.IPv6Address**
IPv6 address.

HIP RVS_HMAC Parameter

For HIP RVS_HMAC parameter as described in RFC 8004, its structure is described as below:

Octets	Bits	Name	Description
0	0	rvs_hmac.type	Parameter Type
1	15	rvs_hmac.critical	Critical Bit
2	16	rvs_hmac.length	Length of Contents
4	32	rvs_hmac.hmac	HMAC
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_RVS_HMAC**

Bases DataType_Parameter

Structure of HIP RVS_HMAC parameter [RFC 8004].

hmac: **bytes**
HMAC.

HIP VIA_RVS Parameter

For HIP VIA_RVS parameter as described in RFC 6028, its structure is described as below:

Octets	Bits	Name	Description
0	0	via_rvs.type	Parameter Type
1	15	via_rvs.critical	Critical Bit
2	16	via_rvs.length	Length of Contents
4	32	via_rvs.ip	Address
?	?	...	...

class pcapkit.protocols.internet.hip.**DataType_Param_Via_RVS**

Bases DataType_Parameter

Structure of HIP VIA_RVS parameter [RFC 6028].

ip: **Tuple[ipaddress.IPv6]**
Array of IPv6 addresses.

HIP RELAY_HMAC Parameter

For HIP RELAY_HMAC parameter as described in RFC 5770, its structure is described as below:

Octets	Bits	Name	Description
0	0	relay_hmac.type	Parameter Type
1	15	relay_hmac.critical	Critical Bit
2	16	relay_hmac.length	Length of Contents
4	32	relay_hmac.hmac	HMAC
?	?		Padding

class pcapkit.protocols.internet.hip.**DataType_Param_Relay_HMAC**

Bases DataType_Parameter

hmac: bytes
HMAC.

HOPOPT - IPv6 Hop-by-Hop Options

pcapkit.protocols.internet.hopopt contains *HOPOPT* only, which implements extractor for IPv6 Hop-by-Hop Options header (HOPOPT)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.next`	Next Header
1	8	`hopopt.length`	Header Extensive Length
2	16	`hopopt.options`	Options

class pcapkit.protocols.internet.hopopt.**HOPOPT**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements IPv6 Hop-by-Hop Options.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[2]

__post_init__(*file*, *length=None*, ***, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

_read_hopopt_options(*length*)
Read HOPOPT options.

Positional arguments: length (int): length of options

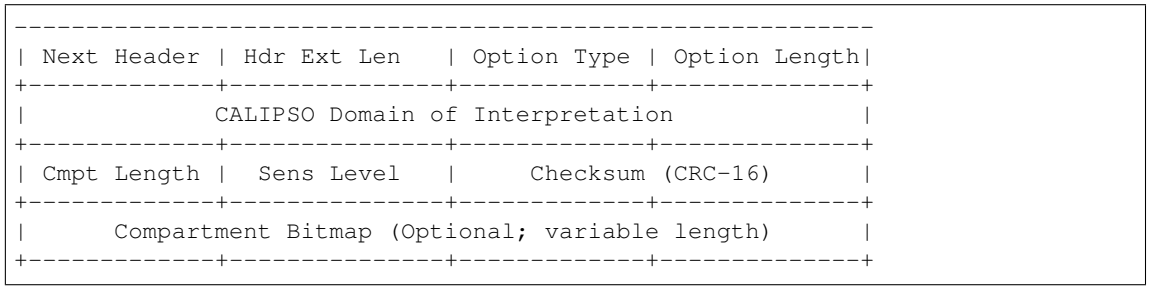
Returns Tuple[Tuple[pcapkit.const.ipv6.option.Option], Dict[str, DataType_Option]]: extracted HOPOPT options

Raises ***ProtocolError*** – If the threshold is **NOT** matching.

[0] https://en.wikipedia.org/wiki/IPv6_packet#Hop-by-hop_options_and_destination_options

_read_opt_calipso(*code*, ***, *desc*)
Read HOPOPT CALIPSO option.

Structure of HOPOPT CALIPSO option [**RFC 5570**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

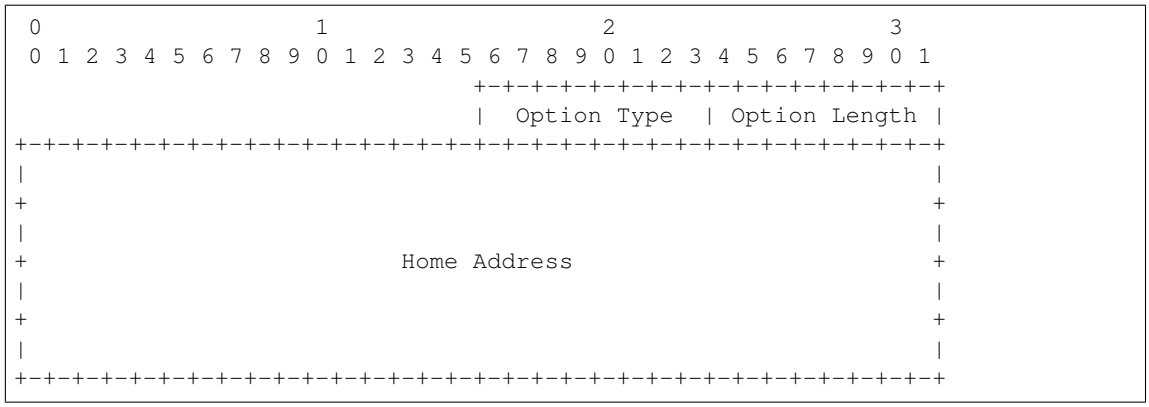
Returns parsed option data

Return type *DataType_Opt_CALIPSO*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_home(*code*, ***, *desc*)
Read HOPOPT Home Address option.

Structure of HOPOPT Home Address option [**RFC 6275**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

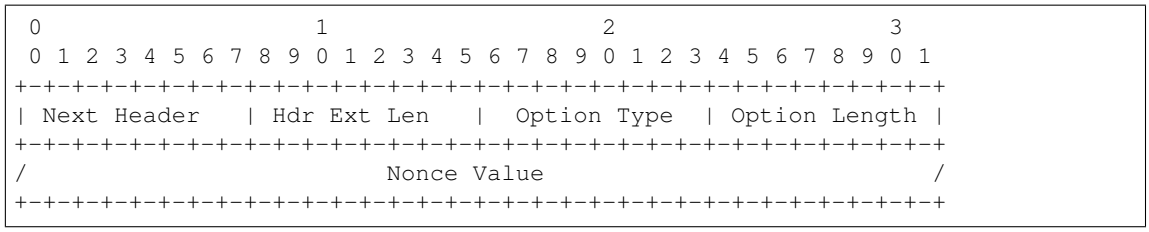
Returns parsed option data

Return type *DataType_Opt_Home*

Raises ***ProtocolError*** – If `hopopt.jumbo.length` is **NOT** 16.

_read_opt_ilnp(*code*, ***, *desc*)
Read HOPOPT ILNP Nonce option.

Structure of HOPOPT ILNP Nonce option [**RFC 6744**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

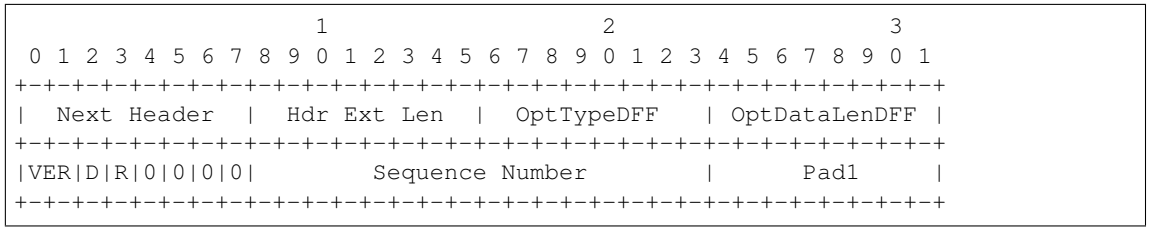
Returns parsed option data

Return type *DataType_Opt_ILNP*

_read_opt_ip_dff(*code*, ***, *desc*)

Read HOPOPT `IP_DFF` option.

Structure of HOPOPT `IP_DFF` option [**RFC 6971**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

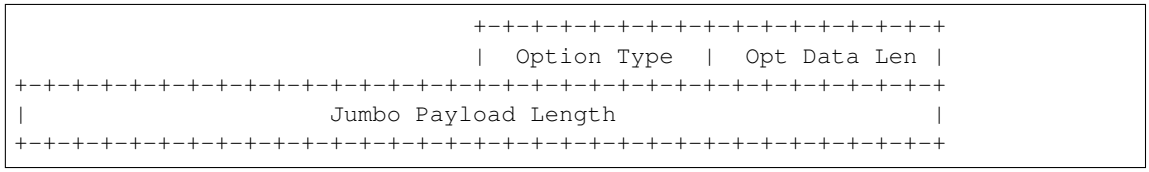
Return type *DataType_Opt_IP_DFF*

Raises ***ProtocolError*** – If `hopopt.ip_dff.length` is **NOT** `2`.

_read_opt_jumbo(*code*, ***, *desc*)

Read HOPOPT Jumbo Payload option.

Structure of HOPOPT Jumbo Payload option [**RFC 2675**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

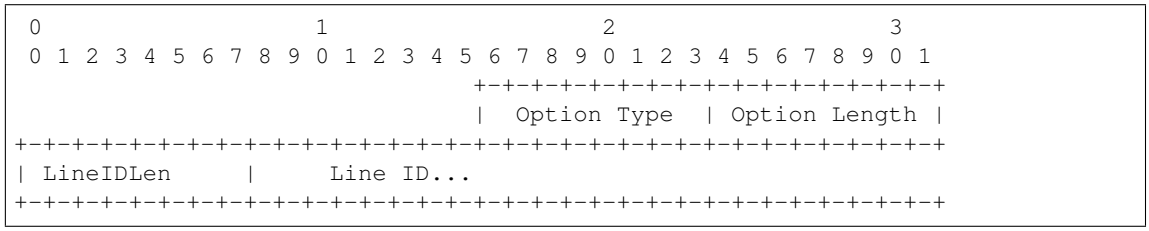
Returns parsed option data

Return type *DataType_Opt_Jumbo*

Raises ***ProtocolError*** – If `hopopt.jumbo.length` is **NOT** `4`.

_read_opt_lio(*code*, ***, *desc*)
Read HOPOPT Line-Identification option.

Structure of HOPOPT Line-Identification option [RFC 6788]:



Parameters **code** (*int*) – option type value

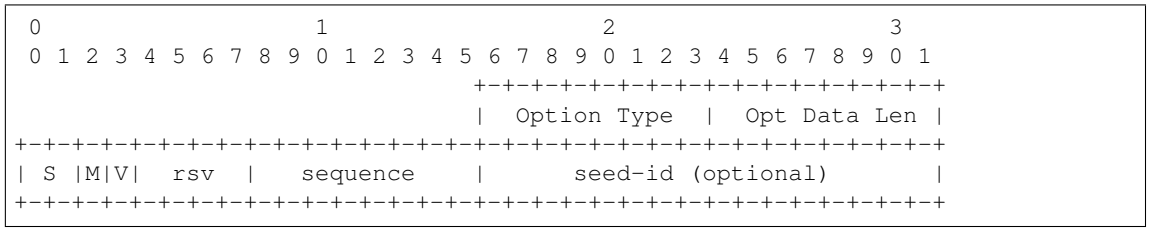
Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_LIO*

_read_opt_mpl(*code*, ***, *desc*)
Read HOPOPT `MPL` option.

Structure of HOPOPT `MPL` option [RFC 7731]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_MPL*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_none(*code*, ***, *desc*)
Read HOPOPT unassigned options.

Structure of HOPOPT unassigned options [RFC 8200]:

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_None*

_read_opt_pad(*code*, ***, *desc*)
Read HOPOPT padding options.

Structure of HOPOPT padding options [**RFC 8200**]:

- `Pad1` option:

```
+-+-+-+-+-+-+-+-+
|       0       |
+-+-+-+-+-+-+-+-+
```

- `PadN` option:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|       1       |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type Union[*DataType_Opt_Pad1*, *DataType_Opt_PadN*]

Raises ***ProtocolError*** – If `code` is **NOT** `0` or `1`.

_read_opt_pdm(*code*, ***, *desc*)
Read HOPOPT `PDM` option.

Structure of HOPOPT `PDM` option [**RFC 8250**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Option Length |    ScaleDTLR  |     ScaleDTLS |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   PSN This Packet             |  PSN Last Received            |
|-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Delta Time Last Received    |  Delta Time Last Sent         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_PDM*

Raises ***ProtocolError*** – If `hopopt.pdm.length` is **NOT** `10`.

_read_opt_qs(*code*, ***, *desc*)
Read HOPOPT Quick Start option.

Structure of HOPOPT Quick-Start option [**RFC 4782**]:

- A Quick-Start Request:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option      |  Length=6     | Func. | Rate  |   QS TTL      |
|               |               | 0000  |Request|               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        QS Nonce                             | R |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Report of Approved Rate:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option      |  Length=6     | Func. | Rate  |   Not Used    |
|               |               | 1000  | Report|               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        QS Nonce                             | R |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_QS*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_ra(*code*, ***, *desc*)

Read HOPOPT Router Alert option.

Structure of HOPOPT Router Alert option [**RFC 2711**]:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0|0 0 1 0 1|0 0 0 0 0 0 1 0|         Value (2 octets)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_RA*

Raises ***ProtocolError*** – If `hopopt.tun.length` is **NOT** `2`.

_read_opt_rpl(*code*, ***, *desc*)

Read HOPOPT `RPL` option.

Structure of HOPOPT `RPL` option [**RFC 6553**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  |  Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(continues on next page)

(continued from previous page)

```
|O|R|F|0|0|0|0|0| RPLInstanceID |           SenderRank          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         (sub-TLVs)                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_RPL*

Raises ***ProtocolError*** – If `hopopt.rpl.length` is **LESS THAN** `4`.

_read_opt_smf_dpd(*code*, ***, *desc*)

Read HOPOPT `SMF_DPD` option.

Structure of HOPOPT `SMF_DPD` option [**RFC 5570**]:

- IPv6 `SMF_DPD` option header in **I-DPD** mode

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               ...               |0|0|0|  01000  | Opt. Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|TidTy| TidLen|             TaggerID (optional) ...           |
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                 |            Identifier  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- IPv6 `SMF_DPD` option header in **H-DPD** mode

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               ...               |0|0|0| OptType | Opt. Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|    Hash Assist Value (HAV) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type Union[*DataType_Opt_SMF_I_PDP*, *DataType_Opt_SMF_H_PDP*]

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_tun(*code*, ***, *desc*)

Read HOPOPT Tunnel Encapsulation Limit option.

Structure of HOPOPT Tunnel Encapsulation Limit option [**RFC 2473**]:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |Hdr Ext Len = 0| Opt Type = 4  |Opt Data Len=1 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Tun Encap Lim |PadN Opt Type=1|Opt Data Len=1 |       0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Opt_TUN*

Raises ***ProtocolError*** – If `hopopt.tun.length` is **NOT** `1`.

_read_opt_type(*kind*)
Read option type field.

Parameters **kind** (*int*) – option kind value

Returns extracted HOPOPT option type field

Return type *DataType_Option_Type*

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, *, *extension=False*, ***kwargs*)
Read IPv6 Hop-by-Hop Options.

Structure of HOPOPT header [**RFC 8200**]:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_HOPOPT*

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['IPv6 Hop-by-Hop Options']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

pcapkit.protocols.internet.hopopt.**_HOPOPT_ACT: Dict[str, str]**
HOPOPT unknown option actions.

Code	Action
00	skip over this option and continue processing the header
01	discard the packet
10	discard the packet and, regardless of whether or not the packet's Destination Address was a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type
11	discard the packet and, only if the packet's Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type

pcapkit.protocols.internet.hopopt.**_HOPOPT_OPT: Dict[int, Tuple[str, str]]**
HOPOPT options.

Code	Acronym	Option	Reference
0x00	pad	Pad1	[**RFC 8200**] 0
0x01	pad	PadN	[**RFC 8200**]
0x04	tun	Tunnel Encapsulation Limit	[**RFC 2473**] 1
0x05	ra	Router Alert	[**RFC 2711**] 2
0x07	calipso	Common Architecture Label IPv6 Security Option	[**RFC 5570**]
0x08	smf_dpd	Simplified Multicast Forwarding	[**RFC 6621**]
0x0F	pdm	Performance and Diagnostic Metrics	[**RFC 8250**] 10
0x26	qs	Quick-Start	[**RFC 4782**][RFC Errata 2034] 6
0x63	rpl	Routing Protocol for Low-Power and Lossy Networks	[**RFC 6553**]
0x6D	mpl	Multicast Protocol for Low-Power and Lossy Networks	[**RFC 7731**]
0x8B	ilnp	Identifier-Locator Network Protocol Nonce	[**RFC 6744**]
0x8C	lio	Line-Identification Option	[**RFC 6788**]
0xC2	jumbo	Jumbo Payload	[**RFC 2675**]
0xC9	home	Home Address	[**RFC 6275**]
0xEE	ip_dff	Depth-First Forwarding	[**RFC 6971**]

pcapkit.protocols.internet.hopopt._HOPOPT_NULL: Dict[int, str]
HOPOPT unknown option descriptions.

Code	Description	Reference
0x1E	RFC3692-style Experiment	[RFC 4727]
0x3E	RFC3692-style Experiment	[RFC 4727]
0x4D	Deprecated	[RFC 7731]
0x5E	RFC3692-style Experiment	[RFC 4727]
0x7E	RFC3692-style Experiment	[RFC 4727]
0x8A	Endpoint Identification	**DEPRECATED**
0x9E	RFC3692-style Experiment	[RFC 4727]
0xBE	RFC3692-style Experiment	[RFC 4727]
0xDE	RFC3692-style Experiment	[RFC 4727]
0xFE	RFC3692-style Experiment	[RFC 4727]

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.hopopt.**DataType_HOPOPT**

Bases TypedDict

Structure of HOPOPT header [RFC 8200].

next: *pcapkit.const.reg.transtype.TransType*
Next header.

length: int
Header extensive length.

options: Tuple[*pcapkit.const.ipv6.option.Option*]
Array of option acronyms.

packet: bytes
Packet data.

class pcapkit.protocols.internet.hopopt.**DataType_Option**

Bases TypedDict

HOPOPT option.

desc: str
Option description.

type: *DataType_Option_Type*
Option type.

length: int
Option length.

Note: This attribute is **NOT** the length specified in the HOPOPT optiona data, rather the *total* length of the current option.

HOPOPT Option Type

For HOPOPT option type field as described in RFC 791, its structure is described as below:

Octets	Bits	Name	Descriptions
0	0	`hopopt.opt.type.value`	Option Number
0	0	`hopopt.opt.type.action`	Action (`00-11`)
0	2	`hopopt.opt.type.change`	Change Flag (`0/1`)

class pcapkit.protocols.internet.hopopt.**DataType_Option_Type**

Bases TypedDict

Structure of option type field [RFC 791].

value: **int**
Option number.

action: **str**
Action.

change: **bool**
Change flag.

HOPOPT Unassigned Options

For HOPOPT unassigned options as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.opt.type`	Option Type
0	0	`hopopt.opt.type.value`	Option Number
0	0	`hopopt.opt.type.action`	Action (`00-11`)
0	2	`hopopt.opt.type.change`	Change Flag (`0/1`)
1	8	`hopopt.opt.length`	Length of Option Data
2	16	`hopopt.opt.data`	Option Data

class pcapkit.protocols.internet.hopopt.**DataType_Opt_None**

Bases DataType_Option

Structure of HOPOPT unassigned options [RFC 8200].

data: **bytes**
Option data.

HOPOPT Padding Options

`Pad1` Option

For HOPOPT `Pad1` option as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.pad.type`	Option Type
0	0	`hopopt.pad.type.value`	Option Number
0	0	`hopopt.pad.type.action`	Action (`00`)
0	2	`hopopt.pad.type.change`	Change Flag (`0`)

class pcapkit.protocols.internet.hopopt.**DataType_Opt_Pad1**

Bases DataType_Option

Structure of HOPOPT padding options [RFC 8200].

length: Literal[1]
Option length.

`PadN` Option

For HOPOPT `PadN` option as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.pad.type`	Option Type
0	0	`hopopt.pad.type.value`	Option Number
0	0	`hopopt.pad.type.action`	Action (`00`)
0	2	`hopopt.pad.type.change`	Change Flag (`0`)
1	8	`hopopt.opt.length`	Length of Option Data
2	16	`hopopt.pad.padding`	Padding

class pcapkit.protocols.internet.hopopt.**DataType_Opt_PadN**

Bases DataType_Option

Structure of HOPOPT padding options [RFC 8200].

padding: bytes
Padding data.

HOPOPT Tunnel Encapsulation Limit Option

For HOPOPT Tunnel Encapsulation Limit option as described in RFC 2473, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.tun.type`	Option Type
0	0	`hopopt.tun.type.value`	Option Number
0	0	`hopopt.tun.type.action`	Action (`00`)
0	2	`hopopt.tun.type.change`	Change Flag (`0`)
1	8	`hopopt.tun.length`	Length of Option Data
2	16	`hopopt.tun.limit`	Tunnel Encapsulation Limit

class pcapkit.protocols.internet.hopopt.**DataType_Opt_TUN**

Bases DataType_Option

Structure of HOPOPT Tunnel Encapsulation Limit option [RFC 2473].

limit: *int*
Tunnel encapsulation limit.

HOPOPT Router Alert Option

For HOPOPT Router Alert option as described in RFC 2711, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.ra.type	Option Type
0	0	hopopt.ra.type.value	Option Number
0	0	hopopt.ra.type.action	Action (00)
0	2	hopopt.ra.type.change	Change Flag (0)
1	8	hopopt.opt.length	Length of Option Data
2	16	hopopt.ra.value	Value

class pcapkit.protocols.internet.hopopt.**DataType_Opt_RA**

Bases DataType_Option

Structure of HOPOPT Router Alert option [RFC 2711].

value: *int*
Router alert code value.

alert: **pcapkit.const.ipv6.router_alter.RouterAlert**
Router alert enumeration.

HOPOPT CALIPSO Option

For HOPOPT CALIPSO option as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.calipso.type	Option Type
0	0	hopopt.calipso.type.value	Option Number
0	0	hopopt.calipso.type.action	Action (00)
0	2	hopopt.calipso.type.change	Change Flag (0)
1	8	hopopt.calipso.length	Length of Option Data
2	16	hopopt.calipso.domain	CALIPSO Domain of Interpretation
6	48	hopopt.calipso.cmpt_len	Cmpt Length
7	56	hopopt.calipso.level	Sens Level
8	64	hopopt.calipso.chksum	Checksum (CRC-16)
9	72	hopopt.calipso.bitmap	Compartment Bitmap

class pcapkit.protocols.internet.hopopt.**DataType_Opt_CALIPSO**

Bases DataType_Option

Structure of HOPOPT CALIPSO option [RFC 5570].

domain: *int*
CALIPSO domain of interpretation.

cmpt_len: *int*
Compartment length.

level: *int*
Sene level.

chksum: *bytes*
Checksum (CRC-16).

bitmap: *Tuple[str]*
Compartment bitmap.

HOPOPT SMF_DPD Option

I-DPD Mode

For IPv6 SMF_DPD option header in I-DPD mode as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.smf_dpd.type	Option Type
0	0	hopopt.smf_dpd.type.value	Option Number
0	0	hopopt.smf_dpd.type.action	Action (00)
0	2	hopopt.smf_dpd.type.change	Change Flag (0)
1	8	hopopt.smf_dpd.length	Length of Option Data
2	16	hopopt.smf_dpd.dpd_type	DPD Type (0)
2	17	hopopt.smf_dpd.tid_type	TaggerID Type
2	20	hopopt.smf_dpd.tid_len	TaggerID Length
3	24	hopopt.smf_dpd.tid	TaggerID
?	?	hopopt.smf_dpd.id	Identifier

class pcapkit.protocols.internet.hopopt.**DataType_Opt_SMF_I_PDP**

Bases DataType_Option

Structure of HOPOPT SMF_DPD option in **I-DPD** mode [RFC 5570].

dpd_type: *Literal[I-DPD]*
DPD type.

tid_type: *pcapkit.const.ipv6.tagger_id.TaggerID*
TaggerID type.

tid_len: *int*
TaggerID length.

tid: *int*
TaggerID.

id: *bytes*
Identifier.

H-DPD Mode

For IPv6 SMF_DPD option header in H-DPD mode as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.smf_dpd.type	Option Type
0	0	hopopt.smf_dpd.type.value	Option Number
0	0	hopopt.smf_dpd.type.action	Action (00)
0	2	hopopt.smf_dpd.type.change	Change Flag (0)
1	8	hopopt.smf_dpd.length	Length of Option Data
2	16	hopopt.smf_dpd.dpd_type	DPD Type (1)
2	17	hopopt.smf_dpd.hav	Hash Assist Value

class pcapkit.protocols.internet.hopopt.**DataType_Opt_SMF_H_PDP**

Bases DataType_Option

Structure of HOPOPT SMF_DPD option in **H-DPD** mode [RFC 5570].

dpd_type: Literal[H-DPD]
DPD type.

hav: str
Hash assist value (as *binary* string).

HOPOPT PDM Option

For HOPOPT PDM option as described in RFC 8250, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.pdm.type	Option Type
0	0	hopopt.pdm.type.value	Option Number
0	0	hopopt.pdm.type.action	Action (00)
0	2	hopopt.pdm.type.change	Change Flag (0)
1	8	hopopt.pdm.length	Length of Option Data
2	16	hopopt.pdm.scaledtlr	Scale Delta Time Last Received
3	24	hopopt.pdm.scaledtls	Scale Delta Time Last Sent
4	32	hopopt.pdm.psntp	Packet Sequence Number This Packet
6	48	hopopt.pdm.psnlr	Packet Sequence Number Last Received
8	64	hopopt.pdm.deltatlr	Delta Time Last Received
10	80	hopopt.pdm.deltatls	Delta Time Last Sent

class pcapkit.protocols.internet.hopopt.**DataType_Opt_PDM**

Bases DataType_Option

Structure of HOPOPT PDM option [RFC 8250].

scaledtlr: datetime.timedelta
Scale delta time last received.

scaledtls: datetime.timedelta
Scale delta time last sent.

psntp: int
Packet sequence number this packet.

psnlr: int
Packet sequence number last received.

deltatlr: datetime.timedelta
Delta time last received.

deltatls: datetime.timedelta
Delta time last sent.

HOPOPT Quick Start Option

For HOPOPT Quick Start option as described in RFC 4782, its structure is described as below:

Octets	Bits	Name	Description
0	0	hopopt.qs.type	Option Type
0	0	hopopt.qs.type.value	Option Number
0	0	hopopt.qs.type.action	Action (00)
0	2	hopopt.qs.type.change	Change Flag (1)
1	8	hopopt.qs.length	Length of Option Data
2	16	hopopt.qs.func	Function (0/8)
2	20	hopopt.qs.rate	Rate Request / Report (in Kbps)
3	24	hopopt.qs.ttl	QS TTL / None
4	32	hopopt.qs.nounce	QS Nounce
7	62		Reserved

class pcapkit.protocols.internet.hopopt.**DataType_Opt_QS**

Bases DataType_Option

Structure of HOPOPT Quick Start option [RFC 8250].

func: *pcapkit.const.ipv6.qs_function.QSFunction*
Function.

rate: float
Rate request and/or report (in *Kbps*).

ttl: Optional[int]
QS TTL.

nounce: int
QS nounce.

HOPOPT RPL Option

For HOPOPT RPL option as described in RFC 6553, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.rpl.type`	Option Type
0	0	`hopopt.rpl.type.value`	Option Number
0	0	`hopopt.rpl.type.action`	Action (`01`)
0	2	`hopopt.rpl.type.change`	Change Flag (`1`)
1	8	`hopopt.rpl.length`	Length of Option Data
2	16	`hopopt.rpl.flags`	RPL Option Flags
2	16	`hopopt.rpl.flags.down`	Down Flag
2	17	`hopopt.rpl.flags.rank_error`	Rank-Error Flag
2	18	`hopopt.rpl.flags.fwd_error`	Forwarding-Error Flag
3	24	`hopopt.rpl.id`	RPL Instance ID
4	32	`hopopt.rpl.rank`	SenderRank
6	48	`hopopt.rpl.data`	Sub-TLVs

class pcapkit.protocols.internet.hopopt.**DataType_Opt_RPL**

Bases DataType_Option

Structure of HOPOPT `RPL` option [**RFC 6553**].

flags: *DataType_RPL_Flags*
RPL option flags.

id: int
RPL instance ID.

rank: int
Sender rank.

data: Optional[bytes]
Sub-TLVs (if `hopopt.rpl.length` is **GREATER THAN** `4`).

class pcapkit.protocols.internet.hopopt.**DataType_RPL_Flags**

Bases TypedDict

RPL option flags.

down: bool
Down flag.

rank_error: bool
Rank-Error flag.

fwd_error: bool
Forwarding-Error flag.

HOPOPT `MPL` Option

For HOPOPT `MPL` option as described in **RFC 7731**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.mpl.type`	Option Type
0	0	`hopopt.mpl.type.value`	Option Number
0	0	`hopopt.mpl.type.action`	Action (`01`)
0	2	`hopopt.mpl.type.change`	Change Flag (`1`)
1	8	`hopopt.mpl.length`	Length of Option Data
2	16	`hopopt.mpl.seed_len`	Seed-ID Length
2	18	`hopopt.mpl.flags`	MPL Option Flags
2	18	`hopopt.mpl.max`	Maximum SEQ Flag
2	19	`hopopt.mpl.verification`	Verification Flag
2	20		Reserved
3	24	`hopopt.mpl.seq`	Sequence
4	32	`hopopt.mpl.seed_id`	Seed-ID

class pcapkit.protocols.internet.hopopt.**DataType_Opt_MPL**

Bases DataType_Option

Structure of HOPOPT `MPL` option [**RFC 7731**].

seed_len: *pcapkit.const.ipv6.seed_id.SeedID*
Seed-ID length.

flags: *DataType_MPL_Flags*
MPL option flags.

seq: **int**
Sequence.

seed_id: **Optional[int]**
Seed-ID.

class pcapkit.protocols.internet.hopopt.**DataType_MPL_Flags**

Bases TypedDict

MPL option flags.

max: **bool**
Maximum sequence flag.

verification: **bool**
Verification flag.

HOPOPT `ILNP` Nounce Option

For HOPOPT `ILNP` Nounce option as described in **RFC 6744**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.ilnp.type`	Option Type
0	0	`hopopt.ilnp.type.value`	Option Number
0	0	`hopopt.ilnp.type.action`	Action (`10`)
0	2	`hopopt.ilnp.type.change`	Change Flag (`0`)
1	8	`hopopt.ilnp.length`	Length of Option Data
2	16	`hopopt.ilnp.value`	Nonce Value

class pcapkit.protocols.internet.hopopt.**DataType_Opt_ILNP**

Bases DataType_Option

Structure of HOPOPT `ILNP` Nonce option [RFC 6744].

value: *bytes*
Nonce value.

HOPOPT Line-Identification Option

For HOPOPT Line-Identification option as described in RFC 6788, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.lio.type`	Option Type
0	0	`hopopt.lio.type.value`	Option Number
0	0	`hopopt.lio.type.action`	Action (`10`)
0	2	`hopopt.lio.type.change`	Change Flag (`0`)
1	8	`hopopt.lio.length`	Length of Option Data
2	16	`hopopt.lio.lid_len`	Line ID Length
3	24	`hopopt.lio.lid`	Line ID

class pcapkit.protocols.internet.hopopt.**DataType_Opt_LIO**

Bases DataType_Option

Structure of HOPOPT Line-Identification option [RFC 6788].

lid_len: *int*
Line ID length.

lid: *bytes*
Line ID.

HOPOPT Jumbo Payload Option

For HOPOPT Jumbo Payload option as described in RFC 2675, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.jumbo.type`	Option Type
0	0	`hopopt.jumbo.type.value`	Option Number
0	0	`hopopt.jumbo.type.action`	Action (`11`)
0	2	`hopopt.jumbo.type.change`	Change Flag (`0`)
1	8	`hopopt.jumbo.length`	Length of Option Data
2	16	`hopopt.jumbo.payload_len`	Jumbo Payload Length

class pcapkit.protocols.internet.hopopt.**DataType_Opt_Jumbo**

Bases DataType_Option

Structure of HOPOPT Jumbo Payload option [RFC 2675].

payload_len: *int*
Jumbo payload length.

HOPOPT Home Address Option

For HOPOPT Home Address option as described in RFC 6275, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.home.type`	Option Type
0	0	`hopopt.home.type.value`	Option Number
0	0	`hopopt.home.type.action`	Action (`11`)
0	2	`hopopt.home.type.change`	Change Flag (`0`)
1	8	`hopopt.home.length`	Length of Option Data
2	16	`hopopt.home.ip`	Home Address

class pcapkit.protocols.internet.hopopt.**DataType_Opt_Home**

Bases DataType_Option

Structure of HOPOPT Home Address option [RFC 6275].

ip: *ipaddress.IPv6Address*
Home address.

HOPOPT `IP_DFF` Option

For HOPOPT `IP_DFF` option as described in RFC 6971, its structure is described as below:

Octets	Bits	Name	Description
0	0	`hopopt.ip_dff.type`	Option Type
0	0	`hopopt.ip_dff.type.value`	Option Number
0	0	`hopopt.ip_dff.type.action`	Action (`11`)
0	2	`hopopt.ip_dff.type.change`	Change Flag (`1`)
1	8	`hopopt.ip_dff.length`	Length of Option Data
2	16	`hopopt.ip_dff.version`	Version
2	18	`hopopt.ip_dff.flags`	Flags
2	18	`hopopt.ip_dff.flags.dup`	`DUP` Flag
2	19	`hopopt.ip_dff.flags.ret`	`RET` Flag
2	20		Reserved
3	24	`hopopt.ip_dff.seq`	Sequence Number

class pcapkit.protocols.internet.hopopt.**DataType_Opt_IP_DFF**

Bases DataType_Option

Structure of HOPOPT `IP_DFF` option [RFC 6971].

version: *int*
Version.

flags: *DataType_IP_DFF_Flags*
Flags.

seq: *int*
Sequence number.

class pcapkit.protocols.internet.hopopt.**DataType_IP_DFF_Flags**

Bases TypedDict

Flags.

dup: bool
DUP flag.

ret: bool
RET flag.

IP - Internet Protocol

pcapkit.protocols.internet.ip contains *IP* only, which is a base class for Internet Protocol (IP) protocol family*[0], eg. *IPv4*, *IPv6*, and *IPsec*.

class pcapkit.protocols.internet.ip.**IP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements all protocols in IP family.

- Internet Protocol version 4 (*IPv4*) [**RFC 791**]
- Internet Protocol version 6 (*IPv6*) [**RFC 2460**]
- Authentication Header (*AH*) [**RFC 4302**]
- Encapsulating Security Payload (ESP) [**RFC 4303**]

classmethod id()
Index ID of the protocol.

Returns Index ID of the protocol.

Return type Tuple[Literal['IPv4'], Literal['IPv6']]

property dst
Destination IP address.

Return type Union[ipaddress.IPv4Address, ipaddress.IPv6Address]

property src
Source IP address.

Return type Union[ipaddress.IPv4Address, ipaddress.IPv6Address]

IPsec - Internet Protocol Security

pcapkit.protocols.internet.ipsec contains *IPsec* only, which is a base class for Internet Protocol Security (IPsec) protocol family*[0], eg. *AH* and ESP†[0].

class pcapkit.protocols.internet.ipsec.**IPsec**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.ip.IP*

Abstract base class for IPsec protocol family.

- Authentication Header (*AH*) [**RFC 4302**]
- Encapsulating Security Payload (ESP) [**RFC 4303**]

classmethod id()
Index ID of the protocol.

[0] https://en.wikipedia.org/wiki/Internet_Protocol
[0] https://en.wikipedia.org/wiki/IPsec
[0] ESP is currently **NOT** implemented.

Returns Index ID of the protocol.

Return type Tuple[Literal['AH'], Literal['ESP']]

property dst
Destination IP address.

Raises ***UnsupportedCall*** – This protocol doesn't support *dst*.

property src
Source IP address.

Raises ***UnsupportedCall*** – This protocol doesn't support *src*.

IPv4 - Internet Protocol version 4

pcapkit.protocols.internet.ipv4 contains *IPv4* only, which implements extractor for Internet Protocol version 4 (IPv4)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	ip.version	Version (4)
0	4	ip.hdr_len	Internal Header Length (IHL)
1	8	ip.dsfield.dscp	Differentiated Services Code Point (DSCP)
1	14	ip.dsfield.ecn	Explicit Congestion Notification (ECN)
2	16	ip.len	Total Length
4	32	ip.id	Identification
6	48		Reserved Bit (must be \x00)
6	49	ip.flags.df	Don't Fragment (DF)
6	50	ip.flags.mf	More Fragments (MF)
6	51	ip.frag_offset	Fragment Offset
8	64	ip.ttl	Time To Live (TTL)
9	72	ip.proto	Protocol (Transport Layer)
10	80	ip.checksum	Header Checksum
12	96	ip.src	Source IP Address
16	128	ip.dst	Destination IP Address
20	160	ip.options	IP Options (if IHL > 5)

class pcapkit.protocols.internet.ipv4.**IPv4**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.ip.IP*

This class implements Internet Protocol version 4.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[20]

_read_ipv4_addr()
Read IP address.

[0] https://en.wikipedia.org/wiki/IPv4

Returns Parsed IP address.

Return type ipaddress.IPv4Address

_read_ipv4_options(*size=None*)
Read IPv4 option list.

Parameters **size** (*Optional[int]*) – buffer size

Returns Tuple[Tuple[pcapkit.const.ipv4.option_number.OptionNumber], Dict[str, Union[DataType_Opt, Tuple[DataType_Opt]]]]: IPv4 option list and extracted IPv4 options

_read_mode_donone(*size*, *kind*)
Read options require no process.

Parameters

- **size** (*int*) – length of option
- **kind** (*int*) – option kind value

Returns extracted option

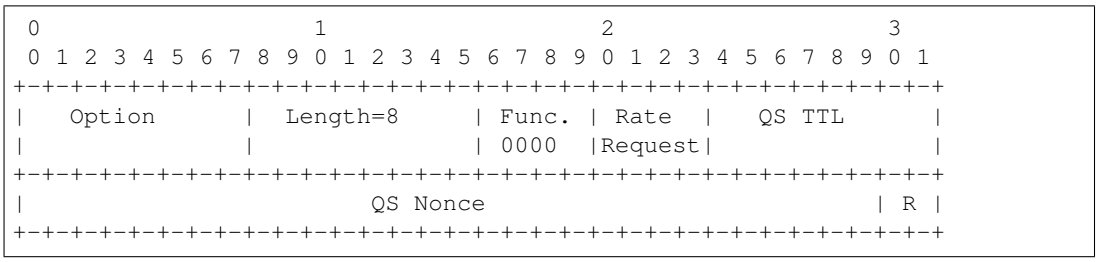
Return type *DataType_Opt_Do_None*

Raises ***ProtocolError*** – If `size` is **LESS THAN** `3`.

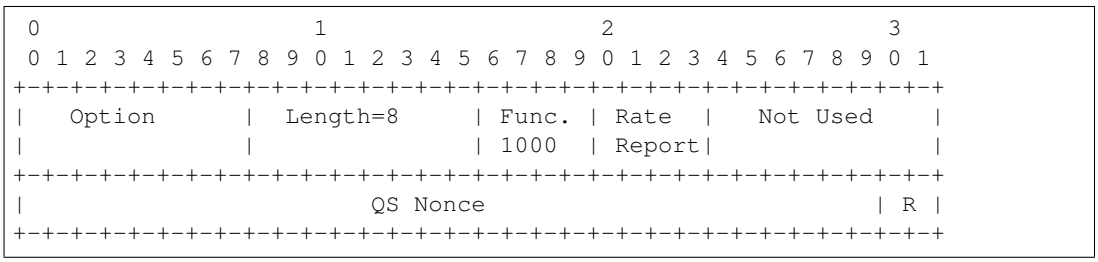
_read_mode_qs(*size*, *kind*)
Read Quick Start option.

Structure of Quick-Start (QS) option [**RFC 4782**]:

- A Quick-Start Request



- Report of Approved Rate



Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[25]*) – option kind value (QS)

Returns extracted Quick Start option

Return type *DataType_Opt_QuickStart*

Raises ***ProtocolError*** – If the option is malformed.

_read_mode_route(*size*, *kind*)
Read options with route data.

Structure of these options [**RFC 791**]:

- Loose Source Route

```
+--------+--------+--------+---------//--------+
|10000011| length | pointer|     route data    |
+--------+--------+--------+---------//--------+
```

- Strict Source Route

```
+--------+--------+--------+---------//--------+
|10001001| length | pointer|     route data    |
+--------+--------+--------+---------//--------+
```

- Record Route

```
+--------+--------+--------+---------//--------+
|00000111| length | pointer|     route data    |
+--------+--------+--------+---------//--------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[7, 131, 137]*) – option kind value (RR/LSR/SSR)

Returns extracted option with route data

Return type *DataType_Opt_Route_Data*

Raises ***ProtocolError*** – If the option is malformed.

_read_mode_rsralt(*size*, *kind*)
Read Router Alert option.

Structure of Router Alert (RTRALT) option [**RFC 2113**]:

```
+--------+--------+--------+--------+
|10010100|00000100|  2 octet value  |
+--------+--------+--------+--------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[140]*) – option kind value (RTRALT)

Returns extracted option with security info

Return type *DataType_Opt_RouterAlert*

Raises ***ProtocolError*** – If `size` is **NOT** `4`.

_read_mode_sec(*size*, *kind*)
Read options with security info.

Structure of these options [**RFC 1108**]:

- Security (SEC)

```
+------------+------------+------------+-------------//----------+
|  10000010  |  XXXXXXXX  |  SSSSSSSS  |  AAAAAAA[1]    AAAAAAA0 |
|            |            |            |         [0]             |
+------------+------------+------------+-------------//----------+
  TYPE = 130     LENGTH   CLASSIFICATION         PROTECTION
                                LEVEL              AUTHORITY
                                                     FLAGS
```

- Extended Security (ESEC)

```
+------------+------------+------------+-------//-------+
|  10000101  |  000LLLLL  |  AAAAAAAA  |  add sec info  |
+------------+------------+------------+-------//-------+
 TYPE = 133      LENGTH     ADDITIONAL      ADDITIONAL
                           SECURITY INFO     SECURITY
                            FORMAT CODE        INFO
```

c

_read_mode_tr(*size*, *kind*)

Read Traceroute option.

Structure of Traceroute (TR) option [RFC 6814]:

```
 0               8              16              24
+-+-+-+-+-+-+-+-+---------------+---------------+---------------+
|F| C |  Number |     Length    |           ID Number           |
+-+-+-+-+-+-+-+-+---------------+---------------+---------------+
|      Outbound Hop Count       |       Return Hop Count        |
+---------------+---------------+---------------+---------------+
|                     Originator IP Address                     |
+---------------+---------------+---------------+---------------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[82]*) – option kind value (TR)

Returns extracted Traceroute option

Return type *DataType_Opt_Traceroute*

Raises ***ProtocolError*** – If `size` is **NOT** `12`.

_read_mode_ts(*size*, *kind*)

Read Time Stamp option.

Structure of Timestamp (TS) option [RFC 791]:

```
+--------+--------+--------+--------+
|01000100| length | pointer|oflw|flg|
+--------+--------+--------+--------+
|         internet address          |
+--------+--------+--------+--------+
|             timestamp             |
+--------+--------+--------+--------+
|                 .                 |
```

(continues on next page)

(continued from previous page)

```
                    .
                    .
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[68]*) – option kind value (TS)

Returns extracted Time Stamp option

Return type *DataType_Opt_TimeStamp*

Raises **ProtocolError** – If the option is malformed.

_read_mode_unpack(*size*, *kind*)
Read options require unpack process.

Parameters

- **size** (*int*) – length of option
- **kind** (*int*) – option kind value

Returns extracted option

Return type *DataType_Opt_Unpack*

Raises **ProtocolError** – If `size` is **LESS THAN** `3`.

_read_opt_type(*kind*)
Read option type field.

Parameters **kind** (*int*) – option kind value

Returns extracted IPv4 option

Return type *DataType_IPv4_Option_Type*

classmethod id()
Index ID of the protocol.

Returns Index ID of the protocol.

Return type Literal['IPv4']

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Internet Protocol version 4 (IPv4).

Structure of IPv4 header [**RFC 791**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
```

(continues on next page)

(continued from previous page)

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Source Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPv4*

property length

Header length of corresponding protocol.

Return type int

property name

Name of corresponding protocol.

Return type Literal['Internet Protocol version 4']

property protocol

Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

pcapkit.protocols.internet.ipv4.**IPv4_OPT: *DataType_IPv4_OPT***

IPv4 option dict parsing mapping.

copy	class	number	kind	length	process	name
0	0	0	0			[RFC 791] End of Option List
0	0	1	1			[RFC 791] No-Operation
0	0	7	7	?	2	[RFC 791] Record Route
0	0	11	11	4	1	[RFC 1063][RFC 1191] MTU Probe
0	0	12	12	4	1	[RFC 1063][RFC 1191] MTU Reply
0	0	25	25	8	3	[RFC 4782] Quick-Start
0	2	4	68	?	4	[RFC 791] Time Stamp
0	2	18	82	?	5	[RFC 1393][RFC 6814] Traceroute
1	0	2	130	?	6	[RFC 1108] Security
1	0	3	131	?	2	[RFC 791] Loose Source Route
1	0	5	133	?	6	[RFC 1108] Extended Security
1	0	8	136	4	1	[RFC 791][RFC 6814] Stream ID
1	0	9	137	?	2	[RFC 791] Strict Source Route
1	0	17	145	?	0	[RFC 1385][RFC 6814] Ext. Inet. Protocol
1	0	20	148	4	7	[RFC 2113] Router Alert

See also:

pcapkit.protocols.internet.ipv4.DataType_IPv4_OPT

pcapkit.protocols.internet.ipv4.**process_opt**: **Dict[int, Callable[[*pcapkit.protocols.interne**

Process method for IPv4 options.

