Edition 2024

SL DPI

Protocol Dictionary

slinkin.tech

Contents

Protocol dictionary

Physical

None

Data Link

Ethernet

> STATUS

Protocol	RFC	Status	Tags
Ethernet	ieee802.3	Fully	basic, network, internet

> FIELDS

Name	Type	Length	Mask	Description
ethernet_type	**uint16**	**2**	ffffffffffffffff	Two-octet field which is used to indicate which protocol is encapsulated in the payload of the frame. 0x0000 - 0x05DC - IEEE802.3 length Field. 0x0101-0x01FF - experimental.
dst_mac	**byte-sequence**	**6**	ffffffffffffffff	Destination MAC address.
src_mac	**byte-sequence**	**6**	ffffffffffffffff	Source MAC address.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

ARP

> STATUS

Protocol	RFC	Status	Tags
ARP	rfc826	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (Ethernet Type)

> FIELDS

Name	Type	Length	Mask	Description
tpa	**byte-sequence**	**0**	ffffffffffffff	Target protocol address. (Internetwork address of the intended receiver)
tha	**byte-sequence**	**0**	ffffffffffffff	Target hardware address. In request, this field is not used. In reply, indicates the address of the host that originated the ARP request.
spa	**byte-sequence**	**0**	ffffffffffffff	Sender protocol address. (Internetwork address of the sender)
sha	**byte-sequence**	**0**	ffffffffffffff	Sender hardware address. In request, indicates the address of the host sending the request. In reply, indicates the address of the host that the request was looking for.
op	**uint16**	**2**	ffffffffffffff	Operation. 1: request, 2: reply.
plen	**uint8**	**1**	ffffffffffffff	Protocol length. (Internetwork addresses length; in octets)
hlen	**uint8**	**1**	ffffffffffffff	Hardware address length. (in octets)
ptype	**uint16**	**2**	ffffffffffffff	Protocol type. Specifies internetwork protocol.
htype	**uint16**	**2**	ffffffffffffff	Hardware type. (Network link protocol type)
root	**uint8**	**1**	ffffffffffffff	Payload data.

RARP

> STATUS

Protocol	RFC	Status	Tags
RARP	rfc903	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (Ethernet Type)

> FIELDS

All **ARP** fields are valid for RARP as well.

VLAN C-TAG

> STATUS

Protocol	RFC	Status	Tags
Vlan C-Tag	ieee802.1q	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (Ethernet Type)

> FIELDS

Name	Type	Length	Mask	Description
ethernet_type	**uint16**	**2**	ffffffffffffffff	Two-octet field which is used to indicate which protocol is encapsulated in the payload of the frame. 0x0000 - 0x05DC - **IEEE802.3** length Field. 0x0101-0x01FF - experimental.
vid	**16-bit-field**	**2**	fff	VLAN identifier.

dei	**16-bit-field**	**2**	1000	Drop eligible indicator.
pcp	**16-bit-field**	**2**	e000	Priority code point.
tci	**uint16**	**2**	ffffffffffffffff	Tag control information.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

GRE

> STATUS

Protocol	RFC	Status	Tags
GRE	rfc2784	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (IP Protocol Type))

> FIELDS

Name	Type	Length	Mask	Description
reserved1	**uint16**	**2**	ffffffffffffffff	The Reserved1 field is reserved for future use, and if present, MUST be transmitted as zero.
checksum	**uint16**	**2**	ffffffffffffffff	The Checksum field contains the IP (one's complement) checksum sum of the all the 16 bit words in the GRE header and the payload packet.
protocol_type	**uint16**	**2**	ffffffffffffffff	The Protocol Type field contains the protocol type of the payload packet. These Protocol Types are defined in rfc1700 as "ETHER TYPES" and in [ETYPES].
version	**uint16**	**2**	7	The Version Number field MUST contain the value zero.

reserved0	**uint16**	**2**	7ff8	A receiver MUST discard a packet where any of bits 1-5 are non-zero, unless that receiver implements rfc1700. Bits 6-12 are reserved for future use.
checksum _flag	**uint16**	**2**	8000	If the Checksum Present bit is set to one, then the Checksum and the Reserved1 fields are present and the Checksum field contains valid information.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

Network

IPv4

> STATUS

Protocol	RFC	Status	Tags
IPv4	rfc791	Partly	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (Ethernet Type)

> FIELDS

Name	Type	Length	Mask	Description
options	**byte-sequence**	**0**	ffffffffffffffff	The options section.
dst_ip	**uint32**	**4**	ffffffffffffffff	The destination address.
src_ip	**uint32**	**4**	ffffffffffffffff	The source address.
checksum	**uint16**	**2**	ffffffffffffffff	A checksum on the header only.
protocol	**uint8**	**1**	ffffffffffffffff	This field indicates the next level protocol used in the data portion of the internet datagram.
ttl	**uint8**	**1**	ffffffffffffffff	This field indicates the maximum time the datagram is allowed to remain in the internet system.
fragment_offset	**16-bit-field**	**2**	1fff	This field indicates where in the datagram this fragment belongs.
mf_flag	**16-bit-field**	**2**	2000	**0** (last fragment), **1** (more fragments).
dm_flag	**16-bit-field**	**2**	4000	**0** (may fragment), **1** (don't fragment).
reserved_flag	**16-bit-field**	**2**	8000	Reserved bit. Must be zero.