Code	Method	Description
0	*_read_mode_donone()*	do nothing
1	*_read_mode_unpack()*	unpack according to size
2	*_read_mode_route()*	unpack route data options
3	*_read_mode_qs()*	unpack Quick-Start
4	*_read_mode_ts()*	unpack Time Stamp
5	*_read_mode_tr()*	unpack Traceroute
6	*_read_mode_sec()*	unpack (Extended) Security
7	*_read_mode_rsralt()*	unpack Router Alert

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.internet.ipv4.**DataType_IPv4**

Bases TypedDict

Structure of IPv4 header [RFC 791].

version: **Literal[4]**
Version (4).

hdr_len: **int**
Internal header length (IHL).

dsfield: ***DataType_DS_Field***
Type of services.

len: **int**
Total length.

id: **int**
Identification.

flags: ***DataType_IPv4_Flags***
Flags.

frag_offset: **int**
Fragment offset.

ttl: **int**
Time to live (TTL).

proto: ***pcapkit.const.reg.transtype.TransType***
Protocol (transport layer).

checksum: **bytes**
Header checksum.

src: **ipaddress.IPv4Address**
Source IP address.

dst: *ipaddress.IPv4Address*
Destination IP address.

opt: *Tuple[pcapkit.const.ipv4.option_number.OptionNumber]*
Tuple of option acronyms.

packet: *bytes*
Rase packet data.

class pcapkit.protocols.internet.ipv4.**DataType_DS_Field**

Bases TypedDict

IPv4 DS fields.

dscp: *DataType_IPv4_DSCP*
Differentiated services code point (DSCP).

ecn: *pcapkit.const.ipv4.tos_ecn.ToSECN*
Explicit congestion notification (ECN).

class pcapkit.protocols.internet.ipv4.**DataType_IPv4_DSCP**

Bases TypedDict

Differentiated services code point (DSCP).

pre: *pcapkit.const.ipv4.tos_pre.ToSPrecedence*
ToS precedence.

del: *pcapkit.const.ipv4.tos_del.ToSDelay*
ToS delay.

thr: *pcapkit.const.ipv4.tos_thr.ToSThroughput*
ToS throughput.

rel: *pcapkit.const.ipv4.tos_rel.ToSReliability*
ToS reliability.

class pcapkit.protocols.internet.ipv4.**DataType_IPv4_Flags**

Bases TypedDict

IPv4 flags.

df: *bool*
Dont's fragment (DF).

mf: *bool*
More fragments (MF).

class pcapkit.protocols.internet.ipv4.**DataType_Opt**

Bases TypedDict

IPv4 option data.

kind: *int*
Option kind.

type: *DataType_IPv4_Option_Type*
Option type info.

length: *int*
Option length.

class pcapkit.protocols.internet.ipv4.**DataType_IPv4_OPT**

Bases TypedDict

IPv4 option `dict` parsing mapping.

flag: **bool**
If the length of option is **GREATER THAN** `1`.

desc: **str**
Description string, also attribute name.

proc: **Optional[int]**
Process method that data bytes need (when *`flag`* is `True`).

See also:

`pcapkit.protocols.internet.ipv4.process_opt`

IPv4 Option Type

For IPv4 option type field as described in **RFC 791**, its structure is described as below:

Octets	Bits	Name	Descriptions
0	0	`ip.opt.type.copy`	Copied Flag (`0`/`1`)
0	1	`ip.opt.type.class`	Option Class (`0`-`3`)
0	3	`ip.opt.type.number`	Option Number

class pcapkit.protocols.internet.ipv4.**DataType_IPv4_Option_Type**

Bases TypedDict

Structure of option type field [**RFC 791**].

copy: **bool**
Copied flag.

class: ***pcapkit.const.ipv4.option_class.OptionClass***
Option class.

number: **int**
Option number.

IPv4 Miscellaneous Options

1-Byte Options

class pcapkit.protocols.internet.ipv4.**DataType_Opt_1_Byte**

Bases DataType_Opt

1-byte options.

length: **Literal[1]**
Option length.

Permission Options

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Permission**

Bases DataType_Opt

Permission options (*length* is 2).

length: **Literal[2]**
Option length.

flag: **Literal[True]**
Permission flag.

No Process Options

For IPv4 options require no process, its structure is described as below:

Octets	Bits	Name	Description
0	0	ip.opt.kind	Kind
0	0	ip.opt.type.copy	Copied Flag
0	1	ip.opt.type.class	Option Class
0	3	ip.opt.type.number	Option Number
1	8	ip.opt.length	Length
2	16	ip.opt.data	Kind-specific Data

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Do_None**

Bases DataType_Opt

Structure of IPv4 options.

data: **bytes**
Kind-specific data.

Unpack Process Options

For IPv4 options require unpack process, its structure is described as below:

Octets	Bits	Name	Description
0	0	ip.opt.kind	Kind
0	0	ip.opt.type.copy	Copied Flag
0	1	ip.opt.type.class	Option Class
0	3	ip.opt.type.number	Option Number
1	8	ip.opt.length	Length
2	16	ip.opt.data	Kind-specific Data

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Unpack**

Bases DataType_Opt

Structure of IPv4 options.

data: **int**
Kind-specific data.

IPv4 Options with Route Data

For IPv4 options with route data as described in RFC 791, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ip.opt.kind`	Kind (`7`/`131`/`137`)
0	0	`ip.opt.type.copy`	Copied Flag (`0`)
0	1	`ip.opt.type.class`	Option Class (`0`/`1`)
0	3	`ip.opt.type.number`	Option Number (`3`/`7`/`9`)
1	8	`ip.opt.length`	Length
2	16	`ip.opt.pointer`	Pointer (`4`)
3	24	`ip.opt.data`	Route Data

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Route_Data**

Bases DataType_Opt

Structure of IPv4 options with route data [RFC 791].

pointer: *int*
Pointer.

data: *Optional[Tuple[ipaddress.IPv4Address]]*
Route data.

IPv4 Quick Start Options

For IPv4 Quick Start options as described in RFC 4782, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ip.qs.kind`	Kind (`25`)
0	0	`ip.qs.type.copy`	Copied Flag (`0`)
0	1	`ip.qs.type.class`	Option Class (`0`)
0	3	`ip.qs.type.number`	Option Number (`25`)
1	8	`ip.qs.length`	Length (`8`)
2	16	`ip.qs.func`	Function (`0`/`8`)
2	20	`ip.qs.rate`	Rate Request / Report (in Kbps)
3	24	`ip.qs.ttl`	QS TTL / `None`
4	32	`ip.qs.nounce`	QS Nounce
7	62		Reserved (`\x00\x00`)

class pcapkit.protocols.internet.ipv4.**DataType_Opt_QuickStart**

Bases DataType_Opt

Structure of Quick-Start (QS) option [RFC 4782].

func: *pcapkit.const.ipv4.qs_function.QSFunction*
Function.

rate: *int*
Rate request / report (in Kbps).

ttl: *Optional[int]*
QS TTL.

nounce: *int*
QS nounce.

IPv4 Time Stamp Option

For IPv4 Time Stamp option as described in RFC 791, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ip.ts.kind`	Kind (`25`)
0	0	`ip.ts.type.copy`	Copied Flag (`0`)
0	1	`ip.ts.type.class`	Option Class (`0`)
0	3	`ip.ts.type.number`	Option Number (`25`)
1	8	`ip.ts.length`	Length (`40`)
2	16	`ip.ts.pointer`	Pointer (`5`)
3	24	`ip.ts.overflow`	Overflow Octets
3	28	`ip.ts.flag`	Flag
4	32	`ip.ts.ip`	Internet Address
8	64	`ip.ts.timestamp`	Timestamp

class pcapkit.protocols.internet.ipv4.**DataType_Opt_TimeStamp**

Bases DataType_Opt

Structure of Timestamp (TS) option [RFC 791].

pointer: *int*
Pointer.

overflow: *int*
Overflow octets.

flag: *int*
Flag.

ip: **Optional[Tuple[ipaddress.IPv4Address]]**
Array of Internet addresses (if *flag* is 1/3).

timestamp: **Optional[Tuple[datetime.datetime]]**
Array of timestamps (if *flag* is 0/1/3).

data: **Optional[bytes]**
Timestamp data (if *flag* is unknown).

IPv4 Traceroute Option

For IPv4 Traceroute option as described in RFC 6814, its structure is described as below:

Octets	Bits	Name	Description
0	0	ip.tr.kind	Kind (82)
0	0	ip.tr.type.copy	Copied Flag (0)
0	1	ip.tr.type.class	Option Class (0)
0	3	ip.tr.type.number	Option Number (18)
1	8	ip.tr.length	Length (12)
2	16	ip.tr.id	ID Number
4	32	ip.tr.ohc	Outbound Hop Count
6	48	ip.tr.rhc	Return Hop Count
8	64	ip.tr.ip	Originator IP Address

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Traceroute**

Bases DataType_Opt

Structure of Traceroute (TR) option [RFC 6814].

id: **int**
ID number.

ohc: **int**
Outbound hop count.

rhc: **int**
Return hop count.

ip: **ipaddress.IPv4Address**
Originator IP address.

IPv4 Options with Security Info

For IPv4 options with security info as described in RFC 1108, its structure is described as below:

Octets	Bits	Name	Description
0	0	ip.sec.kind	Kind (130/133)
0	0	ip.sec.type.copy	Copied Flag (1)
0	1	ip.sec.type.class	Option Class (0)
0	3	ip.sec.type.number	Option Number (2)
1	8	ip.sec.length	Length (3)
2	16	ip.sec.level	Classification Level
3	24	ip.sec.flags	Protection Authority Flags

class pcapkit.protocols.internet.ipv4.**DataType_Opt_Security_Info**

Bases DataType_Opt

Structure of IPv4 options with security info [RFC 791].

level: ***pcapkit.const.ipv4.classification_level.ClassificationLevel***
Classification level.

flags: **Tuple[*DataType_SEC_Flags*]**
Array of protection authority flags.

class pcapkit.protocols.internet.ipv4.**DataType_SEC_Flags**

Bases pcapkit.corekit.infoclass.Info

Protection authority flags, as mapping of protection authority bit assignments *enumeration* and bool flags.

IPv4 Traceroute Option

For IPv4 Router Alert option as described in **RFC 2113**, its structure is described as below:

Octets	Bits	Name	Description
0	0	ip.rsralt.kind	Kind (148)
0	0	ip.rsralt.type.copy	Copied Flag (1)
0	1	ip.rsralt.type.class	Option Class (0)
0	3	ip.rsralt.type.number	Option Number (20)
1	8	ip.rsralt.length	Length (4)
2	16	ip.rsralt.alert	Alert
2	16	ip.rsralt.code	Alert Code

class pcapkit.protocols.internet.ipv4.**DataType_Opt_RouterAlert**

Bases DataType_Opt

Structure of Router Alert (RTRALT) option [**RFC 2113**].

alert: *pcapkit.const.ipv4.router_alert.RouterAlert*
Alert.

code: int
Alert code.

IPv6-Frag - Fragment Header for IPv6

pcapkit.protocols.internet.ipv6_frag contains *IPv6_Frag* only, which implements extractor for Fragment Header for IPv6 (IPv6-Frag)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	frag.next	Next Header
1	8		Reserved
2	16	frag.offset	Fragment Offset
3	29		Reserved
3	31	frag.mf	More Flag
4	32	frag.id	Identification

class pcapkit.protocols.internet.ipv6_frag.**IPv6_Frag**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements Fragment Header for IPv6.

classmethod **__index__**()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

[0] https://en.wikipedia.org/wiki/IPv6_packet#Fragment

__length_hint__()
Return an estimated length for the object.

Return type Literal[8]

__post_init__(*file*, *length=None*, *, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, *, *extension=False*, ***kwargs*)
Read Fragment Header for IPv6.

Structure of IPv6-Frag header **[RFC 8200]**:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |   Reserved    |      Fragment Offset    |Res|M|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identification                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPv6_Frag*

property alias
Acronym of corresponding protocol.

Return type Literal['IPv6-Frag']

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Fragment Header for IPv6']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.ipv6_frag.**DataType_IPv6_Frag**

Bases TypedDict

Structure of IPv6-Frag header [RFC 8200].

next: ***pcapkit.const.reg.transtype.TransType***
Next header.

offset: **int**
Fragment offset.

mf: **bool**
More flag.

id: **int**
Identification.

IPv6-Opts - Destination Options for IPv6

`pcapkit.protocols.internet.ipv6_opts` contains `IPv6_Opts` only, which implements extractor for Destination Options for IPv6 (IPv6-Opts)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`opt.next`	Next Header
1	8	`opt.length`	Header Extensive Length
2	16	`opt.options`	Options

class pcapkit.protocols.internet.ipv6_opts.**IPv6_Opts**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements Destination Options for IPv6.

[0] https://en.wikipedia.org/wiki/IPv6_packet#Hop-by-hop_options_and_destination_options

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[2]

__post_init__(*file*, *length=None*, *, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

_read_ipv6_opts_options(*length*)
Read IPv6-Opts options.

Positional arguments: length (int): length of options

Returns Tuple[Tuple[pcapkit.const.ipv6.option.Option], Dict[str, DataType_Option]]: extracted IPv6-Opts options

Raises ***ProtocolError*** – If the threshold is **NOT** matching.

_read_opt_calipso(*code*, *, *desc*)
Read IPv6-Opts CALIPSO option.

Structure of IPv6-Opts CALIPSO option [**RFC 5570**]:

```
------------------------------------------------------------
| Next Header | Hdr Ext Len   | Option Type | Option Length|
+-------------+---------------+-------------+--------------+
|              CALIPSO Domain of Interpretation             |
+-------------+---------------+-------------+--------------+
| Cmpt Length |  Sens Level   |     Checksum (CRC-16)       |
+-------------+---------------+-------------+--------------+
|      Compartment Bitmap (Optional; variable length)       |
+-------------+---------------+-------------+--------------+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

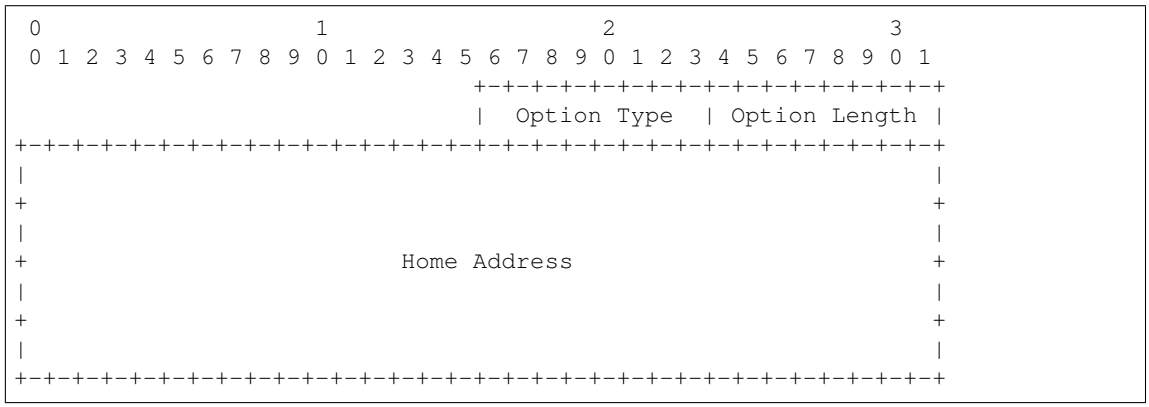
Returns parsed option data

Return type *DataType_Dest_Opt_CALIPSO*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_home(*code*, *, *desc*)
Read IPv6-Opts Home Address option.

Structure of IPv6-Opts Home Address option [RFC 6275]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

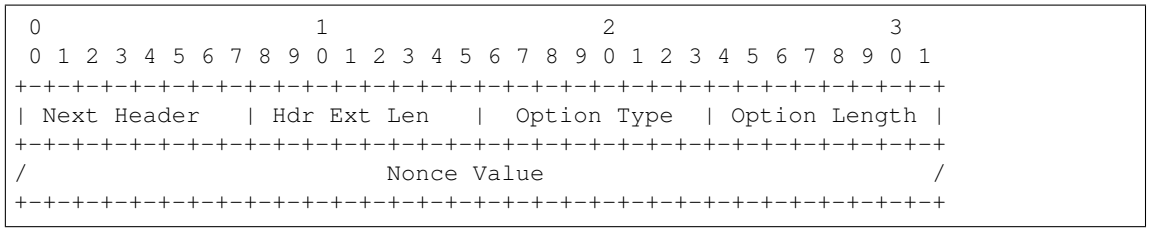
Returns parsed option data

Return type *DataType_Dest_Opt_Home*

Raises ***ProtocolError*** – If `ipv6_opts.jumbo.length` is **NOT** `16`.

_read_opt_ilnp(*code*, *, *desc*)
Read IPv6-Opts `ILNP` Nonce option.

Structure of IPv6-Opts `ILNP` Nonce option [RFC 6744]:



Parameters **code** (*int*) – option type value

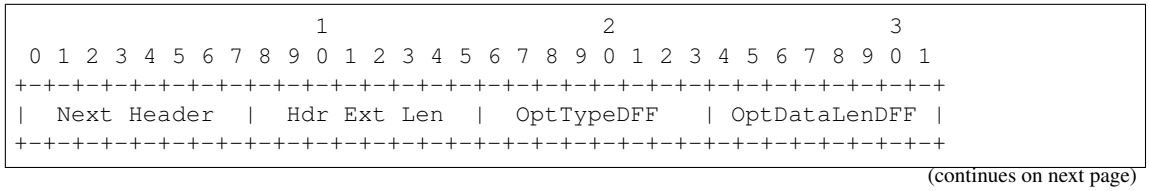
Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_ILNP*

_read_opt_ip_dff(*code*, *, *desc*)
Read IPv6-Opts `IP_DFF` option.

Structure of IPv6-Opts `IP_DFF` option [RFC 6971]:



(continues on next page)

(continued from previous page)

```
|VER|D|R|0|0|0|0|        Sequence Number        |      Pad1     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_IP_DFF*

Raises ***ProtocolError*** – If `ipv6_opts.ip_dff.length` is **NOT** `2`.

_read_opt_jumbo(*code*, ***, *desc*)

Read IPv6-Opts Jumbo Payload option.

Structure of IPv6-Opts Jumbo Payload option [**RFC 2675**]:

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  |  Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Jumbo Payload Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_Jumbo*

Raises ***ProtocolError*** – If `ipv6_opts.jumbo.length` is **NOT** `4`.

_read_opt_lio(*code*, ***, *desc*)

Read IPv6-Opts Line-Identification option.

Structure of IPv6-Opts Line-Identification option [**RFC 6788**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  | Option Length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| LineIDLen     |     Line ID...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_LIO*

_read_opt_mpl(*code*, ***, *desc*)

Read IPv6-Opts `MPL` option.

Structure of IPv6-Opts `MPL` option [**RFC 7731**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  |  Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| S |M|V|  rsv  |   sequence    |      seed-id (optional)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_MPL*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_none(*code*, ***, *desc*)

Read IPv6-Opts unassigned options.

Structure of IPv6-Opts unassigned options [**RFC 8200**]:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|  Option Type  |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_None*

_read_opt_pad(*code*, ***, *desc*)

Read IPv6-Opts padding options.

Structure of IPv6-Opts padding options [**RFC 8200**]:

- `Pad1` option:

```
+-+-+-+-+-+-+-+-+
|       0       |
+-+-+-+-+-+-+-+-+
```

- `PadN` option:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|       1       |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type Union[*DataType_Dest_Opt_Pad1*, *DataType_Dest_Opt_PadN*]

Raises ***ProtocolError*** – If `code` is **NOT** `0` or `1`.

_read_opt_pdm(*code*, ***, *desc*)
Read IPv6-Opts PDM option.

Structure of IPv6-Opts PDM option [**RFC 8250**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Option Length |    ScaleDTLR  |     ScaleDTLS |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   PSN This Packet             |  PSN Last Received            |
|-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Delta Time Last Received    |  Delta Time Last Sent         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_PDM*

Raises ***ProtocolError*** – If `ipv6_opts.pdm.length` is **NOT** `10`.

_read_opt_qs(*code*, ***, *desc*)
Read IPv6-Opts Quick Start option.

Structure of IPv6-Opts Quick-Start option [**RFC 4782**]:

- A Quick-Start Request:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option      |  Length=6     | Func. | Rate  |   QS TTL      |
|               |               | 0000  |Request|               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         QS Nonce                              | R |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Report of Approved Rate:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option      |  Length=6     | Func. | Rate  |   Not Used    |
|               |               | 1000  | Report|               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         QS Nonce                              | R |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

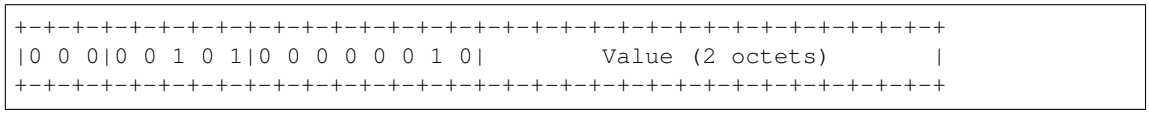
Returns parsed option data

Return type *DataType_Dest_Opt_QS*

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_ra(*code*, ***, *desc*)
Read IPv6-Opts Router Alert option.

Structure of IPv6-Opts Router Alert option [**RFC 2711**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

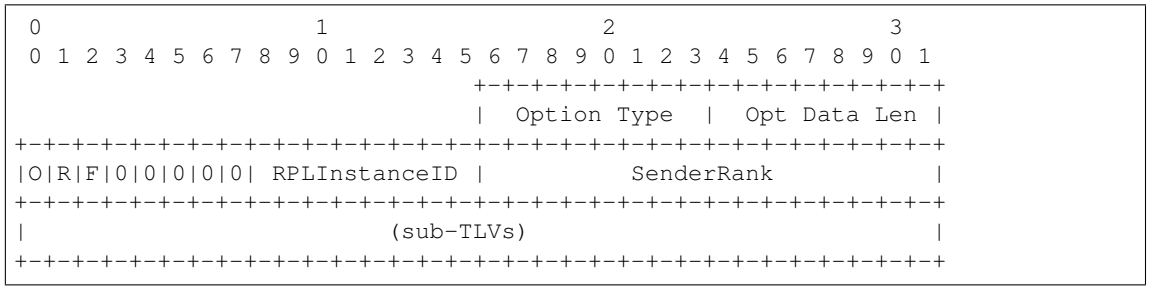
Returns parsed option data

Return type *DataType_Dest_Opt_RA*

Raises ***ProtocolError*** – If `ipv6_opts.tun.length` is **NOT** `2`.

_read_opt_rpl(*code*, ***, *desc*)
Read IPv6-Opts `RPL` option.

Structure of IPv6-Opts `RPL` option [**RFC 6553**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

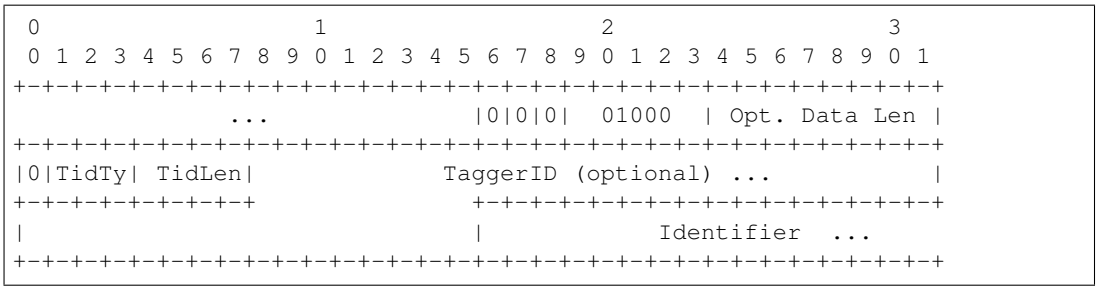
Return type *DataType_Dest_Opt_RPL*

Raises ***ProtocolError*** – If `ipv6_opts.rpl.length` is **LESS THAN** `4`.

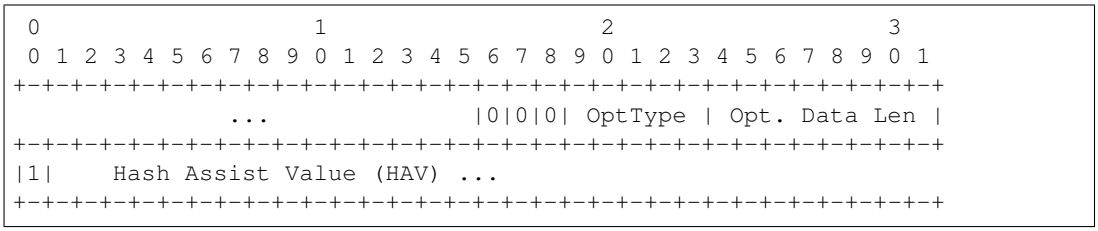
_read_opt_smf_dpd(*code*, ***, *desc*)
Read IPv6-Opts `SMF_DPD` option.

Structure of IPv6-Opts `SMF_DPD` option [**RFC 5570**]:

- IPv6 `SMF_DPD` option header in **I-DPD** mode



- IPv6 `SMF_DPD` option header in **H-DPD** mode



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

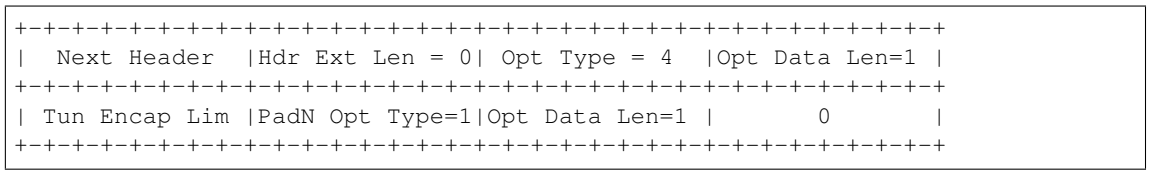
Return type Union[*DataType_Dest_Opt_SMF_I_PDP*, *DataType_Dest_Opt_SMF_H_PDP*]

Raises ***ProtocolError*** – If the option is malformed.

_read_opt_tun(*code*, *, *desc*)

Read IPv6-Opts Tunnel Encapsulation Limit option.

Structure of IPv6-Opts Tunnel Encapsulation Limit option [**RFC 2473**]:



Parameters **code** (*int*) – option type value

Keyword Arguments **desc** (*str*) – option description

Returns parsed option data

Return type *DataType_Dest_Opt_TUN*

Raises ***ProtocolError*** – If `ipv6_opts.tun.length` is **NOT** `1`.

_read_opt_type(*kind*)

Read option type field.

Parameters **kind** (*int*) – option kind value

Returns extracted IPv6-Opts option type field

Return type *DataType_IPv6_Opts_Option_Type*

make(***kwargs*)

Make (construct) packet data.

Keyword Arguments *****kwargs*** – Arbitrary keyword arguments.

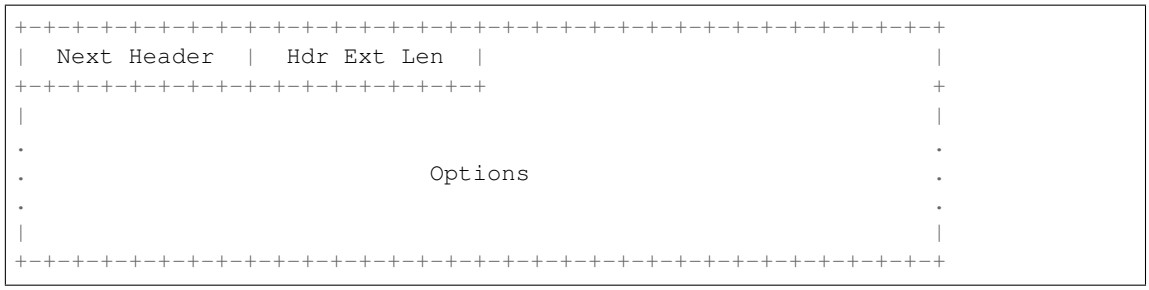
Returns Constructed packet data.

Return type bytes

read(*length=None*, *, *extension=False*, ***kwargs*)

Read Destination Options for IPv6.

Structure of IPv6-Opts header [**RFC 8200**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPv6_Opts*

property alias
Acronym of corresponding protocol.

Return type Literal['IPv6-Opts']

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Destination Options for IPv6']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

pcapkit.protocols.internet.ipv6_opts.**_IPv6_Opts_ACT: Dict[str, str]**
IPv6-Opts unknown option actions.

Code	Action
00	skip over this option and continue processing the header
01	discard the packet
10	discard the packet and, regardless of whether or not the packet's Destination Address was a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type
11	discard the packet and, only if the packet's Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type

pcapkit.protocols.internet.ipv6_opts.**_IPv6_Opts_OPT: Dict[int, Tuple[str, str]]**
IPv6-Opts options.

Code	Acronym	Option	Reference
0x00	pad	Pad1	[RFC 8200] 0
0x01	pad	PadN	[RFC 8200]
0x04	tun	Tunnel Encapsulation Limit	[RFC 2473] 1
0x05	ra	Router Alert	[RFC 2711] 2
0x07	calipso	Common Architecture Label IPv6 Security Option	[RFC 5570]
0x08	smf_dpd	Simplified Multicast Forwarding	[RFC 6621]
0x0F	pdm	Performance and Diagnostic Metrics	[RFC 8250] 10
0x26	qs	Quick-Start	[RFC 4782][RFC Errata 2034] 6
0x63	rpl	Routing Protocol for Low-Power and Lossy Networks	[RFC 6553]
0x6D	mpl	Multicast Protocol for Low-Power and Lossy Networks	[RFC 7731]
0x8B	ilnp	Identifier-Locator Network Protocol Nonce	[RFC 6744]
0x8C	lio	Line-Identification Option	[RFC 6788]
0xC2	jumbo	Jumbo Payload	[RFC 2675]
0xC9	home	Home Address	[RFC 6275]
0xEE	ip_dff	Depth-First Forwarding	[RFC 6971]

pcapkit.protocols.internet.ipv6_opts.**_IPv6_Opts_NULL: Dict[int, str]**
IPv6-Opts unknown option descriptions.

Code	Description	Reference
0x1E	RFC3692-style Experiment	[RFC 4727]
0x3E	RFC3692-style Experiment	[RFC 4727]
0x4D	Deprecated	[RFC 7731]
0x5E	RFC3692-style Experiment	[RFC 4727]
0x7E	RFC3692-style Experiment	[RFC 4727]
0x8A	Endpoint Identification	**DEPRECATED**
0x9E	RFC3692-style Experiment	[RFC 4727]
0xBE	RFC3692-style Experiment	[RFC 4727]
0xDE	RFC3692-style Experiment	[RFC 4727]
0xFE	RFC3692-style Experiment	[RFC 4727]

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.internet.ipv6_opts.**DataType_IPv6_Opts**

Bases TypedDict

Structure of IPv6-Opts header [RFC 8200].

next: *pcapkit.const.reg.transtype.TransType*
Next header.

length: *int*
Header extensive length.

options: *Tuple[pcapkit.const.ipv6.option.Option]*
Array of option acronyms.

packet: *bytes*
Packet data.

class pcapkit.protocols.internet.ipv6_opts.**DataType_Option**

Bases TypedDict

IPv6_Opts option.

desc: *str*
Option description.

type: *DataType_IPv6_Opts_Option_Type*
Option type.

length: *int*
Option length.

Note: This attribute is **NOT** the length specified in the IPv6-Opts optiona data, rather the *total* length of the current option.

IPv6-Opts Option Type

For IPv6-Opts option type field as described in RFC 791, its structure is described as below:

Octets	Bits	Name	Descriptions
0	0	`ipv6_opts.opt.type.value`	Option Number
0	0	`ipv6_opts.opt.type.action`	Action (`00`-`11`)
0	2	`ipv6_opts.opt.type.change`	Change Flag (`0`/`1`)

class pcapkit.protocols.internet.ipv6_opts.**DataType_IPv6_Opts_Option_Type**

Bases TypedDict

Structure of option type field [RFC 791].

value: *int*
Option number.

action: *str*
Action.

change: *bool*
Change flag.

IPv6-Opts Unassigned Options

For IPv6-Opts unassigned options as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.opt.type`	Option Type
0	0	`ipv6_opts.opt.type.value`	Option Number
0	0	`ipv6_opts.opt.type.action`	Action (`00-11`)
0	2	`ipv6_opts.opt.type.change`	Change Flag (`0/1`)
1	8	`ipv6_opts.opt.length`	Length of Option Data
2	16	`ipv6_opts.opt.data`	Option Data

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_None**

Bases DataType_Option

Structure of IPv6-Opts unassigned options [RFC 8200].

data: **bytes**
Option data.

IPv6-Opts Padding Options

`Pad1` Option

For IPv6-Opts `Pad1` option as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.pad.type`	Option Type
0	0	`ipv6_opts.pad.type.value`	Option Number
0	0	`ipv6_opts.pad.type.action`	Action (`00`)
0	2	`ipv6_opts.pad.type.change`	Change Flag (`0`)

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_Pad1**

Bases DataType_Option

Structure of IPv6-Opts padding options [RFC 8200].

length: **Literal[1]**
Option length.

`PadN` Option

For IPv6-Opts `PadN` option as described in RFC 8200, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.pad.type`	Option Type
0	0	`ipv6_opts.pad.type.value`	Option Number
0	0	`ipv6_opts.pad.type.action`	Action (`00`)
0	2	`ipv6_opts.pad.type.change`	Change Flag (`0`)
1	8	`ipv6_opts.opt.length`	Length of Option Data
2	16	`ipv6_opts.pad.padding`	Padding

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_PadN**

Bases DataType_Option

Structure of IPv6-Opts padding options [RFC 8200].

padding: **bytes**
Padding data.

IPv6-Opts Tunnel Encapsulation Limit Option

For IPv6-Opts Tunnel Encapsulation Limit option as described in RFC 2473, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.tun.type	Option Type
0	0	ipv6_opts.tun.type.value	Option Number
0	0	ipv6_opts.tun.type.action	Action (00)
0	2	ipv6_opts.tun.type.change	Change Flag (0)
1	8	ipv6_opts.tun.length	Length of Option Data
2	16	ipv6_opts.tun.limit	Tunnel Encapsulation Limit

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_TUN**

Bases DataType_Option

Structure of IPv6-Opts Tunnel Encapsulation Limit option [RFC 2473].

limit: **int**
Tunnel encapsulation limit.

IPv6-Opts Router Alert Option

For IPv6-Opts Router Alert option as described in RFC 2711, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.ra.type	Option Type
0	0	ipv6_opts.ra.type.value	Option Number
0	0	ipv6_opts.ra.type.action	Action (00)
0	2	ipv6_opts.ra.type.change	Change Flag (0)
1	8	ipv6_opts.opt.length	Length of Option Data
2	16	ipv6_opts.ra.value	Value

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_RA**

Bases DataType_Option

Structure of IPv6-Opts Router Alert option [RFC 2711].

value: **int**
Router alert code value.

alert: **pcapkit.const.ipv6.router_alter.RouterAlert**
Router alert enumeration.

IPv6-Opts CALIPSO Option

For IPv6-Opts CALIPSO option as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.calipso.type	Option Type
0	0	ipv6_opts.calipso.type.value	Option Number
0	0	ipv6_opts.calipso.type.action	Action (00)
0	2	ipv6_opts.calipso.type.change	Change Flag (0)
1	8	ipv6_opts.calipso.length	Length of Option Data
2	16	ipv6_opts.calipso.domain	CALIPSO Domain of Interpretation
6	48	ipv6_opts.calipso.cmpt_len	Cmpt Length
7	56	ipv6_opts.calipso.level	Sens Level
8	64	ipv6_opts.calipso.chksum	Checksum (CRC-16)
9	72	ipv6_opts.calipso.bitmap	Compartment Bitmap

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_CALIPSO**

Bases DataType_Option

Structure of IPv6-Opts CALIPSO option [RFC 5570].

domain: **int**
CALIPSO domain of interpretation.

cmpt_len: **int**
Compartment length.

level: **int**
Sene level.

chksum: **bytes**
Checksum (CRC-16).

bitmap: **Tuple[str]**
Compartment bitmap.

IPv6-Opts SMF_DPD Option

I-DPD Mode

For IPv6 SMF_DPD option header in I-DPD mode as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.smf_dpd.type	Option Type
0	0	ipv6_opts.smf_dpd.type.value	Option Number
0	0	ipv6_opts.smf_dpd.type.action	Action (00)
0	2	ipv6_opts.smf_dpd.type.change	Change Flag (0)
1	8	ipv6_opts.smf_dpd.length	Length of Option Data
2	16	ipv6_opts.smf_dpd.dpd_type	DPD Type (0)
2	17	ipv6_opts.smf_dpd.tid_type	TaggerID Type
2	20	ipv6_opts.smf_dpd.tid_len	TaggerID Length
3	24	ipv6_opts.smf_dpd.tid	TaggerID
?	?	ipv6_opts.smf_dpd.id	Identifier

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_SMF_I_PDP**

Bases DataType_Option

Structure of IPv6-Opts `SMF_DPD` option in **I-DPD** mode [RFC 5570].

dpd_type: **Literal[I-DPD]**
DPD type.

tid_type: ***pcapkit.const.ipv6.tagger_id.TaggerID***
TaggerID type.

tid_len: **int**
TaggerID length.

tid: **int**
TaggerID.

id: **bytes**
Identifier.

H-DPD Mode

For IPv6 `SMF_DPD` option header in H-DPD mode as described in RFC 5570, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.smf_dpd.type`	Option Type
0	0	`ipv6_opts.smf_dpd.type.value`	Option Number
0	0	`ipv6_opts.smf_dpd.type.action`	Action (`00`)
0	2	`ipv6_opts.smf_dpd.type.change`	Change Flag (`0`)
1	8	`ipv6_opts.smf_dpd.length`	Length of Option Data
2	16	`ipv6_opts.smf_dpd.dpd_type`	DPD Type (`1`)
2	17	`ipv6_opts.smf_dpd.hav`	Hash Assist Value

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_SMF_H_PDP**

Bases DataType_Option

Structure of IPv6-Opts `SMF_DPD` option in **H-DPD** mode [RFC 5570].

dpd_type: **Literal[H-DPD]**
DPD type.

hav: **str**
Hash assist value (as *binary* string).

IPv6-Opts `PDM` Option

For IPv6-Opts `PDM` option as described in RFC 8250, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.pdm.type`	Option Type
0	0	`ipv6_opts.pdm.type.value`	Option Number
0	0	`ipv6_opts.pdm.type.action`	Action (`00`)
0	2	`ipv6_opts.pdm.type.change`	Change Flag (`0`)
1	8	`ipv6_opts.pdm.length`	Length of Option Data
2	16	`ipv6_opts.pdm.scaledtlr`	Scale Delta Time Last Received
3	24	`ipv6_opts.pdm.scaledtls`	Scale Delta Time Last Sent
4	32	`ipv6_opts.pdm.psntp`	Packet Sequence Number This Packet
6	48	`ipv6_opts.pdm.psnlr`	Packet Sequence Number Last Received
8	64	`ipv6_opts.pdm.deltatlr`	Delta Time Last Received
10	80	`ipv6_opts.pdm.deltatls`	Delta Time Last Sent

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_PDM**

Bases DataType_Option

Structure of IPv6-Opts `PDM` option [**RFC 8250**].

scaledtlr: **datetime.timedelta**
Scale delta time last received.

scaledtls: **datetime.timedelta**
Scale delta time last sent.

psntp: **int**
Packet sequence number this packet.

psnlr: **int**
Packet sequence number last received.

deltatlr: **datetime.timedelta**
Delta time last received.

deltatls: **datetime.timedelta**
Delta time last sent.

IPv6-Opts Quick Start Option

For IPv6-Opts Quick Start option as described in **RFC 4782**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.qs.type`	Option Type
0	0	`ipv6_opts.qs.type.value`	Option Number
0	0	`ipv6_opts.qs.type.action`	Action (`00`)
0	2	`ipv6_opts.qs.type.change`	Change Flag (`1`)
1	8	`ipv6_opts.qs.length`	Length of Option Data
2	16	`ipv6_opts.qs.func`	Function (`0`/`8`)
2	20	`ipv6_opts.qs.rate`	Rate Request / Report (in Kbps)
3	24	`ipv6_opts.qs.ttl`	QS TTL / `None`
4	32	`ipv6_opts.qs.nounce`	QS Nounce
7	62		Reserved

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_QS**

Bases DataType_Option

Structure of IPv6-Opts Quick Start option [RFC 8250].

func: *pcapkit.const.ipv6.qs_function.QSFunction*
Function.

rate: float
Rate request and/or report (in *Kbps*).

ttl: Optional[int]
QS TTL.

nounce: int
QS nounce.

IPv6-Opts RPL Option

For IPv6-Opts RPL option as described in RFC 6553, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.rpl.type	Option Type
0	0	ipv6_opts.rpl.type.value	Option Number
0	0	ipv6_opts.rpl.type.action	Action (01)
0	2	ipv6_opts.rpl.type.change	Change Flag (1)
1	8	ipv6_opts.rpl.length	Length of Option Data
2	16	ipv6_opts.rpl.flags	RPL Option Flags
2	16	ipv6_opts.rpl.flags.down	Down Flag
2	17	ipv6_opts.rpl.flags.rank_error	Rank-Error Flag
2	18	ipv6_opts.rpl.flags.fwd_error	Forwarding-Error Flag
3	24	ipv6_opts.rpl.id	RPL Instance ID
4	32	ipv6_opts.rpl.rank	SenderRank
6	48	ipv6_opts.rpl.data	Sub-TLVs

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_RPL**

Bases DataType_Option

Structure of IPv6-Opts RPL option [RFC 6553].

flags: *DataType_RPL_Flags*
RPL option flags.

id: int
RPL instance ID.

rank: int
Sender rank.

data: Optional[bytes]
Sub-TLVs (if ipv6_opts.rpl.length is **GREATER THAN** 4).

class pcapkit.protocols.internet.ipv6_opts.**DataType_RPL_Flags**

Bases TypedDict

RPL option flags.

down: bool
Down flag.

rank_error: bool
Rank-Error flag.

fwd_error: bool
Forwarding-Error flag.

IPv6-Opts MPL Option

For IPv6-Opts MPL option as described in RFC 7731, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.mpl.type	Option Type
0	0	ipv6_opts.mpl.type.value	Option Number
0	0	ipv6_opts.mpl.type.action	Action (01)
0	2	ipv6_opts.mpl.type.change	Change Flag (1)
1	8	ipv6_opts.mpl.length	Length of Option Data
2	16	ipv6_opts.mpl.seed_len	Seed-ID Length
2	18	ipv6_opts.mpl.flags	MPL Option Flags
2	18	ipv6_opts.mpl.max	Maximum SEQ Flag
2	19	ipv6_opts.mpl.verification	Verification Flag
2	20		Reserved
3	24	ipv6_opts.mpl.seq	Sequence
4	32	ipv6_opts.mpl.seed_id	Seed-ID

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_MPL**

Bases DataType_Option

Structure of IPv6-Opts MPL option [RFC 7731].

seed_len: *pcapkit.const.ipv6.seed_id.SeedID*
Seed-ID length.

flags: *DataType_MPL_Flags*
MPL option flags.

seq: int
Sequence.

seed_id: Optional[int]
Seed-ID.

class pcapkit.protocols.internet.ipv6_opts.**DataType_MPL_Flags**

Bases TypedDict

MPL option flags.

max: bool
Maximum sequence flag.

verification: bool
Verification flag.

IPv6-Opts ILNP Nounce Option

For IPv6-Opts ILNP Nounce option as described in RFC 6744, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.ilnp.type	Option Type
0	0	ipv6_opts.ilnp.type.value	Option Number
0	0	ipv6_opts.ilnp.type.action	Action (10)
0	2	ipv6_opts.ilnp.type.change	Change Flag (0)
1	8	ipv6_opts.ilnp.length	Length of Option Data
2	16	ipv6_opts.ilnp.value	Nonce Value

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_ILNP**

Bases DataType_Option

Structure of IPv6-Opts ILNP Nonce option [RFC 6744].

value: **bytes**
Nonce value.

IPv6-Opts Line-Identification Option

For IPv6-Opts Line-Identification option as described in RFC 6788, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.lio.type	Option Type
0	0	ipv6_opts.lio.type.value	Option Number
0	0	ipv6_opts.lio.type.action	Action (10)
0	2	ipv6_opts.lio.type.change	Change Flag (0)
1	8	ipv6_opts.lio.length	Length of Option Data
2	16	ipv6_opts.lio.lid_len	Line ID Length
3	24	ipv6_opts.lio.lid	Line ID

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_LIO**

Bases DataType_Option

Structure of IPv6-Opts Line-Identification option [RFC 6788].

lid_len: **int**
Line ID length.

lid: **bytes**
Line ID.

IPv6-Opts Jumbo Payload Option

For IPv6-Opts Jumbo Payload option as described in RFC 2675, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.jumbo.type`	Option Type
0	0	`ipv6_opts.jumbo.type.value`	Option Number
0	0	`ipv6_opts.jumbo.type.action`	Action (`11`)
0	2	`ipv6_opts.jumbo.type.change`	Change Flag (`0`)
1	8	`ipv6_opts.jumbo.length`	Length of Option Data
2	16	`ipv6_opts.jumbo.payload_len`	Jumbo Payload Length

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_Jumbo**

Bases DataType_Option

Structure of IPv6-Opts Jumbo Payload option [RFC 2675].

payload_len: *int*
Jumbo payload length.

IPv6-Opts Home Address Option

For IPv6-Opts Home Address option as described in RFC 6275, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipv6_opts.home.type`	Option Type
0	0	`ipv6_opts.home.type.value`	Option Number
0	0	`ipv6_opts.home.type.action`	Action (`11`)
0	2	`ipv6_opts.home.type.change`	Change Flag (`0`)
1	8	`ipv6_opts.home.length`	Length of Option Data
2	16	`ipv6_opts.home.ip`	Home Address

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_Home**

Bases DataType_Option

Structure of IPv6-Opts Home Address option [RFC 6275].

ip: *ipaddress.IPv6Address*
Home address.

IPv6-Opts `IP_DFF` Option

For IPv6-Opts `IP_DFF` option as described in RFC 6971, its structure is described as below:

Octets	Bits	Name	Description
0	0	ipv6_opts.ip_dff.type	Option Type
0	0	ipv6_opts.ip_dff.type.value	Option Number
0	0	ipv6_opts.ip_dff.type.action	Action (11)
0	2	ipv6_opts.ip_dff.type.change	Change Flag (1)
1	8	ipv6_opts.ip_dff.length	Length of Option Data
2	16	ipv6_opts.ip_dff.version	Version
2	18	ipv6_opts.ip_dff.flags	Flags
2	18	ipv6_opts.ip_dff.flags.dup	DUP Flag
2	19	ipv6_opts.ip_dff.flags.ret	RET Flag
2	20		Reserved
3	24	ipv6_opts.ip_dff.seq	Sequence Number

class pcapkit.protocols.internet.ipv6_opts.**DataType_Dest_Opt_IP_DFF**

Bases DataType_Option

Structure of IPv6-Opts IP_DFF option [**RFC 6971**].

version: **int**
Version.

flags: ***DataType_IP_DFF_Flags***
Flags.

seq: **int**
Sequence number.

class pcapkit.protocols.internet.ipv6_opts.**DataType_IP_DFF_Flags**

Bases TypedDict

Flags.

dup: **bool**
DUP flag.

ret: **bool**
RET flag.

IPv6-Route - Routing Header for IPv6

pcapkit.protocols.internet.ipv6_route contains *IPv6_Route* only, which implements extractor for Routing Header for IPv6 (IPv6-Route)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	route.next	Next Header
1	8	route.length	Header Extensive Length
2	16	route.type	Routing Type
3	24	route.seg_left	Segments Left
4	32	route.data	Type-Specific Data

class pcapkit.protocols.internet.ipv6_route.**IPv6_Route**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.internet.internet.Internet*

[0] https://en.wikipedia.org/wiki/IPv6_packet#Routing

This class implements Routing Header for IPv6.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[4]

__post_init__(*file*, *length=None*, *, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

_read_data_type_2(*length*)
Read IPv6-Route Type 2 data.

Structure of IPv6-Route Type 2 data [**RFC 6275**]:

Parameters **length** (*int*) – route data length

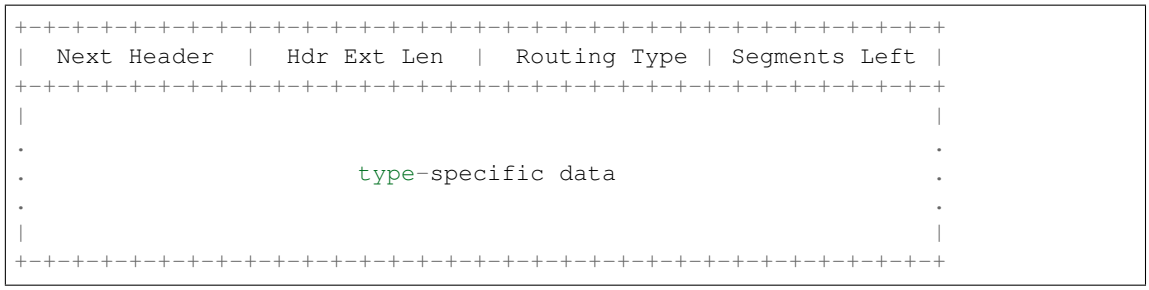
Returns parsed route data

Return type *DataType_IPv6_Route_2*

Raises ***ProtocolError*** – If `length` is **NOT** `20`.

_read_data_type_none(*length*)
Read IPv6-Route unknown type data.

Structure of IPv6-Route unknown type data [**RFC 8200**][**RFC 5095**]:



Parameters **length** (*int*) – route data length

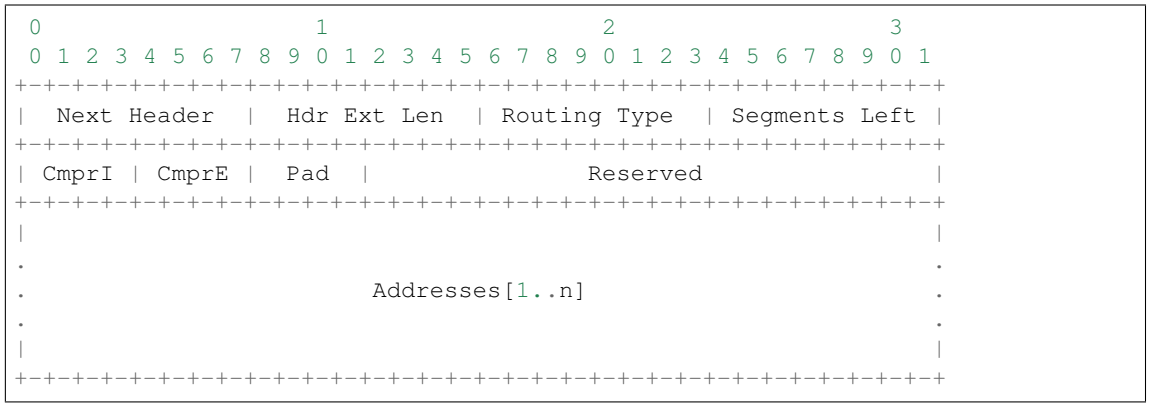
Returns parsed route data

Return type *DataType_IPv6_Route_None*

_read_data_type_rpl(*length*)

Read IPv6-Route RPL Source data.

Structure of IPv6-Route RPL Source data [**RFC 6554**]:



Parameters **length** (*int*) – route data length

Returns parsed route data

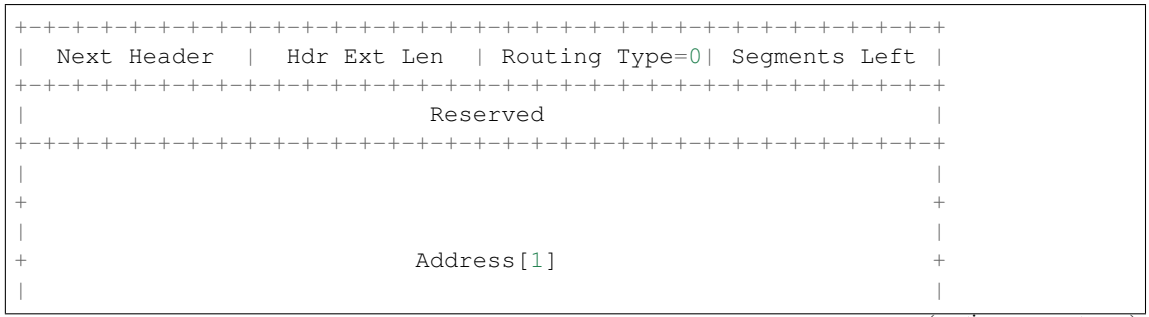
Return type *DataType_IPv6_Route_RPL*

Raises ***ProtocolError*** – If `length` is **NOT** `20`.

_read_data_type_src(*length*)

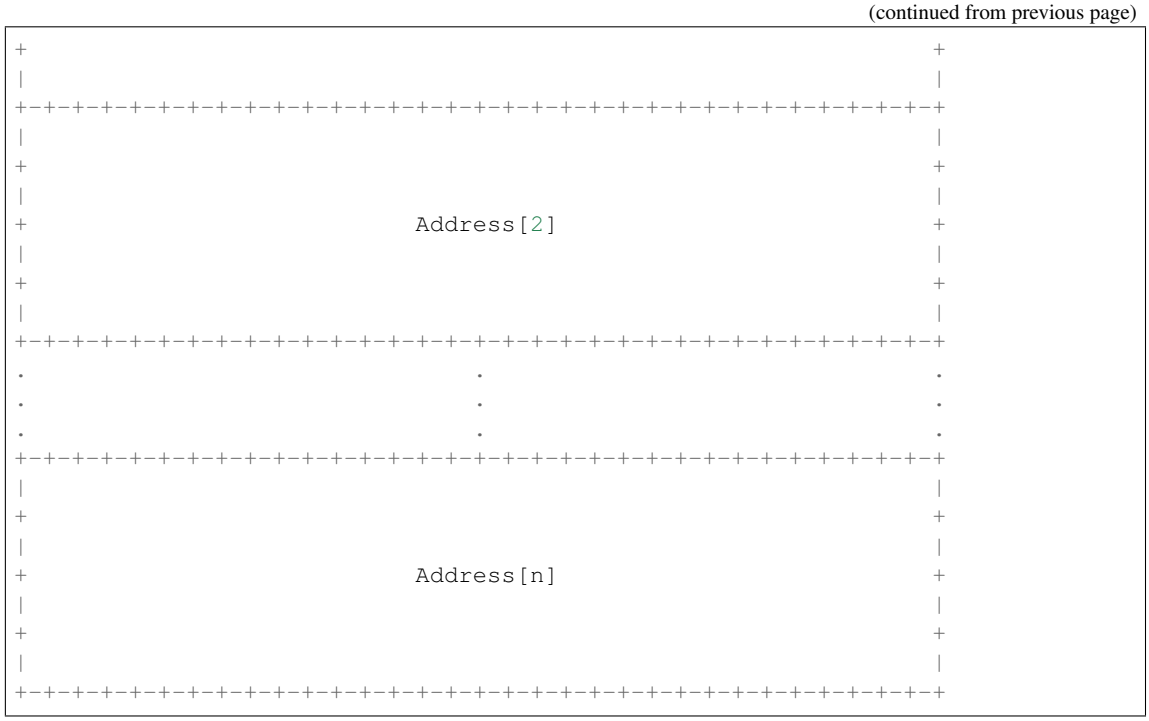
Read IPv6-Route Source Route data.

Structure of IPv6-Route Source Route data [**RFC 5095**]:



(continues on next page)

(continued from previous page)



Parameters **length** (*int*) – route data length

Returns parsed route data

Return type *DataType_IPv6_Route_Source*

make(***kwargs*)

Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

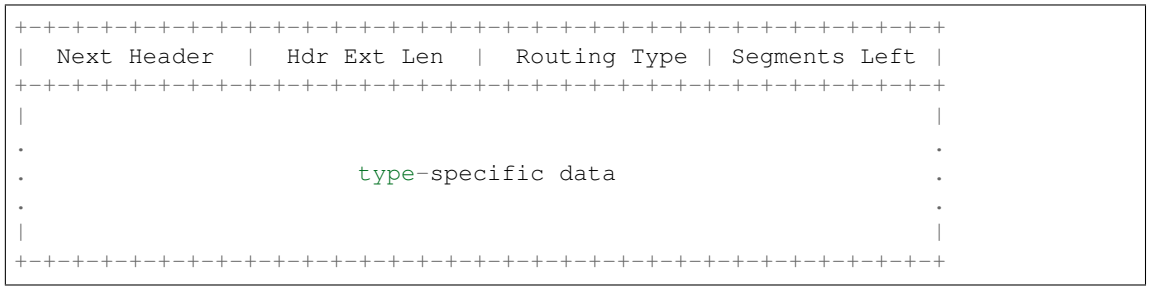
Returns Constructed packet data.

Return type bytes

read(*length=None*, ***, *extension=False*, ***kwargs*)

Read Routing Header for IPv6.

Structure of IPv6-Route header [**RFC 8200**][**RFC 5095**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.

- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPv6_Route*

property alias
Acronym of corresponding protocol.

Return type Literal['IPv6-Route']

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Routeing Header for IPv6']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

pcapkit.protocols.internet.ipv6_route.**_ROUTE_PROC: Dict[int, str]**
IPv6 routing processors.

Code	Processor	Note
0	*_read_data_type_src()*	[RFC 5095] DEPRECATED
2	*_read_data_type_2()*	[RFC 6275]
3	*_read_data_type_rpl()*	[RFC 6554]

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.internet.ipv6_route.**DataType_IPv6_Route**
Structure of IPv6-Route header [RFC 8200][RFC 5095].

next: ***pcapkit.const.reg.transtype.TransType***
Next header.

length: **int**
Header extensive length.

type: ***pcapkit.const.ipv6.routing.Routing***
Routing type.

seg_left: **int**
Segments left.

packet: *bytes*
Raw packet data.

IPv6-Route Unknown Type

For IPv6-Route unknown type data as described in RFC 8200 and RFC 5095, its structure is described as below:

Octets	Bits	Name	Description
0	0	`route.next`	Next Header
1	8	`route.length`	Header Extensive Length
2	16	`route.type`	Routing Type
3	24	`route.seg_left`	Segments Left
4	32	`route.data`	Type-Specific Data

class pcapkit.protocols.internet.ipv6_route.**DataType_IPv6_Route_None**

Bases TypedDict

Structure of IPv6-Route unknown type data [RFC 8200][RFC 5095].

data: *bytes*
Type-specific data.

IPv6-Route Source Route

For IPv6-Route Source Route data as described in RFC 5095, its structure is described as below:

Octets	Bits	Name	Description
0	0	`route.next`	Next Header
1	8	`route.length`	Header Extensive Length
2	16	`route.type`	Routing Type
3	24	`route.seg_left`	Segments Left
4	32		Reserved
8	64	`route.ip`	Address

class pcapkit.protocols.internet.ipv6_route.**DataType_IPv6_Route_Source**

Bases TypedDict

Structure of IPv6-Route Source Route data [RFC 5095].

ip: *Tuple[ipaddress.IPv6Address]*
Array of IPv6 addresses.

IPv6-Route Type 2

For IPv6-Route Type 2 data as described in RFC 6275, its structure is described as below:

Octets	Bits	Name	Description
0	0	`route.next`	Next Header
1	8	`route.length`	Header Extensive Length
2	16	`route.type`	Routing Type
3	24	`route.seg_left`	Segments Left
4	32		Reserved
8	64	`route.ip`	Home Address

class pcapkit.protocols.internet.ipv6_route.**DataType_IPv6_Route_2**

Bases TypedDict

Structure of IPv6-Route Type 2 data [RFC 6275].

ip: **ipaddress.IPv6Address**
Home IPv6 addresses.

IPv6-Route RPL Source

For IPv6-Route RPL Source data as described in RFC 6554, its structure is described as below:

Octets	Bits	Name	Description
0	0	`route.next`	Next Header
1	8	`route.length`	Header Extensive Length
2	16	`route.type`	Routing Type
3	24	`route.seg_left`	Segments Left
4	32	`route.cmpr_i`	CmprI
4	36	`route.cmpr_e`	CmprE
5	40	`route.pad`	Pad Size
5	44		Reserved
8	64	`route.ip`	Addresses

class pcapkit.protocols.internet.ipv6_route.**DataType_IPv6_Route_RPL**

Bases TypedDict

Structure of IPv6-Route RPL Source data [RFC 6554].

cmpr_i: **int**
CmprI.

cmpr_e: **int**
CmprE.

pad: **int**
Pad size.

ip: **Tuple[Union[ipaddress.IPv4Address, ipaddress.IPv6Address]]**
Array of IPv4 and/or IPv6 addresses.

IPv6 - Internet Protocol version 6

pcapkit.protocols.internet.ipv6 contains *IPv6* only, which implements extractor for Internet Protocol version 6 (IPv6)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	ip.version	Version (6)
0	4	ip.class	Traffic Class
1	12	ip.label	Flow Label
4	32	ip.payload	Payload Length (header excludes)
6	48	ip.next	Next Header
7	56	ip.limit	Hop Limit
8	64	ip.src	Source Address
24	192	ip.dst	Destination Address

class pcapkit.protocols.internet.ipv6.**IPv6**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.internet.ip.IP*

This class implements Internet Protocol version 6.

classmethod __index__()

Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()

Return an estimated length for the object.

Return type Literal[40]

_decode_next_layer(*ipv6*, *proto=None*, *length=None*)

Decode next layer extractor.

Parameters

- **ipv6** (DataType_IPv6) – info buffer
- **proto** (*str*) – next layer protocol name
- **length** (*int*) – valid (*not padding*) length

Returns current protocol with next layer extracted

Return type *DataType_IPv6*

_read_ip_addr()

Read IP address.

Returns Parsed IP address.

Return type ipaddress.IPv6Address

_read_ip_hextet()

Read first four hextets of IPv6.

Returns Parsed hextets data, including version number, traffic class and flow label.

Return type Tuple[int, int, int]

[0] https://en.wikipedia.org/wiki/IPv6_packet

classmethod id()
Index ID of the protocol.

Returns Index ID of the protocol.

Return type Literal['IPv6']

make(***kwargs*)
Make (construct) packet data.

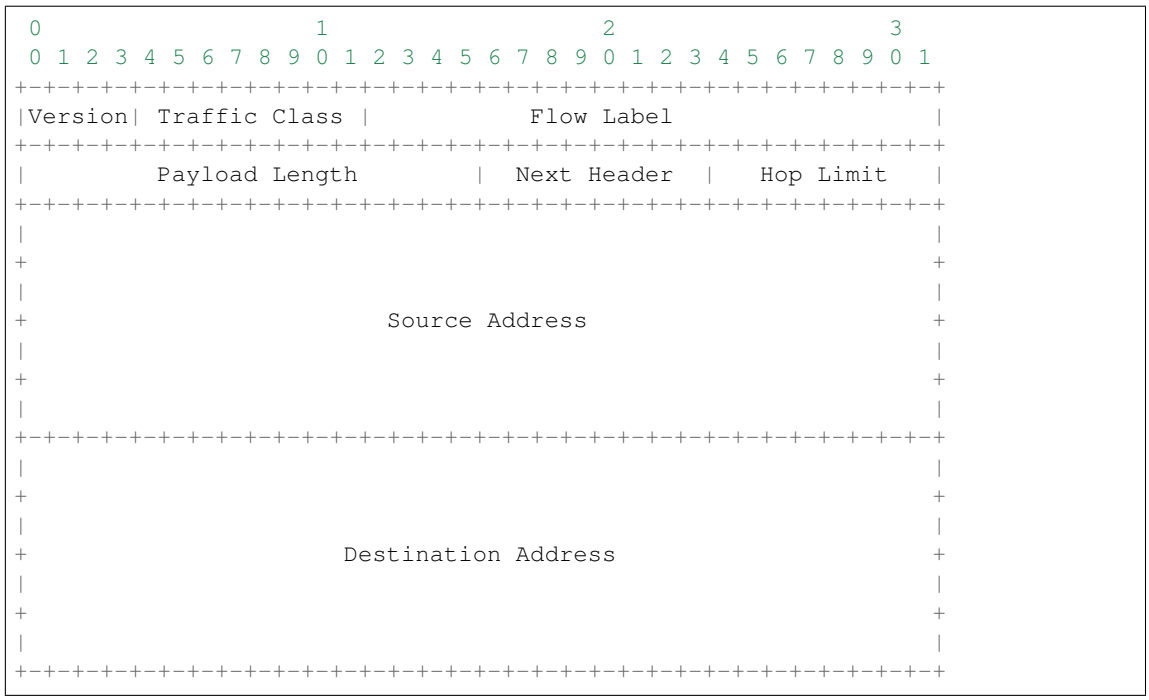
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Internet Protocol version 6 (IPv6).

Structure of IPv6 header [**RFC 2460**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPv6*

property length
Header length of corresponding protocol.

Return type int

property name
Name of corresponding protocol.

Return type Literal['Internet Protocol version 6']

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.ipv6.**DataType_IPv6**

Bases TypedDict

Structure of IPv6 header [RFC 2460].

version: Literal[6]
Version.

class: int
Traffic class.

label: int
Flow label.

payload: int
Payload length.

next: *pcapkit.const.reg.transtype.TransType*
Next header.

limit: int
Hop limit.

src: ipaddress.IPv6Address
Source address.

dst: ipaddress.IPv6Address
Destination address.

packet: bytes
Raw packet data.

IPX - Internetwork Packet Exchange

`pcapkit.protocols.internet.ipx` contains *IPX* only, which implements extractor for Internetwork Packet Exchange (IPX)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`ipx.cksum`	Checksum
2	16	`ipx.len`	Packet Length (header includes)
4	32	`ipx.count`	Transport Control (hop count)
5	40	`ipx.type`	Packet Type
6	48	`ipx.dst`	Destination Address
18	144	`ipx.src`	Source Address

[0] https://en.wikipedia.org/wiki/Internetwork_Packet_Exchange

class pcapkit.protocols.internet.ipx.**IPX**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements Internetwork Packet Exchange.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[30]

_read_ipx_address()
Read IPX address field.

Returns Parsed IPX address field.

Return type *DataType_IPX_Address*

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Internetwork Packet Exchange.

Args: length (Optional[int]): Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_IPX*

property dst
Destination IPX address.

Return type str

property length
Header length of corresponding protocol.

Return type Literal[30]

property name
Name of corresponding protocol.

Return type Literal['Internetwork Packet Exchange']

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

property src
Source IPX address.

Return type str

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.internet.ipx.**DataType_IPX**

Bases TypedDict

Structure of IPX header [RFC 1132].

chksum: **bytes**
Checksum.

len: **int**
Packet length (header includes).

count: **int**
Transport control (hop count).

type: ***pcapkit.const.ipx.packet.Packet***
Packet type.

dst: ***DataType_IPX_Address***
Destination address.

src: ***DataType_IPX_Address***
Source address.

For IPX address field, its structure is described as below:

Octets	Bits	Name	Description
0	0	`ipx.addr.network`	Network Number
4	32	`ipx.addr.node`	Node Number
10	80	`ipx.addr.socket`	Socket Number

class pcapkit.protocols.internet.ipx.**DataType_IPX_Address**

Bases TypedDict

Structure of IPX address.

network: **str**
Network number (`:` separated).

node: **str**
Node number (– separated).

socket: ***pcapkit.const.ipx.socket.Socket***
Socket number.

addr: **str**
Full address (`:` separated).

MH - Mobility Header

pcapkit.protocols.internet.mh contains *MH* only, which implements extractor for Mobility Header (MH)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`mh.next`	Next Header
1	8	`mh.length`	Header Length
2	16	`mh.type`	Mobility Header Type
3	24		Reserved
4	32	`mh.chksum`	Checksum
6	48	`mh.data`	Message Data

class pcapkit.protocols.internet.mh.**MH**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.internet.internet.Internet*

This class implements Mobility Header.

classmethod **__index__**()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[6]

__post_init__(*file*, *length=None*, *, *extension=False*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the protocol is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

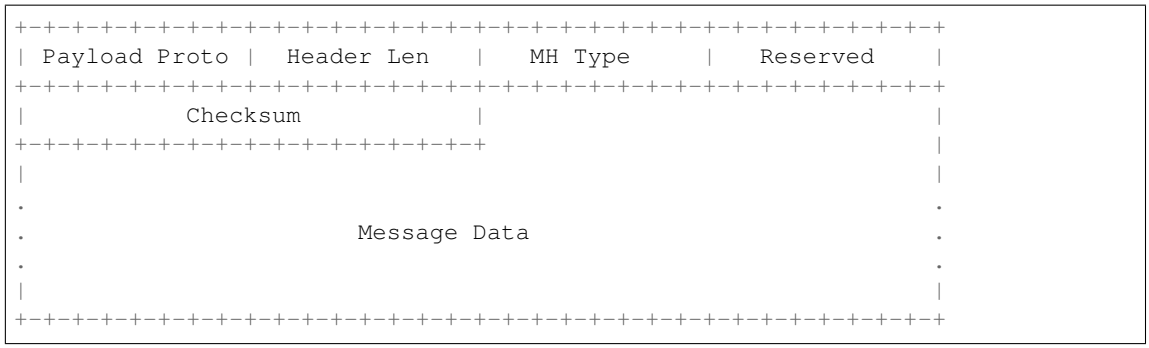
Returns Constructed packet data.

Return type bytes

read(*length=None*, *, *extension=False*, ***kwargs*)
Read Mobility Header.

Structure of MH header [**RFC 6275**]:

[0] https://en.wikipedia.org/wiki/Mobile_IP#Changes_in_IPv6_for_Mobile_IPv6



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **extension** (*bool*) – If the packet is used as an IPv6 extension header.
- ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_MH*

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Mobility Header']

property payload
Payload of current instance.

Raises ***UnsupportedCall*** – if the protocol is used as an IPv6 extension header

Return type *pcapkit.protocols.protocol.Protocol*

property protocol
Name of next layer protocol.

Return type *pcapkit.const.reg.transtype.TransType*

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.internet.mh.**DataType_MH**

Bases TypedDict

next: ***pcapkit.const.reg.transtype.TransType***
Next header.

length: **int**
Header length.

type: *pcapkit.const.mh.packet.Packet*
Mobility header type.

chksum: bytes
Checksum.

data: bytes
Message data.

Base Protocol

pcapkit.protocols.internet.internet contains *Internet*, which is a base class for internet layer protocols, eg. *AH*, *IPsec*, *IPv4*, *IPv6*, *IPX*, and etc.

class pcapkit.protocols.internet.internet.**Internet**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.protocol.Protocol*

Abstract base class for internet layer protocol family.

__layer__ = 'Internet'
Layer of protocol.

__proto__: DefaultDict[int, Tuple[str, str]]
Protocol index mapping for decoding next layer, c.f. *self._decode_next_layer* & *self._import_next_layer*. The values should be a tuple representing the module name and class name.

Code	Module	Class
0	*pcapkit.protocols.internet.hopopt*	*HOPOPT*
4	*pcapkit.protocols.internet.ipv4*	*IPv4*
6	*pcapkit.protocols.transport.tcp*	*TCP*
17	*pcapkit.protocols.transport.udp*	*UDP*
41	*pcapkit.protocols.internet.ipv6*	*IPv6*
43	*pcapkit.protocols.internet.ipv6_route*	*IPv6_Route*
44	*pcapkit.protocols.internet.ipv6_frag*	*IPv6_Frag*
51	*pcapkit.protocols.internet.ah*	*AH*
60	*pcapkit.protocols.internet.ipv6_opts*	*IPv6_Opts*
111	*pcapkit.protocols.internet.ipx*	*IPX*
135	*pcapkit.protocols.internet.mh*	*MH*
139	*pcapkit.protocols.internet.hip*	*HIP*

_decode_next_layer(*dict_*, *proto=None*, *length=None*, ***, *version=4*, *ipv6_exthdr=None*)
Decode next layer extractor.

Parameters

- **dict** (*dict*) – info buffer
- **proto** (*int*) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Keyword Arguments

- **version** (*Literal[4, 6]*) – IP version
- **ipv6_exthdr** (pcapkit.corekit.protochain.ProtoChain) – protocol chain of IPv6 extension headers

Returns current protocol with next layer extracted

Return type dict

_import_next_layer(*proto*, *length=None*, ***, *version=4*, *extension=False*)

Import next layer extractor.

This method currently supports following protocols as registered in *TransType*:

proto	Class
0	*HOPOPT*
4	*IPv4*
6	*TCP*
17	*UDP*
41	*IPv6*
43	*IPv6_Route*
44	*IPv6_Frag*
51	*AH*
60	*IPv6_Opts*
111	*IPX*
135	*MH*
139	*HIP*

Parameters

- **proto** (*int*) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Keyword Arguments

- **version** (*Literal[4, 6]*) – IP protocol version
- **extension** (*bool*) – if is extension header

Returns instance of next layer

Return type *pcapkit.protocols.protocol.Protocol*

_read_protos(*size*)

Read next layer protocol type.

Parameters **size** (*int*) – buffer size

Returns next layer's protocol enumeration

Return type *pcapkit.const.reg.transtype.TransType*

classmethod register(*code*, *module*, *class_*)

Register a new protocol class.

Parameters

- **code** (*int*) – protocol code as in *TransType*
- **module** (*str*) – module name
- **class** (*str*) – class name

Notes

The full qualified class name of the new protocol class should be as `{module}.{class_}`.

---

property layer
Protocol layer.

Return type Literal['Internet']

1.3.4 Transport Layer Protocols

pcapkit.protocols.transport is collection of all protocols in transport layer, with detailed implementation and methods.

UDP - User Datagram Protocol

pcapkit.protocols.transport.udp contains *UDP* only, which implements extractor for User Datagram Protocol (UDP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`udp.srcport`	Source Port
2	16	`udp.dstport`	Destination Port
4	32	`udp.len`	Length (header includes)
6	48	`udp.checksum`	Checksum

class pcapkit.protocols.transport.udp.**UDP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.transport.transport.Transport*

This class implements User Datagram Protocol.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

__length_hint__()
Return an estimated length for the object.

Return type Literal[8]

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read User Datagram Protocol (UDP).

Structure of UDP header [**RFC 768**]:

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
```

(continues on next page)

[0] https://en.wikipedia.org/wiki/User_Datagram_Protocol

(continued from previous page)

```
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_UDP*

property dst
Destination port.

Return type int

property length
Header length of current protocol.

Return type Literal[8]

property name
Name of current protocol.

Return type Literal['User Datagram Protocol']

property src
Source port.

Return type int

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.transport.udp.**DataType_UDP**

Bases TypedDict

Structure of UDP header [**RFC 768**].

srcport: **int**
Source port.

dstport: **int**
Destination port.

len: **int**
Length.

checksum: **bytes**
Checksum.

TCP - Transmission Control Protocol

pcapkit.protocols.transport.tcp contains *TCP* only, which implements extractor for Transmission Control Protocol (TCP)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.srcport`	Source Port
2	16	`tcp.dstport`	Destination Port
4	32	`tcp.seq`	Sequence Number
8	64	`tcp.ack`	Acknowledgement Number (if ACK set)
12	96	`tcp.hdr_len`	Data Offset
12	100		Reserved (must be `\x00`)
12	103	`tcp.flags.ns`	ECN Concealment Protection (NS)
13	104	`tcp.flags.cwr`	Congestion Window Reduced (CWR)
13	105	`tcp.flags.ece`	ECN-Echo (ECE)
13	106	`tcp.flags.urg`	Urgent (URG)
13	107	`tcp.flags.ack`	Acknowledgement (ACK)
13	108	`tcp.flags.psh`	Push Function (PSH)
13	109	`tcp.flags.rst`	Reset Connection (RST)
13	110	`tcp.flags.syn`	Synchronize Sequence Numbers (SYN)
13	111	`tcp.flags.fin`	Last Packet from Sender (FIN)
14	112	`tcp.window_size`	Size of Receive Window
16	128	`tcp.checksum`	Checksum
18	144	`tcp.urgent_pointer`	Urgent Pointer (if URG set)
20	160	`tcp.opt`	TCP Options (if data offset > 5)

class pcapkit.protocols.transport.tcp.**TCP**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.transport.transport.Transport*

This class implements Transmission Control Protocol.

_syn: **bool**
SYN flag.

_ack: **bool**
ACK flag.

classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol in IANA.

Return type *pcapkit.const.reg.transtype.TransType*

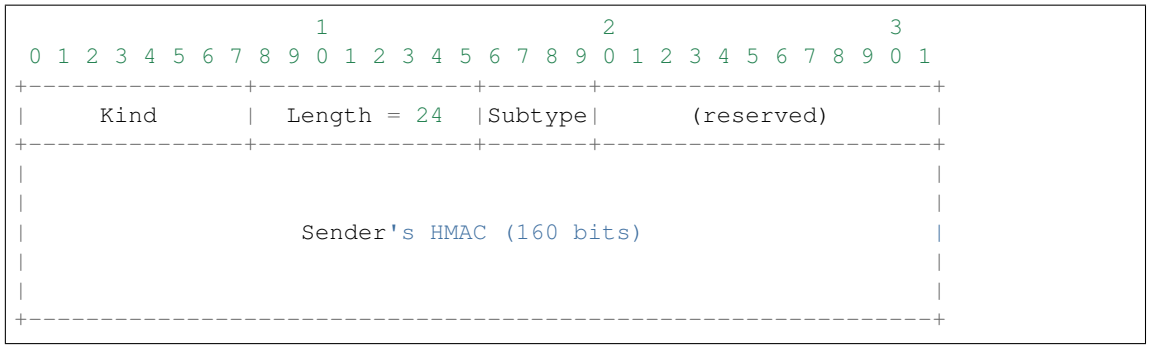
__length_hint__()
Return an estimated length for the object.

Return type Literal[20]

_read_join_ack(*bits*, *size*, *kind*)
Read Join Connection option for Third ACK.

Structure of `MP_JOIN-ACK` [**RFC 6824**]:

[0] https://en.wikipedia.org/wiki/Transmission_Control_Protocol



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

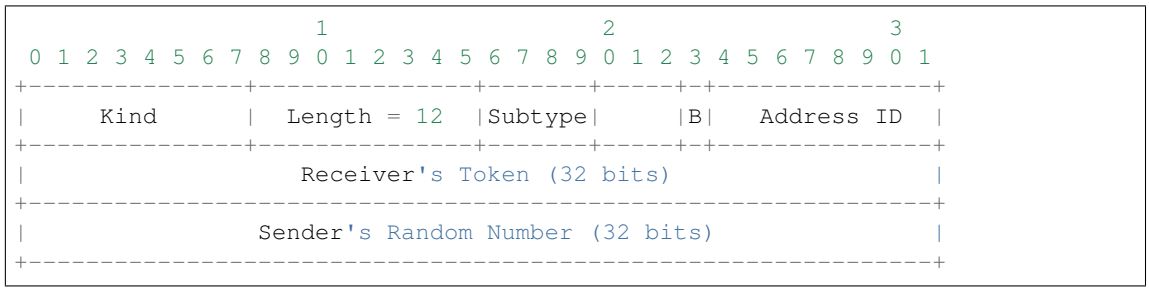
Returns extracted Join Connection (MP_JOIN-ACK) option for Third ACK

Return type *DataType_TCP_Opt_MP_JOIN_ACK*

_read_join_syn(*bits*, *size*, *kind*)

Read Join Connection option for Initial SYN.

Structure of MP_JOIN-SYN [**RFC 6824**]:



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

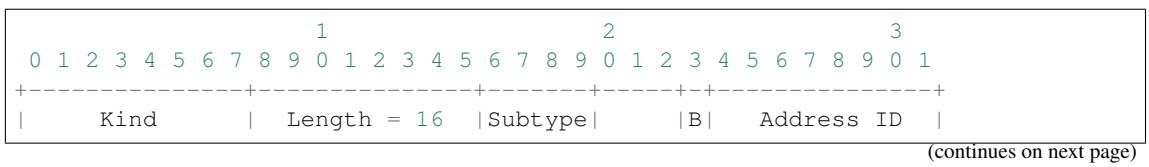
Returns extracted Join Connection (MP_JOIN-SYN) option for Initial SYN

Return type *DataType_TCP_Opt_MP_JOIN_SYN*

_read_join_synack(*bits*, *size*, *kind*)

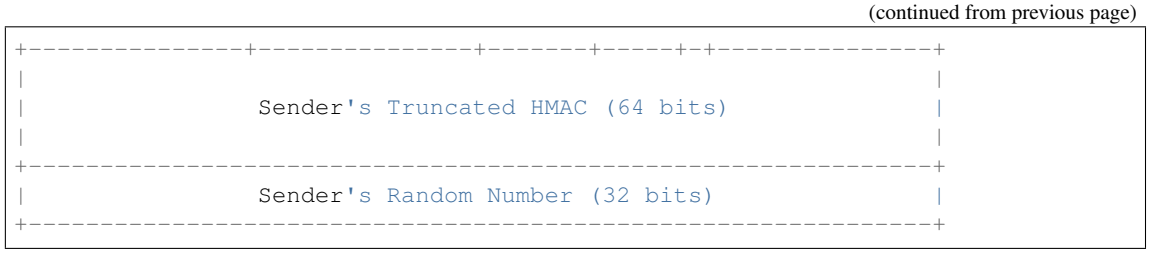
Read Join Connection option for Responding SYN/ACK.

Structure of MP_JOIN-SYN/ACK [**RFC 6824**]:



(continues on next page)

(continued from previous page)



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Join Connection (MP_JOIN-SYN/ACK) option for Responding SYN/ACK

Return type *DataType_TCP_Opt_MP_JOIN_SYNACK*

_read_mode_acopt(*size*, *kind*)

Read Alternate Checksum Request option.

Structure of TCP CHKSUM-REQ [**RFC 1146**][**RFC 6247**]:

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[14]*) – option kind value (Alt-Chksum Request)

Returns extracted Alternate Checksum Request (CHKSUM-REQ) option

Return type *DataType_TCP_Opt_ACOPT*

_read_mode_donone(*size*, *kind*)

Read options request no process.

Parameters

- **size** (*int*) – length of option
- **kind** (*int*) – option kind value

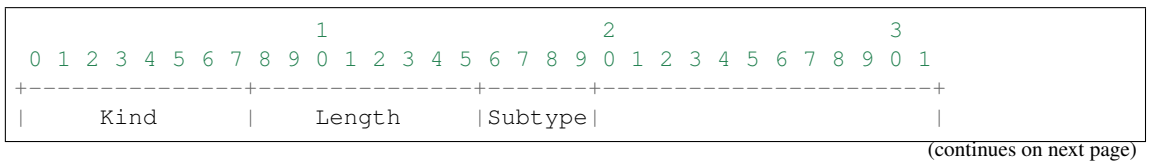
Returns Extracted option with no operation.

Return type *DataType_TCP_Opt_DONONE*

_read_mode_mptcp(*size*, *kind*)

Read Multipath TCP option.

Structure of MP-TCP [**RFC 6824**]:



(continues on next page)

(continued from previous page)

```
+--------------+---------------+-------+                        |
|                      Subtype-specific data                     |
|                        (variable length)                       |
+----------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Multipath TCP (MP-TCP) option

Return type *DataType_TCP_Opt_MPTCP*

_read_mode_pocsp(*size*, *kind*)

Read Partial Order Connection Service Profile option.

Structure of TCP POC-SP Option [**RFC 1693**][**RFC 6247**]:

```
                          1 bit        1 bit     6 bits
+----------+----------+------------+----------+--------+
|  Kind=10 | Length=3 | Start_flag | End_flag | Filler |
+----------+----------+------------+----------+--------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[10]*) – option kind value (POC-Serv Profile)

Returns extracted Partial Order Connection Service Profile (POC-SP) option

Return type *DataType_TCP_Opt_POCSP*

_read_mode_qsopt(*size*, *kind*)

Read Quick-Start Response option.

Structure of TCP QSopt [**RFC 4782**]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Kind      |  Length=8     | Resv. | Rate  |   TTL Diff    |
|               |               |       |Request|               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    QS Nonce                                 | R |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[27]*) – option kind value (Quick-Start Response)

Returns extracted Quick-Start Response (QS) option

Return type *DataType_TCP_Opt_QSOPT*

_read_mode_tcpao(*size*, *kind*)
Read Authentication option.

Structure of TCP `AOopt` [**RFC 5925**]:

```
+------------+------------+------------+------------+
|  Kind=29   |   Length   |   KeyID    | RNextKeyID |
+------------+------------+------------+------------+
|                      MAC            ...
+-----------------------------------...

...-----------------+
...  MAC (con't)    |
...-----------------+
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[29]*) – option kind value (TCP Authentication Option)

Returns extracted Authentication (`AO`) option

Return type *DataType_TCP_Opt_TCPAO*

_read_mode_tsopt(*size*, *kind*)
Read Timestamps option.

Structure of TCP `TSopt` [**RFC 7323**]:

```
+-------+-------+---------------------+---------------------+
|Kind=8 |  10   |   TS Value (TSval)  |TS Echo Reply (TSecr)|
+-------+-------+---------------------+---------------------+
    1       1              4                     4
```

Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[8]*) – option kind value (Timestamps)

Returns extracted Timestamps (`TS`) option

Return type *DataType_TCP_Opt_TS*

_read_mode_unpack(*size*, *kind*)
Read options request unpack process.

Parameters

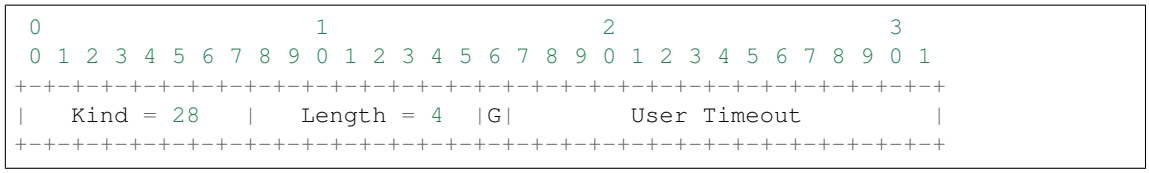
- **size** (*int*) – length of option
- **kind** (*int*) – option kind value

Returns Extracted option which unpacked.

Return type *DataType_TCP_Opt_UNPACK*

_read_mode_utopt(*size*, *kind*)
Read User Timeout option.

Structure of TCP `TIMEOUT` [**RFC 5482**]:



Parameters

- **size** (*int*) – length of option
- **kind** (*Literal[28]*) – option kind value (User Timeout Option)

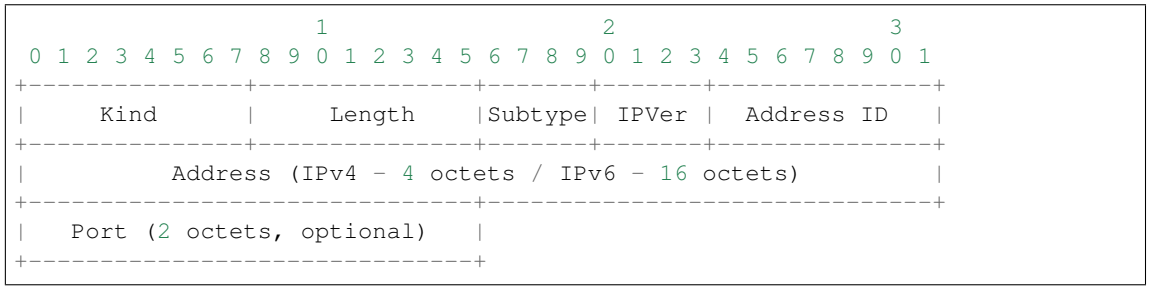
Returns extracted User Timeout (TIMEOUT) option

Return type *DataType_TCP_Opt_UTOPT*

_read_mptcp_add(*bits*, *size*, *kind*)

Read Add Address option.

Structure of ADD_ADDR [**RFC 6824**]:



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Add Address (ADD_ADDR) option

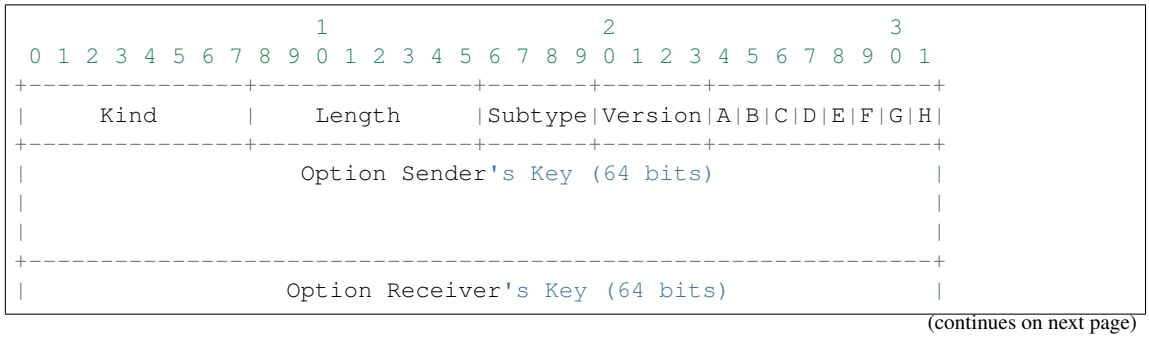
Return type *DataType_TCP_Opt_ADD_ADDR*

Raises ***ProtocolError*** – If the option is malformed.

_read_mptcp_capable(*bits*, *size*, *kind*)

Read Multipath Capable option.

Structure of MP_CAPABLE [**RFC 6824**]:



(continues on next page)

(continued from previous page)

```
|                      (if option Length == 20)                 |
|                                                               |
+---------------------------------------------------------------+
```

Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Multipath Capable (MP_CAPABLE) option

Return type *DataType_TCP_Opt_MP_CAPABLE*

_read_mptcp_dss(*bits*, *size*, *kind*)

Read Data Sequence Signal (Data ACK and Data Sequence Mapping) option.

Structure of DSS [**RFC 6824**]:

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------+----------------------+
|     Kind      |    Length     |Subtype| (reserved) |F|m|M|a|A|
+---------------+---------------+-------+----------------------+
|                                                               |
|           Data ACK (4 or 8 octets, depending on flags)        |
|                                                               |
+---------------------------------------------------------------+
|                                                               |
|   Data sequence number (4 or 8 octets, depending on flags)    |
|                                                               |
+---------------------------------------------------------------+
|               Subflow Sequence Number (4 octets)              |
+-------------------------------+-------------------------------+
|  Data-Level Length (2 octets) |      Checksum (2 octets)      |
+-------------------------------+-------------------------------+
```

Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Data Sequence Signal (DSS) option

Return type *DataType_TCP_Opt_DSS*

_read_mptcp_fail(*bits*, *size*, *kind*)

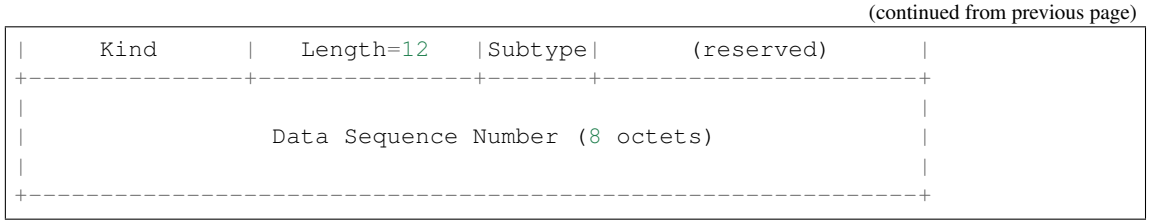
Read Fallback option.

Structure of MP_FAIL [**RFC 6824**]:

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------+----------------------+
```

(continues on next page)

(continued from previous page)



Parameters

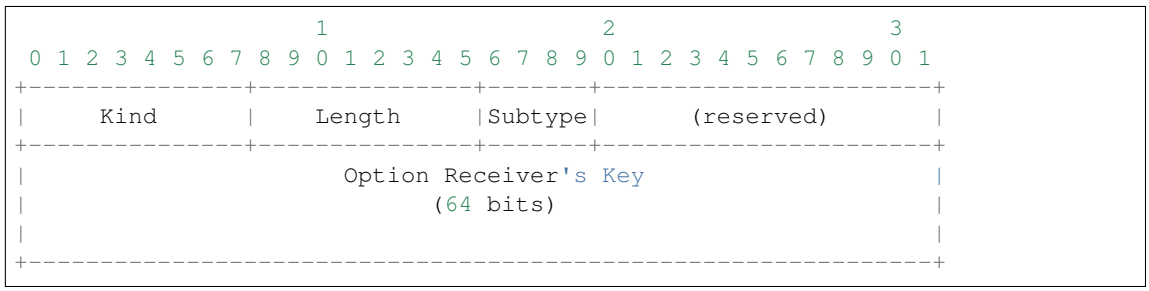
- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Fallback (MP_FAIL) option

Return type *DataType_TCP_Opt_MP_FAIL*

_read_mptcp_fastclose(*bits*, *size*, *kind*)
Read Fast Close option.

Structure of MP_FASTCLOSE [RFC 6824]:



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Fast Close (MP_FASTCLOSE) option

Return type *DataType_TCP_Opt_MP_FAIL*

_read_mptcp_join(*bits*, *size*, *kind*)
Read Join Connection option.

Parameters

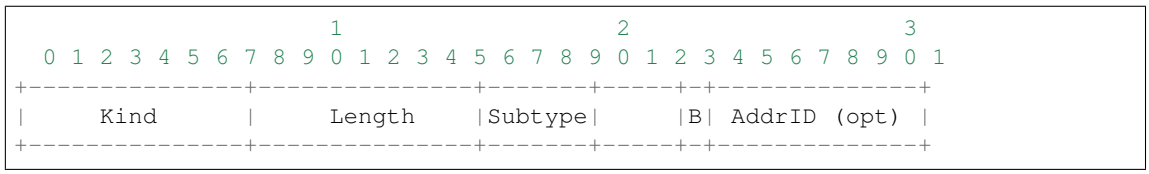
- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Join Connection (MP_JOIN) option

Return type *DataType_TCP_Opt_MP_JOIN*

_read_mptcp_prio(*bits*, *size*, *kind*)
Read Change Subflow Priority option.

Structure of `MP_PRIO` [RFC 6824]:



Parameters

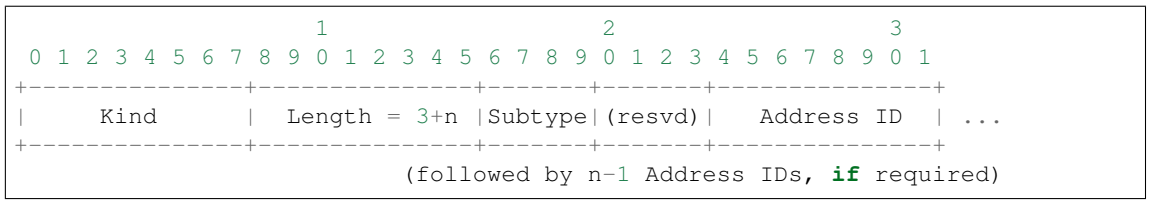
- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Change Subflow Priority (`MP_PRIO`) option

Return type *DataType_TCP_Opt_REMOVE_ADDR*

_read_mptcp_remove(*bits*, *size*, *kind*)
Read Remove Address option.

Structure of `REMOVE_ADDR` [**RFC 6824**]:



Parameters

- **bits** (*str*) – 4-bit data (after subtype)
- **size** (*int*) – length of option
- **kind** (*Literal[30]*) – option kind value (Multipath TCP)

Returns extracted Remove Address (`REMOVE_ADDR`) option

Return type *DataType_TCP_Opt_REMOVE_ADDR*

_read_tcp_options(*size*)
Read TCP option list.

Parameters **size** (*int*) – length of option list

Returns Tuple of TCP option list and extracted TCP options.

Return type Tuple[Tuple[*pcapkit.const.tcp.option.Option*], *DataType_TCP_Opt*]

make(***kwargs*)
Make (construct) packet data.

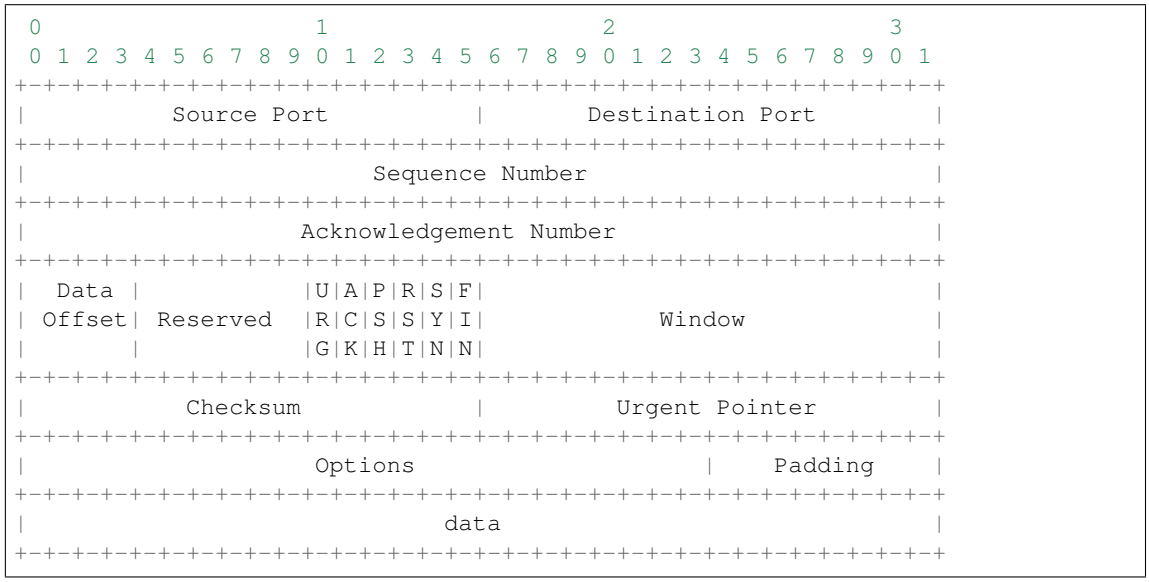
Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Transmission Control Protocol (TCP).

Structure of TCP header [**RFC 793**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_TCP*

property dst
Destination port.

Return type int

property length
Header length of current protocol.

Return type int

property name
Name of current protocol.

Return type Literal['Transmission Control Protocol']

property src
Source port.

Return type int

pcapkit.protocols.transport.tcp.**TCP_OPT**: ***DataType_TCP_OPT***
TCP option dict parsing mapping.

kind	length	type	process	comment	name
0					[RFC 793] End of Option List
1					[RFC 793] No-Operation
2	4	H	1		[RFC 793] Maximum Segment Size
3	3	B	1		[RFC 7323] Window Scale
4	2	?		True	[RFC 2018] SACK Permitted
5	?	P	0	2+8*N	[RFC 2018] SACK
6	6	P	0		[RFC 1072][RFC 6247] Echo
7	6	P	0		[RFC 1072][RFC 6247] Echo Reply
8	10	II	2		[RFC 7323] Timestamps
9	2	?		True	[RFC 1693][RFC 6247] POC Permitted
10	3	??P	3		[RFC 1693][RFC 6247] POC-Serv Profile
11	6	P	0		[RFC 1693][RFC 6247] Connection Count
12	6	P	0		[RFC 1693][RFC 6247] CC.NEW
13	6	P	0		[RFC 1693][RFC 6247] CC.ECHO
14	3	B	4		[RFC 1146][RFC 6247] Alt-Chksum Request
15	?	P	0		[RFC 1146][RFC 6247] Alt-Chksum Data
19	18	P	0		[RFC 2385] MD5 Signature Option
27	8	P	5		[RFC 4782] Quick-Start Response
28	4	P	6		[RFC 5482] User Timeout Option
29	?	P	7		[RFC 5925] TCP Authentication Option
30	?	P	8		[RFC 6824] Multipath TCP
34	?	P	0		[RFC 7413] Fast Open

See also:

pcapkit.protocols.transport.tcp.DataType_TCP_OPT

pcapkit.protocols.transport.tcp.**process_opt**: **Dict[int, Callable[[*pcapkit.protocols.transpo***

Process method for TCP options.

Code	Method	Description
0	*_read_mode_donone()*	do nothing
1	*_read_mode_unpack()*	unpack according to size
2	*_read_mode_tsopt()*	timestamps
3	*_read_mode_pocsp()*	POC service profile
4	*_read_mode_acopt()*	alternate checksum request
5	*_read_mode_qsopt()*	Quick-Start response
6	*_read_mode_utopt()*	user timeout option
7	*_read_mode_tcpao()*	TCP authentication option
8	*_read_mode_mptcp()*	multipath TCP

pcapkit.protocols.transport.tcp.**mptcp_opt**: **Dict[int, Callable[[*pcapkit.protocols.transport***

Process method for multipath TCP options [RFC 6824].

Code	Method	Description
0	*_read_mptcp_capable()*	MP_CAPABLE
1	*_read_mptcp_join()*	MP_JOIN
2	*_read_mptcp_dss()*	DSS
3	*_read_mptcp_add()*	ADD_ADDR
4	*_read_mptcp_remove()*	REMOVE_ADDR
5	*_read_mptcp_prio()*	MP_PRIO
6	*_read_mptcp_fail()*	MP_FAIL
7	*_read_mptcp_fastclose()*	MP_FASTCLOSE

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.transport.tcp.**DataType_TCP**

Bases TypedDict

Structure of TCP header [**RFC 793**].

srcport: **int**
Source port.

dstport: **int**
Description port.

seq: **int**
Sequence number.

ack: **int**
Acknowledgement number.

hdr_len: **int**
Data offset.

flags: ***DataType_TCP_Flags***
Flags.

window_size: **int**
Size of receive window.

checksum: **bytes**
Checksum.

urgent_pointer: **int**
Urgent pointer.

opt: **Tuple[*pcapkit.const.tcp.option.Option*]**
Array of TCP options.

packet: **bytes**
Raw packet data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Flags**

Bases TypedDict

Flags.

ns: bool
ECN concealment protection.

cwr: bool
Congestion window reduced.

ece: bool
ECN-Echo.

urg: bool
Urgent.

ack: bool
Acknowledgement.

psh: bool
Push function.

rst: bool
Reset connection.

syn: bool
Synchronize sequence numbers.

fin: bool
Last packet from sender.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt**

Bases TypedDict

Structure of TCP options.

kind: int
Option kind value.

length: int
Length of option.

class pcapkit.protocols.transport.tcp.**DataType_TCP_OPT**

Bases TypedDict

TCP option `dict` parsing mapping.

flag: bool
If the length of option is **GREATER THAN** `1`.

desc: str
Description string, also attribute name.

func: Optional[Callable[[int], int]]
Function, length of data bytes.

proc: Optional[int]
Process method that data bytes need (when *`flag`* is `True`).

See also:

pcapkit.protocols.transport.tcp.process_opt

TCP Miscellaneous Options

No Process Options

For TCP options require no process, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.opt.kind`	Kind
1	8	`tcp.opt.length`	Length
2	16	`tcp.opt.data`	Kind-specific Data

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_DONONE**

Bases DataType_TCP_Opt

Structure of TCP options.

data: **bytes**
Kind-specific data.

Unpack Process Options

For TCP options require unpack process, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.opt.kind`	Kind
1	8	`tcp.opt.length`	Length
2	16	`tcp.opt.data`	Kind-specific Data

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_UNPACK**

Bases DataType_TCP_Opt

Structure of TCP options.

data: **bytes**
Kind-specific data.

Timestamps Option

For TCP Timestamps (`TS`) option as described in **RFC 7323**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.ts.kind`	Kind (`8`)
1	8	`tcp.ts.length`	Length (`10`)
2	16	`tcp.ts.val`	Timestamp Value
6	48	`tcp.ts.ecr`	Timestamps Echo Reply

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_TS**

Bases DataType_TCP_Opt

Structure of TCP `TSopt` [**RFC 7323**].

val: *int*
Timestamp value.

ecr: *int*
Timestamps echo reply.

Partial Order Connection Service Profile Option

For TCP Partial Order Connection Service Profile (`POC-SP`) option as described in **RFC 1693** and **RFC 6247**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.pocsp.kind`	Kind (`10`)
1	8	`tcp.pocsp.length`	Length (`3`)
2	16	`tcp.pocsp.start`	Start Flag
2	17	`tcp.pocsp.end`	End Flag
2	18	`tcp.pocsp.filler`	Filler

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_POCSP**

Bases DataType_TCP_Opt

Structure of TCP `POC-SP` Option [**RFC 1693**][**RFC 6247**].

start: *bool*
Start flag.

end: *bool*
End flag.

filler: *bytes*
Filler.

Alternate Checksum Request Option

For TCP Alternate Checksum Request (`CHKSUM-REQ`) option as described in **RFC 1146** and **RFC 6247**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.chksumreq.kind`	Kind (`14`)
1	8	`tcp.chksumreq.length`	Length (`3`)
2	16	`tcp.chksumreq.ac`	Checksum Algorithm

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_ACOPT**

Bases DataType_TCP_Opt

Structure of TCP `CHKSUM-REQ` [**RFC 1146**][**RFC 6247**].

ac: *pcapkit.const.tcp.checksum.Checksum*
Checksum algorithm.

Quick-Start Response Option

For TCP Quick-Start Response (`QS`) option as described in **RFC 4782**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.qs.kind`	Kind (`27`)
1	8	`tcp.qs.length`	Length (`8`)
2	16		Reserved (must be `\x00`)
2	20	`tcp.qs.req_rate`	Request Rate
3	24	`tcp.qs.ttl_diff`	TTL Difference
4	32	`tcp.qs.nounce`	QS Nounce
7	62		Reserved (must be `\x00`)

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_QSOPT**

Bases DataType_TCP_Opt

Structure of TCP `QSopt` [**RFC 4782**].

req_rate: **int**
Request rate.

ttl_diff: **int**
TTL difference.

nounce: **int**
QS nounce.

User Timeout Option

For TCP User Timeout (`TIMEOUT`) option as described in **RFC 5482**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.timeout.kind`	Kind (`28`)
1	8	`tcp.timeout.length`	Length (`4`)
2	16	`tcp.timeout.granularity`	Granularity
2	17	`tcp.timeout.timeout`	User Timeout

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_UTOPT**

Bases DataType_TCP_Opt

Structure of TCP `TIMEOUT` [**RFC 5482**].

granularity: **Literal[minutes, seconds]**
Granularity.

timeout: **datetime.timedelta**
User timeout.

Authentication Option

For Authentication (AO) option as described in RFC 5925, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.ao.kind	Kind (29)
1	8	tcp.ao.length	Length
2	16	tcp.ao.key_id	KeyID
3	24	tcp.ao.r_next_key_id	RNextKeyID
4	32	tcp.ao.mac	Message Authentication Code

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_TCPAO**

Bases DataType_TCP_Opt

Structure of TCP AOopt [RFC 5925].

key_id: **int**
KeyID.

r_next_key_id: **int**
RNextKeyID.

mac: **bytes**
Message authentication code.

Multipath TCP Options

For Multipath TCP (MP-TCP) options as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.mp.kind	Kind (30)
1	8	tcp.mp.length	Length
2	16	tcp.mp.subtype	Subtype
2	20	tcp.mp.data	Subtype-specific Data

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MPTCP**

Bases DataType_TCP_Opt

Structure of MP-TCP [RFC 6824].

subtype: **pcapkit.const.tcp.mp_tcp_option.MPTCPOption**
Subtype.

data: **Optional[bytes]**
Subtype-specific data.

Multipath Capable Option

For Multipath Capable (`MP_CAPABLE`) options as described in **RFC 6824**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length (`12`/`20`)
2	16	`tcp.mp.subtype`	Subtype (`0`)
2	20	`tcp.mp.capable.version`	Version
3	24	`tcp.mp.capable.flags.req`	Checksum Require Flag (`A`)
3	25	`tcp.mp.capable.flags.ext`	Extensibility Flag (`B`)
3	26	`tcp.mp.capable.flags.res`	Unassigned (`C` - `G`)
3	31	`tcp.mp.capable.flags.hsa`	HMAC-SHA1 Flag (`H`)
4	32	`tcp.mp.capable.skey`	Option Sender's Key
12	96	`tcp.mp.capable.rkey`	Option Receiver's Key (only if option length is `20`)

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_CAPABLE**

Bases DataType_TCP_Opt_MPTCP

Structure of `MP_CAPABLE` [**RFC 6824**].

capable: *DataType_TCP_Opt_MP_CAPABLE_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_CAPABLE_Data**

Bases TypedDict

Structure of `MP_CAPABLE` [**RFC 6824**].

version: **int**
Version.

flags: *DataType_TCP_Opt_MP_CAPABLE_Flags*
Flags.

skey: **int**
Option sender's key.

rkey: **Optional[int]**
Option receiver's key.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_CAPABLE_Flags**

Bases TypedDict

Flags.

req: **bool**
Checksum require flag.

ext: **bool**
Extensibility flag.

res: **Tuple[bool, bool, bool, bool, bool]**
Unassigned flags.

hsa: **bool**
HMAC-SHA1 flag.

Join Connection Option

For Join Connection (`MP_JOIN`) options as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length
2	16	`tcp.mp.subtype`	Subtype (`1`)
2	20	`tcp.mp.data`	Handshake-specific Data

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN**

Bases DataType_TCP_Opt_MPTCP

Structure of `MP_JOIN` [RFC 6824].

connection: Optional[Literal[SYN/ACK, SYN, ACK]]
Join connection type.

join: *DataType_TCP_Opt_MP_JOIN_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_Data**

Bases TypedDict

Structure of `MP_JOIN` [RFC 6824].

data: Optional[bytes]
Unknown type data.

MP_JOIN-SYN

For Join Connection (`MP_JOIN-SYN`) option for Initial SYN as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length (`12`)
2	16	`tcp.mp.subtype`	Subtype (`1` | `SYN`)
2	20		Reserved (must be `\x00`)
2	23	`tcp.mp.join.syn.backup`	Backup Path (`B`)
3	24	`tcp.mp.join.syn.addr_id`	Address ID
4	32	`tcp.mp.join.syn.token`	Receiver's Token
8	64	`tcp.mp.join.syn.rand_num`	Sender's Random Number

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_SYN**

Bases DataType_TCP_Opt_MP_JOIN_Data

Structure of `MP_JOIN-SYN` [RFC 6824].

syn: *DataType_TCP_Opt_MP_JOIN_SYN_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_SYN_Data**

Bases TypedDict

Structure of `MP_JOIN-SYN` [RFC 6824].

backup: **bool**
Backup path.

addr_id: **int**
Address ID.

token: **int**
Receiver's token.

rand_num: **int**
Sender's random number.

MP_JOIN-SYN/ACK

For Join Connection (`MP_JOIN-SYN/ACK`) option for Responding SYN/ACK as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length (`16`)
2	16	`tcp.mp.subtype`	Subtype (`1` | `SYN/ACK`)
2	20		Reserved (must be `\x00`)
2	23	`tcp.mp.join.synack.backup`	Backup Path (`B`)
3	24	`tcp.mp.join.synack.addr_id`	Address ID
4	32	`tcp.mp.join.synack.hmac`	Sender's Truncated HMAC
12	96	`tcp.mp.join.synack.rand_num`	Sender's Random Number

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_SYNACK**

Bases DataType_TCP_Opt_MP_JOIN_Data

Structure of `MP_JOIN-SYN/ACK` [RFC 6824].

syn: ***DataType_TCP_Opt_MP_JOIN_SYNACK_Data***
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_SYNACK_Data**

Bases TypedDict

Structure of `MP_JOIN-SYN/ACK` [RFC 6824].

backup: **bool**
Backup path.

addr_id: **int**
Address ID.

hmac: **bytes**
Sender's truncated HMAC.

rand_num: **int**
Sender's random number.

MP_JOIN-ACK

For Join Connection (MP_JOIN-ACK) option for Third ACK as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.mp.kind	Kind (30)
1	8	tcp.mp.length	Length (16)
2	16	tcp.mp.subtype	Subtype (1 | ACK)
2	20		Reserved (must be \x00)
4	32	tcp.mp.join.ack.hmac	Sender's HMAC

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_ACK**

Bases DataType_TCP_Opt_MP_JOIN_Data

Structure of MP_JOIN-ACK [RFC 6824].

syn: *DataType_TCP_Opt_MP_JOIN_ACK_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_JOIN_ACK_Data**

Bases TypedDict

Structure of MP_JOIN-ACK [RFC 6824].

hmac: bytes
Sender's HMAC.

Data Sequence Signal Option

For Data Sequence Signal (DSS) options as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.mp.kind	Kind (30)
1	8	tcp.mp.length	Length
2	16	tcp.mp.subtype	Subtype (2)
2	20		Reserved (must be \x00)
3	27	tcp.mp.dss.flags.fin	DATA_FIN (F)
3	28	tcp.mp.dss.flags.dsn_len	DSN Length (m)
3	29	tcp.mp.dss.flags.data_pre	DSN, SSN, Data-Level Length, CHKSUM Present (M)
3	30	tcp.mp.dss.flags.ack_len	ACK Length (a)
3	31	tcp.mp.dss.flags.ack_pre	Data ACK Present (A)
4	32	tcp.mp.dss.ack	Data ACK (4 / 8 octets)
8/12	64/96	tcp.mp.dss.dsn	DSN (4 / 8 octets)
12/20	48/160	tcp.mp.dss.ssn	Subflow Sequence Number
16/24	128/192	tcp.mp.dss.dl_len	Data-Level Length
18/26	144/208	tcp.mp.dss.checksum	Checksum

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_DSS**

Bases DataType_TCP_Opt_MPTCP

Structure of DSS [RFC 6824].

dss: *DataType_TCP_Opt_DSS_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_DSS_Data**

Bases TypedDict

Structure of DSS [RFC 6824].

flags: *DataType_TCP_Opt_DSS_Flags*
Flags.

ack: **Optional[int]**
Data ACK.

dsn: **Optional[int]**
DSN.

ssn: **Optional[int]**
Subflow sequence number.

dl_len: **int**
Data-level length.

checksum: **bytes**
Checksum.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_DSS_Flags**

Bases TypedDict

Flags.

fin: **bool**
DATA_FIN.

dsn_len: **int**
DSN length.

data_pre: **int**
DSN, SSN, data-level length, checksum present.

ack_len: **int**
ACK length.

ack_pre: **bool**
ACK present.

Add Address Option

For Add Address (ADD_ADDR) options as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.mp.kind	Kind (30)
1	8	tcp.mp.length	Length
2	16	tcp.mp.subtype	Subtype (3)
2	20	tcp.mp.add_addr.ip_ver	IP Version
3	24	tcp.mp.add_addr.addr_id	Address ID
4	32	tcp.mp.add_addr.addr	IP Address (4 / 16)
8/20	64/160	tcp.mp.add_addr.port	Port (optional)

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_ADD_ADDR**

Bases DataType_TCP_Opt_MPTCP

Structure of ADD_ADDR [RFC 6824].

add_addr: *DataType_TCP_Opt_ADD_ADDR_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_ADD_ADDR_Data**

Bases TypedDict

Structure of ADD_ADDR [RFC 6824].

ip_ver: **Literal[4, 6]**
IP version.

addr_id: **int**
Address ID.

addr: **Union[ipaddress.IPv4Address, ipaddress.IPv6Address]**
IP address.

port: **Optional[int]**
Port.

Remove Address Option

For Remove Address (REMOVE_ADDR) options as described in RFC 6824, its structure is described as below:

Octets	Bits	Name	Description
0	0	tcp.mp.kind	Kind (30)
1	8	tcp.mp.length	Length
2	16	tcp.mp.subtype	Subtype (4)
2	20		Reserved (must be \x00)
3	24	tcp.mp.remove_addr.addr_id	Address ID (optional list)

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_REMOVE_ADDR**

Bases DataType_TCP_Opt_MPTCP

Structure of REMOVE_ADDR [RFC 6824].

remove_addr: *DataType_TCP_Opt_REMOVE_ADDR_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_REMOVE_ADDR_Data**

Bases TypedDict

Structure of REMOVE_ADDR [RFC 6824].

addr_id: **Tuple[int]**
Array of address IDs.

Change Subflow Priority Option

For Change Subflow Priority (`MP_PRIO`) options as described in **RFC 6824**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length
2	16	`tcp.mp.subtype`	Subtype (`4`)
2	23	`tcp.mp.prio.backup`	Backup Path (`B`)
3	24	`tcp.mp.prio.addr_id`	Address ID (optional)

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_PRIO**

Bases DataType_TCP_Opt_MPTCP

Structure of `MP_PRIO` [**RFC 6824**].

prio: *DataType_TCP_Opt_MP_PRIO_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_PRIO_Data**

Bases TypedDict

Structure of `MP_PRIO` [**RFC 6824**].

backup: **bool**
Backup path.

addr_id: **Optional[int]**
Address ID.

Fallback Option

For Fallback (`MP_FAIL`) options as described in **RFC 6824**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length
2	16	`tcp.mp.subtype`	Subtype (`4`)
2	23		Reserved (must be `\x00`)
4	32	`tcp.mp.fail.dsn`	Data Sequence Number

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_FAIL**

Bases DataType_TCP_Opt_MPTCP

Structure of `MP_FAIL` [**RFC 6824**].

fail: *DataType_TCP_Opt_MP_FAIL_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_FAIL_Data**

Bases TypedDict

Structure of `MP_FAIL` [**RFC 6824**].

dsn: *int*
Data sequence number.

Fast Close Option

For Fast Close (`MP_FASTCLOSE`) options as described in **RFC 6824**, its structure is described as below:

Octets	Bits	Name	Description
0	0	`tcp.mp.kind`	Kind (`30`)
1	8	`tcp.mp.length`	Length
2	16	`tcp.mp.subtype`	Subtype (`4`)
2	23		Reserved (must be `\x00`)
4	32	`tcp.mp.fastclose.rkey`	Option Receiver's Key

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_FASTCLOSE**

Bases DataType_TCP_Opt_MPTCP

Structure of `MP_FASTCLOSE` [**RFC 6824**].

fastclose: *DataType_TCP_Opt_MP_FASTCLOSE_Data*
Subtype-specific data.

class pcapkit.protocols.transport.tcp.**DataType_TCP_Opt_MP_FASTCLOSE_Data**

Bases TypedDict

Structure of `MP_FASTCLOSE` [**RFC 6824**].

rkey: *int*
Option receiver's key.

Base Protocol

pcapkit.protocols.transport.transport contains *Transport*, which is a base class for transport layer protocols, eg. `TCP` and `UDP`.

class pcapkit.protocols.transport.transport.**Transport**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.protocol.Protocol*

Abstract base class for transport layer protocol family.

__layer__ = 'Transport'
Layer of protocol.

_import_next_layer(*proto*, *length=None*)
Import next layer extractor.

Parameters

- **proto** (*str*) – next layer protocol name
- **length** (*int*) – valid (*non-padding*) length

Returns instance of next layer

Return type *pcapkit.protocols.protocol.Protocol*

property layer
Protocol layer.

Return type Literal['Transport']

1.3.5 Application Layer Protocols

pcapkit.protocols.application is collection of all protocols in application layer, with detailed implementation and methods.

FTP - File Transfer Protocol

pcapkit.protocols.application.ftp contains *FTP* only, which implements extractor for File Transfer Protocol (FTP)*[0].

class pcapkit.protocols.application.ftp.**FTP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.application.application.Application*

This class implements File Transfer Protocol.

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read File Transfer Protocol (FTP).

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type Union[*DataType_FTP_Request*, *DataType_FTP_Response*]

Raises ***ProtocolError*** – If the packet is malformed.

property length
Header length of current protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *length*.

property name
Name of current protocol.

Return type Literal['File Transfer Protocol']

[0] https://en.wikipedia.org/wiki/File_Transfer_Protocol

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.application.ftp.**DataType_FTP_Request**

Bases TypedDict

Structure of FTP request packet [RFC 959].

type: **Literal[request]**
Packet type.

command: ***pcapkit.corekit.infoclass.Info***
FTP command.

arg: **Optional[str]**
FTP command arguments.

raw: **bytes**
Raw packet data.

class pcapkit.protocols.application.ftp.**DataType_FTP_Response**

Bases TypedDict

Structure of FTP response packet [RFC 959].

type: **Literal[response]**
Packet type.

code: ***pcapkit.const.ftp.return_code.ReturnCode***
FTP response code.

arg: **Optional[str]**
FTP response arguments (messages).

mf: **bool**
More fragmented messages flag.

raw: **bytes**
Raw packet data.

HTTP - Hypertext Transfer Protocol

pcapkit.protocols.application.http contains *HTTP* only, which is a base class for Hypertext Transfer Protocol (HTTP)*[0] family, eg. `HTTP/1.*` and `HTTP/2`.

class pcapkit.protocols.application.http.**HTTP**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.application.application.Application*

This class implements all protocols in HTTP family.

- Hypertext Transfer Protocol (HTTP/1.1) [RFC 7230]
- Hypertext Transfer Protocol version 2 (HTTP/2) [RFC 7540]

classmethod id()
Index ID of the protocol.

[0] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Returns Index ID of the protocol.

Return type Tuple[Literal['HTTPv1'], Literal['HTTPv2']]

property length
Header length of current protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *length*.

property name
Name of current protocol.

Return type Literal['Hypertext Transfer Protocol']

HTTP/1.* - Hypertext Transfer Protocol

pcapkit.protocols.application.httpv1 contains *HTTPv1* only, which implements extractor for Hypertext Transfer Protocol (HTTP/1.*)*[0], whose structure is described as below:

```
METHOD URL HTTP/VERSION\r\n :==: REQUEST LINE
<key> : <value>\r\n         :==: REQUEST HEADER
............  (Ellipsis)    :==: REQUEST HEADER
\r\n                        :==: REQUEST SEPARATOR
<body>                      :==: REQUEST BODY (optional)

HTTP/VERSION CODE DESP \r\n :==: RESPONSE LINE
<key> : <value>\r\n         :==: RESPONSE HEADER
............  (Ellipsis)    :==: RESPONSE HEADER
\r\n                        :==: RESPONSE SEPARATOR
<body>                      :==: RESPONSE BODY (optional)
```

class pcapkit.protocols.application.httpv1.**HTTPv1**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.application.http.HTTP*

This class implements Hypertext Transfer Protocol (HTTP/1.*).

_read_http_body(*body*)
Read HTTP/1.* body.

Parameters **body** (*bytes*) – HTTP body data.

Returns Raw HTTP body.

Return type str

_read_http_header(*header*)
Read HTTP/1.* header.

Structure of HTTP/1.* header [**RFC 7230**]:

```
start-line      :==:    request-line / status-line
request-line    :==:    method SP request-target SP HTTP-version CRLF
status-line     :==:    HTTP-version SP status-code SP reason-phrase CRLF
header-field    :==:    field-name ":" OWS field-value OWS
```

Parameters **header** (*bytes*) – HTTP header data.

Returns Parsed packet data.

[0] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Return type Union[*DataType_HTTP_Request_Header*, *DataType_HTTP_Response_Header*]

Raises **ProtocolError** – If the packet is malformed.

classmethod id()
Index ID of the protocol.

Returns Index ID of the protocol.

Return type Literal['HTTPv1']

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read Hypertext Transfer Protocol (HTTP/1.*).

Structure of HTTP/1.* packet [**RFC 7230**]:

```
HTTP-message    :==:    start-line
                        *( header-field CRLF )
                        CRLF
                        [ message-body ]
```

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_HTTP*

Raises **ProtocolError** – If the packet is malformed.

_receipt = None
Type of HTTP receipt.

Type Literal['request', 'response']

property alias
Acronym of current protocol.

Return type Literal['HTTP/0.9', 'HTTP/1.0', 'HTTP/1.1']

pcapkit.protocols.application.httpv1.**HTTP_METHODS = ['GET', 'HEAD', 'POST', 'PUT', 'DELETE**
Supported HTTP method.

pcapkit.protocols.application.httpv1.**_RE_METHOD = re.compile(b'GET|HEAD|POST|PUT|DELETE|TRA**
Regular expression to match HTTP methods.

pcapkit.protocols.application.httpv1.**_RE_STATUS = re.compile(b'\\d{3}')**
Regular expression to match HTTP status code.

pcapkit.protocols.application.httpv1.**_RE_VERSION = re.compile(b'HTTP/(?P<version>\\d\\.\\d)**
Regular expression to match HTTP version string.

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the `pcapkit` module.

class pcapkit.protocols.application.httpv1.**DataType_HTTP**

Bases TypedDict

Structure of HTTP/1.* packet [RFC 7230].

receipt: **Literal[request, response]**
HTTP packet receipt.

header: **Union[*DataType_HTTP_Request_Header*, *DataType_HTTP_Response_Header*]**
Parsed HTTP header data.

body: **bytes**
HTTP body data.

raw: ***DataType_HTTP_Raw***
Raw HTTP packet data.

class pcapkit.protocols.application.httpv1.**DataType_HTTP_Raw**

Bases TypedDict

Raw HTTP packet data.

header: **bytes**
Raw HTTP header data.

body: **bytes**
Raw HTTP body data.

packet: **bytes**
Raw HTTP packet data.

class pcapkit.protocols.application.httpv1.**DataType_HTTP_Request_Header**

Bases TypedDict

HTTP request header.

request: ***DataType_HTTP_Request_Header_Meta***
Request metadata.

class pcapkit.protocols.application.httpv1.**DataType_HTTP_Response_Header**

Bases TypedDict

HTTP response header.

response: ***DataType_HTTP_Response_Header_Meta***
Response metadata.

class pcapkit.protocols.application.httpv1.**DataType_HTTP_Request_Header_Meta**

Bases TypedDict

Request metadata.

method: **str**
HTTP request method.

target: **str**
HTTP request target URI.

version: **Literal[0.9, 1.0, 1.1]**
HTTP version string.

class pcapkit.protocols.application.httpv1.**DataType_HTTP_Response_Header_Meta**

Bases TypedDict

Response metadata.

version: **Literal[0.9, 1.0, 1.1]**
HTTP version string.

status: **int**
HTTP response status code.

phrase: **str**
HTTP response status reason.

HTTP/2 - Hypertext Transfer Protocol

pcapkit.protocols.application.httpv2 contains *HTTPv2* only, which implements extractor for Hypertext Transfer Protocol (HTTP/2)*[0], whose structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.payload	Frame Payload

class pcapkit.protocols.application.httpv2.**HTTPv2**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.application.http.HTTP*

This class implements Hypertext Transfer Protocol (HTTP/2).

__length_hint__()
Total length of corresponding protocol.

Return type Literal[9]

_read_http_continuation(*size*, *kind*, *flag*)
Read HTTP/2 CONTINUATION frames.

Structure of HTTP/2 CONTINUATION frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------------------------------------------------------+
```

(continues on next page)

[0] https://en.wikipedia.org/wiki/HTTP/2

(continued from previous page)

```
|                   Header Block Fragment (*)                   ...
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_CONTINUATION*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_data(*size*, *kind*, *flag*)

Read HTTP/2 `DATA` frames.

Structure of HTTP/2 `DATA` frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------+-----------------------------------------------+
|Pad Length? (8)|
+---------------+-----------------------------------------------+
|                            Data (*)                         ...
+---------------------------------------------------------------+
|                           Padding (*)                       ...
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_DATA*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_goaway(*size*, *kind*, *flag*)

Read HTTP/2 `GOAWAY` frames.

Structure of HTTP/2 `GOAWAY` frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
```

(continues on next page)

(continued from previous page)

```
|R|                 Stream Identifier (31)                        |
+-+-------------+---------------+-------------------------------+
|R|                  Last-Stream-ID (31)                          |
+-+-------------------------------------------------------------+
|                       Error Code (32)                         |
+---------------------------------------------------------------+
|                   Additional Debug Data (*)                   |
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_GOAWAY*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_headers(*size*, *kind*, *flag*)

Read HTTP/2 HEADERS frames.

Structure of HTTP/2 HEADERS frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------+-----------------------------------------------+
|Pad Length? (8)|
+-+-------------+-----------------------------------------------+
|E|                 Stream Dependency? (31)                     |
+-+-------------+-----------------------------------------------+
|  Weight? (8)  |
+-+-------------+-----------------------------------------------+
|                   Header Block Fragment (*)                 ...
+---------------------------------------------------------------+
|                           Padding (*)                       ...
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_HEADERS*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_none(*size*, *kind*, *flag*)
Read HTTP packet with unassigned type.

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

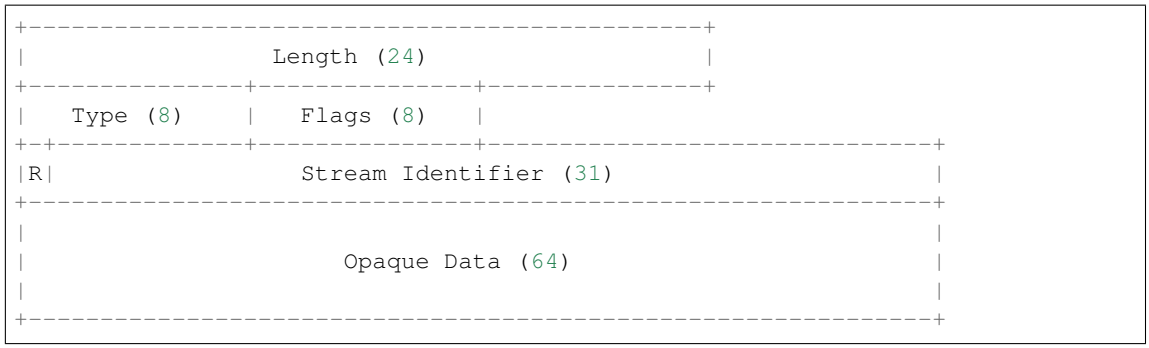
Returns Parsed packet data.

Return type *DataType_HTTPv2_Unassigned*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_ping(*size*, *kind*, *flag*)
Read HTTP/2 `PING` frames.

Structure of HTTP/2 `PING` frame [**RFC 7540**]:



Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

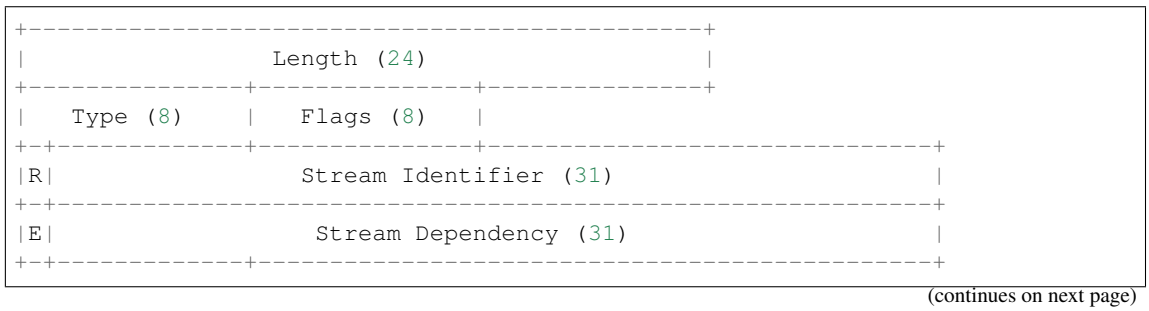
Returns Parsed packet data.

Return type *DataType_HTTPv2_PING*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_priority(*size*, *kind*, *flag*)
Read HTTP/2 `PRIORITY` frames.

Structure of HTTP/2 `PRIORITY` frame [**RFC 7540**]:



(continues on next page)

(continued from previous page)

```
|   Weight (8)  |
+-+-------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_PRIORITY*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_push_promise(*size*, *kind*, *flag*)

Read HTTP/2 `PUSH_PROMISE` frames.

Structure of HTTP/2 `PUSH_PROMISE` frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------+-----------------------------------------------+
|Pad Length? (8)|
+-+-------------+-----------------------------------------------+
|R|                  Promised Stream ID (31)                    |
+-+-----------------------------+-------------------------------+
|                   Header Block Fragment (*)                 ...
+---------------------------------------------------------------+
|                           Padding (*)                       ...
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_PUSH_PROMISE*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_rst_stream(*size*, *kind*, *flag*)

Read HTTP/2 `RST_STREAM` frames.

Structure of HTTP/2 `RST_STREAM` frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
```

(continues on next page)

(continued from previous page)

```
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------------------------------------------------------+
|                         Error Code (32)                       |
+---------------------------------------------------------------+
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_RST_STREAM*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_settings(*size*, *kind*, *flag*)

Read HTTP/2 `SETTINGS` frames.

Structure of HTTP/2 `SETTINGS` frame [**RFC 7540**]:

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+---------------------------------------------------------------+
|       Identifier (16)         |
+-------------------------------+-------------------------------+
|                         Value (32)                            |
+---------------------------------------------------------------+
|                           ......                              |
```

Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

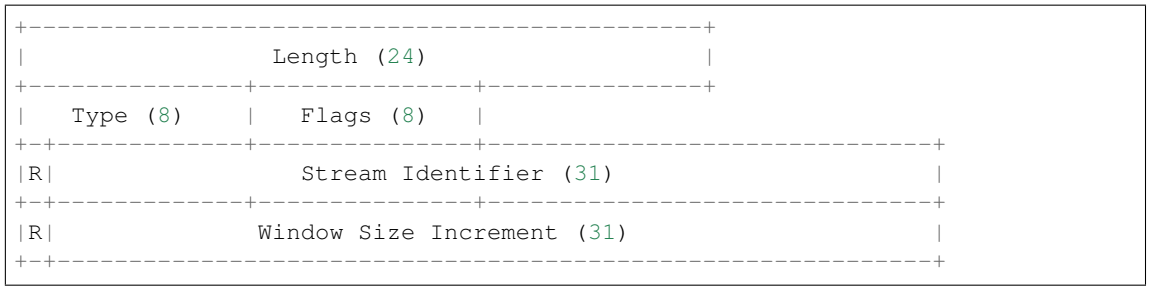
Return type *DataType_HTTPv2_SETTINGS*

Raises ***ProtocolError*** – If the packet is malformed.

_read_http_window_update(*size*, *kind*, *flag*)

Read HTTP/2 `WINDOW_UPDATE` frames.

Structure of HTTP/2 `WINDOW_UPDATE` frame [**RFC 7540**]:



Parameters

- **size** (*int*) – length of packet data
- **kind** (*int*) – packet type
- **flag** (*str*) – packet flags (8 bits)

Returns Parsed packet data.

Return type *DataType_HTTPv2_WINDOW_UPDATE*

Raises ***ProtocolError*** – If the packet is malformed.

classmethod id()

Index ID of the protocol.

Returns Index ID of the protocol.

Return type Literal['HTTPv2']

make(***kwargs*)

Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

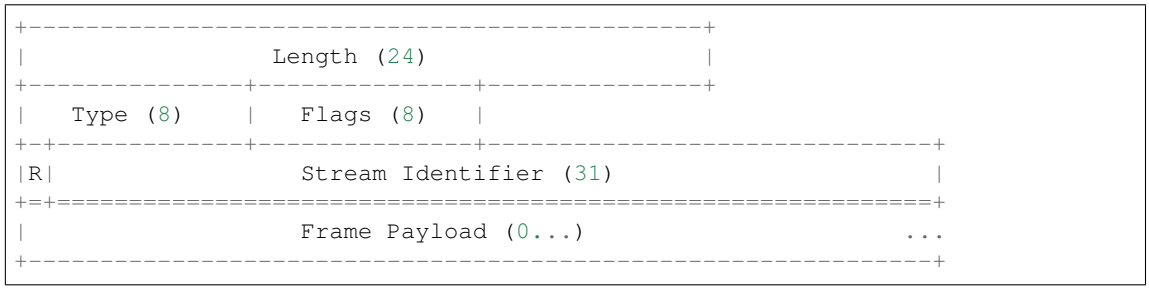
Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)

Read Hypertext Transfer Protocol (HTTP/2).

Structure of HTTP/2 packet [**RFC 7540**]:



Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type *DataType_HTTPv2*

Raises *ProtocolError* – If the packet is malformed.

property alias
Acronym of current protocol.

Return type Literal['HTTP/2']

pcapkit.protocols.application.httpv2.**_HTTP_FUNC: Dict[int, Callable[[pcapkit.protocols.app.**
Process method for HTTP/2 packets.

Code	Method	Description
N/A	*_read_http_none()*	Unsigned
0x00	*_read_http_data()*	DATA
0x01	*_read_http_headers()*	HEADERS
0x02	*_read_http_priority()*	PRIORITY
0x03	*_read_http_rst_stream()*	RST_STREAM
0x04	*_read_http_settings()*	SETTINGS
0x05	*_read_http_push_promise()*	PUSH_PROMISE
0x06	*_read_http_ping()*	PING
0x07	*_read_http_goaway()*	GOAWAY
0x08	*_read_http_window_update()*	WINDOW_UPDATE
0x09	*_read_http_continuation()*	CONTINUATION

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2**

Bases TypedDict

Structure of HTTP/2 packet [**RFC 7540**].

length: **int**
Length.

type: *pcapkit.const.http.frame.Frame*
Type.

sid: **int**
Stream identifier.

packet: **bytes**
Raw packet data.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_Frame**

Bases TypedDict

HTTP/2 packet data.

HTTP/2 Unassigned Frame

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_Unassigned**

Bases DataType_HTTPv2_Frame

flags: **Literal[None]**
HTTP/2 packet flags.

payload: **Optional[types]**
Raw packet payload.

HTTP/2 DATA Frame

For HTTP/2 DATA frame as described in **RFC 7540**, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (0)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.pad_len	Pad Length (Optional)
10	80	http.data	Data
?	?		Padding (Optional)

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_DATA**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 DATA frame [**RFC 7540**].

flags: ***DataType_HTTPv2_DATA_Flags***
HTTP/2 packet flags.

data: **bytes**
HTTP/2 transferred data.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_DATA_Flags**

Bases TypedDict

HTTP/2 DATA frame packet flags.

END_STREAM: **bool**

Bit 0

PADDED: **bool**

Bit 3

HTTP/2 HEADERS Frame

For HTTP/2 HEADERS frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (1)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.pad_len	Pad Length (Optional)
10	80	http.exclusive	Exclusive Flag
10	81	http.deps	Stream Dependency (Optional)
14	112	http.weight	Weight (Optional)
15	120	http.frag	Header Block Fragment
?	?		Padding (Optional)

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_HEADERS**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 HEADERS frame [RFC 7540].

flags: *DataType_HTTPv2_HEADERS_Flags*
HTTP/2 packet flags.

frag: **Optional[bytes]**
Header block fragment.

pad_len: **int**
Pad length.

exclusive: **bool**
Exclusive flag.

deps: **int**
Stream dependency.

weight: **int**
Weight.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_HEADERS_Flags**

Bases TypedDict

HTTP/2 HEADERS frame packet flags.

END_STREAM: **bool**

Bit 0

END_HEADERS: **bool**

Bit 2

PADDED: **bool**

Bit 3

PRIORITY: **bool**

Bit 5

HTTP/2 PRIORITY Frame

For HTTP/2 PRIORITY frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (2)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.exclusive	Exclusive Flag
9	73	http.deps	Stream Dependency
13	104	http.weight	Weight

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_PRIORITY**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 PRIORITY frame [RFC 7540].

flags: **Literal[None]**
HTTP/2 packet flags.

exclusive: **bool**
Exclusive flag.

deps: **int**
Stream dependency.

weight: **int**
Weight.

HTTP/2 RST_STREAM Frame

For HTTP/2 RST_STREAM frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (3)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.error	Error Code

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_RST_STREAM**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 PRIORITY frame [RFC 7540].

flags: **Literal[None]**
HTTP/2 packet flags.

error: ***pcapkit.const.http.error_code.ErrorCode***
Error code.

HTTP/2 SETTINGS Frame

For HTTP/2 SETTINGS frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (4)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.settings	Settings
9	72	http.settings.id	Identifier
10	80	http.settings.value	Value

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_SETTINGS**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 SETTINGS frame [RFC 7540].

flags: *DataType_HTTPv2_SETTINGS_Flags*
HTTP/2 packet flags.

settings: **Tuple[***pcapkit.const.http.setting.Setting***]**
Array of HTTP/2 settings.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_SETTINGS_Flags**

Bases TypedDict

HTTP/2 packet flags.

ACK: bool

Bit 0

HTTP/2 PUSH_PROMISE Frame

For HTTP/2 PUSH_PROMISE frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (5)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.pad_len	Pad Length (Optional)
10	80		Reserved
10	81	http.pid	Promised Stream ID
14	112	http.frag	Header Block Fragment
?	?		Padding (Optional)

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_PUSH_PROMISE**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 PUSH_PROMISE frame [RFC 7540].

flags: *DataType_HTTPv2_PUSH_PROMISE_Flags*
HTTP/2 packet flags.

pid: int
Promised stream ID.

frag: Optional[bytes]
Header block fragment.

pad_len: int
Pad length.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_PUSH_PROMISE_Flags**

Bases TypedDict

HTTP/2 packet flags.

END_HEADERS: bool

Bit 2

PADDED: bool

Bit 3

HTTP/2 PING Frame

For HTTP/2 PING frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (6)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72	http.data	Opaque Data

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_PING**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 PING frame [RFC 7540].

flags: *DataType_HTTPv2_PING_Flags*
HTTP/2 packet flags.

data: bytes
Opaque data.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_PING_Flags**

Bases TypedDict

HTTP/2 packet flags.

ACK: bool

Bit 0

HTTP/2 GOAWAY Frame

For HTTP/2 GOAWAY frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (7)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72		Reserved
9	73	http.last_sid	Last Stream ID
13	104	http.error	Error Code
17	136	http.data	Additional Debug Data (Optional)

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_GOAWAY**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 GOAWAY frame [RFC 7540].

flags: **Literal[None]**
HTTP/2 packet flags.

last_sid: **int**
Last stream ID.

error: ***pcapkit.const.http.error_code.ErrorCode***
Error code.

data: **Optional[None]**
Additional debug data.

HTTP/2 WINDOW_UPDATE Frame

For HTTP/2 WINDOW_UPDATE frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (8)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	72		Reserved
9	73	http.window	Window Size Increment

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_WINDOW_UPDATE**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 WINDOW_UPDATE frame [RFC 7540].

flags: **Literal[None]**
HTTP/2 packet flags.

window: *int*
Window size increment.

HTTP/2 CONTINUATION Frame

For HTTP/2 CONTINUATION frame as described in RFC 7540, its structure is described as below:

Octets	Bits	Name	Description
0	0	http.length	Length
3	24	http.type	Type (9)
4	32	http.flags	Flags
5	40		Reserved
5	41	http.sid	Stream Identifier
9	73	http.frag	Header Block Fragment

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_CONTINUATION**

Bases DataType_HTTPv2_Frame

Structure of HTTP/2 CONTINUATION frame [RFC 7540].

flags: *DataType_HTTPv2_CONTINUATION_Flags*
HTTP/2 packet flags.

frag: *bytes*
Header block fragment.

class pcapkit.protocols.application.httpv2.**DataType_HTTPv2_CONTINUATION_Flags**

Bases TypedDict

HTTP/2 packet flags.

END_HEADERS: *bool*

Bit 2

Base Protocol

pcapkit.protocols.application.application contains only *Application*, which is a base class for application layer protocols, eg. HTTP/1.*, HTTP/2 and etc.

class pcapkit.protocols.application.application.**Application**(*file=None*, *length=None*, ***kwargs*)

Bases: *pcapkit.protocols.protocol.Protocol*

Abstract base class for transport layer protocol family.

__layer__ = 'Application'
Layer of protocol.

classmethod __index__()
Numeral registry index of the protocol.

Raises *IntError* – This protocol doesn't support *__index__()*.

__post_init__(*file=None*, *length=None*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to `make()`.

_decode_next_layer(*dict_*, *proto=None*, *length=None*)
Decode next layer protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *_decode_next_layer()*.

_import_next_layer(*proto*, *length=None*)
Import next layer extractor.

Raises ***UnsupportedCall*** – This protocol doesn't support *_import_next_layer()*.

property layer
Protocol layer.

Return type Literal['Application']

1.3.6 Miscellaneous Protocols

Raw Packet Data

pcapkit.protocols.raw contains *Raw* only, which implements extractor for unknown protocol, and constructs a *Protocol* like object.

class pcapkit.protocols.raw.**Raw**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.protocol.Protocol*

This class implements universal unknown protocol.

classmethod __index__()
Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__post_init__(*file*, *length=None*, ***, *error=None*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*io.BytesIO*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **error** (*Optional[str]*) – Parsing errors if any (for parsing).
- ****kwargs** – Arbitrary keyword arguments.

Would *pcapkit* encounter malformed packets, the original parsing error message will be provided as in `error`.

See also:

For construction argument, please refer to *make()*.

make(***kwargs*)
Make raw packet data.

Keyword Arguments

- **packet** (*bytes*) – Raw packet data.
- ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, *, *error=None*, ***kwargs*)
Read raw packet data.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **error** (*Optional[str]*) – Parsing errors if any.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed packet data.

Return type *DataType_Raw*

property length
Header length of current protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *length*.

property name
Name of current protocol.

Return type Literal['Unknown']

property protocol
Name of next layer protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *protocol*.

Data Structure

Important: Following classes are only for *documentation* purpose. They do **NOT** exist in the *pcapkit* module.

class pcapkit.protocols.raw.**DataType_Raw**

Bases TypedDict

Raw packet data.

packet: **bytes**
raw packet data

error: **Optional[str]**
optional error message

No-Payload Packet

pcapkit.protocols.null contains *NoPayload* only, which implements a *Protocol* like object whose payload is recursively *NoPayload* itself.

class pcapkit.protocols.null.**NoPayload**(*file=None*, *length=None*, ***kwargs*)
Bases: *pcapkit.protocols.protocol.Protocol*

This class implements no-payload protocol.

classmethod __index__()
Numeral registry index of the protocol.

Raises ***UnsupportedCall*** – This protocol has no registry entry.

__post_init__(*file=None*, *length=None*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

_decode_next_layer(**args*, ***kwargs*)
Decode next layer protocol.

Parameters ***args** – arbitrary positional arguments

Keyword Arguments ****kwargs** – arbitrary keyword arguments

Raises ***UnsupportedCall*** – This protocol doesn't support *_decode_next_layer()*.

_import_next_layer(**args*, ***kwargs*)
Import next layer extractor.

Parameters ***args** – arbitrary positional arguments

Keyword Arguments ****kwargs** – arbitrary keyword arguments

Raises ***UnsupportedCall*** – This protocol doesn't support *_import_next_layer()*.

make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

read(*length=None*, ***kwargs*)
Read (parse) packet data.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type dict

property length
Header length of current protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *length*.

property name
Name of current protocol.

Return type Literal['Null']

property protocol
Name of next layer protocol.

Raises ***UnsupportedCall*** – This protocol doesn't support *protocol*.

1.3.7 Base Protocol

class pcapkit.protocols.protocol.**Protocol**(*file=None*, *length=None*, ***kwargs*)
Bases: object

Abstract base class for all protocol family.

__layer__ = None
Layer of protocol. Can be one of Link, Internet, Transport and Application.

Type Literal['Link', 'Internet', 'Transport', 'Application']

__proto__ = {}
Protocol index mapping for decoding next layer, c.f. *self._decode_next_layer* & *self._import_next_layer*. The values should be a tuple representing the module name and class name.

Type DefaultDict[int, Tuple[str, str]]

__bytes__()
Returns source data stream in bytes.

__contains__(*name*)
Returns if name is in self._info.

Parameters **name** (*Any*) – name to search

Returns if name exists

Return type bool

classmethod __eq__(*other*)
Returns if other is of the same protocol as the current object.

Parameters **other** (*Union[Protocol, Type[Protocol]]*) – Comparision against the object.

Returns If other is of the same protocol as the current object.

Return type bool

__getitem__(*key*)
Subscription (getitem) support.

- If key is a :obj`slice` object, *ProtocolUnbound* will be raised.
- If key is a *Protocol* object, the method will fetch its indexes (*id()*).
- Later, search the packet's chain of protocols with the calculated key.
- If no matches, then raises *ProtocolNotFound*.

Parameters **key** (*Union[str, Protocol, Type[Protocol]]*) – Indexing key.

Returns The sub-packet from the current packet of indexed protocol.

Return type *pcapkit.protocols.protocol.Protocol*

Raises

- ***ProtocolUnbound*** – If `key` is a `slice` object.
- ***ProtocolNotFound*** – If `key` is not in the current packet.

__hash__()
Return the hash value for `self._data`.

abstract classmethod __index__()
Numeral registry index of the protocol.

Returns Numeral registry index of the protocol.

Return type enum.IntEnum

__init__(*file=None*, *length=None*, ***kwargs*)
Initialisation.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments

- **_error** (*bool*) – If the object is initiated after parsing errors (*self._onerror*).
- **_layer** (*str*) – Parse packet until `_layer` (*self._onerror*).
- **_protocol** (*str*) – Parse packet until `_protocol` (*self._onerror*).
- ****kwargs** – Arbitrary keyword arguments.

__iter__()
Iterate through `self._data`.

__length_hint__()
Return an estimated length for the object.

__post_init__(*file=None*, *length=None*, ***kwargs*)
Post initialisation hook.

Parameters

- **file** (*Optional[io.BytesIO]*) – Source packet stream.
- **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

See also:

For construction argument, please refer to *make()*.

__repr__()
Returns representation of parsed protocol data.

Example

```
>>> protocol
<Frame Info(..., ethernet=Info(...), protocols='Ethernet:IPv6:Raw')>
```

_check_term_threshold()
Check if reached termination threshold.

Returns if reached termination threshold

Return type bool

_decode_next_layer(*dict_*, *proto=None*, *length=None*)
Decode next layer protocol.

Parameters

- **dict** (*dict*) – info buffer
- **proto** (*int*) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Returns current protocol with next layer extracted

Return type dict

_import_next_layer(*proto*, *length=None*)
Import next layer extractor.

Parameters

- **proto** (*int*) – next layer protocol index
- **length** (*int*) – valid (*non-padding*) length

Returns instance of next layer

Return type *pcapkit.protocols.protocol.Protocol*

classmethod _make_index(*name*, *default=None*, ***, *namespace=None*, *reversed=False*, *pack=False*, *size=4*, *signed=False*, *lilendian=False*)
Return first index of `name` from a `dict` or enumeration.

Parameters

- **name** (*Union[str, int, enum.IntEnum]*) – item to be indexed
- **default** (*int*) – default value

Keyword Arguments

- **namespace** (*Union[dict, enum.EnumMeta]*) – namespace for item
- **reversed** (*bool*) – if namespace is `str -> int` pairs
- **pack** (*bool*) – if need `struct.pack()` to pack the result
- **size** (*int*) – buffer size
- **signed** (*bool*) – signed flag
- **lilendian** (*bool*) – little-endian flag

Returns Index of `name` from a dict or enumeration. If `packet` is `True`, returns `bytes`; otherwise, returns `int`.

Return type Union[int, bytes]

Raises ***ProtocolNotImplemented*** – If `name` is **NOT** in `namespace` and `default` is `None`.

classmethod _make_pack(*integer*, ***, *size=1*, *signed=False*, *lilendian=False*)
Pack integers to bytes.

Parameters **integer** (*int*) –

Keyword Arguments

- **size** (*int*) – buffer size
- **signed** (*bool*) – signed flag
- **lilendian** (*bool*) – little-endian flag

Returns Packed data upon success.

Return type bytes

Raises ***StructError*** – If failed to pack the integer.

_read_binary(*size=1*)
Read bytes and convert into binaries.

Parameters **size** (*int*) – buffer size

Returns binary bits (`0`/`1`)

Return type str

_read_fileng(**args*, ***kwargs*)
Read file buffer (`self._file`).

This method wraps the `file.read()` call.

Parameters ***args** – arbitrary positional arguments

Keyword Arguments ****kwargs** – arbitrary keyword arguments

Returns Data read from file buffer.

Return type bytes

_read_packet(*length=None*, ***, *header=None*, *payload=None*, *discard=False*)
Read raw packet data.

Parameters **length** (*int*) – length of the packet

Keyword Arguments

- **header** (*Optional[int]*) – length of the packet header
- **payload** (*Optional[int]*) – length of the packet payload
- **discard** (*bool*) – flag if discard header data

Returns

- If `header` omits, returns the whole packet data in `bytes`.
- If `discard` is set as `True`, returns the packet body (in `bytes`) only.
- Otherwise, returns the header and payload data as a `dict`:

```
class Packet(TypedDict):
    """Header and payload data."""

    #: packet header
    header: bytes
    #: packet payload
    payload: bytes
```

_read_protos(*size*)
Read next layer protocol type.

Parameters **size** (*int*) – buffer size

Returns

- If *succeed*, returns the name of next layer protocol (str).
- If *fail*, returns None.

_read_unpack(*size=1*, *, *signed=False*, *lilendian=False*, *quiet=False*)
Read bytes and unpack for integers.

Parameters **size** (*int*) – buffer size

Keyword Arguments

- **signed** (*bool*) – signed flag
- **lilendian** (*bool*) – little-endian flag
- **quiet** (*bool*) – quiet (no exception) flag

Returns unpacked data upon success

Return type Optional[int]

Raises ***StructError*** – If unpack (struct.pack()) failed, and struct.error raised.

static decode(*byte*, *, *encoding=None*, *errors='strict'*)
Decode bytes into str.

Should decoding failed using encoding, the method will try again decoding the bytes as 'unicode_escape'.

Parameters **byte** (*bytes*) – Source bytestring.

Keyword Arguments

- **encoding** (*Optional[str]*) – The encoding with which to decode the bytes. If not provided, *pcapkit* will first try detecting its encoding using chardet. The fallback encoding would is **UTF-8**.
- **errors** (*str*) – The error handling scheme to use for the handling of decoding errors. The default is 'strict' meaning that decoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error() that can handle UnicodeDecodeError.

Returns Decoede string.

Return type str

See also:

bytes.decode()

classmethod id()
Index ID of the protocol.

By default, it returns the name of the protocol.

Returns Index ID of the protocol.

Return type Union[str, Tuple[str]]

See also:

pcapkit.protocols.protocol.Protocol.__getitem__()

abstract make(***kwargs*)
Make (construct) packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Constructed packet data.

Return type bytes

abstract read(*length=None*, ***kwargs*)
Read (parse) packet data.

Parameters **length** (*Optional[int]*) – Length of packet data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Parsed packet data.

Return type dict

static unquote(*url*, ***, *encoding='utf-8'*, *errors='replace'*)
Unquote URLs into readable format.

Should decoding failed , the method will try again replacing `'%'` with `'\x'` then decoding the `url` as `'unicode_escape'`.

Parameters **url** (*str*) – URL string.

Keyword Arguments

- **encoding** (*str*) – The encoding with which to decode the `bytes`.
- **errors** (*str*) – The error handling scheme to use for the handling of decoding errors. The default is `'strict'` meaning that decoding errors raise a `UnicodeDecodeError`. Other possible values are `'ignore'` and `'replace'` as well as any other name registered with `codecs.register_error()` that can handle `UnicodeDecodeError`.

Returns Unquoted string.

Return type str

See also:

`urllib.parse.unquote()`

_exlayer
Parse packet until such layer.

Type str

_exproto
Parse packet until such protocol.

Type str

_onerror
If the object is initiated after parsing errors.

Type bool

_seekset
Initial offset of `self._file`

Type int

_sigterm
If terminate parsing next layer of protocol.

Type bool

property alias
Acronym of current protocol.

Return type str

property data
Binary packet data of current instance.

Return type bytes

property info
Info dict of current instance.

Return type *pcapkit.corekit.infoclass.Info*

abstract property length
Header length of current protocol.

Return type int

abstract property name
Name of current protocol.

Return type str

property payload
Payload of current instance.

Return type *pcapkit.protocols.protocol.Protocol*

property protochain
Protocol chain of current instance.

Return type *pcapkit.corekit.protochain.ProtoChain*

property protocol
Name of next layer protocol (if any).

Return type Optional[str]

1.4 Reassembly Packets & Datagrams

pcapkit.reassembly bases on algorithms described in **RFC 815**, implements datagram reassembly of IP and TCP packets.

1.4.1 Fragmented Packets Reassembly

pcapkit.reassembly.reassembly contains class:~*pcapkit.reassembly.reassembly.Reassembly* only, which is an abstract base class for all reassembly classes, bases on algorithms described in **RFC 815**, implements datagram reassembly of IP and TCP packets.

class pcapkit.reassembly.reassembly.**Reassembly**(*, *strict=True*)
Bases: object

Base class for reassembly procedure.

__call__(*packet*)
Call packet reassembly.

Parameters **packet** (*dict*) – packet dict to be reassembled (detailed format described in corresponding protocol)

__init__(*, *strict=True*)
Initialise packet reassembly.

Keyword Arguments **strict** (*bool*) – if return all datagrams (including those not implemented) when submit

fetch()
Fetch datagram.

Returns Tuple of reassembled datagrams.

Return type Tuple[dict]

Fetch reassembled datagrams from *_dtgram* and returns a *tuple* of such datagrams.

If *_newflg* set as `True`, the method will call *submit()* to (*force*) obtain newly reassembled payload. Otherwise, the already calculated *_dtgram* will be returned.

index(*pkt_num*)
Return datagram index.

Parameters **pkt_num** (*int*) – index of packet

Returns reassembled datagram index which was from No. `pkt_num` packet; if not found, returns `None`

Return type Optional[int]

abstract reassembly(*info*)
Reassembly procedure.

Parameters **info** (pcapkit.corekit.infoclass.Info) – info dict of packets to be reassembled

run(*packets*)
Run automatically.

Parameters **packets** (*List[dict]*) – list of packet dicts to be reassembled

abstract submit(*buf*, ***kwargs*)
Submit reassembled payload.

Parameters **buf** (*dict*) – buffer dict of reassembled packets

_buffer
dict buffer field

_dtgram
list reassembled datagram

_newflg
if new packets reassembled flag

Type bool

_strflg
strict mode flag

Type bool

property count
Total number of reassembled packets.

Return type int

property datagram
Reassembled datagram.

Return type tuple

abstract property name
Protocol of current packet.

Return type str

abstract property protocol
Protocol of current reassembly object.

Return type str

1.4.2 IP Datagram Reassembly

pcapkit.reassembly.ip contains *IP_Reassembly* only, which reconstructs fragmented IP packets back to origin. The following algorithm implement is based on IP reassembly procedure introduced in **RFC 791**, using RCVBT (fragment receivedbit table). Though another algorithm is explained in **RFC 815**, replacing RCVBT, however, this implement still used the elder one.

Notation

FO	Fragment Offset
IHL	Internet Header Length
MF	More Fragments Flag
TTL	Time To Live
NFB	Number of Fragment Blocks
TL	Total Length
TDL	Total Data Length
BUFID	Buffer Identifier
RCVBT	Fragment Received Bit Table
TLB	Timer Lower Bound

Algorithm

```
DO {
   BUFID <- source|destination|protocol|identification;

   IF (FO = 0 AND MF = 0) {
      IF (buffer with BUFID is allocated) {
         flush all reassembly for this BUFID;
         Submit datagram to next step;
         DONE.
      }
   }

   IF (no buffer with BUFID is allocated) {
      allocate reassembly resources with BUFID;
      TIMER <- TLB;
      TDL <- 0;
      put data from fragment into data buffer with BUFID
         [from octet FO*8 to octet (TL-(IHL*4))+FO*8];
      set RCVBT bits [from FO to FO+((TL-(IHL*4)+7)/8)];
   }

   IF (MF = 0) {
      TDL <- TL-(IHL*4)+(FO*8)
   }

   IF (FO = 0) {
      put header in header buffer
   }

   IF (TDL # 0 AND all RCVBT bits [from 0 to (TDL+7)/8] are set) {
      TL <- TDL+(IHL*4)
      Submit datagram to next step;
      free all reassembly resources for this BUFID;
      DONE.
   }

   TIMER <- MAX(TIMER,TTL);

} give up until (next fragment or timer expires);

timer expires: {
   flush all reassembly with this BUFID;
   DONE.
}
```

Implementation

class pcapkit.reassembly.ip.**IP_Reassembly**(*, *strict=True*)

Bases: *pcapkit.reassembly.reassembly.Reassembly*

Reassembly for IP payload.

reassembly(*info*)

Reassembly procedure.

Parameters **info** (pcapkit.corekit.infoclass.Info) – info dict of packets to be reassembled

submit (*buf*, ***, *checked=False*)
Submit reassembled payload.

Parameters **buf** (*dict*) – buffer dict of reassembled packets

Keyword Arguments **bufid** (tuple) – buffer identifier

Returns reassembled packets

Return type list

1.4.3 IPv4 Datagram Reassembly

pcapkit.reassembly.ipv4 contains *IPv4_Reassembly* only, which reconstructs fragmented IPv4 packets back to origin. Please refer to *IP Datagram Reassembly* for more information.

Data Structure

ipv4.packet Data structure for **IPv4 datagram reassembly** (*reassembly()*) is as following:

ipv4.datagram Data structure for **reassembled IPv4 datagram** (element from *datagram* *tuple*) is as following:

ipv4.buffer Data structure for internal buffering when performing reassembly algorithms (*_buffer*) is as following:

```
(dict) buffer --> memory buffer for reassembly
 |--> (tuple) BUFID : (dict)
 |      |--> ipv4.src        |
 |      |--> ipc6.dst        |
 |      |--> ipv4.label      |
 |      |--> ipv4_frag.next  |
 |                           |--> 'TDL' : (int) total data length
 |                           |--> RCVBT : (bytearray) fragment received bit table
 |                           |              |--> (bytes) b'\x00' -> not received
 |                           |              |--> (bytes) b'\x01' -> received
 |                           |              |--> (bytes) ...
 |                           |--> 'index' : (list) list of reassembled packets
 |                           |               |--> (int) packet range number
 |                           |--> 'header' : (bytearray) header buffer
 |                           |--> 'datagram' : (bytearray) data buffer, holes set␣
↪to b'\x00'
 |--> (tuple) BUFID ...
```

Implementation

class pcapkit.reassembly.ipv4.**IPv4_Reassembly**(***, *strict=True*)
Bases: *pcapkit.reassembly.ip.IP_Reassembly*

Reassembly for IPv4 payload.

Example

```
>>> from pcapkit.reassembly import IPv4_Reassembly
# Initialise instance:
>>> ipv4_reassembly = IPv4_Reassembly()
# Call reassembly:
>>> ipv4_reassembly(packet_dict)
```

(continues on next page)

(continued from previous page)

```
# Fetch result:
>>> result = ipv4_reassembly.datagram
```

---

property name
Protocol of current packet.

Return type Literal['Internet Protocol version 4']

property protocol
Protocol of current reassembly object.

Return type Literal['IPv4']

1.4.4 IPv6 Datagram Reassembly

pcapkit.reassembly.ipv6 contains *IPv6_Reassembly* only, which reconstructs fragmented IPv6 packets back to origin. Please refer to *IP Datagram Reassembly* for more information.

Data Structure

ipv6.packet Data structure for **IPv6 datagram reassembly** (*reassembly()*) is as following:

```
packet_dict = dict(
  bufid = tuple(
      ipv6.src,                   # source IP address
      ipv6.dst,                   # destination IP address
      ipv6.label,                 # label
      ipv6_frag.next,             # next header field in IPv6 Fragment Header
  ),
  num = frame.number,             # original packet range number
  fo = ipv6_frag.offset,          # fragment offset
  ihl = ipv6.hdr_len,             # header length, only headers before IPv6-Frag
  mf = ipv6_frag.mf,              # more fragment flag
  tl = ipv6.len,                  # total length, header includes
  header = ipv6.header,           # raw bytearray type header before IPv6-Frag
  payload = ipv6.payload,         # raw bytearray type payload after IPv6-Frag
)
```

ipv6.datagram Data structure for **reassembled IPv6 datagram** (element from *datagram* *tuple*) is as following:

```
(tuple) datagram
 |--> (dict) data
 |      |--> 'NotImplemented' : (bool) True --> implemented
 |      |--> 'index' : (tuple) packet numbers
 |      |               |--> (int) original packet range number
 |      |--> 'packet' : (Optional[bytes]) reassembled IPv6 packet
 |--> (dict) data
 |      |--> 'NotImplemented' : (bool) False --> not implemented
 |      |--> 'index' : (tuple) packet numbers
 |      |               |--> (int) original packet range number
 |      |--> 'header' : (Optional[bytes]) IPv6 header
 |      |--> 'payload' : (Optional[tuple]) partially reassembled IPv6 payload
 |                        |--> (Optional[bytes]) IPv4 payload fragment
 |--> (dict) data ...
```

ipv6.buffer Data structure for internal buffering when performing reassembly algorithms (*_buffer*) is as following:

```
(dict) buffer --> memory buffer for reassembly
 |--> (tuple) BUFID : (dict)
 |      |--> ipv6.src        |
 |      |--> ipc6.dst        |
 |      |--> ipv6.label      |
 |      |--> ipv6_frag.next  |
 |                           |--> 'TDL' : (int) total data length
 |                           |--> RCVBT : (bytearray) fragment received bit table
 |                           |              |--> (bytes) b'\x00' -> not received
 |                           |              |--> (bytes) b'\x01' -> received
 |                           |              |--> (bytes) ...
 |                           |--> 'index' : (list) list of reassembled packets
 |                           |                |--> (int) packet range number
 |                           |--> 'header' : (bytearray) header buffer
 |                           |--> 'datagram' : (bytearray) data buffer, holes set␣
↪to b'\x00'
 |--> (tuple) BUFID ...
```

Implementation

class pcapkit.reassembly.ipv6.**IPv6_Reassembly**(*, *strict=True*)

Bases: *pcapkit.reassembly.ip.IP_Reassembly*

Reassembly for IPv6 payload.

Example

```
>>> from pcapkit.reassembly import IPv6_Reassembly
# Initialise instance:
>>> ipv6_reassembly = IPv6_Reassembly()
# Call reassembly:
>>> ipv6_reassembly(packet_dict)
# Fetch result:
>>> result = ipv6_reassembly.datagram
```

---

property name

Protocol of current packet.

Return type Literal['Internet Protocol version 6']

property protocol

Protocol of current reassembly object.

Return type Literal['IPv6']

1.4.5 TCP Datagram Reassembly

pcapkit.reassembly.tcp contains *TCP_Reassembly* only, which reconstructs fragmented TCP packets back to origin. The algorithm for TCP reassembly is described as below.

Notation

DSN	Data Sequence Number
ACK	TCP Acknowledgement
SYN	TCP Synchronisation Flag
FIN	TCP Finish Flag
RST	TCP Reset Connection Flag
BUFID	Buffer Identifier
HDL	Hole Discriptor List
ISN	Initial Sequence Number
src	source IP
dst	destination IP
srcport	source TCP port
dstport	destination TCP port

Algorithm

```
DO {
   BUFID <- src|dst|srcport|dstport|ACK;
   IF (SYN is true) {
      IF (buffer with BUFID is allocated) {
         flush all reassembly for this BUFID;
         submit datagram to next step;
      }
   }

   IF (no buffer with BUFID is allocated) {
      allocate reassembly resources with BUFID;
      ISN <- DSN;
      put data from fragment into data buffer with BUFID
         [from octet fragment.first to octet fragment.last];
      update HDL;
   }

   IF (FIN is true or RST is true) {
      submit datagram to next step;
      free all reassembly resources for this BUFID;
      BREAK.
   }
} give up until (next fragment);

update HDL: {
   DO {
      select the next hole descriptor from HDL;

      IF (fragment.first >= hole.first) CONTINUE.
      IF (fragment.last <= hole.first) CONTINUE.
```

(continues on next page)

(continued from previous page)

```
      delete the current entry from HDL;

      IF (fragment.first >= hole.first) {
         create new entry "new_hole" in HDL;
         new_hole.first <- hole.first;
         new_hole.last <- fragment.first - 1;
         BREAK.
      }

      IF (fragment.last <= hole.last) {
         create new entry "new_hole" in HDL;
         new_hole.first <- fragment.last + 1;
         new_hole.last <- hole.last;
         BREAK.
      }
   } give up until (no entry from HDL)
}
```

The following algorithm implement is based on **IP Datagram Reassembly Algorithm** introduced in **RFC 815**. It described an algorithm dealing with `RCVBT` (fragment received bit table) appeared in **RFC 791**. And here is the process:

1. Select the next hole descriptor from the hole descriptor list. If there are no more entries, go to step eight.
2. If `fragment.first` is greater than `hole.last`, go to step one.
3. If `fragment.last` is less than `hole.first`, go to step one.
4. Delete the current entry from the hole descriptor list.
5. If `fragment.first` is greater than `hole.first`, then create a new hole descriptor `new_hole` with `new_hole.first` equal to `hole.first`, and `new_hole.last` equal to `fragment.first` minus one (`-1`).
6. If `fragment.last` is less than `hole.last` and `fragment.more_fragments` is `true`, then create a new hole descriptor `new_hole`, with `new_hole.first` equal to `fragment.last` plus one (`+1`) and `new_hole.last` equal to `hole.last`.
7. Go to step one.
8. If the hole descriptor list is now empty, the datagram is now complete. Pass it on to the higher level protocol processor for further handling. Otherwise, return.

Data Structure

tcp.packet Data structure for **TCP datagram reassembly** (*`reassembly()`*) is as following:

```
packet_dict = Info(
  bufid = tuple(
      ip.src,                     # source IP address
      ip.dst,                     # destination IP address
      tcp.srcport,                # source port
      tcp.dstport,                # destination port
  ),
  num = frame.number,             # original packet range number
  syn = tcp.flags.syn,            # synchronise flag
  fin = tcp.flags.fin,            # finish flag
  rst = tcp.flags.rst,            # reset connection flag
```

(continues on next page)

(continued from previous page)

```
  len = tcp.raw_len,              # payload length, header excludes
  first = tcp.seq,                # this sequence number
  last = tcp.seq + tcp.raw_len,   # next (wanted) sequence number
  payload = tcp.raw,              # raw bytearray type payload
)
```

tcp.datagram Data structure for **reassembled TCP datagram** (element from *datagram tuple*) is as following:

```
(tuple) datagram
 |--> (Info) data
 |      |--> 'NotImplemented' : (bool) True --> implemented
 |      |--> 'id' : (Info) original packet identifier
 |      |             |--> 'src' --> (tuple)
 |      |             |               |--> (str) ip.src
 |      |             |               |--> (int) tcp.srcport
 |      |             |--> 'dst' --> (tuple)
 |      |             |               |--> (str) ip.dst
 |      |             |               |--> (int) tcp.dstport
 |      |             |--> 'ack' --> (int) original packet ACK number
 |      |--> 'index' : (tuple) packet numbers
 |      |               |--> (int) original packet range number
 |      |--> 'payload' : (Optional[bytes]) reassembled application layer data
 |      |--> 'packets' : (Tuple[Analysis]) analysed payload
 |--> (Info) data
 |      |--> 'NotImplemented' : (bool) False --> not implemented
 |      |--> 'id' : (Info) original packet identifier
 |      |             |--> 'src' --> (tuple)
 |      |             |               |--> (str) ip.src
 |      |             |               |--> (int) tcp.srcport
 |      |             |--> 'dst' --> (tuple)
 |      |             |               |--> (str) ip.dst
 |      |             |               |--> (int) tcp.dstport
 |      |             |--> 'ack' --> (int) original packet ACK number
 |      |--> 'ack' : (int) original packet ACK number
 |      |--> 'index' : (tuple) packet numbers
 |      |               |--> (int) original packet range number
 |      |--> 'payload' : (Optional[tuple]) partially reassembled payload
 |      |                  |--> (Optional[bytes]) payload fragment
 |      |--> 'packets' : (Tuple[Analysis]) analysed payloads
 |--> (Info) data ...
```

tcp.buffer Data structure for internal buffering when performing reassembly algorithms (*_buffer*) is as following:

```
(dict) buffer --> memory buffer for reassembly
 |--> (tuple) BUFID : (dict)
 |       |--> ip.src      |
 |       |--> ip.dst      |
 |       |--> tcp.srcport |
 |       |--> tcp.dstport |
 |                        |--> 'hdl' : (list) hole descriptor list
 |                        |             |--> (Info) hole --> hole descriptor
 |                        |                    |--> "first" --> (int) start of hole
 |                        |                    |--> "last" --> (int) stop of hole
 |                        |--> (int) ACK : (dict)
 |                        |                 |--> 'ind' : (list) list of␣
↪reassembled packets
 |                        |                 |            |--> (int) packet range␣
↪number
```

(continues on next page)

(continued from previous page)

```
 |                          |                  |--> 'isn' : (int) ISN of payload␣
↪buffer
 |                          |                  |--> 'len' : (int) length of payload␣
↪buffer
 |                          |                  |--> 'raw' : (bytearray) reassembled␣
↪payload, holes set to b'\x00'
 |                          |--> (int) ACK ...
 |                          |--> ...
 |--> (tuple) BUFID ...
```

Implementation

class pcapkit.reassembly.tcp.**TCP_Reassembly**(*, *strict=True*)

Bases: *pcapkit.reassembly.reassembly.Reassembly*

Reassembly for TCP payload.

Example

```
>>> from pcapkit.reassembly import TCP_Reassembly
# Initialise instance:
>>> tcp_reassembly = TCP_Reassembly()
# Call reassembly:
>>> tcp_reassembly(packet_dict)
# Fetch result:
>>> result = tcp_reassembly.datagram
```

---

reassembly(*info*)

Reassembly procedure.

Parameters **info** (pcapkit.corekit.infoclass.Info) – *info* dict of packets to be reassembled

submit(*buf*, *, *bufid*)

Submit reassembled payload.

Parameters **buf** (*dict*) – *buffer* dict of reassembled packets

Keyword Arguments **bufid** (tuple) – buffer identifier

Returns reassembled *packets*

Return type List[dict]

property name

Protocol of current packet.

Return type Literal['Transmission Control Protocol']

property protocol

Protocol of current reassembly object.

Return type Literal['TCP']

1.5 Core Utilities

pcapkit.corekit is the collection of core utilities for *pcapkit* implementation, including dict like class *Info*, tuple like class *VersionInfo*, and protocol collection class *ProtoChain*.

1.5.1 Info Class

pcapkit.corekit.infoclass contains dict like class *Info* only, which is originally designed to work alike dataclasses.dataclass() as introduced in **PEP 557**.

class pcapkit.corekit.infoclass.**Info**(*dict_=None*, ***kwargs*)
Bases: collections.abc.Mapping

Turn dictionaries into object like instances.

Notes

- *Info* objects inherit from dict type
- *Info* objects are *iterable*, and support all functions as dict
- *Info* objects are **one-time-modeling**, thus cannot set or delete attributes after initialisation

---

static **__new__**(*cls*, *dict_=None*, ***kwargs*)
Create a new instance.

Parameters **dict** (*Dict[str, Any]*) – Source dict data.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Notes

Keys with the same names as the builtin methods will be renamed with 2 suffix implicitly and internally.

---

info2dict()
Convert *Info* into dict.

Returns Converted dict.

Return type Dict[str, Any]

1.5.2 Protocol Chain

pcapkit.corekit.protochain contains special protocol collection class *ProtoChain*.

class pcapkit.corekit.protochain.**ProtoChain**(*proto=None*, *alias=None*, ***, *basis=None*)
Bases: collections.abc.Container

Protocols chain.

__alias__: ***pcapkit.corekit.protochain._AliasList***
Protocol aliases chain.

__proto__: ***pcapkit.corekit.protochain._ProtoList***
Protocol classes chain.

__contains__(*name*)
Returns if `name` is in the chain.

Parameters **name** (*Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]*) – name to search

Returns if `name` is in the chain

Return type bool

__getitem__(*index*)
Subscription (`getitem`) support.

Parameters **index** (*int*) – Indexing key.

Returns Protocol alias at such index.

Return type str

__init__(*proto=None*, *alias=None*, ***, *basis=None*)
Initialisation.

Parameters

- **proto** (*Optional[pcapkit.protocols.protocol.Protocol]*) – New protocol class on the top stack.
- **alias** (*Optional[str]*) – New protocol alias on the top stack.

Keyword Arguments **basis** (pcapkit.corekit.protochain.ProtoChain) – Original protocol chain as base stacks.

__repr__()
Returns representation of protocol chain data.

Example

```
>>> protochain
ProtoChain(<class 'pcapkit.protocols.link.ethernet.Ethernet'>, ...)
```

__str__()
Returns formatted hex representation of source data stream.

Example

```
>>> protochain
ProtoChain(<class 'pcapkit.protocols.link.ethernet.Ethernet'>, ...)
>>> print(protochain)
Ethernet:IPv6:Raw
```

count(*value*)
Number of occurrences of `value`.

Parameters **value** – (Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]): value to search

Returns Number of occurrences of `value`.

Return type int

See also:

This method calls *self.__alias__.count* for the actual processing.

index (*value*, *start=None*, *stop=None*)
First index of `value`.

Parameters

- **value** (*Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]*) – value to search
- **start** (*int*) – start offset
- **stop** (*int*) – stop offset

Returns First index of `value`.

Return type int

Raises ***IntError*** – If the value is not present.

See also:

This method calls *self.__alias__.index* for the actual processing.

property alias
Protocol aliases chain.

Return type pcapkit.corekit.protocol._AliasList

property chain
Protocol chain string.

Return type str

property proto
Protocol classes chain.

Return type pcapkit.corekit.protocol._ProtoList

tuple
Protocol names.

Return type Tuple[str]

class pcapkit.corekit.protochain.**_AliasList**(*data=None*, *, *base=None*)
Bases: collections.abc.Sequence

List of protocol aliases for ProtoChain

__data__: List[str]
Protocol aliases chain data.

__contains__(*x*)
Returns if `x` is in the chain.

Parameters **x** (*Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]*) – name to search

Returns if `x` is in the chain

Return type bool

__getitem__(*index*)
Subscription (`getitem`) support.

Parameters **index** (*int*) – Indexing key.

Returns Protocol alias at such index.

Return type str

__init__(*data=None*, *, *base=None*)
Initialisation.

Parameters **data** (*Optional[str]*) – New protocol alias on top stack.

Keyword Arguments **base** (*Union[pcapkit.corekit.protochain._AliasLists, List[str]]*) – Original protocol alias chain as base stacks.

__iter__()
Iterate through the protocol chain.

Return type Iterator[str]

__len__()
Length of the protocol chain.

Return type int

__reversed__()
Reverse the protocol alias chain.

Return type List[str]

count(*value*)
Number of occurrences of `value`.

Parameters **value** – (Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]): value to search

Returns Number of occurrences of `value`.

Return type int

index(*value*, *start=0*, *stop=None*)
First index of `value`.

Parameters

- **value** (*Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]*) – value to search
- **start** (*int*) – start offset
- **stop** (*int*) – stop offset

Returns First index of `value`.

Return type int

Raises ***IntError*** – If the value is not present.

property data
Protocol alias data.

Return type List[str]

class pcapkit.corekit.protochain.**_ProtoList**(*data=None*, *, *base=None*)
Bases: collections.abc.Collection

List of protocol classes for *ProtoChain*.

__data__: List[*pcapkit.protocols.protocol.Protocol*]
Protocol classes chain data.

__contains__(*x*)
Returns if x is in the chain.

Parameters **x** (*Union[str, pcapkit.protocols.protocol.Protocol, Type[pcapkit.protocols.protocol, Protocol]]*) – name to search

Returns if x is in the chain

Return type bool

__init__(*data=None*, *, *base=None*)
Initialisation.

Parameters **data** (*Optional[pcapkit.protocols.protocol.Protocol]*) – New protocol class on the top stack.

Keyword Arguments **base** (*Union[pcapkit.corekit.protochain._ProtoList, List[pcapkit.protocols.protocol.Protocol]]*) – Original protocol class chain as base stacks.

__iter__()
Iterate through the protocol chain.

Return type Iterator[*pcapkit.protocols.protocol.Protocol*]

__len__()
Length of the protocol chain.

Return type int

property data
Protocol data.

Return type List[*pcapkit.protocols.protocol.Protocol*]

1.5.3 Version Info

pcapkit.corekit.version contains tuple like class *VersionInfo*, which is originally designed alike sys.version_info.

class pcapkit.corekit.version.**VersionInfo**(*major*, *minor*)
Bases: tuple

VersionInfo is alike sys.version_info.

_asdict()
Return a new dict which maps field names to their values.

classmethod _make(*iterable*)
Make a new VersionInfo object from a sequence or iterable

_replace(**kwds*)
Return a new VersionInfo object replacing specified fields with new values

_field_defaults = {}

_fields = ('major', 'minor')

_fields_defaults = {}

major
Alias for field number 0

minor
Alias for field number 1

1.6 Dump Utilities

pcapkit.dumpkit is the collection of dumpers for *pcapkit* implementation, which is alike those described in dictdumper.

1.6.1 PCAP Dumper

1.6.2 Undefined Dumper

class pcapkit.dumpkit.**NotImplementedIO**(*fname*, ***kwargs*)
Bases: dictdumper.dumper.Dumper

Unspecified output format.

__call__(*value*, *name=None*)
Dump a new frame.

Parameters

- **value** (Dict[str, Any]) – content to be dumped
- **name** (Optional[str]) – name of current content block

Returns the dumper class itself (to support chain calling)

Return type PCAP

_append_value(*value*, *file*, *name*)
Call this function to write contents.

Parameters

- **value** (*Dict[str, Any]*) – content to be dumped
- **file** (*io.TextIOWrapper*) – output file
- **name** (*str*) – name of current content block

_dump_header(***kwargs*)
Initially dump file heads and tails.

Keyword Arguments ****kwargs** – arbitrary keyword arguments

property kind
File format of current dumper.

Return type Literal[NotImplemented]

1.7 Compatibility Tools

pcapkit.toolkit provides several utility functions for compatibility of multiple engine support.

1.7.1 Default (PyPCAPKit) Tools

pcapkit.toolkit.default contains all you need for *pcapkit* handy usage. All functions returns with a flag to indicate if usable for its caller.

pcapkit.toolkit.default.**ipv4_reassembly**(*frame*)
Make data for IPv4 reassembly.

Parameters **frame** (pcapkit.protocols.pcap.frame.Frame) – PCAP frame.

Returns

A tuple of data for IPv4 reassembly.

- If the frame can be used for IPv4 reassembly. A frame can be reassembled if it contains IPv4 layer (*pcapkit.protocols.internet.ipv4.IPv4*) and the **DF** (*IPv4.flags.df*) flag is False.
- If the frame can be reassembled, then the dict mapping of data for IPv4 reassembly (c.f. *ipv4.packet*) will be returned; otherwise, returns None.

Return type Tuple[bool, Dict[str, Any]]

See also:

IPv4Reassembly

pcapkit.toolkit.default.**ipv6_reassembly**(*frame*)
Make data for IPv6 reassembly.

Parameters **frame** (pcapkit.protocols.pcap.frame.Frame) – PCAP frame.

Returns

A tuple of data for IPv6 reassembly.

- If the frame can be used for IPv6 reassembly. A frame can be reassembled if it contains IPv6 layer (*pcapkit.protocols.internet.ipv6.IPv6*) and IPv6 Fragment header (**RFC 2460#section-4.5**, *pcapkit.protocols.internet.ipv6_frag.IPv6_Frag*).
- If the frame can be reassembled, then the dict mapping of data for IPv6 reassembly (*ipv6.packet*) will be returned; otherwise, returns None.

Return type Tuple[bool, Dict[str, Any]]

See also:

IPv6Reassembly

pcapkit.toolkit.default.**tcp_reassembly**(*frame*)
Make data for TCP reassembly.

Parameters **frame** (pcapkit.protocols.pcap.frame.Frame) – PCAP frame.

Returns

A tuple of data for TCP reassembly.

- If the `frame` can be used for TCP reassembly. A frame can be reassembled if it contains TCP layer (*pcapkit.protocols.transport.tcp.TCP*).
- If the `frame` can be reassembled, then the `dict` mapping of data for TCP reassembly (*tcp.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`TCPReassembly`

pcapkit.toolkit.default.**tcp_traceflow**(*frame*, *, *data_link*)

Trace packet flow for TCP.

Parameters **frame** (pcapkit.protocols.pcap.frame.Frame) – PCAP frame.

Keyword Arguments **data_link** (*str*) – Data link layer protocol (from global header).

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP flow tracing. A frame can be reassembled if it contains TCP layer (*pcapkit.protocols.transport.tcp.TCP*).
- If the `frame` can be reassembled, then the `dict` mapping of data for TCP flow tracing (*trace.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

TraceFlow

1.7.2 DPKT Tools

pcapkit.toolkit.dpkt contains all you need for *pcapkit* handy usage with DPKT engine. All reforming functions returns with a flag to indicate if usable for its caller.

pcapkit.toolkit.dpkt.**ipv4_reassembly**(*packet*, *, *count=NotImplemented*)

Make data for IPv4 reassembly.

Parameters **packet** (*dpkt.dpkt.Packet*) – DPKT packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for IPv4 reassembly.

- If the `packet` can be used for IPv4 reassembly. A packet can be reassembled if it contains IPv4 layer (`dpkt.ip.IP`) and the **DF** (`dpkt.ip.IP.df`) flag is `False`.
- If the `packet` can be reassembled, then the `dict` mapping of data for IPv4 reassembly (*ipv4.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`IPv4Reassembly`

pcapkit.toolkit.dpkt.**ipv6_hdr_len**(*ipv6*)

Calculate length of headers before IPv6 Fragment header.

Parameters **ipv6** (*dpkt.ip6.IP6*) – DPKT IPv6 packet.

Returns Length of headers before IPv6 Fragment header `dpkt.ip6.IP6FragmentHeader` (**RFC 2460#section-4.5**).

Return type int

As specified in **RFC 2460#section-4.1**, such headers (before the IPv6 Fragment Header) includes Hop-by-Hop Options header `dpkt.ip6.IP6HopOptsHeader` (**RFC 2460#section-4.3**), Destination Options header `dpkt.ip6.IP6DstOptHeader` (**RFC 2460#section-4.6**) and Routing header `dpkt.ip6.IP6RoutingHeader` (**RFC 2460#section-4.4**).

`pcapkit.toolkit.dpkt.`**`ipv6_reassembly`**(*packet*, *, *count=NotImplemented*)

Make data for IPv6 reassembly.

Parameters **packet** (*dpkt.dpkt.Packet*) – DPKT packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for IPv6 reassembly.

- If the `packet` can be used for IPv6 reassembly. A packet can be reassembled if it contains IPv6 layer (`dpkt.ip6.IP6`) and IPv6 Fragment header (**RFC 2460#section-4.5**, `dpkt.ip6.IP6FragmentHeader`).
- If the `packet` can be reassembled, then the `dict` mapping of data for IPv6 reassembly (*ipv6.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`IPv6Reassembly`

`pcapkit.toolkit.dpkt.`**`packet2chain`**(*packet*)

Fetch DPKT packet protocol chain.

Parameters **packet** (*dpkt.dpkt.Packet*) – DPKT packet.

Returns Colon (`:`) seperated list of protocol chain.

Return type str

`pcapkit.toolkit.dpkt.`**`packet2dict`**(*packet*, *timestamp*, *, *data_link*)

Convert DPKT packet into `dict`.

Parameters **packet** (*c*) – Scapy packet.

Returns A `dict` mapping of packet data.

Return type Dict[str, Any]

`pcapkit.toolkit.dpkt.`**`tcp_reassembly`**(*packet*, *, *count=NotImplemented*)

Make data for TCP reassembly.

Parameters **packet** (*dpkt.dpkt.Packet*) – DPKT packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP reassembly. A packet can be reassembled if it contains TCP layer (`dpkt.tcp.TCP`).
- If the `packet` can be reassembled, then the `dict` mapping of data for TCP reassembly (*tcp.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`TCPReassembly`

pcapkit.toolkit.dpkt.**tcp_traceflow**(*packet*, *timestamp*, *, *data_link*, *count=NotImplemented*)

Trace packet flow for TCP.

Parameters

- **packet** (*dpkt.dpkt.Packet*) – DPKT packet.
- **timestamp** (*float*) – Timestamp of the packet.

Keyword Arguments

- **data_link** (*str*) – Data link layer protocol (from global header).
- **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP flow tracing. A packet can be reassembled if it contains TCP layer (`dpkt.tcp.TCP`).
- If the `packet` can be reassembled, then the `dict` mapping of data for TCP flow tracing (*trace.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

TraceFlow

1.7.3 PyShark Tools

pcapkit.toolkit.pyshark contains all you need for *pcapkit* handy usage with PyShark engine. All reforming functions returns with a flag to indicate if usable for its caller.

pcapkit.toolkit.pyshark.**packet2dict**(*packet*)

Convert PyShark packet into `dict`.

Parameters **packet** (*pyshark.packet.packet.Packet*) – Scapy packet.

Returns A `dict` mapping of packet data.

Return type Dict[str, Any]

pcapkit.toolkit.pyshark.**tcp_traceflow**(*packet*)

Trace packet flow for TCP.

Parameters **packet** (*pyshark.packet.packet.Packet*) – Scapy packet.

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP flow tracing. A packet can be reassembled if it contains TCP layer.
- If the `packet` can be reassembled, then the `dict` mapping of data for TCP flow tracing (*trace.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

TraceFlow

1.7.4 Scapy Tools

pcapkit.toolkit.scapy contains all you need for *pcapkit* handy usage with Scapy engine. All reforming functions returns with a flag to indicate if usable for its caller.

pcapkit.toolkit.scapy.**ipv4_reassembly**(*packet*, *, *count=NotImplemented*)

Make data for IPv4 reassembly.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for IPv4 reassembly.

- If the `packet` can be used for IPv4 reassembly. A packet can be reassembled if it contains IPv4 layer (`scapy.layers.inet.IP`) and the **DF** (`scapy.layers.inet.IP.flags.DF`) flag is `False`.
- If the `packet` can be reassembled, then the `dict` mapping of data for IPv4 reassembly (*ipv4.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`IPv4Reassembly`

pcapkit.toolkit.scapy.**ipv6_reassembly**(*packet*, *, *count=NotImplemented*)

Make data for IPv6 reassembly.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for IPv6 reassembly.

- If the `packet` can be used for IPv6 reassembly. A packet can be reassembled if it contains IPv6 layer (`scapy.layers.inet6.IPv6`) and IPv6 Fragment header (**RFC 2460#section-4.5**, `scapy.layers.inet6.IPv6ExtHdrFragment`).
- If the `packet` can be reassembled, then the `dict` mapping of data for IPv6 reassembly (*ipv6.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

Raises ***ModuleNotFound*** – If Scapy is not installed.

See also:

`IPv6Reassembly`

pcapkit.toolkit.scapy.**packet2chain**(*packet*)
Fetch Scapy packet protocol chain.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Returns Colon (`:`) seperated list of protocol chain.

Return type str

Raises ***ModuleNotFound*** – If Scapy is not installed.

pcapkit.toolkit.scapy.**packet2dict**(*packet*)
Convert Scapy packet into `dict`.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Returns A `dict` mapping of packet data.

Return type Dict[str, Any]

Raises ***ModuleNotFound*** – If Scapy is not installed.

pcapkit.toolkit.scapy.**tcp_reassembly**(*packet*, *, *count=NotImplemented*)
Store data for TCP reassembly.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP reassembly. A packet can be reassembled if it contains TCP layer (`scapy.layers.inet.TCP`).
- If the `packet` can be reassembled, then the `dict` mapping of data for TCP reassembly (*tcp.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

`TCPReassembly`

pcapkit.toolkit.scapy.**tcp_traceflow**(*packet*, *, *count=NotImplemented*)
Trace packet flow for TCP.

Parameters **packet** (*scapy.packet.Packet*) – Scapy packet.

Keyword Arguments **count** (*int*) – Packet index. If not provided, default to `NotImplemented`.

Returns

A tuple of data for TCP reassembly.

- If the `packet` can be used for TCP flow tracing. A packet can be reassembled if it contains TCP layer (`scapy.layers.inet.TCP`).
- If the `packet` can be reassembled, then the `dict` mapping of data for TCP flow tracing (*trace.packet*) will be returned; otherwise, returns `None`.

Return type Tuple[bool, Dict[str, Any]]

See also:

TraceFlow

1.8 Utility Functions & Classes

pcapkit.utilities contains several useful functions and classes which are fundations of *pcapkit*, including decorater function *seekset()* and *beholder()*, and several user-refined exceptions and validations.

1.8.1 Decorator Functions

pcapkit.utilities.decorators contains several useful decorators, including *seekset()* and *beholder()*.

@pcapkit.utilities.decorators.**seekset**
Read file from start then set back to original.

Important: This decorator function is designed for decorating *class methods*.

The decorator will keep the current offset of `self._file`, then call the decorated function. Afterwards, it will rewind the offset of `self._file` to the original and returns the return value from the decorated function.

Note: The decorated function should have following signature:

```
func(self, *args, **kw)
```

See also:

pcapkit.protocols.protocol.Protocol._read_packet()

@pcapkit.utilities.decorators.**seekset_ng**
Read file from start then set back to original.

Important: This decorator function is designed for decorating *plain functions*.

The decorator will rewind the offset of `file` to `seekset`, then call the decorated function and returns its return value.

Note: The decorated function should have following signature:

```
func(protocol, file, *args, seekset=os.SEEK_SET, **kw)
```

c.f. *pcapkit.foundation.analysis._analyse()*.

See also:

pcapkit.foundation.analysis

@pcapkit.utilities.decorators.**beholder**
Behold extraction procedure.

Important: This decorator function is designed for decorating *class methods*.

This decorate first keep the current offset of `self._file`, then try to call the decorated function. Should any exception raised, it will re-parse the `self._file` as *Raw* protocol.

Note: The decorated function should have following signature:

```
func(self, proto, length, *args, **kwargs)
```

See also:

pcapkit.protocols.protocol.Protocol._decode_next_layer()

@pcapkit.utilities.decorators.**beholder_ng**
Behold analysis procedure.

Important: This decorator function is designed for decorating *plain functions*.

This decorate first keep the current offset of `file`, then try to call the decorated function. Should any exception raised, it will re-parse the `file` as *Raw* protocol.

Note: The decorated function should have following signature:

```
func(file, length, *args, **kwargs)
```

See also:

pcapkit.protocols.transport.transport.Transport._import_next_layer()

Important: *pcapkit.utilities.decorators.seekset()* and *pcapkit.utilities.decorators.beholder()* are designed for decorating *class methods*.

1.8.2 User Defined Exceptions

`pcapkit.exceptions` refined built-in exceptions. Make it possible to show only user error stack infomation*[0], when exception raised on user's operation.

exception pcapkit.utilities.exceptions.**BaseError**(**args*, *quiet=False*, ***kwargs*)
Bases: `Exception`

Base error class of all kinds.

Important:

- Turn off system-default traceback function by set `sys.tracebacklimit` to `0`.
- But bugs appear in Python 3.6, so we have to set `sys.tracebacklimit` to `None`.

  Note: This note is deprecated since Python fixed the problem above.

- In Python 2.7, `trace.print_stack(limit)()` dose not support negative limit.

See also:

pcapkit.utilities.exceptions.stacklevel()

__init__(**args*, *quiet=False*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

exception pcapkit.utilities.exceptions.**BoolError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

The argument(s) must be `bool` type.

exception pcapkit.utilities.exceptions.**BytearrayError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

The argument(s) must be `bytearray` type.

exception pcapkit.utilities.exceptions.**BytesError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

The argument(s) must be `bytes` type.

exception pcapkit.utilities.exceptions.**CallableError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

The argument(s) must be *callable*.

exception pcapkit.utilities.exceptions.**ComparisonError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

Rich comparison not supported between instances.

exception pcapkit.utilities.exceptions.**ComplexError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, `TypeError`

The function is not defined for complex instance.

[0] See `tbtrim` project for a modern Pythonic implementation.

exception pcapkit.utilities.exceptions.**DictError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be dict type.

exception pcapkit.utilities.exceptions.**DigitError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be (a) number(s).

exception pcapkit.utilities.exceptions.**EndianError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, ValueError

Invalid endian (byte order).

exception pcapkit.utilities.exceptions.**EnumError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be *enumeration protocol* type.

exception pcapkit.utilities.exceptions.**FileError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, OSError

[Errno 5] Wrong file format.

exception pcapkit.utilities.exceptions.**FileExists**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, FileExistsError

[Errno 17] File already exists.

exception pcapkit.utilities.exceptions.**FileNotFound**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, FileNotFoundError

[Errno 2] File not found.

exception pcapkit.utilities.exceptions.**FormatError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, AttributeError

Unknown format(s).

exception pcapkit.utilities.exceptions.**FragmentError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, KeyError

Invalid fragment dict.

exception pcapkit.utilities.exceptions.**IOObjError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be *file-like object*.

exception pcapkit.utilities.exceptions.**IPError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be *IP address*.

exception pcapkit.utilities.exceptions.**IndexNotFound**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, ValueError

Protocol not in ProtoChain.

exception pcapkit.utilities.exceptions.**InfoError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be *Info* instance.

exception pcapkit.utilities.exceptions.**IntError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be integral.

exception pcapkit.utilities.exceptions.**IterableError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be *iterable*.

exception pcapkit.utilities.exceptions.**ListError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be `list` type.

exception pcapkit.utilities.exceptions.**ModuleNotFound**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, ModuleNotFoundError

Module not found.

exception pcapkit.utilities.exceptions.**PacketError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, KeyError

Invalid packet dict.

exception pcapkit.utilities.exceptions.**ProtocolError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, ValueError

Invalid protocol format.

exception pcapkit.utilities.exceptions.**ProtocolNotFound**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, IndexError

Protocol not found in ProtoChain.

exception pcapkit.utilities.exceptions.**ProtocolNotImplemented**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, NotImplementedError

Protocol not implemented.

exception pcapkit.utilities.exceptions.**ProtocolUnbound**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

Protocol slice unbound.

exception pcapkit.utilities.exceptions.**RealError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The function is not defined for real number.

exception pcapkit.utilities.exceptions.**StringError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be `str` type.

exception pcapkit.utilities.exceptions.**StructError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, struct.error

Unpack failed.

exception pcapkit.utilities.exceptions.**TupleError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, TypeError

The argument(s) must be tuple type.

exception pcapkit.utilities.exceptions.**UnsupportedCall**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, AttributeError

Unsupported function or property call.

exception pcapkit.utilities.exceptions.**VendorNotImplemented**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, NotImplementedError

Vendor not implemented.

exception pcapkit.utilities.exceptions.**VersionError**(**args*, *quiet=False*, ***kwargs*)
Bases: *pcapkit.utilities.exceptions.BaseError*, ValueError

Unknown IP version.

pcapkit.utilities.exceptions.**stacklevel**()
Fetch current stack level.

The function will walk through the straceback stack (traceback.extract_stack()), and fetch the stack level where the path contains /pcapkit/. So that it won't display any disturbing internal traceback information when raising errors.

Returns Stack level until internal stacks, i.e. contains /pcapkit/.

Return type int

pcapkit.utilities.exceptions.**DEVMODE = False**
Development mode (DEVMODE) flag.

1.8.3 Logging System

pcapkit.utilities.logging contains naïve integration of the Python logging system, i.e. a logging.Logger instance as *logger*.

pcapkit.utilities.logging.**logger = <Logger pcapkit (WARNING)>**
Logger instance named after pcapkit.

Type logging.Logger

1.8.4 Validation Utilities

pcapkit.utilities.validations contains functions to validate arguments for functions and classes. It was first used in PyNTLib as validators.

pcapkit.utilities.validations.**_ip_frag_check**(**args*, *stacklevel=3*)
Check if arguments are valid IP fragments (*IPv4* and/or *IPv6* packet).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

See also:

- *pcapkit.toolkit.default.ipv4_reassembly()*
- *pcapkit.toolkit.default.ipv6_reassembly()*

pcapkit.utilities.validations.**_tcp_frag_check**(**args*, *stacklevel=3*)
Check if arguments are valid TCP fragments (*TCP packet*).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

See also:

pcapkit.toolkit.default.tcp_reassembly()

pcapkit.utilities.validations.**bool_check**(**args*, *stacklevel=2*)
Check if arguments are `bool` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***BoolError*** – If any of the arguments is **NOT** `bool` type.

pcapkit.utilities.validations.**bytearray_check**(**args*, *stacklevel=2*)
Check if arguments are `bytearray` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***BytearrayError*** – If any of the arguments is **NOT** `bytearray` type.

pcapkit.utilities.validations.**bytes_check**(**args*, *stacklevel=2*)
Check if arguments are `bytes` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***BytesError*** – If any of the arguments is **NOT** `bytes` type.

pcapkit.utilities.validations.**complex_check**(**args*, *stacklevel=2*)
Check if arguments are *complex numbers* (`complex`).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***ComplexError*** – If any of the arguments is **NOT** *complex number* (`complex`).

pcapkit.utilities.validations.**dict_check**(**args*, *stacklevel=2*)
Check if arguments are `dict` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***DictError*** – If any of the arguments is **NOT** `dict` type.

`pcapkit.utilities.validations.`**`enum_check`**(**args*, *stacklevel=2*)
Check if arguments are of *enumeration protocol* type (`enum.EnumMeta` and/or `aenum.EnumMeta`).

Parameters

- **`*args`** – Arguments to check.
- **`stacklevel`** (*int*) – Stack level to fetch originated function name.

Raises ***EnumError*** – If any of the arguments is **NOT** *enumeration protocol* type (`enum.EnumMeta` and/or `aenum.EnumMeta`).

`pcapkit.utilities.validations.`**`frag_check`**(**args*, *protocol*, *stacklevel=3*)
Check if arguments are valid fragments.

Parameters

- **`*args`** – Arguments to check.
- **`protocol`** (*str*) – Originated fragmentation protocol (IPv4, IPv6 or TCP).
- **`stacklevel`** (*int*) – Stack level to fetch originated function name.

- If the protocol is IPv4, the fragment should be as an IPv4 *fragmentation*.
- If the protocol is IPv6, the fragment should be as an IPv6 *fragmentation*.
- If the protocol is TCP, the fragment should be as an TCP *fragmentation*.

Raises ***FragmentError*** – If any of the arguments is **NOT** valid fragment.

See also:

- *pcapkit.utilities.validations._ip_frag_check()*
- *pcapkit.utilities.validations._tcp_frag_check()*

`pcapkit.utilities.validations.`**`info_check`**(**args*, *stacklevel=2*)
Check if arguments are *Info* instances.

Parameters

- **`*args`** – Arguments to check.
- **`stacklevel`** (*int*) – Stack level to fetch originated function name.

Raises ***InfoError*** – If any of the arguments is **NOT** *Info* instance.

`pcapkit.utilities.validations.`**`int_check`**(**args*, *stacklevel=2*)
Check if arguments are *integrals* (`int`).

Parameters

- **`*args`** – Arguments to check.
- **`stacklevel`** (*int*) – Stack level to fetch originated function name.

Raises ***IntError*** – If any of the arguments is **NOT** *integral* (`int`).

`pcapkit.utilities.validations.`**`io_check`**(**args*, *stacklevel=2*)
Check if arguments are *file-like object* (`io.IOBase`).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***IOObjError*** – If any of the arguments is **NOT** *file-like object* (`io.IOBase`).

pcapkit.utilities.validations.**ip_check**(**args*, *stacklevel=2*)
Check if arguments are *IP addresses* (`ipaddress.IPv4Address` and/or `ipaddress.IPv6Address`).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***IPError*** – If any of the arguments is **NOT** *IP address* (`ipaddress.IPv4Address` and/or `ipaddress.IPv6Address`).

pcapkit.utilities.validations.**list_check**(**args*, *stacklevel=2*)
Check if arguments are `list` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***ListError*** – If any of the arguments is **NOT** `list` type.

pcapkit.utilities.validations.**number_check**(**args*, *stacklevel=2*)
Check if arguments are *numbers*.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***DigitError*** – If any of the arguments is **NOT** *number* (`int`, `float` and/or `complex`).

pcapkit.utilities.validations.**pkt_check**(**args*, *stacklevel=3*)
Check if arguments are valid packets (*TCP packet*).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***PacketError*** – If any of the arguments is **NOT** valid packet.

See also:

pcapkit.toolkit.default.tcp_traceflow()

pcapkit.utilities.validations.**real_check**(**args*, *stacklevel=2*)
Check if arguments are *real numbers* (`int` and/or `float`).

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***RealError*** – If any of the arguments is **NOT** *real number* (`int` and/or `float`).

pcapkit.utilities.validations.**str_check**(**args*, *stacklevel=2*)
Check if arguments are `str` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***StringError*** – If any of the arguments is **NOT** `str` type.

pcapkit.utilities.validations.**tuple_check**(**args*, *stacklevel=2*)
Check if arguments are `tuple` type.

Parameters

- ***args** – Arguments to check.
- **stacklevel** (*int*) – Stack level to fetch originated function name.

Raises ***TupleError*** – If any of the arguments is **NOT** `tuple` type.

1.8.5 User Defined Warnings

`pcapkit.warnings` refined built-in warnings.

exception pcapkit.utilities.warnings.**AttributeWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `RuntimeWarning`

Unsupported attribute.

exception pcapkit.utilities.warnings.**BaseWarning**(**args*, ***kwargs*)
Bases: `UserWarning`

Base warning class of all kinds.

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

exception pcapkit.utilities.warnings.**DPKTWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `ResourceWarning`

Warnings on DPKT usage.

exception pcapkit.utilities.warnings.**DevModeWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `RuntimeWarning`

Run in development mode.

exception pcapkit.utilities.warnings.**EngineWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `ImportWarning`

Unsupported extraction engine.

exception pcapkit.utilities.warnings.**FileWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `RuntimeWarning`

Warning on file(s).

exception pcapkit.utilities.warnings.**FormatWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `ImportWarning`

Warning on unknown format(s).

exception pcapkit.utilities.warnings.**InvalidVendorWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, `ImportWarning`

Vendor CLI invalid updater.

exception pcapkit.utilities.warnings.**LayerWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, RuntimeWarning

Unrecognised layer.

exception pcapkit.utilities.warnings.**ProtocolWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, RuntimeWarning

Unrecognised protocol.

exception pcapkit.utilities.warnings.**PySharkWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, ResourceWarning

Warnings on PyShark usage.

exception pcapkit.utilities.warnings.**ScapyWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, ResourceWarning

Warnings on Scapy usage.

exception pcapkit.utilities.warnings.**VendorRequestWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, RuntimeWarning

Vendor request connection failed.

exception pcapkit.utilities.warnings.**VendorRuntimeWarning**(**args*, ***kwargs*)
Bases: *pcapkit.utilities.warnings.BaseWarning*, RuntimeWarning

Vendor failed during runtime.

pcapkit.utilities.warnings.**warn**(*message*, *category*, *stacklevel=None*)
Wrapper function of warnings.warn().

Parameters

- **message** (*Union[str, Warning]*) – Warning message.
- **category** (*Type[Warning]*) – Warning category.
- **stacklevel** (*Optional[int]*) – Warning stack level.

1.9 Constant Enumerations

1.9.1 ARP Constant Enumerations

ARP Hardware Types*[0]

*

class pcapkit.const.arp.hardware.**Hardware**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)
Bases: aenum.IntEnum

[Hardware] Hardware Types [RFC 826][RFC 5494]

classmethod **_missing_**(*value*)
Lookup function used when value is not found.

static **get**(*key*, *default=- 1*)
Backport support for original codes.

[0] https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-2

AEthernet = 257
AEthernet [Geoffroy Gramaize]

ARCNET = 7
ARCNET [RFC 1201]

ARPSec = 30
ARPSec [Jerome Etienne]

Amateur_Radio_AX_25 = 3
Amateur Radio AX.25 [Philip Koch]

Asynchronous_Transmission_Mode_16 = 16
Asynchronous Transmission Mode (ATM) [JXB2]

Asynchronous_Transmission_Mode_19 = 19
Asynchronous Transmission Mode (ATM) [RFC 2225]

Asynchronous_Transmission_Mode_21 = 21
Asynchronous Transmission Mode (ATM) [Mike Burrows]

Autonet_Short_Address = 10
Autonet Short Address [Mike Burrows]

Chaos = 5
Chaos [Gill Pratt]

EUI_64 = 27
EUI-64 [Kenji Fujisawa]

Ethernet = 1
Ethernet (10Mb) [Jon Postel]

Experimental_Ethernet = 2
Experimental Ethernet (3Mb) [Jon Postel]

Fibre_Channel = 18
Fibre Channel [RFC 4338]

Frame_Relay = 15
Frame Relay [Andy Malis]

HDLC = 17
HDLC [Jon Postel]

HFI = 37
HFI [Tseng-Hui Lin]

HIPARP = 28
HIPARP [Jean Michel Pittet]

HW_EXP1 = 36
HW_EXP1 [RFC 5494]

HW_EXP2 = 256
HW_EXP2 [RFC 5494]

Hyperchannel = 8
Hyperchannel [Jon Postel]

IEEE_1394_1995 = 24
IEEE 1394.1995 [Myron Hattig]

IEEE_802_Networks = 6
IEEE 802 Networks [Jon Postel]

IP_and_ARP_over_ISO_7816_3 = 29
IP and ARP over ISO 7816-3 [Scott Guthery]

IPsec_tunnel = 31
IPsec tunnel [**RFC 3456**]

InfiniBand = 32
InfiniBand (TM) [**RFC 4391**]

Lanstar = 9
Lanstar [Tom Unger]

LocalNet = 12
LocalNet (IBM PCNet or SYTEK LocalNET) [Joseph Murdock]

LocalTalk = 11
LocalTalk [Joyce K Reynolds]

MAPOS = 25
MAPOS [Mitsuru Maruyama][**RFC 2176**]

MIL_STD_188_220 = 22
MIL-STD-188-220 [Herb Jensen]

Metricom = 23
Metricom [Jonathan Stone]

Proteon_ProNET_Token_Ring = 4
Proteon ProNET Token Ring [Avri Doria]

Pure_IP = 35
Pure IP [Inaky Perez-Gonzalez]

Reserved_0 = 0
Reserved [**RFC 5494**]

Reserved_65535 = 65535
Reserved [**RFC 5494**]

SMDS = 14
SMDS [George Clapp]

Serial_Line = 20
Serial Line [Jon Postel]

TIA_102_Project_25_Common_Air_Interface = 33
TIA-102 Project 25 Common Air Interface (CAI) [Jeff Anderson, Telecommunications Industry of America (TIA) TR-8.5 Formulating Group, <cja015&motorola.com>, June 2004]

Twinaxial = 26
Twinaxial [Marion Pitts]

Ultra_link = 13
Ultra link [Rajiv Dhingra]

Wiegand_Interface = 34
Wiegand Interface [Scott Guthery 2]

Operation Codes†[0]

†

class pcapkit.const.arp.operation.**Operation**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Operation] Operation Codes [RFC 826][RFC 5494]

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ARP_NAK = 10
ARP-NAK [RFC 1577]

DRARP_Error = 7
DRARP-Error [RFC 1931]

DRARP_Reply = 6
DRARP-Reply [RFC 1931]

DRARP_Request = 5
DRARP-Request [RFC 1931]

InARP_Reply = 9
InARP-Reply [RFC 2390]

InARP_Request = 8
InARP-Request [RFC 2390]

MAPOS_UNARP = 23
MAPOS-UNARP [Mitsuru Maruyama][RFC 2176]

MARS_Grouplist_Reply = 21
MARS-Grouplist-Reply [Grenville Armitage]

MARS_Grouplist_Request = 20
MARS-Grouplist-Request [Grenville Armitage]

MARS_Join = 14
MARS-Join [Grenville Armitage]

MARS_Leave = 15
MARS-Leave [Grenville Armitage]

MARS_MServ = 13
MARS-MServ [Grenville Armitage]

MARS_Multi = 12
MARS-Multi [Grenville Armitage]

MARS_NAK = 16
MARS-NAK [Grenville Armitage]

MARS_Redirect_Map = 22
MARS-Redirect-Map [Grenville Armitage]

MARS_Request = 11
MARS-Request [Grenville Armitage]

[0] https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-1

MARS_SJoin = 18
MARS-SJoin [Grenville Armitage]

MARS_SLeave = 19
MARS-SLeave [Grenville Armitage]

MARS_Unserv = 17
MARS-Unserv [Grenville Armitage]

OP_EXP1 = 24
OP_EXP1 [RFC 5494]

OP_EXP2 = 25
OP_EXP2 [RFC 5494]

REPLY = 2
REPLY [RFC 826][RFC 5227]

REQUEST = 1
REQUEST [RFC 826][RFC 5227]

Reserved_0 = 0
Reserved [RFC 5494]

Reserved_65535 = 65535
Reserved [RFC 5494]

reply_Reverse = 4
reply Reverse [RFC 903]

request_Reverse = 3
request Reverse [RFC 903]

1.9.2 FTP Constant Enumerations

FTP Commands*[0]

*

class pcapkit.const.ftp.command.**defaultInfo**(*dict_=None*, ***kwargs*)
Bases: *pcapkit.corekit.infoclass.Info*

Extended *Info* with default values.

__getitem__(*key*)
Missing keys as specified in RFC 3659.

pcapkit.const.ftp.command.**Command = Info(ABOR=Info(...), ACCT=Info(...), ADAT=Info(...), Al**
FTP Command

[0] https://www.iana.org/assignments/ftp-commands-extensions/ftp-commands-extensions.xhtml#ftp-commands-extensions-2

FTP Return Codes†[0]

†

class pcapkit.const.ftp.return_code.**ReturnCode**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ReturnCode] FTP Server Return Code

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CODE_110 = 110
Restart marker replay. In this case, the text is exact and not left to the particular implementation; it must read: MARK yyyy = mmmm where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").

CODE_120 = 120
Service ready in nnn minutes.

CODE_125 = 125
Data connection already open; transfer starting.

CODE_150 = 150
File status okay; about to open data connection.

CODE_202 = 202
Command not implemented, superfluous at this site.

CODE_211 = 211
System status, or system help reply.

CODE_212 = 212
Directory status.

CODE_213 = 213
File status.

CODE_214 = 214
Help message. Explains how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.

CODE_215 = 215
NAME system type. Where NAME is an official system name from the registry kept by IANA.

CODE_220 = 220
Service ready for new user.

CODE_221 = 221
Service closing control connection.

CODE_225 = 225
Data connection open; no transfer in progress.

CODE_226 = 226
Closing data connection. Requested file action successful (for example, file transfer or file abort).

[0] https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes

CODE_227 = 227
Entering Passive Mode (h1,h2,h3,h4,p1,p2).

CODE_228 = 228
Entering Long Passive Mode (long address, port).

CODE_229 = 229
Entering Extended Passive Mode (|||port|).

CODE_230 = 230
User logged in, proceed. Logged out if appropriate.

CODE_231 = 231
User logged out; service terminated.

CODE_232 = 232
Logout command noted, will complete when transfer done.

CODE_234 = 234
Specifies that the server accepts the authentication mechanism specified by the client, and the exchange of security data is complete. A higher level nonstandard code created by Microsoft.

CODE_250 = 250
Requested file action okay, completed.

CODE_257 = 257
"PATHNAME" created.

CODE_331 = 331
User name okay, need password.

CODE_332 = 332
Need account for login.

CODE_350 = 350
Requested file action pending further information

CODE_421 = 421
Service not available, closing control connection. This may be a reply to any command if the service knows it must shut down.

CODE_425 = 425
Can't open data connection.

CODE_426 = 426
Connection closed; transfer aborted.

CODE_430 = 430
Invalid username or password

CODE_434 = 434
Requested host unavailable.

CODE_450 = 450
Requested file action not taken.

CODE_451 = 451
Requested action aborted. Local error in processing.

CODE_452 = 452
Requested action not taken. Insufficient storage space in system. File unavailable (e. g. , file busy).

CODE_501 = 501
Syntax error in parameters or arguments.

CODE_502 = 502
Command not implemented.

CODE_503 = 503
Bad sequence of commands.

CODE_504 = 504
Command not implemented for that parameter.

CODE_530 = 530
Not logged in.

CODE_532 = 532
Need account for storing files.

CODE_534 = 534
Could Not Connect to Server - Policy Requires SSL

CODE_550 = 550
Requested action not taken. File unavailable (e. g. , file not found, no access).

CODE_551 = 551
Requested action aborted. Page type unknown.

CODE_552 = 552
Requested file action aborted. Exceeded storage allocation (for current directory or dataset).

CODE_553 = 553
Requested action not taken. File name not allowed.

CODE_631 = 631
Integrity protected reply.

CODE_632 = 632
Confidentiality and integrity protected reply.

CODE_633 = 633
Confidentiality protected reply.

pcapkit.const.ftp.return_code.**INFO = {'0': 'Syntax', '1': 'Information', '2': 'Connectio**
Grouping information.

pcapkit.const.ftp.return_code.**KIND = {'1': 'Positive Preliminary', '2': 'Positive Complet**
Response kind; whether the response is good, bad or incomplete.

1.9.3 HIP Constant Enumerations

HIP Certificate Types*[0]

*

class pcapkit.const.hip.certificate.**Certificate**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)
Bases: aenum.IntEnum

[Certificate] HIP Certificate Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#certificate-types

static get(*key*, *default=- 1*)
Backport support for original codes.

Distinguished_Name_of_X_509_v3 = 7
Distinguished Name of X.509 v3 [RFC 8002]

Hash_and_URL_of_X_509_v3 = 3
Hash and URL of X.509 v3 [RFC 8002]

LDAP_URL_of_X_509_v3 = 5
LDAP URL of X.509 v3 [RFC 8002]

Obsoleted_2 = 2
Obsoleted [RFC 8002]

Obsoleted_4 = 4
Obsoleted [RFC 8002]

Obsoleted_6 = 6
Obsoleted [RFC 8002]

Obsoleted_8 = 8
Obsoleted [RFC 8002]

Reserved = 0
Reserved [RFC 8002]

X_509_v3 = 1
X.509 v3 [RFC 8002]

HIP Cipher IDs†[0]

†

class pcapkit.const.hip.cipher.**Cipher**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)
Bases: aenum.IntEnum

[Cipher] Cipher IDs

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AES_128_CBC = 2
AES-128-CBC [RFC 7401]

AES_256_CBC = 4
AES-256-CBC [RFC 7401]

NULL_ENCRYPT = 1
NULL-ENCRYPT [RFC 7401]

RESERVED_0 = 0
RESERVED [RFC 7401]

RESERVED_3 = 3
RESERVED [RFC 7401]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-cipher-id

DI-Types‡[0]

‡

class pcapkit.const.hip.di.**DITypes**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[DITypes] DI-Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

FQDN = 1
FQDN [RFC 7401]

NAI = 2
NAI [RFC 7401]

none_included = 0
none included [RFC 7401]

ECDSA Curve Label§[0]

class pcapkit.const.hip.ecdsa_curve.**ECDSACurve**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ECDSACurve] ECDSA Curve Label

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

NIST_P_256 = 1
NIST P-256 [RFC 7401]

NIST_P_384 = 2
NIST P-384 [RFC 7401]

RESERVED = 0
RESERVED [RFC 7401]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-7
[159] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#ecdsa-curve-label

ECDSA_LOW Curve Label¶[0]

¶

class pcapkit.const.hip.ecdsa_low_curve.**ECDSALowCurve**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ECDSALowCurve] ECDSA_LOW Curve Label

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

RESERVED = 0
RESERVED [RFC 7401]

SECP160R1 = 1
SECP160R1 [RFC 7401]

ESP Transform Suite IDs[35]

class pcapkit.const.hip.esp_transform_suite.**ESPTransformSuite**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ESPTransformSuite] ESP Transform Suite IDs

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AES_128_CBC_with_HMAC_SHA1 = 1
AES-128-CBC with HMAC-SHA1 [RFC 3602][RFC 2404]

AES_128_CBC_with_HMAC_SHA_256 = 8
AES-128-CBC with HMAC-SHA-256 [RFC 3602][RFC 4868]

AES_256_CBC_with_HMAC_SHA_256 = 9
AES-256-CBC with HMAC-SHA-256 [RFC 3602][RFC 4868]

AES_CCM_16 = 11
AES-CCM-16 [RFC 4309]

AES_CCM_8 = 10
AES-CCM-8 [RFC 4309]

AES_CMAC_96 = 14
AES-CMAC-96 [RFC 4493][RFC 4494]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#ecdsa-low-curve-label
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#esp-transform-suite-ids

AES_GCM_with_a_16_octet_ICV = 13
AES-GCM with a 16 octet ICV [RFC 4106]

AES_GCM_with_an_8_octet_ICV = 12
AES-GCM with an 8 octet ICV [RFC 4106]

AES_GMAC = 15
AES-GMAC [RFC 4543]

DEPRECATED_2 = 2
DEPRECATED [RFC 7402]

DEPRECATED_3 = 3
DEPRECATED [RFC 7402]

DEPRECATED_4 = 4
DEPRECATED [RFC 7402]

DEPRECATED_5 = 5
DEPRECATED [RFC 7402]

DEPRECATED_6 = 6
DEPRECATED [RFC 7402]

NULL_with_HMAC_SHA_256 = 7
NULL with HMAC-SHA-256 [RFC 2410][RFC 4868]

RESERVED = 0
RESERVED [RFC 7402]

Group IDs[0]

class pcapkit.const.hip.group.**Group**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Group] Group IDs

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Group_1536_bit_MODP_group = 3
1536-bit MODP group [RFC 7401]

Group_2048_bit_MODP_group = 11
2048-bit MODP group [RFC 7401]

Group_3072_bit_MODP_group = 4
3072-bit MODP group [RFC 7401]

Group_384_bit_group = 1
384-bit group [RFC 5201] DEPRECATED

Group_6144_bit_MODP_group = 5
6144-bit MODP group [RFC 5201] DEPRECATED

Group_8192_bit_MODP_group = 6
8192-bit MODP group [RFC 5201] DEPRECATED

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-5

NIST_P_256 = 7
NIST P-256 [RFC 7401]

NIST_P_384 = 8
NIST P-384 [RFC 7401]

NIST_P_521 = 9
NIST P-521 [RFC 7401]

OAKLEY_well_known_group_1 = 2
OAKLEY well known group 1 [RFC 5201] DEPRECATED

Reserved = 0
Reserved [RFC 7401]

SECP160R1 = 10
SECP160R1 [RFC 7401]

HI Algorithm[0]

class pcapkit.const.hip.hi_algorithm.**HIAlgorithm**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[HIAlgorithm] HI Algorithm

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

DSA = 3
DSA [RFC 7401]

ECDSA = 7
ECDSA [RFC 7401]

ECDSA_LOW = 9
ECDSA_LOW [RFC 7401]

NULL_ENCRYPT = 1
NULL-ENCRYPT [RFC 2410]

RESERVED = 0
RESERVED [RFC 7401]

RSA = 5
RSA [RFC 7401]

Unassigned_2 = 2
Unassigned

Unassigned_4 = 4
Unassigned

Unassigned_6 = 6
Unassigned

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hi-algorithm

Unassigned_8 = 8
Unassigned

HIT Suite ID[0]

class pcapkit.const.hip.hit_suite.**HITSuite**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[HITSuite] HIT Suite ID

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ECDSA_LOW_SHA_1 = 3
ECDSA_LOW/SHA-1 [RFC 7401]

ECDSA_SHA_384 = 2
ECDSA/SHA-384 [RFC 7401]

RESERVED = 0
RESERVED [RFC 7401]

RSA_DSA_SHA_256 = 1
RSA,DSA/SHA-256 [RFC 7401]

HIP NAT Traversal Modes[0]

class pcapkit.const.hip.nat_traversal.**NATTraversal**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[NATTraversal] HIP NAT Traversal Modes

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ICE_STUN_UDP = 2
ICE-STUN-UDP [RFC 5770]

Reserved = 0
Reserved [RFC 5770]

UDP_ENCAPSULATION = 1
UDP-ENCAPSULATION [RFC 5770]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hit-suite-id
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#nat-traversal

Notify Message Types**[0]

**

class pcapkit.const.hip.notify_message.**NotifyMessage**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[NotifyMessage] Notify Message Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AUTHENTICATION_FAILED = 24
AUTHENTICATION_FAILED [RFC 7401]

BLOCKED_BY_POLICY = 42
BLOCKED_BY_POLICY [RFC 7401]

CHECKSUM_FAILED = 26
CHECKSUM_FAILED [RFC 7401]

CONNECTIVITY_CHECKS_FAILED = 61
CONNECTIVITY_CHECKS_FAILED [RFC 5770]

CREDENTIALS_REQUIRED = 48
CREDENTIALS_REQUIRED [RFC 8002]

ENCRYPTION_FAILED = 32
ENCRYPTION_FAILED [RFC 7401]

HIP_MAC_FAILED = 28
HIP_MAC_FAILED [RFC 7401]

I2_ACKNOWLEDGEMENT = 16384
I2_ACKNOWLEDGEMENT [RFC 7401]

INVALID_CERTIFICATE = 50
INVALID_CERTIFICATE [RFC 8002]

INVALID_DH_CHOSEN = 15
INVALID_DH_CHOSEN [RFC 7401]

INVALID_ESP_TRANSFORM_CHOSEN = 19
INVALID_ESP_TRANSFORM_CHOSEN [RFC 7402]

INVALID_HIP_CIPHER_CHOSEN = 17
INVALID_HIP_CIPHER_CHOSEN [RFC 7401]

INVALID_HIT = 40
INVALID_HIT [RFC 7401]

INVALID_SYNTAX = 7
INVALID_SYNTAX [RFC 7401]

LOCATOR_TYPE_UNSUPPORTED = 46
LOCATOR_TYPE_UNSUPPORTED [RFC 8046]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-9

MESSAGE_NOT_RELAYED = 62
MESSAGE_NOT_RELAYED [RFC 5770]

NO_DH_PROPOSAL_CHOSEN = 14
NO_DH_PROPOSAL_CHOSEN [RFC 7401]

NO_ESP_PROPOSAL_CHOSEN = 18
NO_ESP_PROPOSAL_CHOSEN [RFC 7402]

NO_HIP_PROPOSAL_CHOSEN = 16
NO_HIP_PROPOSAL_CHOSEN [RFC 7401]

NO_VALID_HIP_TRANSPORT_MODE = 100
NO_VALID_HIP_TRANSPORT_MODE [RFC 6261]

NO_VALID_NAT_TRAVERSAL_MODE_PARAMETER = 60
NO_VALID_NAT_TRAVERSAL_MODE_PARAMETER [RFC 5770]

OVERLAY_TTL_EXCEEDED = 70
OVERLAY_TTL_EXCEEDED [RFC 6079]

REG_REQUIRED = 51
REG_REQUIRED [RFC 8003]

RESPONDER_BUSY_PLEASE_RETRY = 44
RESPONDER_BUSY_PLEASE_RETRY [RFC 7401]

Reserved = 0
Reserved [RFC 7401]

UNKNOWN_NEXT_HOP = 90
UNKNOWN_NEXT_HOP [RFC 6028]

UNSUPPORTED_CRITICAL_PARAMETER_TYPE = 1
UNSUPPORTED_CRITICAL_PARAMETER_TYPE [RFC 7401]

UNSUPPORTED_HIT_SUITE = 20
UNSUPPORTED_HIT_SUITE [RFC 7401]

Unassigned_25 = 25
Unassigned

Unassigned_27 = 27
Unassigned

Unassigned_41 = 41
Unassigned

Unassigned_43 = 43
Unassigned

Unassigned_45 = 45
Unassigned

Unassigned_47 = 47
Unassigned

Unassigned_49 = 49
Unassigned

Packet Types††[0]

††

class pcapkit.const.hip.packet.**Packet**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Packet] HIP Packet Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CLOSE = 18
CLOSE [RFC 7401] the HIP Association Closing Packet

CLOSE_ACK = 19
CLOSE_ACK [RFC 7401] the HIP Closing Acknowledgment Packet

HDRR = 20
HDRR [RFC 6537] HIP Distributed Hash Table Resource Record

HIP_DATA = 32
HIP_DATA [RFC 6078]

I1 = 1
I1 [RFC 7401] the HIP Initiator Packet

I2 = 3
I2 [RFC 7401] the Second HIP Initiator Packet

NOTIFY = 17
NOTIFY [RFC 7401] the HIP Notify Packet

R1 = 2
R1 [RFC 7401] the HIP Responder Packet

R2 = 4
R2 [RFC 7401] the Second HIP Responder Packet

Reserved = 0
Reserved [RFC 7401]

UPDATE = 16
UPDATE [RFC 7401] the HIP Update Packet

Parameter Types‡‡[0]

‡‡

class pcapkit.const.hip.parameter.**Parameter**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Parameter] HIP Parameter Types

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-1
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-4

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ACK = 449
ACK [RFC 7401]

ACK_DATA = 4545
ACK_DATA [RFC 6078]

CERT = 768
CERT [RFC 7401][RFC 8002]

DH_GROUP_LIST = 511
DH_GROUP_LIST [RFC 7401]

DIFFIE_HELLMAN = 513
DIFFIE_HELLMAN [RFC 7401]

ECHO_REQUEST_SIGNED = 897
ECHO_REQUEST_SIGNED [RFC 7401]

ECHO_REQUEST_UNSIGNED = 63661
ECHO_REQUEST_UNSIGNED [RFC 7401]

ECHO_RESPONSE_SIGNED = 961
ECHO_RESPONSE_SIGNED [RFC 7401]

ECHO_RESPONSE_UNSIGNED = 63425
ECHO_RESPONSE_UNSIGNED [RFC 7401]

ENCRYPTED = 641
ENCRYPTED [RFC 7401]

ESP_INFO = 65
ESP_INFO [RFC 7402] 12

ESP_TRANSFORM = 4095
ESP_TRANSFORM [RFC 7402]

FROM = 65498
FROM [RFC 8004] 16

HIP_CIPHER = 579
HIP_CIPHER [RFC 7401]

HIP_MAC = 61505
HIP_MAC [RFC 7401]

HIP_MAC_2 = 61569
HIP_MAC_2 [RFC 7401]

HIP_SIGNATURE = 61697
HIP_SIGNATURE [RFC 7401]

HIP_SIGNATURE_2 = 61633
HIP_SIGNATURE_2 [RFC 7401]

HIP_TRANSFORM = 577
HIP_TRANSFORM [RFC 5201] (v1 only)

HIP_TRANSPORT_MODE = 7680
HIP_TRANSPORT_MODE [RFC 6261]

HIT_SUITE_LIST = 715
HIT_SUITE_LIST [RFC 7401]

HOST_ID = 705
HOST_ID [RFC 7401]

LOCATOR_SET = 193
LOCATOR_SET [RFC 8046]

NAT_TRAVERSAL_MODE = 608
NAT_TRAVERSAL_MODE [RFC 5770]

NOTIFICATION = 832
NOTIFICATION [RFC 7401]

OVERLAY_ID = 4592
OVERLAY_ID [RFC 6079]

OVERLAY_TTL = 64011
OVERLAY_TTL [RFC 6079] 4

PAYLOAD_MIC = 4577
PAYLOAD_MIC [RFC 6078]

PUZZLE = 257
PUZZLE [RFC 7401] 12

R1_COUNTER = 129
R1_COUNTER [RFC 7401] 12

R1_Counter = 128
R1_Counter [RFC 5201] 12 (v1 only)

REG_FAILED = 936
REG_FAILED [RFC 8003]

REG_FROM = 950
REG_FROM [RFC 5770] 20

REG_INFO = 930
REG_INFO [RFC 8003]

REG_REQUEST = 932
REG_REQUEST [RFC 8003]

REG_RESPONSE = 934
REG_RESPONSE [RFC 8003]

RELAY_FROM = 63998
RELAY_FROM [RFC 5770] 20

RELAY_HMAC = 65520
RELAY_HMAC [RFC 5770]

RELAY_TO = 64002
RELAY_TO [RFC 5770] 20

ROUTE_DST = 4601
ROUTE_DST [RFC 6028]

ROUTE_VIA = 64017
ROUTE_VIA [RFC 6028]

RVS_HMAC = 65500
RVS_HMAC [RFC 8004]

SEQ = 385
SEQ [RFC 7401] 4

SEQ_DATA = 4481
SEQ_DATA [RFC 6078] 4

SOLUTION = 321
SOLUTION [RFC 7401] 20

TRANSACTION_ID = 4580
TRANSACTION_ID [RFC 6078]

TRANSACTION_PACING = 610
TRANSACTION_PACING [RFC 5770] 4

TRANSPORT_FORMAT_LIST = 2049
TRANSPORT_FORMAT_LIST [RFC 7401]

Unassigned_512 = 512
Unassigned

Unassigned_578 = 578
Unassigned

Unassigned_609 = 609
Unassigned

Unassigned_65499 = 65499
Unassigned

Unassigned_65501 = 65501
Unassigned

Unassigned_931 = 931
Unassigned

Unassigned_933 = 933
Unassigned

Unassigned_935 = 935
Unassigned

VIA_RVS = 65502
VIA_RVS [RFC 8004]

Registration Types§§[0]

§

class pcapkit.const.hip.registration.**Registration**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Registration] Registration Types

[159] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-11

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

RELAY_UDP_HIP = 2
RELAY_UDP_HIP [RFC 5770]

RENDEZVOUS = 1
RENDEZVOUS [RFC 8004]

Unassigned = 0
Unassigned

Registration Failure Types¶¶[0]

¶¶

class pcapkit.const.hip.registration_failure.**RegistrationFailure**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[RegistrationFailure] Registration Failure Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Bad_certificate = 4
Bad certificate [RFC 8003]

Certificate_expired = 6
Certificate expired [RFC 8003]

Certificate_other = 7
Certificate other [RFC 8003]

Insufficient_resources = 2
Insufficient resources [RFC 8003]

Invalid_certificate = 3
Invalid certificate [RFC 8003]

Registration_requires_additional_credentials = 0
Registration requires additional credentials [RFC 8003]

Registration_type_unavailable = 1
Registration type unavailable [RFC 8003]

Unknown_CA = 8
Unknown CA [RFC 8003]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-13

Unsupported_certificate = 5
Unsupported certificate [RFC 8003]

Suite IDs#[35]

class pcapkit.const.hip.suite.**Suite**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Suite] Suite IDs

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AES_CBC_with_HMAC_SHA1 = 1
AES-CBC with HMAC-SHA1 [RFC 5201]

BLOWFISH_CBC_with_HMAC_SHA1 = 4
BLOWFISH-CBC with HMAC-SHA1 [RFC 5201]

NULL_ENCRYPT_with_HMAC_MD5 = 6
NULL-ENCRYPT with HMAC-MD5 [RFC 5201]

NULL_ENCRYPT_with_HMAC_SHA1 = 5
NULL-ENCRYPT with HMAC-SHA1 [RFC 5201]

Reserved = 0
Reserved [RFC 5201]

Suite_3DES_CBC_with_HMAC_MD5 = 3
3DES-CBC with HMAC-MD5 [RFC 5201]

Suite_3DES_CBC_with_HMAC_SHA1 = 2
3DES-CBC with HMAC-SHA1 [RFC 5201]

HIP Transport Modes[0]

class pcapkit.const.hip.transport.**Transport**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Transport] HIP Transport Modes

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

DEFAULT = 1
DEFAULT [RFC 6261]

ESP = 2
ESP [RFC 6261]

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-6
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#transport-modes

ESP_TCP = 3
ESP-TCP [RFC 6261]

RESERVED = 0
RESERVED [RFC 6261]

1.9.4 HTTP Constant Enumerations

HTTP/2 Error Code*[0]

*

class pcapkit.const.http.error_code.**ErrorCode**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ErrorCode] HTTP/2 Error Code

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CANCEL = 8
desc

COMPRESSION_ERROR = 9
desc

CONNECT_ERROR = 10
desc

ENHANCE_YOUR_CALM = 11
desc

FLOW_CONTROL_ERROR = 3
desc

FRAME_SIZE_ERROR = 6
desc

HTTP_1_1_REQUIRED = 13
desc

INADEQUATE_SECURITY = 12
desc

INTERNAL_ERROR = 2
desc

NO_ERROR = 0
desc

PROTOCOL_ERROR = 1
desc

REFUSED_STREAM = 7
desc

[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#error-code

SETTINGS_TIMEOUT = 4
desc

STREAM_CLOSED = 5
desc

HTTP/2 Frame Type†[0]

†

class pcapkit.const.http.frame.**Frame**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Frame] HTTP/2 Frame Type

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ALTSVC = 10
ALTSVC [**:rfc:`7838, Section 4`**]

CONTINUATION = 9
CONTINUATION [**:rfc:`7540, Section 6.10`**]

DATA = 0
DATA [**:rfc:`7540, Section 6.1`**]

GOAWAY = 7
GOAWAY [**:rfc:`7540, Section 6.8`**]

HEADERS = 1
HEADERS [**:rfc:`7540, Section 6.2`**]

ORIGIN = 12
ORIGIN [**RFC 8336**]

PING = 6
PING [**:rfc:`7540, Section 6.7`**]

PRIORITY = 2
PRIORITY [**:rfc:`7540, Section 6.3`**]

PUSH_PROMISE = 5
PUSH_PROMISE [**:rfc:`7540, Section 6.6`**]

RST_STREAM = 3
RST_STREAM [**:rfc:`7540, Section 6.4`**]

SETTINGS = 4
SETTINGS [**:rfc:`7540, Section 6.5`**]

Unassigned = 11
Unassigned

WINDOW_UPDATE = 8
WINDOW_UPDATE [**:rfc:`7540, Section 6.9`**]

[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#frame-type

HTTP/2 Settings‡[0]

‡

class pcapkit.const.http.setting.**Setting**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Setting] HTTP/2 Settings

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ENABLE_PUSH = 2
ENABLE_PUSH [**:rfc:`7540, Section 6.5.2`**] 1

HEADER_TABLE_SIZE = 1
HEADER_TABLE_SIZE [**:rfc:`7540, Section 6.5.2`**] 4096

INITIAL_WINDOW_SIZE = 4
INITIAL_WINDOW_SIZE [**:rfc:`7540, Section 6.5.2`**] 65535

MAX_CONCURRENT_STREAMS = 3
MAX_CONCURRENT_STREAMS [**:rfc:`7540, Section 6.5.2`**] infinite

MAX_FRAME_SIZE = 5
MAX_FRAME_SIZE [**:rfc:`7540, Section 6.5.2`**] 16384

MAX_HEADER_LIST_SIZE = 6
MAX_HEADER_LIST_SIZE [**:rfc:`7540, Section 6.5.2`**] infinite

Reserved = 0
Reserved [**RFC 7540**]

SETTINGS_ENABLE_CONNECT_PROTOCOL = 8
SETTINGS_ENABLE_CONNECT_PROTOCOL [**RFC 8441**] 0

TLS_RENEG_PERMITTED = 16
TLS_RENEG_PERMITTED [MS-HTTP2E][Gabriel Montenegro] 0x00

Unassigned = 7
Unassigned

1.9.5 IPv4 Constant Enumerations

Classification Level Encodings

class pcapkit.const.ipv4.classification_level.**ClassificationLevel**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#settings

[ClassificationLevel] Classification Level Encodings

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Confidential = 150

Reserved_1 = 241

Reserved_2 = 204

Reserved_3 = 102

Reserved_4 = 1

Secret = 90

Top_Secret = 61

Unclassified = 171

Option Classes

class pcapkit.const.ipv4.option_class.**OptionClass**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[OptionClass] Option Classes

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

control = 0

debugging_and_measurement = 2

reserved_for_future_use_1 = 1

reserved_for_future_use_3 = 3

IP Option Numbers*[0]

*

class pcapkit.const.ipv4.option_number.**OptionNumber**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[OptionNumber] IP Option Numbers

classmethod _missing_(*value*)
Lookup function used when value is not found.

[0] https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ip-parameters-1

static get(*key*, *default=- 1*)
Backport support for original codes.

ADDEXT = 147
ADDEXT - Address Extension [Ullmann IPv7][RFC 6814]

CIPSO = 134
CIPSO - Commercial Security [draft-ietf-cipso-ipsecurity-01]

DPS = 151
DPS - Dynamic Packet State [Andy Malis][RFC 6814]

EIP = 145
EIP - Extended Internet Protocol [RFC 1385][RFC 6814]

ENCODE = 15
ENCODE [VerSteeg][RFC 6814]

EOOL = 0
EOOL - End of Options List [RFC 791][Jon Postel]

EXP_158 = 158
EXP - RFC3692-style Experiment [RFC 4727]

EXP_222 = 222
EXP - RFC3692-style Experiment [RFC 4727]

EXP_30 = 30
EXP - RFC3692-style Experiment [RFC 4727]

EXP_94 = 94
EXP - RFC3692-style Experiment [RFC 4727]

E_SEC = 133
E-SEC - Extended Security [RFC 1108]

FINN = 205
FINN - Experimental Flow Control [Greg Finn]

IMITD = 144
IMITD - IMI Traffic Descriptor [Lee]

LSR = 131
LSR - Loose Source Route [RFC 791][Jon Postel]

MTUP = 11
MTUP - MTU Probe [RFC 1063][RFC 1191]

MTUR = 12
MTUR - MTU Reply [RFC 1063][RFC 1191]

NOP = 1
NOP - No Operation [RFC 791][Jon Postel]

QS = 25
QS - Quick-Start [RFC 4782]

RR = 7
RR - Record Route [RFC 791][Jon Postel]

RTRALT = 148
RTRALT - Router Alert [RFC 2113]

SDB = 149
SDB - Selective Directed Broadcast [Charles Bud Graff][RFC 6814]

SEC = 130
SEC - Security [RFC 1108]

SID = 136
SID - Stream ID [RFC 791][Jon Postel][RFC 6814]

SSR = 137
SSR - Strict Source Route [RFC 791][Jon Postel]

TR = 82
TR - Traceroute [RFC 1393][RFC 6814]

TS = 68
TS - Time Stamp [RFC 791][Jon Postel]

UMP = 152
UMP - Upstream Multicast Pkt. [Dino Farinacci][RFC 6814]

Unassigned_150 = 150
Unassigned (Released 18 October 2005)

VISA = 142
VISA - Experimental Access Control [Deborah Estrin][RFC 6814]

ZSU = 10
ZSU - Experimental Measurement [ZSu]

Protection Authority Bit Assignments

class pcapkit.const.ipv4.protection_authority.**ProtectionAuthority**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ProtectionAuthority] Protection Authority Bit Assignments

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

DOE = 4

Field_Termination_Indicator = 7

GENSER = 0

NSA = 3

SCI = 2

SIOP_ESI = 1

Unassigned_5 = 5

Unassigned_6 = 6

QS Functions

class pcapkit.const.ipv4.qs_function.**QSFunction**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[QSFunction] QS Functions

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Quick_Start_Request = 0

Report_of_Approved_Rate = 8

IPv4 Router Alert Option Values†[0]

†

class pcapkit.const.ipv4.router_alert.**RouterAlert**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[RouterAlert] IPv4 Router Alert Option Values

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Aggregated_Reservation_Nesting_Level_0 = 1
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_1 = 2
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_10 = 11
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_11 = 12
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_12 = 13
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_13 = 14
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_14 = 15
Aggregated Reservation Nesting Level [**RFC 3175**]

Aggregated_Reservation_Nesting_Level_15 = 16
Aggregated Reservation Nesting Level [**RFC 3175**]

[0] https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ipv4-router-alert-option-values

Aggregated_Reservation_Nesting_Level_16 = 17
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_17 = 18
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_18 = 19
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_19 = 20
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_2 = 3
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_20 = 21
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_21 = 22
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_22 = 23
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_23 = 24
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_24 = 25
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_25 = 26
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_26 = 27
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_27 = 28
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_28 = 29
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_29 = 30
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_3 = 4
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_30 = 31
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_31 = 32
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_4 = 5
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_5 = 6
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_6 = 7
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_7 = 8
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_8 = 9
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_9 = 10
Aggregated Reservation Nesting Level [RFC 3175]

NSIS_NATFW_NSLP = 65
NSIS NATFW NSLP [RFC 5973]

QoS_NSLP_Aggregation_Level_0 = 33
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_1 = 34
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_10 = 43
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_11 = 44
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_12 = 45
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_13 = 46
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_14 = 47
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_15 = 48
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_16 = 49
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_17 = 50
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_18 = 51
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_19 = 52
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_2 = 35
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_20 = 53
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_21 = 54
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_22 = 55
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_23 = 56
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_24 = 57
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_25 = 58
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_26 = 59
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_27 = 60
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_28 = 61
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_29 = 62
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_3 = 36
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_30 = 63
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_31 = 64
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_4 = 37
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_5 = 38
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_6 = 39
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_7 = 40
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_8 = 41
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_9 = 42
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

Reserved = 65535
Reserved [RFC 5350]

ToS (DS Field) Delay

class pcapkit.const.ipv4.tos_del.**ToSDelay**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ToSDelay] ToS (DS Field) Delay

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

LOW = 1

NORMAL = 0

ToS ECN Field

class pcapkit.const.ipv4.tos_ecn.**ToSECN**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ToSECN] ToS ECN Field

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CE = 3

ECT_0b01 = 1

ECT_0b10 = 2

Not_ECT = 0

ToS (DS Field) Precedence

class pcapkit.const.ipv4.tos_pre.**ToSPrecedence**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ToSPrecedence] ToS (DS Field) Precedence

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CRITIC_ECP = 5

Flash = 3

Flash_Override = 4

Immediate = 2

Internetwork_Control = 6

Network_Control = 7

Priority = 1

Routine = 0

ToS (DS Field) Reliability

class pcapkit.const.ipv4.tos_rel.**ToSReliability**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ToSReliability] ToS (DS Field) Reliability

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

HIGH = 1

NORMAL = 0

ToS (DS Field) Throughput

class pcapkit.const.ipv4.tos_thr.**ToSThroughput**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ToSThroughput] ToS (DS Field) Throughput

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

HIGH = 1

NORMAL = 0

1.9.6 IPv6 Constant Enumerations

IPv6 Extension Header Types*[0]

*

class pcapkit.const.ipv6.extension_header.**ExtensionHeader**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[ExtensionHeader] IPv6 Extension Header Types

static get(*key*, *default=- 1*)
Backport support for original codes.

AH = 51
AH [RFC 4302] Authentication Header

ESP = 50
ESP [RFC 4303] Encap Security Payload

[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#extension-header

HIP = 139
HIP [RFC 7401] Host Identity Protocol

HOPOPT = 0
HOPOPT [RFC 8200] IPv6 Hop-by-Hop Option

IPv6_Frag = 44
IPv6-Frag [Steve Deering] Fragment Header for IPv6

IPv6_Opts = 60
IPv6-Opts [RFC 8200] Destination Options for IPv6

IPv6_Route = 43
IPv6-Route [Steve Deering] Routing Header for IPv6

Mobility_Header = 135
Mobility Header [RFC 6275]

Shim6 = 140
Shim6 [RFC 5533] Shim6 Protocol

Use_for_experimentation_and_testing_253 = 253
Use for experimentation and testing [RFC 3692]

Use_for_experimentation_and_testing_254 = 254
Use for experimentation and testing [RFC 3692]

Destination Options and Hop-by-Hop Options†[0]

†

class pcapkit.const.ipv6.option.**Option**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Option] Destination Options and Hop-by-Hop Options

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

CALIPSO = 7
CALIPSO [RFC 5570]

DEPRECATED = 138
DEPRECATED [CHARLES LYNN]

Deprecated = 77
Deprecated [RFC 7731]

HOME = 201
HOME [RFC 6275]

ILNP = 139
ILNP [RFC 6744]

IOAM_TEMPORARY_registered_2020_04_16_expires_2021_04_16_0x11 = 17
IOAM TEMPORARY - registered 2020-04-16, expires 2021-04-16 [draft-ietf- ippm-ioam-ipv6-options]

[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2

IOAM_TEMPORARY_registered_2020_04_16_expires_2021_04_16_0x31 = 49
IOAM TEMPORARY - registered 2020-04-16, expires 2021-04-16 [draft-ietf- ippm-ioam-ipv6-options]

IP_DFF = 238
IP_DFF [RFC 6971]

JUMBO = 194
JUMBO [RFC 2675]

LIO = 140
LIO [RFC 6788]

MPL = 109
MPL [RFC 7731]

PAD = 0
PAD [IPV6]

PADN = 1
PADN [IPV6]

PDM = 15
PDM [RFC 8250]

Path_MTU_Record_Option_TEMPORARY_registered_2019_09_03_expires_2020_09_03 = 48
Path MTU Record Option TEMPORARY - registered 2019-09-03, expires 2020-09-03 [draft-ietf-6man-mtu-option]

QS = 38
QS [RFC 4782][RFC Errata 2034]

RA = 5
RA [RFC 2711]

RFC3692_style_Experiment_0x1E = 30
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0x3E = 62
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0x5E = 94
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0x7E = 126
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0x9E = 158
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0xBE = 190
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0xDE = 222
RFC3692-style Experiment [RFC 4727]

RFC3692_style_Experiment_0xFE = 254
RFC3692-style Experiment [RFC 4727]

RPL_0x63 = 99
RPL [RFC 6553][RFC-ietf-roll-useofrplinfo-31]

RPL_Option_0x23 = 35
RPL Option [RFC-ietf-roll-useofrplinfo-31]

SMF_DPD = 8
SMF_DPD [RFC 6621]

TUN = 4
TUN [RFC 2473]

IPv6 QS Functions

class pcapkit.const.ipv6.qs_function.**QSFunction**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[QSFunction] QS Functions

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Quick_Start_Request = 0

Report_of_Approved_Rate = 8

IPv6 Router Alert Option Values‡[0]

‡

class pcapkit.const.ipv6.router_alert.**RouterAlert**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[RouterAlert] IPv6 Router Alert Option Values

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Aggregated_Reservation_Nesting_Level_0 = 4
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_1 = 5
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_10 = 14
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_11 = 15
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_12 = 16
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_13 = 17
Aggregated Reservation Nesting Level [RFC 3175]

[0] https://www.iana.org/assignments/ipv6-routeralert-values/ipv6-routeralert-values.xhtml#ipv6-routeralert-values-1

Aggregated_Reservation_Nesting_Level_14 = 18
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_15 = 19
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_16 = 20
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_17 = 21
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_18 = 22
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_19 = 23
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_2 = 6
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_20 = 24
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_21 = 25
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_22 = 26
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_23 = 27
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_24 = 28
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_25 = 29
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_26 = 30
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_27 = 31
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_28 = 32
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_29 = 33
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_3 = 7
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_30 = 34
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_31 = 35
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_4 = 8
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_5 = 9
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_6 = 10
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_7 = 11
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_8 = 12
Aggregated Reservation Nesting Level [RFC 3175]

Aggregated_Reservation_Nesting_Level_9 = 13
Aggregated Reservation Nesting Level [RFC 3175]

Datagram_contains_RSVP_message = 1
Datagram contains RSVP message [RFC 2711]

Datagram_contains_a_Multicast_Listener_Discovery_message = 0
Datagram contains a Multicast Listener Discovery message [RFC 2710]

Datagram_contains_an_Active_Networks_message = 2
Datagram contains an Active Networks message [RFC 2711]

MPLS_OAM = 69
MPLS OAM [RFC 7506]

NSIS_NATFW_NSLP = 68
NSIS NATFW NSLP [RFC 5973]

QoS_NSLP_Aggregation_Level_0 = 36
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_1 = 37
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_10 = 46
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_11 = 47
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_12 = 48
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_13 = 49
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_14 = 50
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_15 = 51
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_16 = 52
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_17 = 53
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_18 = 54
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_19 = 55
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_2 = 38
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_20 = 56
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_21 = 57
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_22 = 58
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_23 = 59
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_24 = 60
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_25 = 61
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_26 = 62
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_27 = 63
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_28 = 64
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_29 = 65
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_3 = 39
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_30 = 66
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_31 = 67
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_4 = 40
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_5 = 41
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_6 = 42
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_7 = 43
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_8 = 44
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

QoS_NSLP_Aggregation_Level_9 = 45
QoS NSLP Aggregation Levels 0-31 [RFC 5974]

Reserved_3 = 3
Reserved [RFC 5350]

Reserved_65535 = 65535
Reserved [The Internet Assigned Numbers Authority]

Routing Types§[0]

class pcapkit.const.ipv6.routing.**Routing**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Routing] IPv6 Routing Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Nimrod = 1
Nimrod DEPRECATED 2009-05-06

RFC3692_style_Experiment_1 = 253
RFC3692-style Experiment 1 [RFC 4727]

RFC3692_style_Experiment_2 = 254
RFC3692-style Experiment 2 [RFC 4727]

RPL_Source_Route_Header = 3
RPL Source Route Header [RFC 6554]

Reserved = 255
Reserved

Segment_Routing_Header = 4
Segment Routing Header [RFC 8754] SRH

Source_Route = 0
Source Route [IPV6][RFC 5095] DEPRECATED

Type_2_Routing_Header = 2
Type 2 Routing Header [RFC 6275]

Seed-ID Types

class pcapkit.const.ipv6.seed_id.**SeedID**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[SeedID] Seed-ID Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

IPV6_SOURCE_ADDRESS = 0

[159] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-3

SEEDID_128_BIT_UNSIGNED_INTEGER = 3

SEEDID_16_BIT_UNSIGNED_INTEGER = 1

SEEDID_64_BIT_UNSIGNED_INTEGER = 2

TaggerId Types¶[0]

¶

class pcapkit.const.ipv6.tagger_id.**TaggerID**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[TaggerID] TaggerID Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

DEFAULT = 1
DEFAULT [RFC 6621]

IPv4 = 2
IPv4 [RFC 6621]

IPv6 = 3
IPv6 [RFC 6621]

NULL = 0
NULL [RFC 6621]

1.9.7 IPX Constant Enumerations

IPX Packet Types*[0]

*

class pcapkit.const.ipx.packet.**Packet**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Packet] IPX Packet Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Echo_Packet = 2
Echo Packet

Error_Packet = 3
Error Packet

[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#taggerId-types
[0] https://en.wikipedia.org/wiki/Internetwork_Packet_Exchange#IPX_packet_structure

NCP = 17
NCP - NetWare Core Protocol

PEP = 4
PEP - Packet Exchange Protocol, used for SAP (Service Advertising Protocol)

RIP = 1
RIP - Routing Information Protocol ([RFC 1582], [RFC 2091])

SPX = 5
SPX - Sequenced Packet Exchange

Unknown = 0
Unknown

IPX Socket Types†[0]

†

class pcapkit.const.ipx.socket.**Socket**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Socket] Socket Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Diagnostic_Packet = 1110
Diagnostic Packet

Echo_Protocol_Packet = 2
Echo Protocol Packet

Error_Handling_Packet = 3
Error Handling Packet

IPX = 32864
IPX

IPXF = 37011
IPXF - IPX Fragmentation Protocol

NetBIOS = 1109
NetBIOS

NetWare_Core_Protocol = 1105
NetWare Core Protocol - NCP – used by Novell NetWare servers

Routing_Information_Packet = 1
Routing Information Packet

Routing_Information_Protocol = 1107
Routing Information Protocol - RIP

Serialization_Packet = 1111
Serialization Packet - used for NCP as well

[0] https://en.wikipedia.org/wiki/Internetwork_Packet_Exchange#Socket_number

Service_Advertising_Protocol = 1106
Service Advertising Protocol - SAP

TCP_over_IPXF = 37009
TCP over IPXF

UDP_over_IPXF = 37010
UDP over IPXF

Used_by_Novell_NetWare_Client = 16387
Used by Novell NetWare Client

1.9.8 MH Constant Enumerations

Mobility Header Types*[0]

*

class pcapkit.const.mh.packet.**Packet**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)
Bases: aenum.IntEnum

[Packet] Mobility Header Types - for the MH Type field in the Mobility Header

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Binding_Acknowledgement = 6
Binding Acknowledgement [RFC 6275]

Binding_Error = 7
Binding Error [RFC 6275]

Binding_Refresh_Request = 0
Binding Refresh Request [RFC 6275]

Binding_Revocation_Message = 16
Binding Revocation Message [RFC 5846]

Binding_Update = 5
Binding Update [RFC 6275]

Care_of_Test = 4
Care-of Test [RFC 6275]

Care_of_Test_Init = 2
Care-of Test Init [RFC 6275]

Experimental_Mobility_Header = 11
Experimental Mobility Header [RFC 5096]

Fast_Binding_Acknowledgment = 9
Fast Binding Acknowledgment [RFC 5568]

Fast_Binding_Update = 8
Fast Binding Update [RFC 5568]

[0] https://www.iana.org/assignments/mobility-parameters/mobility-parameters.xhtml#mobility-parameters-1

Fast_Neighbor_Advertisement = 10
Fast Neighbor Advertisement [RFC 5568] (Deprecated)

Flow_Binding_Message = 21
Flow Binding Message [RFC 7109]

Handover_Acknowledge_Message = 15
Handover Acknowledge Message [RFC 5568]

Handover_Initiate_Message = 14
Handover Initiate Message [RFC 5568]

Heartbeat_Message = 13
Heartbeat Message [RFC 5847]

Home_Agent_Switch_Message = 12
Home Agent Switch Message [RFC 5142]

Home_Test = 3
Home Test [RFC 6275]

Home_Test_Init = 1
Home Test Init [RFC 6275]

Localized_Routing_Acknowledgment = 18
Localized Routing Acknowledgment [RFC 6705]

Localized_Routing_Initiation = 17
Localized Routing Initiation [RFC 6705]

Subscription_Query = 22
Subscription Query [RFC 7161]

Subscription_Response = 23
Subscription Response [RFC 7161]

Update_Notification = 19
Update Notification [RFC 7077]

Update_Notification_Acknowledgement = 20
Update Notification Acknowledgement [RFC 7077]

1.9.9 OSPF Constant Enumerations

Authentication Codes*[0]

*

class pcapkit.const.ospf.authentication.**Authentication**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Authentication] Authentication Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

[0] https://www.iana.org/assignments/ospf-authentication-codes/ospf-authentication-codes.xhtml#authentication-codes

static get(*key*, *default=- 1*)
Backport support for original codes.

Cryptographic_Authentication_with_Extended_Sequence_Numbers = 3
Cryptographic Authentication with Extended Sequence Numbers [RFC 7474]

Cryptographic_authentication = 2
Cryptographic authentication [RFC 2328][RFC 5709]

No_Authentication = 0
No Authentication [RFC 1583]

Simple_Password_Authentication = 1
Simple Password Authentication [RFC 1583]

OSPF Packet Type†[0]

†

class pcapkit.const.ospf.packet.**Packet**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)
Bases: aenum.IntEnum

[Packet] OSPF Packet Types

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Database_Description = 2
Database Description [RFC 2328]

Hello = 1
Hello [RFC 2328]

Link_State_Ack = 5
Link State Ack [RFC 2328]

Link_State_Request = 3
Link State Request [RFC 2328]

Link_State_Update = 4
Link State Update [RFC 2328]

Reserved = 0
Reserved

[0] https://www.iana.org/assignments/ospfv2-parameters/ospfv2-parameters.xhtml#ospfv2-parameters-3

1.9.10 Protocol Type Registry Constant Enumerations

LINK-LAYER HEADER TYPES*[0]

*

class pcapkit.const.reg.linktype.**LinkType**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[LinkType] Link-Layer Header Type Values

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

APPLE_IP_OVER_IEEE1394 = 138
DLT_APPLE_IP_OVER_IEEE1394

ARCNET_BSD = 7
DLT_ARCNET

ARCNET_LINUX = 129
DLT_ARCNET_LINUX

ATM_RFC1483 = 100
DLT_ATM_RFC1483

ATSC_ALP = 289
DLT_ATSC_ALP

AX25 = 3
DLT_AX25

AX25_KISS = 202
DLT_AX25_KISS

BACNET_MS_TP = 165
DLT_BACNET_MS_TP

BLUETOOTH_BREDR_BB = 255
DLT_BLUETOOTH_BREDR_BB

BLUETOOTH_HCI_H4 = 187
DLT_BLUETOOTH_HCI_H4

BLUETOOTH_HCI_H4_WITH_PHDR = 201
DLT_BLUETOOTH_HCI_H4_WITH_PHDR

BLUETOOTH_LE_LL = 251
DLT_BLUETOOTH_LE_LL

BLUETOOTH_LE_LL_WITH_PHDR = 256
DLT_BLUETOOTH_LE_LL_WITH_PHDR

BLUETOOTH_LINUX_MONITOR = 254
DLT_BLUETOOTH_LINUX_MONITOR

CAN_SOCKETCAN = 227
DLT_CAN_SOCKETCAN

[0] http://www.tcpdump.org/linktypes.html

C_HDLC = 104
DLT_C_HDLC

C_HDLC_WITH_DIR = 205
DLT_C_HDLC_WITH_DIR

DBUS = 231
DLT_DBUS

DISPLAYPORT_AUX = 275
DLT_DISPLAYPORT_AUX

DOCSIS = 143
DLT_DOCSIS

DOCSIS31_XRA31 = 273
DLT_DOCSIS31_XRA31

DSA_TAG_BRCM = 281
DLT_DSA_TAG_BRCM

DSA_TAG_BRCM_PREPEND = 282
DLT_DSA_TAG_BRCM_PREPEND

DSA_TAG_DSA = 284
DLT_DSA_TAG_DSA

DSA_TAG_EDSA = 285
DLT_DSA_TAG_EDSA

DVB_CI = 235
DLT_DVB_CI

EBHSCR = 279
DLT_EBHSCR

ELEE = 286
DLT_ELEE

EPON = 259
DLT_EPON

ERF = 197
DLT_ERF

ETHERNET = 1
DLT_EN10MB

ETHERNET_MPACKET = 274
DLT_ETHERNET_MPACKET

FC_2 = 224
DLT_FC_2

FC_2_WITH_FRAME_DELIMS = 225
DLT_FC_2_WITH_FRAME_DELIMS

FDDI = 10
DLT_FDDI

FRELAY = 107
DLT_FRELAY

FRELAY_WITH_DIR = 206
DLT_FRELAY_WITH_DIR

GPF_F = 171
DLT_GPF_F

GPF_T = 170
DLT_GPF_T

GPRS_LLC = 169
DLT_GPRS_LLC

IEEE802_11 = 105
DLT_IEEE802_11

IEEE802_11_AVS = 163
DLT_IEEE802_11_RADIO_AVS

IEEE802_11_PRISM = 119
DLT_PRISM_HEADER

IEEE802_11_RADIOTAP = 127
DLT_IEEE802_11_RADIO

IEEE802_15_4_NOFCS = 230
DLT_IEEE802_15_4_NOFCS

IEEE802_15_4_NONASK_PHY = 215
DLT_IEEE802_15_4_NONASK_PHY

IEEE802_15_4_TAP = 283
DLT_IEEE802_15_4_TAP

IEEE802_15_4_WITHFCS = 195
DLT_IEEE802_15_4_WITHFCS

IEEE802_5 = 6
DLT_IEEE802

INFINIBAND = 247
DLT_INFINIBAND

IPMB_LINUX = 209
DLT_IPMB_LINUX

IPMI_HPM_2 = 260
DLT_IPMI_HPM_2

IPNET = 226
DLT_IPNET

IPOIB = 242
DLT_IPOIB

IPV4 = 228
DLT_IPV4

IPV6 = 229
DLT_IPV6

IP_OVER_FC = 122
DLT_IP_OVER_FC

ISO_14443 = 264
DLT_ISO_14443

LAPB_WITH_DIR = 207
DLT_LAPB_WITH_DIR

LAPD = 203
DLT_LAPD

LINUX_IRDA = 144
DLT_LINUX_IRDA

LINUX_LAPD = 177
DLT_LINUX_LAPD

LINUX_SLL = 113
DLT_LINUX_SLL

LINUX_SLL2 = 276
DLT_LINUX_SLL2

LOOP = 108
DLT_LOOP

LORATAP = 270
DLT_LORATAP

LTALK = 114
DLT_LTALK

MFR = 182
DLT_MFR

MPEG_2_TS = 243
DLT_MPEG_2_TS

MTP2 = 140
DLT_MTP2

MTP2_WITH_PHDR = 139
DLT_MTP2_WITH_PHDR

MTP3 = 141
DLT_MTP3

MUX27010 = 236
DLT_MUX27010

NETANALYZER = 240
DLT_NETANALYZER

NETANALYZER_TRANSPARENT = 241
DLT_NETANALYZER_TRANSPARENT

NETLINK = 253
DLT_NETLINK

NFC_LLCP = 245
DLT_NFC_LLCP

NFLOG = 239
DLT_NFLOG

NG40 = 244
DLT_NG40

NORDIC_BLE = 272
DLT_NORDIC_BLE

NULL = 0
DLT_NULL

OPENVIZSLA = 278
DLT_OPENVIZSLA

PFLOG = 117
DLT_PFLOG

PKTAP = 258
DLT_PKTAP

PPI = 192
DLT_PPI

PPP = 9
DLT_PPP

PPP_ETHER = 51
DLT_PPP_ETHER

PPP_HDLC = 50
DLT_PPP_SERIAL

PPP_PPPD = 166
DLT_PPP_PPPD

PPP_WITH_DIR = 204
DLT_PPP_WITH_DIR

PROFIBUS_DL = 257
DLT_PROFIBUS_DL

RAW = 101
DLT_RAW

RDS = 265
DLT_RDS

RTAC_SERIAL = 250
DLT_RTAC_SERIAL

SCCP = 142
DLT_SCCP

SCTP = 248
DLT_SCTP

SDLC = 268
DLT_SDLC

SITA = 196
DLT_SITA

SLIP = 8
DLT_SLIP

STANAG_5066_D_PDU = 237
DLT_STANAG_5066_D_PDU

SUNATM = 123
DLT_SUNATM

USBPCAP = 249
DLT_USBPCAP

USB_2_0 = 288
DLT_USB_2_0

USB_DARWIN = 266
DLT_USB_DARWIN

USB_LINUX = 189
DLT_USB_LINUX

USB_LINUX_MMAPPED = 220
DLT_USB_LINUX_MMAPPED

USER_0 = 147
DLT_USER_0

USER_1 = 148
DLT_USER_1

USER_10 = 157
DLT_USER_10

USER_11 = 158
DLT_USER_11

USER_12 = 159
DLT_USER_12

USER_13 = 160
DLT_USER_13

USER_14 = 161
DLT_USER_14

USER_15 = 162
DLT_USER_15

USER_2 = 149
DLT_USER_2

USER_3 = 150
DLT_USER_3

USER_4 = 151
DLT_USER_4

USER_5 = 152
DLT_USER_5

USER_6 = 153
DLT_USER_6

USER_7 = 154
DLT_USER_7

USER_8 = 155
DLT_USER_8

USER_9 = 156
DLT_USER_9

VPP_DISPATCH = 280
DLT_VPP_DISPATCH

VSOCK = 271
DLT_VSOCK

WATTSTOPPER_DLM = 263
DLT_WATTSTOPPER_DLM

ZWAVE_R1_R2 = 261
DLT_ZWAVE_R1_R2

ZWAVE_R3 = 262
DLT_ZWAVE_R3

Z_WAVE_SERIAL = 287
DLT_Z_WAVE_SERIAL

ETHER TYPES†[0]

†

class pcapkit.const.reg.ethertype.**EtherType**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[EtherType] Ethertype IEEE 802 Numbers

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

ARAI_Bunkichi = 33188
ARAI Bunkichi [Neil Sembower]

ATOMIC = 34527
ATOMIC [Joe Touch]

AT_T_0x8008 = 32776
AT&T [Neil Sembower]

AT_T_0x8046 = 32838
AT&T [Neil Sembower]

AT_T_0x8047 = 32839
AT&T [Neil Sembower]

AT_T_0x8069 = 32873
AT&T [Neil Sembower]

Address_Resolution_Protocol = 2054
Address Resolution Protocol (ARP) [RFC 7042]

[0] https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml#ieee-802-numbers-1

Aeonic_Systems = 32822
Aeonic Systems [Neil Sembower]

Alpha_Micro = 33098
Alpha Micro [Neil Sembower]

Apollo_Computer = 33015
Apollo Computer [Neil Sembower]

Apollo_Domain = 32793
Apollo Domain [Neil Sembower]

AppleTalk_AARP = 33011
AppleTalk AARP (Kinetics) [Neil Sembower]

Appletalk = 32923
Appletalk [Neil Sembower]

Applitek_Corporation = 32967
Applitek Corporation [Neil Sembower]

Autophon = 32874
Autophon [Neil Sembower]

BBN_Simnet = 21000
BBN Simnet [Neil Sembower]

BBN_VITAL_LanBridge_cache = 65280
BBN VITAL-LanBridge cache [Neil Sembower]

BIIN_0x814D = 33101
BIIN [Neil Sembower]

BIIN_0x814E = 33102
BIIN [Neil Sembower]

Banyan_Systems_0x80C4 = 32964
Banyan Systems [Neil Sembower]

Banyan_Systems_0x80C5 = 32965
Banyan Systems [Neil Sembower]

Banyan_VINES = 2989
Banyan VINES [Neil Sembower]

Berkeley_Trailer_nego = 4096
Berkeley Trailer nego [Neil Sembower]

Cabletron = 28724
Cabletron [Neil Sembower]

Chaosnet = 2052
Chaosnet [Neil Sembower]

ComDesign = 32876
ComDesign [Neil Sembower]

Computgraphic_Corp = 32877
Computgraphic Corp. [Neil Sembower]

Counterpoint_Computers = 32866
Counterpoint Computers [Neil Sembower]

Cronus_Direct = 32772
Cronus Direct [RFC 824][Daniel Tappan]

Cronus_VLN = 32771
Cronus VLN [RFC 824][Daniel Tappan]

Customer_VLAN_Tag_Type = 33024
Customer VLAN Tag Type (C-Tag, formerly called the Q-Tag) (initially Wellfleet) [RFC 7042]

DEC_Customer_Protocol = 24582
DEC Customer Protocol [Neil Sembower]

DEC_DECNET_Phase_IV_Route = 24579
DEC DECNET Phase IV Route [Neil Sembower]

DEC_Diagnostic_Protocol = 24581
DEC Diagnostic Protocol [Neil Sembower]

DEC_Ethernet_Encryption = 32829
DEC Ethernet Encryption [Neil Sembower]

DEC_LANBridge = 32824
DEC LANBridge [Neil Sembower]

DEC_LAN_Traffic_Monitor = 32831
DEC LAN Traffic Monitor [Neil Sembower]

DEC_LAT = 24580
DEC LAT [Neil Sembower]

DEC_LAVC_SCA = 24583
DEC LAVC, SCA [Neil Sembower]

DEC_MOP_Dump_Load = 24577
DEC MOP Dump/Load [Neil Sembower]

DEC_MOP_Remote_Console = 24578
DEC MOP Remote Console [Neil Sembower]

DEC_Unassigned_0x6000 = 24576
DEC Unassigned (Exp.) [Neil Sembower]

DEC_Unassigned_0x803E = 32830
DEC Unassigned [Neil Sembower]

DLOG_0x0660 = 1632
DLOG [Neil Sembower]

DLOG_0x0661 = 1633
DLOG [Neil Sembower]

Dansk_Data_Elektronik = 32891
Dansk Data Elektronik [Neil Sembower]

Delta_Controls = 34526
Delta Controls [Neil Sembower]

ECMA_Internet = 2051
ECMA Internet [Neil Sembower]

EtherType_3Com_TCP_IP_Sys = 36866
3Com(Bridge) TCP-IP Sys [Neil Sembower]

EtherType_3Com_XNS_Sys_Mgmt = 36865
3Com(Bridge) XNS Sys Mgmt [Neil Sembower]

EtherType_3Com_loop_detect = 36867
3Com(Bridge) loop detect [Neil Sembower]

Evans_Sutherland = 32861
Evans & Sutherland [Neil Sembower]

Excelan = 32784
Excelan [Neil Sembower]

ExperData = 32841
ExperData [Neil Sembower]

Frame_Relay_ARP = 2056
Frame Relay ARP [RFC 1701]

General_Dynamics = 32872
General Dynamics [Neil Sembower]

General_Switch_Management_Protocol = 34828
General Switch Management Protocol (GSMP) [RFC 7042]

GeoNetworking_as_defined_in_ETSI_EN_302_636_4_1 = 35143
GeoNetworking as defined in ETSI EN 302 636-4-1 [IEEE]

HIPPI_FP_encapsulation = 33152
HIPPI-FP encapsulation [Neil Sembower]

HP_Probe = 32773
HP Probe [Neil Sembower]

Hayes_Microcomputers = 33072
Hayes Microcomputers [Neil Sembower]

IBM_SNA_Service_on_Ether = 32981
IBM SNA Service on Ether [Neil Sembower]

IEEE_Std_802_11_Fast_Roaming_Remote_Request = 35085
IEEE Std 802.11 - Fast Roaming Remote Request (802.11r) [IEEE]

IEEE_Std_802_11_Pre_Authentication = 35015
IEEE Std 802.11 - Pre-Authentication (802.11i) [IEEE]

IEEE_Std_802_1AB_Link_Layer_Discovery_Protocol = 35020
IEEE Std 802.1AB - Link Layer Discovery Protocol (LLDP) [IEEE]

IEEE_Std_802_1AE_Media_Access_Control_Security = 35045
IEEE Std 802.1AE - Media Access Control Security [IEEE]

IEEE_Std_802_1Q_Multiple_Multicast_Registration_Protocol = 35062
IEEE Std 802.1Q - Multiple Multicast Registration Protocol (MMRP) [IEEE]

IEEE_Std_802_1Q_Multiple_VLAN_Registration_Protocol = 35061
IEEE Std 802.1Q - Multiple VLAN Registration Protocol (MVRP) [IEEE]

IEEE_Std_802_1Q_Service_VLAN_tag_identifier = 34984
IEEE Std 802.1Q - Service VLAN tag identifier (S-Tag) [IEEE]

IEEE_Std_802_1Qbe_Multiple_I_SID_Registration_Protocol = 35113
IEEE Std 802.1Qbe - Multiple I-SID Registration Protocol [IEEE]

IEEE_Std_802_1Qbg_ECP_Protocol = 35136
IEEE Std 802.1Qbg - ECP Protocol (also used in 802.1BR) [IEEE]

IEEE_Std_802_1X_Port_based_network_access_control = 34958
IEEE Std 802.1X - Port-based network access control [IEEE]

IEEE_Std_802_21_Media_Independent_Handover_Protocol = 35095
IEEE Std 802.21 - Media Independent Handover Protocol [IEEE]

IEEE_Std_802_3_Ethernet_Passive_Optical_Network = 34824
IEEE Std 802.3 - Ethernet Passive Optical Network (EPON) [EPON][RFC 7042]

IEEE_Std_802_Local_Experimental_Ethertype_0x88B5 = 34997
IEEE Std 802 - Local Experimental Ethertype [IEEE]

IEEE_Std_802_Local_Experimental_Ethertype_0x88B6 = 34998
IEEE Std 802 - Local Experimental Ethertype [IEEE]

IEEE_Std_802_OUI_Extended_Ethertype = 34999
IEEE Std 802 - OUI Extended Ethertype [IEEE]

IP_Autonomous_Systems = 34668
IP Autonomous Systems [RFC 1701]

Internet_Protocol_version_4 = 2048
Internet Protocol version 4 (IPv4) [RFC 7042]

Internet_Protocol_version_6 = 34525
Internet Protocol version 6 (IPv6) [RFC 7042]

L2_IS_IS = 8948
L2-IS-IS [RFC 6325]

Little_Machines = 32864
Little Machines [Neil Sembower]

LoWPAN_encapsulation = 41197
LoWPAN encapsulation [RFC 7973]

Logicraft = 33096
Logicraft [Neil Sembower]

Loopback = 36864
Loopback [Neil Sembower]

MPLS = 34887
MPLS [RFC 5332]

MPLS_with_upstream_assigned_label = 34888
MPLS with upstream-assigned label [RFC 5332]

Matra = 32890
Matra [Neil Sembower]

Merit_Internodal = 32892
Merit Internodal [Hans Werner Braun]

Motorola_Computer = 33165
Motorola Computer [Neil Sembower]

Multi_Topology = 39458
Multi-Topology [RFC 8377]

Multicast_Channel_Allocation_Protocol = 34913
Multicast Channel Allocation Protocol (MCAP) [RFC 7042]

NBS_Internet = 2050
NBS Internet [Neil Sembower]

NSH = 35151
NSH (Network Service Header) [RFC 8300]

Nestar = 32774
Nestar [Neil Sembower]

Network_Computing_Devices = 33097
Network Computing Devices [Neil Sembower]

Nixdorf = 1024
Nixdorf [Neil Sembower]

Nixdorf_Computers = 32931
Nixdorf Computers [Neil Sembower]

PCS_Basic_Block_Protocol = 16962
PCS Basic Block Protocol [Neil Sembower]

PPP_over_Ethernet_Discovery_Stage = 34915
PPP over Ethernet (PPPoE) Discovery Stage [RFC 2516]

PPP_over_Ethernet_Session_Stage = 34916
PPP over Ethernet (PPPoE) Session Stage [RFC 2516]

PUP_Addr_Trans_0x0201 = 513
PUP Addr Trans (see 0A01) [Neil Sembower]

PUP_Addr_Trans_0x0A01 = 2561
PUP Addr Trans [Neil Sembower]

Pacer_Software = 32966
Pacer Software [Neil Sembower]

Planning_Research_Corp = 32836
Planning Research Corp. [Neil Sembower]

Point_to_Point_Protocol = 34827
Point-to-Point Protocol (PPP) [RFC 7042]

Proteon = 28720
Proteon [Neil Sembower]

Provider_Backbone_Bridging_Instance_tag = 35047
Provider Backbone Bridging Instance tag [IEEE Std 802.1Q-2014]

Rational_Corp = 33104
Rational Corp [Neil Sembower]

Raw_Frame_Relay = 25945
Raw Frame Relay [RFC 1701]

Reserved = 65535
Reserved [RFC 1701]

Reserved_for_HIPPI_6400_0x8182 = 33154
Reserved for HIPPI-6400 [Neil Sembower]

Reserved_for_HIPPI_6400_0x8183 = 33155
Reserved for HIPPI-6400 [Neil Sembower]

Retix = 33010
Retix [Neil Sembower]

Reverse_Address_Resolution_Protocol = 32821
Reverse Address Resolution Protocol (RARP) [RFC 903][Joseph Murdock]

SECTRA = 34523
SECTRA [Neil Sembower]

SGI_Time_Warner_prop = 33150
SGI/Time Warner prop. [Neil Sembower]

SGI_bounce_server = 32790
SGI bounce server [Andrew Cherenson]

SGI_diagnostics = 32787
SGI diagnostics [Andrew Cherenson]

SGI_network_games = 32788
SGI network games [Andrew Cherenson]

SGI_reserved = 32789
SGI reserved [Andrew Cherenson]

SNMP = 33100
SNMP [Joyce K Reynolds]

STP_HIPPI_ST = 33153
STP, HIPPI-ST [Neil Sembower]

Secure_Data = 34669
Secure Data [RFC 1701]

Spider_Systems_Ltd = 32927
Spider Systems Ltd. [Neil Sembower]

Stanford_V_Kernel_exp = 32859
Stanford V Kernel exp. [Neil Sembower]

Stanford_V_Kernel_prod = 32860
Stanford V Kernel prod. [Neil Sembower]

Symbolics_Private = 2076
Symbolics Private [David Plummer]

TCP_IP_Compression = 34667
TCP/IP Compression [RFC 1144][RFC 1701]

TRILL = 8947
TRILL [RFC 6325]

TRILL_Fine_Grained_Labeling = 35131
TRILL Fine Grained Labeling (FGL) [RFC 7172]

TRILL_RBridge_Channel = 35142
TRILL RBridge Channel [RFC 7178]

Technically_Elite_Concept = 33103
Technically Elite Concept [Neil Sembower]

The_Ethertype_will_be_used_to_identify_a_Channel_in_which_control_messages_are_encapsu
The Ethertype will be used to identify a "Channel" in which control messages are encapsulated as payload of GRE packets. When a GRE packet tagged with the Ethertype is received, the payload will be handed to the network processor for processing. [RFC 8157]

Tigan_Inc = 32815
Tigan, Inc. [Neil Sembower]

Trans_Ether_Bridging = 25944
Trans Ether Bridging [RFC 1701]

Tymshare = 32814
Tymshare [Neil Sembower]

Ungermann_Bass_dia_loop = 28674
Ungermann-Bass dia/loop [Neil Sembower]

Ungermann_Bass_download = 28672
Ungermann-Bass download [Neil Sembower]

Ungermann_Bass_net_debugr = 2304
Ungermann-Bass net debugr [Neil Sembower]

Univ_of_Mass_Amherst_0x8065 = 32869
Univ. of Mass. @ Amherst [Neil Sembower]

Univ_of_Mass_Amherst_0x8066 = 32870
Univ. of Mass. @ Amherst [Neil Sembower]

VG_Laboratory_Systems = 33073
VG Laboratory Systems [Neil Sembower]

VINES_Echo = 2991
VINES Echo [RFC 1701]

VINES_Loopback = 2990
VINES Loopback [RFC 1701]

Valid_Systems = 5632
Valid Systems [Neil Sembower]

Varian_Associates = 32989
Varian Associates [Neil Sembower]

Veeco_Integrated_Auto = 32871
Veeco Integrated Auto. [Neil Sembower]

Vitalink_TransLAN_III = 32896
Vitalink TransLAN III [Neil Sembower]

Wellfleet_Communications = 33023
Wellfleet Communications [Neil Sembower]

XEROX_NS_IDP = 1536
Data Link Layer and Physical Layer Specification", AA-K759B-TK, Digital Equipment Corporation, Maynard, MA. Also as: "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980. And: "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications", Digital, Intel and Xerox, November 1982. And: XEROX, "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specification", X3T51/80-50, Xerox Corporation, Stamford, CT., October 1980.][Neil Sembower]

Type XEROX NS IDP ["The Ethernet, A Local Area Network

XEROX_PUP = 512
XEROX PUP (see 0A00) [Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP: An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.][Neil Sembower]

XNS_Compatability = 2055
XNS Compatability [Neil Sembower]

XTP = 33149
XTP [Neil Sembower]

X_25_Level_3 = 2053
X.25 Level 3 [Neil Sembower]

X_75_Internet = 2049
X.75 Internet [Neil Sembower]

Xerox_IEEE802_3_PUP = 2560
Xerox IEEE802.3 PUP [Neil Sembower]

Assigned Internet Protocol Numbers‡[0]

‡

class pcapkit.const.reg.transtype.**TransType**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[TransType] Transport Layer Protocol Numbers

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AH = 51
[RFC 4302] Authentication Header

ARGUS = 13
[Robert W Scheifler] ARGUS (deprecated))

ARIS = 104
[Nancy Feldman] ARIS

AX_25 = 93
[Brian Kantor] AX.25 Frames

A_N = 107
[Bob Braden] Active Networks

BBN_RCC_MON = 10
[Steve Chipman] BBN RCC Monitoring

BNA = 49
[Gary Salamon] BNA

BR_SAT_MON = 76
[Steven Blumenthal] Backroom SATNET Monitoring

[0] https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml#protocol-numbers-1

CBT = 7
[Tony Ballardie] CBT

CFTP = 62
[Forsdick, H., "CFTP", Network Message, Bolt Beranek and Newman, January 1982.][Harry Forsdick] CFTP

CHAOS = 16
[J Noel Chiappa] Chaos

CPHB = 73
[David Mittnacht] Computer Protocol Heart Beat

CPNX = 72
[David Mittnacht] Computer Protocol Network Executive

CRTP = 126
[Robert Sautter] Combat Radio Transport Protocol

CRUDP = 127
[Robert Sautter] Combat Radio User Datagram

Compaq_Peer = 110
[Victor Volpe] Compaq Peer Protocol

DCCP = 33
[RFC 4340] Datagram Congestion Control Protocol

DCN_MEAS = 19
[David Mills] DCN Measurement Subsystems

DDP = 37
[Wesley Craig] Datagram Delivery Protocol

DDX = 116
[John Worley] D-II Data Exchange (DDX)

DGP = 86
[M/A-COM Government Systems, "Dissimilar Gateway Protocol Specification, Draft Version", Contract no. CS901145, November 16, 1987.][Mike Little] Dissimilar Gateway Protocol

DSR = 48
[RFC 4728] Dynamic Source Routing Protocol

EGP = 8
[RFC 888][David Mills] Exterior Gateway Protocol

EIGRP = 88
[RFC 7868] EIGRP

EMCON = 14
[<mystery contact>] EMCON

ENCAP = 98
[RFC 1241][Robert Woodburn] Encapsulation Header

ESP = 50
[RFC 4303] Encap Security Payload

ETHERIP = 97
[RFC 3378] Ethernet-within-IP Encapsulation

Ethernet = 143
[draft-ietf-spring-srv6-network-programming] Ethernet (TEMPORARY - registered 2020-01-31, expires 2021-01-31)

FC = 133
[Murali Rajagopal][RFC 6172] Fibre Channel

FIRE = 125
[Criag Partridge]

GGP = 3
[RFC 823] Gateway-to-Gateway

GMTP = 100
[RXB5] GMTP

GRE = 47
[RFC 2784][Tony Li] Generic Routing Encapsulation

HIP = 139
[RFC 7401] Host Identity Protocol

HMP = 20
[RFC 869][Bob Hinden] Host Monitoring

HOPOPT = 0
[RFC 8200] IPv6 Hop-by-Hop Option

IATP = 117
[John Murphy] Interactive Agent Transfer Protocol

ICMP = 1
[RFC 792] Internet Control Message

IDPR = 35
[Martha Steenstrup] Inter-Domain Policy Routing Protocol

IDPR_CMTP = 38
[Martha Steenstrup] IDPR Control Message Transport Proto

IDRP = 45
[Sue Hares] Inter-Domain Routing Protocol

IFMP = 101
[Bob Hinden][November 1995, 1997.] Ipsilon Flow Management Protocol

IGMP = 2
[RFC 1112] Internet Group Management

IGP = 9
[Internet Assigned Numbers Authority] any private interior gateway (used by Cisco for their IGRP)

IL = 40
[Dave Presotto] IL Transport Protocol

IPCV = 71
[Steven Blumenthal] Internet Packet Core Utility

IPComp = 108
[RFC 2393] IP Payload Compression Protocol

IPIP = 94
[John Ioannidis] IP-within-IP Encapsulation Protocol

IPLT = 129
[Hollbach]

IPPC = 67
[Steven Blumenthal] Internet Pluribus Packet Core

IPTM = 84
[Jim Stevens] Internet Protocol Traffic Manager

IPX_in_IP = 111
[CJ Lee] IPX in IP

IPv4 = 4
[RFC 2003] IPv4 encapsulation

IPv6 = 41
[RFC 2473] IPv6 encapsulation

IPv6_Frag = 44
[Steve Deering] Fragment Header for IPv6

IPv6_ICMP = 58
[RFC 8200] ICMP for IPv6

IPv6_NoNxt = 59
[RFC 8200] No Next Header for IPv6

IPv6_Opts = 60
[RFC 8200] Destination Options for IPv6

IPv6_Route = 43
[Steve Deering] Routing Header for IPv6

IRTP = 28
[RFC 938][Trudy Miller] Internet Reliable Transaction

ISIS_over_IPv4 = 124
[Tony Przygienda]

ISO_IP = 80
[Marshall T Rose] ISO Internet Protocol

ISO_TP4 = 29
[RFC 905][<mystery contact>] ISO Transport Protocol Class 4

I_NLSP = 52
[K Robert Glenn] Integrated Net Layer Security TUBA

KRYPTOLAN = 65
[Paul Liu] Kryptolan

L2TP = 115
[RFC 3931][Bernard Aboba] Layer Two Tunneling Protocol

LARP = 91
[Brian Horn] Locus Address Resolution Protocol

LEAF_1 = 25
[Barry Boehm] Leaf-1

LEAF_2 = 26
[Barry Boehm] Leaf-2

MERIT_INP = 32
[Hans Werner Braun] MERIT Internodal Protocol

MFE_NSP = 31
[Shuttleworth, B., "A Documentary of MFENet, a National Computer Network", UCRL-52317, Lawrence Livermore Labs, Livermore, California, June 1977.][Barry Howard] MFE Network Services Protocol

MICP = 95
[John Ioannidis] Mobile Internetworking Control Pro. (deprecated))

MOBILE = 55
[Charlie Perkins] IP Mobility

MPLS_in_IP = 137
[RFC 4023]

MTP = 92
[Susie Armstrong] Multicast Transport Protocol

MUX = 18
[Cohen, D. and J. Postel, "Multiplexing Protocol", IEN 90, USC/Information Sciences Institute, May 1979.][Jon Postel] Multiplexing

Mobility_Header = 135
[RFC 6275]

NARP = 54
[RFC 1735] NBMA Address Resolution Protocol

NETBLT = 30
[RFC 969][David Clark] Bulk Data Transfer Protocol

NSFNET_IGP = 85
[Hans Werner Braun] NSFNET-IGP

NVP_II = 11
[RFC 741][Steve Casner] Network Voice Protocol

OSPFIGP = 89
[RFC 1583][RFC 2328][RFC 5340][John Moy] OSPFIGP

PGM = 113
[Tony Speakman] PGM Reliable Transport Protocol

PIM = 103
[RFC 7761][Dino Farinacci] Protocol Independent Multicast

PIPE = 131
[Bernhard Petri] Private IP Encapsulation within IP

PNNI = 102
[Ross Callon] PNNI over IP

PRM = 21
[Zaw Sing Su] Packet Radio Measurement

PTP = 123
[Michael Welzl] Performance Transparency Protocol

PUP = 12
An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.][XEROX] PUP

Type [Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP

PVP = 75
[Steve Casner] Packet Video Protocol

QNX = 106
[Michael Hunter] QNX

RDP = 27
[RFC 908][Bob Hinden] Reliable Data Protocol

ROHC = 142
[RFC 5858] Robust Header Compression

RSVP = 46
[RFC 2205][RFC 3209][Bob Braden] Reservation Protocol

RSVP_E2E_IGNORE = 134
[RFC 3175]

RVD = 66
[Michael Greenwald] MIT Remote Virtual Disk Protocol

Reserved = 255
[Internet Assigned Numbers Authority]

SAT_EXPAK = 64
[Steven Blumenthal] SATNET and Backroom EXPAK

SAT_MON = 69
[Steven Blumenthal] SATNET Monitoring

SCC_SP = 96
[Howard Hart] Semaphore Communications Sec. Pro.

SCPS = 105
[Robert Durst] SCPS

SCTP = 132
[Randall R Stewart] Stream Control Transmission Protocol

SDRP = 42
[Deborah Estrin] Source Demand Routing Protocol

SECURE_VMTP = 82
[Dave Cheriton] SECURE-VMTP

SKIP = 57
[Tom Markson] SKIP

SM = 122
[Jon Crowcroft][draft-perlman-simple-multicast] Simple Multicast Protocol (deprecated))

SMP = 121
[Leif Ekblad] Simple Message Protocol

SNP = 109
[Manickam R Sridhar] Sitara Networks Protocol

SPS = 130
[Bill McIntosh] Secure Packet Shield

SRP = 119
[Mark Hamilton] SpectraLink Radio Protocol

SSCOPMCE = 128
[Kurt Waber]

ST = 5
[RFC 1190][RFC 1819] Stream

STP = 118
[Jean Michel Pittet] Schedule Transfer Protocol

SUN_ND = 77
[William Melohn] SUN ND PROTOCOL-Temporary

SWIPE = 53
[John Ioannidis] IP with Encryption (deprecated))

Shim6 = 140
[RFC 5533] Shim6 Protocol

Sprite_RPC = 90
[Welch, B., "The Sprite Remote Procedure Call System", Technical Report, UCB/Computer Science Dept., 86/302, University of California at Berkeley, June 1986.][Bruce Willins] Sprite RPC Protocol

TCF = 87
[Guillermo A Loyola] TCF

TCP = 6
[RFC 793] Transmission Control

TLSP = 56
[Christer Oberg] Transport Layer Security Protocol using Kryptonet key management

TP = 39
[Dirk Fromhein] TP++ Transport Protocol

TRUNK_1 = 23
[Barry Boehm] Trunk-1

TRUNK_2 = 24
[Barry Boehm] Trunk-2

TTP = 84
[Jim Stevens] Transaction Transport Protocol

TransType_3PC = 34
[Stuart A Friedberg] Third Party Connect Protocol

UDP = 17
[RFC 768][Jon Postel] User Datagram

UDPLite = 136
[RFC 3828]

UTI = 120
[Peter Lothberg] UTI

Use_for_experimentation_and_testing_253 = 253
[RFC 3692] Use for experimentation and testing

Use_for_experimentation_and_testing_254 = 254
[RFC 3692] Use for experimentation and testing

VINES = 83
[Brian Horn] VINES

VISA = 70
[Gene Tsudik] VISA Protocol

VMTP = 81
[Dave Cheriton] VMTP

VRRP = 112
[RFC 5798] Virtual Router Redundancy Protocol

WB_EXPAK = 79
[Steven Blumenthal] WIDEBAND EXPAK

WB_MON = 78
[Steven Blumenthal] WIDEBAND Monitoring

WESP = 141
[RFC 5840] Wrapped Encapsulating Security Payload

WSN = 74
[Victor Dafoulas] Wang Span Network

XNET = 15
[Haverty, J., "XNET Formats for Internet Protocol Version 4", IEN 158, October 1980.][Jack Haverty] Cross Net Debugger

XNS_IDP = 22
Data Link Layer and Physical Layer Specification", AA-K759B-TK, Digital Equipment Corporation, Maynard, MA. Also as: "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980. And: "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications", Digital, Intel and Xerox, November 1982. And: XEROX, "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specification", X3T51/80-50, Xerox Corporation, Stamford, CT., October 1980.][XEROX] XEROX NS IDP

Type ["The Ethernet, A Local Area Network

XTP = 36
[Greg Chesson] XTP

any_0_hop_protocol = 114
[Internet Assigned Numbers Authority] any 0-hop protocol

any_distributed_file_system = 68
[Internet Assigned Numbers Authority] any distributed file system

any_host_internal_protocol = 61
[Internet Assigned Numbers Authority] any host internal protocol

any_local_network = 63
[Internet Assigned Numbers Authority] any local network

any_private_encryption_scheme = 99
[Internet Assigned Numbers Authority] any private encryption scheme

manet = 138
[RFC 5498] MANET Protocols

1.9.11 TCP Constant Enumerations

TCP Checksum*[0]

*

class pcapkit.const.tcp.checksum.**Checksum**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Checksum] TCP Checksum [RFC 1146]

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

Checksum_16_bit_Fletcher_s_algorithm = 2

Checksum_8_bit_Fletcher_s_algorithm = 1

Redundant_Checksum_Avoidance = 3

TCP_checksum = 0

TCP Option Kind Numbers†[0]

†

class pcapkit.const.tcp.option.**Option**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[Option] TCP Option Kind Numbers

classmethod _missing_(*value*)
Lookup function used when value is not found.

static get(*key*, *default=- 1*)
Backport support for original codes.

AO = 29
TCP Authentication Option (TCP-AO) [RFC 5925]

Bubba = 17
Bubba [Stev Knowles]

CC = 11
CC (obsolete) [RFC 1644][RFC 6247]

CCECHO = 13
CC.ECHO (obsolete) [RFC 1644][RFC 6247]

CCNEW = 12
CC.NEW (obsolete) [RFC 1644][RFC 6247]

CHKREQ = 14
TCP Alternate Checksum Request (obsolete) [RFC 1146][RFC 6247]

[0] https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-2
[0] https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1

CHKSUM = 15
TCP Alternate Checksum Data (obsolete) [RFC 1146][RFC 6247]

Corruption_experienced = 23
Corruption experienced [Keith Scott]

ECHO = 6
Echo (obsoleted by option 8) [RFC 1072][RFC 6247]

ECHORE = 7
Echo Reply (obsoleted by option 8) [RFC 1072][RFC 6247]

EOOL = 0
End of Option List [RFC 793]

Encryption_Negotiation = 69
Encryption Negotiation (TCP-ENO) [RFC 8547]

FASTOPEN = 34
TCP Fast Open Cookie [RFC 7413]

MP = 30
Multipath TCP (MPTCP) [RFC 8684]

MSS = 2
Maximum Segment Size [RFC 793]

NOP = 1
No-Operation [RFC 793]

POC = 9
Partial Order Connection Permitted (obsolete) [RFC 1693][RFC 6247]

POCSP = 10
Partial Order Service Profile (obsolete) [RFC 1693][RFC 6247]

QS = 27
Quick-Start Response [RFC 4782]

RFC3692_style_Experiment_1 = 253
RFC3692-style Experiment 1 (also improperly used for shipping products) [*] [RFC 4727]

RFC3692_style_Experiment_2 = 254
RFC3692-style Experiment 2 (also improperly used for shipping products) [*] [RFC 4727]

Record_Boundaries = 22
Record Boundaries [Keith Scott]

Reserved_31 = 31
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_32 = 32
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_33 = 33
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_70 = 70
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_76 = 76
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_77 = 77
Reserved (known unauthorized use without proper IANA assignment) [**]

Reserved_78 = 78
Reserved (known unauthorized use without proper IANA assignment) [**]

SACK = 5
SACK [RFC 2018]

SACKPMT = 4
SACK Permitted [RFC 2018]

SCPS_Capabilities = 20
SCPS Capabilities [Keith Scott]

SIG = 19
MD5 Signature Option (obsoleted by option 29) [RFC 2385]

SNAP = 24
SNAP [Vladimir Sukonnik]

Selective_Negative_Acknowledgements = 21
Selective Negative Acknowledgements [Keith Scott]

Skeeter = 16
Skeeter [Stev Knowles]

TCP_Compression_Filter = 26
TCP Compression Filter [Steve Bellovin]

TIMEOUT = 28
User Timeout Option (also, other known unauthorized use) [***][1] [RFC 5482]

TS = 8
Timestamps [RFC 7323]

Trailer_Checksum_Option = 18
Trailer Checksum Option [Subbu Subramaniam][Monroe Bridges]

Unassigned = 25
Unassigned (released 2000-12-18)

WS = 3
Window Scale [RFC 7323]

1.9.12 VLAN Constant Enumerations

Priority Levels*[0]

*

class pcapkit.const.vlan.priority_level.**PriorityLevel**(*value=<object object>*, *names=None*, *module=None*, *type=None*, *start=1*)

Bases: aenum.IntEnum

[PriorityLevel] Priority levels defined in IEEE 802.1p.

classmethod _missing_(*value*)
Lookup function used when value is not found.

[0] https://en.wikipedia.org/wiki/IEEE_P802.1p#Priority_levels

static get (*key*, *default=- 1*)
Backport support for original codes.

BE = 0
1 - Best effort (default)

BK = 1
0 - Background (lowest)

CA = 3
3 - Critical applications

EE = 2
2 - Excellent effort

IC = 6
6 - Internetwork control

NC = 7
7 - Network control (highest)

VI = 4
4 - Video, < 100 ms latency and jitter

VO = 5
5 - Voice, < 10 ms latency and jitter

1.10 Web Crawlers for Constant Enumerations

1.10.1 ARP Vendor Crawlers

ARP Hardware Types*[0]

*

class pcapkit.vendor.arp.hardware.**Hardware**
Bases: *pcapkit.vendor.default.Vendor*

Hardware Types [**RFC 826**][**RFC 5494**]

FLAG = 'isinstance(value, int) and 0 <= value <= 65535'
Value limit checker.

LINK = 'https://www.iana.org/assignments/arp-parameters/arp-parameters-2.csv'
Link to registry.

[0] https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-2

Operation Codes†[0]

†

class pcapkit.vendor.arp.operation.**Operation**
Bases: *pcapkit.vendor.default.Vendor*

Operation Codes [RFC 826][RFC 5494]

FLAG = 'isinstance(value, int) and 0 <= value <= 65535'
Value limit checker.

LINK = 'https://www.iana.org/assignments/arp-parameters/arp-parameters-1.csv'
Link to registry.

1.10.2 FTP Vendor Crawlers

FTP Commands*[0]

*

FTP Return Codes†[0]

†

1.10.3 HIP Vendor Crawler

HIP Certificate Types*[0]

*

HIP Cipher IDs†[0]

†

DI-Types‡[0]

‡

[0] https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-1
[0] https://www.iana.org/assignments/ftp-commands-extensions/ftp-commands-extensions.xhtml#ftp-commands-extensions-2
[0] https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#certificate-types
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-cipher-id
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-7

ECDSA Curve Label§[0]

ECDSA_LOW Curve Label¶[0]

¶

ESP Transform Suite IDs[35]

Group IDs[0]

HI Algorithm[0]

HIT Suite ID[0]

HIP NAT Traversal Modes[0]

Notify Message Types**[0]

**

Packet Types††[0]

††

Parameter Types‡‡[0]

‡‡

Registration Types§§[0]

§

[159] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#ecdsa-curve-label
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#ecdsa-low-curve-label
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#esp-transform-suite-ids
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-5
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hi-algorithm
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hit-suite-id
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#nat-traversal
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-9
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-1
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-4
[159] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-11

Registration Failure Types¶¶[0]

¶¶

Suite IDs#[35]

HIP Transport Modes[0]

1.10.4 HTTP Vendor Crawler

HTTP/2 Error Code*[0]

*

HTTP/2 Frame Type†[0]

†

HTTP/2 Settings‡[0]

‡

1.10.5 IPv4 Vendor Crawler

Classification Level Encodings

Option Classes

IP Option Numbers*[0]

*

Protection Authority Bit Assignments

QS Functions

IPv4 Router Alert Option Values†[0]

†

[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-13
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#hip-parameters-6
[0] https://www.iana.org/assignments/hip-parameters/hip-parameters.xhtml#transport-modes
[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#error-code
[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#frame-type
[0] https://www.iana.org/assignments/http2-parameters/http2-parameters.xhtml#settings
[0] https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ip-parameters-1
[0] https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ipv4-router-alert-option-values

ToS (DS Field) Delay

ToS ECN Field

ToS (DS Field) Precedence

ToS (DS Field) Reliability

ToS (DS Field) Throughput

1.10.6 IPv6 Vendor Crawler

IPv6 Extension Header Types*[0]

*

Destination Options and Hop-by-Hop Options†[0]

†

IPv6 QS Functions

IPv6 Router Alert Option Values‡[0]

‡

Routing Types§[0]

Seed-ID Types

TaggerId Types¶[0]

¶

1.10.7 IPX Vendor Crawler

IPX Packet Types*[0]

*

[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#extension-header
[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2
[0] https://www.iana.org/assignments/ipv6-routeralert-values/ipv6-routeralert-values.xhtml#ipv6-routeralert-values-1
[159] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-3
[0] https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#taggerId-types
[0] https://en.wikipedia.org/wiki/Internetwork_Packet_Exchange#IPX_packet_structure

IPX Socket Types†[0]

†

1.10.8 MH Vendor Crawler

Mobility Header Types*[0]

*

1.10.9 OSPF Vendor Crawler

Authentication Codes*[0]

*

OSPF Packet Type†[0]

†

1.10.10 Protocol Type Registry Vendor Crawlers

LINK-LAYER HEADER TYPES*[0]

*

class pcapkit.vendor.reg.linktype.**LinkType**

Bases: *pcapkit.vendor.default.Vendor*

Link-Layer Header Type Values

count(*data*)

Count field records.

process(*data*)

Process registry data.

Parameters **data** (*List[str]*) – Registry data.

Returns Enumeration fields. List[str]: Missing fields.

Return type List[str]

request(*text*)

Fetch registry table.

Parameters **text** (*str*) – Context from *LINK*.

Returns Rows (`tr`) from registry table (`table`).

Return type List[str]

[0] https://en.wikipedia.org/wiki/Internetwork_Packet_Exchange#Socket_number
[0] https://www.iana.org/assignments/mobility-parameters/mobility-parameters.xhtml#mobility-parameters-1
[0] https://www.iana.org/assignments/ospf-authentication-codes/ospf-authentication-codes.xhtml#authentication-codes
[0] https://www.iana.org/assignments/ospfv2-parameters/ospfv2-parameters.xhtml#ospfv2-parameters-3
[0] http://www.tcpdump.org/linktypes.html

FLAG = 'isinstance(value, int) and 0x00000000 <= value <= 0xFFFFFFFF'
Value limit checker.

LINK = 'http://www.tcpdump.org/linktypes.html'
Link to registry.

ETHER TYPES†[0]

†

class pcapkit.vendor.reg.ethertype.**EtherType**
Bases: *pcapkit.vendor.default.Vendor*

Ethertype IEEE 802 Numbers

count(*data*)
Count field records.

Parameters **data** (*List[str]*) – CSV data.

Returns Field recordings.

Return type Counter

process(*data*)
Process CSV data.

Parameters **data** (*List[str]*) – CSV data.

Returns Enumeration fields. List[str]: Missing fields.

Return type List[str]

rename(*name*, *code*)
Rename duplicated fields.

Parameters

- **name** (*str*) – Field name.
- **code** (*str*) – Field code (hex).

Keyword Arguments **original** (*str*) – Original field name (extracted from CSV records).

Returns Revised field name.

Return type str

FLAG = 'isinstance(value, int) and 0x0000 <= value <= 0xFFFF'
Value limit checker.

LINK = 'https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers-1.csv'
Link to registry.

[0] https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml#ieee-802-numbers-1

Assigned Internet Protocol Numbers‡[0]

‡

class pcapkit.vendor.reg.transtype.**TransType**
Bases: *pcapkit.vendor.default.Vendor*

Transport Layer Protocol Numbers

count(*data*)
Count field records.

Parameters **data** (*List[str]*) – CSV data.

Returns Field recordings.

Return type Counter

process(*data*)
Process CSV data.

Parameters **data** (*List[str]*) – CSV data.

Returns Enumeration fields. List[str]: Missing fields.

Return type List[str]

FLAG = 'isinstance(value, int) and 0 <= value <= 255'
Value limit checker.

LINK = 'https://www.iana.org/assignments/protocol-numbers/protocol-numbers-1.csv'
Link to registry.

1.10.11 TCP Vendor Crawler

TCP Checksum*[0]

*

TCP Option Kind Numbers†[0]

†

1.10.12 VLAN Vendor Crawler

Priority Levels*[0]

*

[0] https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml#protocol-numbers-1
[0] https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-2
[0] https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1
[0] https://en.wikipedia.org/wiki/IEEE_P802.1p#Priority_levels

1.10.13 Base Generator

class pcapkit.vendor.default.**Vendor**
Bases: object

Default vendor generator.

Inherit this class with *FLAG* & *LINK* attributes, etc. to implement a new vendor generator.

__init__()
Generate new constant files.

static __new__(*cls*)
Subclassing checkpoint.

Raises ***VendorNotImplemented*** – If `cls` is not a subclass of *Vendor*.

_request()
Fetch CSV data from *LINK*.

This is the low-level call of *request()*.

If *LINK* is `None`, it will directly call the upper method *request()* with **NO** arguments.

The method will first try to *GET* the content of *LINK*. Should any exception raised, it will first try with proxy settings from *get_proxies()*.

Note: Since some *LINK* links are from Wikipedia, etc., they might not be available in certain areas, e.g. the amazing PRC :)

Would proxies failed again, it will prompt for user intervention, i.e. it will use `webbrowser.open()` to open the page in browser for you, and you can manually load that page and save the HTML source at the location it provides.

Returns CSV data.

Return type List[str]

Warns **VendorRequestWarning** – If connection failed with and/or without proxies.

See also:

request()

context(*data*)
Generate constant context.

Parameters **data** (*List[str]*) – CSV data.

Returns Constant context.

Return type str

count(*data*)
Count field records.

Parameters **data** (*List[str]*) – CSV data.

Returns Field recordings.

Return type Counter

process(*data*)
Process CSV data.

Parameters **data** (*List[str]*) – CSV data.

Returns Enumeration fields. List[str]: Missing fields.

Return type List[str]

rename(*name*, *code*, *, *original=None*)
Rename duplicated fields.

Parameters

- **name** (*str*) – Field name.
- **code** (*int*) – Field code.

Keyword Arguments **original** (*str*) – Original field name (extracted from CSV records).

Returns Revised field name.

Return type str

Example

If `name` has multiple occurrences in the source registry, the field name will be sanitised as `${name}_${code}`.

Otherwise, the plain `name` will be returned.

---

request(*text=None*)
Fetch CSV file.

Parameters **text** (*str*) – Context from *LINK*.

Returns CSV data.

Return type List[str]

safe_name(*name*)
Convert enumeration name to `enum.Enum` friendly.

Parameters **name** (*str*) – original enumeration name

Returns Converted enumeration name.

Return type str

static wrap_comment(*text*)
Wraps long-length text to shorter lines of comments.

Parameters **text** (*str*) – Source text.

Returns Wrapped comments.

DOCS
Docstring of constant enumeration.

Type str

FLAG = None
Value limit checker.

Type str

LINK = None
Link to registry.

Type str

NAME
Name of constant enumeration.

Type str

pcapkit.vendor.default.**LINE**(*NAME*, *DOCS*, *FLAG*, *ENUM*, *MISS*)
Default constant template of enumerate registry from IANA CSV.

pcapkit.vendor.default.**get_proxies**()
Get proxy for blocked sites.

The function will read PCAPKIT_HTTP_PROXY and PCAPKIT_HTTPS_PROXY, if any, for the proxy settings of requests.

Returns Proxy settings for requests.

Return type Dict[str, str]

1.10.14 Command Line Tool

```
usage: pcapkit-vendor [-h] [-V] ...

update constant enumerations

positional arguments:
  target         update targets, supply none to update all

optional arguments:
  -h, --help     show this help message and exit
  -V, --version  show program's version number and exit
```

In *pcapkit*, all files can be described as following eight different components.

- Interface (*pcapkit.interface*)

  user interface for the *pcapkit* library, which standardise and simplify the usage of this library

- Foundation (*pcapkit.foundation*)

  synthesise file I/O and protocol analysis, coordinate information exchange in all network layers

- Reassembly (*pcapkit.reassembly*)

  base on algorithms described in **RFC 815**, implement datagram reassembly of IP and TCP packets

- Protocols (*pcapkit.protocols*)

  collection of all protocol family, with detailed implementation and methods

- Utilities (*pcapkit.utilities*)

  collection of utility functions and classes

- CoreKit (*pcapkit.corekit*)

  core utilities for *pcapkit* implementation

- ToolKit (*pcapkit.toolkit*)

  utility tools for *pcapkit* implementation

- DumpKit (*pcapkit.dumpkit*)

  dump utilities for *pcapkit* implementation

1.11 Library Index

pcapkit has defined various and numerous functions and classes, which have different features and purposes. To make a simple index for this library, *pcapkit.all* contains all things from *pcapkit*.

CHAPTER

TWO

COMMAND LINE INTERFACE

pcapkit.__main__ was originally the module file of jspcapy, which is now deprecated and merged with *pcapkit*.

```
usage: pcapkit-cli [-h] [-V] [-o file-name] [-f format] [-j] [-p] [-t] [-a]
                   [-v] [-F] [-E PKG] [-P PROTOCOL] [-L LAYER]
                   input-file-name

PCAP file extractor and formatted dumper

positional arguments:
  input-file-name       The name of input pcap file. If ".pcap" omits, it will
                        be automatically appended.

optional arguments:
  -h, --help            show this help message and exit
  -V, --version         show program's version number and exit
  -o file-name, --output file-name
                        The name of input pcap file. If format extension
                        omits, it will be automatically appended.
  -f format, --format format
                        Print a extraction report in the specified output
                        format. Available are all formats supported by
                        dictdumper, e.g.: json, plist, and tree.
  -j, --json            Display extraction report as json. This will yield
                        "raw" output that may be used by external tools. This
                        option overrides all other options.
  -p, --plist           Display extraction report as macOS Property List
                        (plist). This will yield "raw" output that may be used
                        by external tools. This option overrides all other
                        options.
  -t, --tree            Display extraction report as tree view text. This will
                        yield "raw" output that may be used by external tools.
                        This option overrides all other options.
  -a, --auto-extension  If output file extension omits, append automatically.
  -v, --verbose         Show more information.
  -F, --files           Split each frame into different files.
  -E PKG, --engine PKG  Indicate extraction engine. Note that except default
                        or pcapkit engine, all other engines need support of
                        corresponding packages.
  -P PROTOCOL, --protocol PROTOCOL
                        Indicate extraction stops after which protocol.
  -L LAYER, --layer LAYER
                        Indicate extract frames until which layer.
```

CHAPTER

THREE

ABOUT

PyPCAPKit is an independent open source library, using only `DictDumper` as its formatted output dumper.

Note: There is a project called `jspcapy` works on *pcapkit*, which is a command line tool for PCAP extraction but now ***DEPRECATED***.

Unlike popular PCAP file extractors, such as `Scapy`, `dpkt`, `PyShark`, and etc, *pcapkit* uses **streaming** strategy to read input files. That is to read frame by frame, decrease occupation on memory, as well as enhance efficiency in some way.

3.1 Module Structure

In *pcapkit*, all files can be described as following eight parts.

- Interface (*pcapkit.interface*)

  User interface for the *pcapkit* library, which standardise and simplify the usage of this library.

- Foundation (*pcapkit.foundation*)

  Synthesise file I/O and protocol analysis, coordinate information exchange in all network layers.

- Reassembly (*pcapkit.reassembly*)

  Based on algorithms described in **RFC 815**, implement datagram reassembly of IP and TCP packets.

- Protocols (*pcapkit.protocols*)

  Collection of all protocol family, with detail implementation and methods, as well as constructors.

- CoreKit (*pcapkit.corekit*)

  Core utilities for *pcapkit* implementation.

- TookKit (*pcapkit.toolkit*)

  Compatibility tools for *pcapkit* implementation.

- DumpKit (*pcapkit.dumpkit*

  Dump utilities for *pcapkit* implementation.

- Utilities (*pcapkit.utilities*)

  Collection of four utility functions and classes.

3.2 Engine Comparison

Besides, due to complexity of *pcapkit*, its extraction procedure takes around *0.0009* seconds per packet, which is not ideal enough. Thus *pcapkit* introduced alternative extractionengines to accelerate this procedure. By now *pcapkit* supports Scapy, DPKT, and PyShark. Plus, *pcapkit* supports two strategies of multiprocessing (`server` & `pipeline`). For more information, please refer to the documentation.

3.2.1 Test Environment

Operating System	macOS Mojave
Processor Name	Intel Core i7
Processor Speed	2.6 GHz
Total Number of Cores	6
Memory	16 GB

3.2.2 Test Results

Engine	Performance (seconds per packet)
`dpkt`	0.00017389218012491862
`scapy`	0.00036091208457946774
`default`	0.0009537641207377116
`pipeline`	0.0009694552421569824
`server`	0.018088217973709107
`pyshark`	0.04200994372367859

CHAPTER

FOUR

INSTALLATION

Note: *pcapkit* supports Python versions **since 3.4**.

Simply run the following to install the current version from PyPI:

```
pip install pypcapkit
```

Or install the latest version from the gi repository:

```
git clone https://github.com/JarryShaw/PyPCAPKit.git
cd pypcapkit
pip install -e .
# and to update at any time
git pull
```

And since *pcapkit* supports various extraction engines, and extensive plug-in functions, you may want to install the optional ones:

```
# for DPKT only
pip install pypcapkit[DPKT]
# for Scapy only
pip install pypcapkit[Scapy]
# for PyShark only
pip install pypcapkit[PyShark]
# and to install all the optional packages
pip install pypcapkit[all]
# or to do this explicitly
pip install pypcapkit dpkt scapy pyshark
```

CHAPTER

FIVE

SAMPLES

5.1 Usage Samples

As described above, :mo:d`pcapkit` is quite easy to use, with simply three verbs as its main interface. Several scenarios are shown as below.

1. extract a PCAP file and dump the result to a specific file (with no reassembly)

```
import pcapkit
# dump to a PLIST file with no frame storage (property frame disabled)
plist = pcapkit.extract(fin='in.pcap', fout='out.plist', format='plist',␣
↪store=False)
# dump to a JSON file with no extension auto-complete
json = pcapkit.extract(fin='in.cap', fout='out.json', format='json',␣
↪extension=False)
# dump to a folder with each tree-view text file per frame
tree = pcapkit.extract(fin='in.pcap', fout='out', format='tree', files=True)
```

2. extract a PCAP file and fetch IP packet (both IPv4 and IPv6) from a frame (with no output file)

```
>>> import pcapkit
>>> extraction = pcapkit.extract(fin='in.pcap', nofile=True)
>>> frame0 = extraction.frame[0]
# check if IP in this frame, otherwise ProtocolNotFound will be raised
>>> flag = pcapkit.IP in frame0
>>> tcp = frame0[pcapkit.IP] if flag else None
```

3. extract a PCAP file and reassemble TCP payload (with no output file nor frame storage)

```
import pcapkit
# set strict to make sure full reassembly
extraction = pcapkit.extract(fin='in.pcap', store=False, nofile=True, tcp=True,␣
↪strict=True)
# print extracted packet if HTTP in reassembled payloads
for packet in extraction.reassembly.tcp:
    for reassembly in packet.packets:
        if pcapkit.HTTP in reassembly.protochain:
            print(reassembly.info)
```

5.2 CLI Samples

The CLI (command line interface) of *pcapkit* has two different access.

- through console scripts

  Use command name `pcapkit [...]` directly (as shown in samples).

- through Python module

  `python -m pypcapkit [...]` works exactly the same as above.

Here are some usage samples:

1. export to a macOS Property List (Xcode has special support for this format)

```
$ pcapkit in --format plist --verbose
Loading file 'in.pcap'
 - Frame   1: Ethernet:IPv6:ICMPv6
 - Frame   2: Ethernet:IPv6:ICMPv6
 - Frame   3: Ethernet:IPv4:TCP
 - Frame   4: Ethernet:IPv4:TCP
 - Frame   5: Ethernet:IPv4:TCP
 - Frame   6: Ethernet:IPv4:UDP
Report file stored in 'out.plist'
```

2. export to a JSON file (with no format specified)

```
$ pcapkit in --output out.json --verbose
Loading file 'in.pcap'
 - Frame   1: Ethernet:IPv6:ICMPv6
 - Frame   2: Ethernet:IPv6:ICMPv6
 - Frame   3: Ethernet:IPv4:TCP
 - Frame   4: Ethernet:IPv4:TCP
 - Frame   5: Ethernet:IPv4:TCP
 - Frame   6: Ethernet:IPv4:UDP
Report file stored in 'out.json'
```

3. export to a text tree view file (without extension autocorrect)

```
$ pcapkit in --output out --format tree --verbose
Loading file 'in.pcap'
 - Frame   1: Ethernet:IPv6:ICMPv6
 - Frame   2: Ethernet:IPv6:ICMPv6
 - Frame   3: Ethernet:IPv4:TCP
 - Frame   4: Ethernet:IPv4:TCP
 - Frame   5: Ethernet:IPv4:TCP
 - Frame   6: Ethernet:IPv4:UDP
Report file stored in 'out'
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z