id	**uint16**	**2**	ffffffffffffffff	An identifying value assigned by the sender to aid in assembling the fragments of a datagram.
total_length	**uint16**	**2**	ffffffffffffffff	**Total Length** is the length of the datagram, measured in octets, including internet header and data.
ecn	**8-bit-field**	**1**	3	Explicit Congestion Notification.
dscp	**8-bit-field**	**1**	fc	Differentiated Services Code Point.
tos	**uint8**	**1**	ffffffffffffffff	Type of Service provides an indication of the abstract parameters of the quality of service desired.
ihl	**8-bit-field**	**1**	f	Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data.
ip_version	**8-bit-field**	**1**	f0	The Version field indicates the format of the internet header.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

> LIMITATION

- Options are not supported

ICMP

> STATUS

Protocol	RFC	Status	Tags
ICMP	rfc792	Partly	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (IP Protocol Type)

> FIELDS

Name	Type	Length	Mask	Description
padding	**uint32**	**4**	ffffffffffffffff	Four-byte field, contents vary based on the ICMP type and code.
checksum	**uint16**	**2**	ffffffffffffffff	The 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type.
code	**uint8**	**1**	ffffffffffffffff	Additional context information for the message.
type	**uint8**	**1**	ffffffffffffffff	Type of message.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

Transport

IPv4

> STATUS

Protocol	RFC	Status	Tags
UDP	rfc768	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (IP Protocol Type)

> FIELDS

Name	Type	Length	Mask	Description
checksum	**uint16**	**2**	ffffffffffffffff	Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.
length	**uint16**	**2**	ffffffffffffffff	Length is the length in octets of this user datagram including this header and the data.
dst_port	**uint16**	**2**	ffffffffffffffff	Destination port.
src_port	**uint16**	**2**	ffffffffffffffff	Source port.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

TCP

> STATUS

Protocol	RFC	Status	Tags
TCP	rfc793	Partly	basic, network, internet

> LAYER DETECTION METHODS

- Explicit detection (IP Protocol Type)

> FIELDS

Name	Type	Length	Mask	Description
urgent_pointer	**uint16**	**2**	ffffffffffffffff	This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment.
checksum	**uint16**	**2**	ffffffffffffffff	The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text.
window_size	**uint16**	**2**	ffffffffffffffff	The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.
fin	**8-bit-field**	**1**	1	No more data from sender.
syn	**8-bit-field**	**1**	2	Synchronize sequence numbers.
rst	**8-bit-field**	**1**	4	Reset the connection.
psh	**8-bit-field**	**1**	8	Push Function.
ack	**8-bit-field**	**1**	10	Acknowledgment field significant.
urg	**8-bit-field**	**1**	20	Urgent Pointer field significant.

ece	**8-bit-field**	**1**	40	ECN-Echo flag.
cwr	**8-bit-field**	**1**	80	Congestion Window Reduced flag.
flags	**uint8**	**1**	ffffffffffffff	The field which contains tcp flags which are used to indicate a particular state of connection.
reserved	**8-bit-field**	**1**	f	Reserved for future use. Must be zero.
data_offset	**8-bit-field**	**1**	f0	The number of 32 bit words in the TCP Header.
ack_number	**uint32**	**4**	ffffffffffffff	If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive.
sequence_number	**uint32**	**4**	ffffffffffffff	The sequence number of the first data octet in this segment (except when SYN is present).
dst_port	**uint16**	**2**	ffffffffffffff	The destination port number.
src_port	**uint16**	**2**	ffffffffffffff	The source port number.
root	**uint8**	**1**	ffffffffffffff	Payload data.

> LIMITATION

- Options are not supported

QUIC

> STATUS

Protocol	RFC	Status	Tags
Quic	rfc9000 rfc9001	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Layer structure test

> PORTS

- 443 (**udp**)

> FIELDS

Name	Type	Length	Mask	Description
reason_phrase	**byte-sequence**	**0**	ffffffffffffff	Additional diagnostic information for the closure. This can be zero length if the sender chooses not to give details beyond the Error Code value.
reason_phrase_length	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer specifying the length of the reason phrase in bytes.
triggered_frame_type	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer encoding the type of frame that triggered the error. A value of 0 (equivalent to the mention of the PADDING frame) is used when the frame type is unknown. The field is presented only when frame type is 0x1d.
error_code	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer that indicates the reason for closing this connection. Error codes for 0x1c and 0x1d frame types have different description.
stream_data	**byte-sequence**	**0**	ffffffffffffff	The bytes from the designated stream to be delivered.
stream_data_length	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer specifying the length of the Stream Data field in this STREAM frame. This field is present when the LEN bit is set to **1**. When the LEN bit is set to **0**, the Stream Data field consumes all the remaining bytes in the packet.
stream_offset	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer specifying the byte offset in the stream for the data in this STREAM frame. This field is present when the **OFF** bit is set to **1**. When the Offset field is absent, the offset is **0**.

stateless_reset_token	**byte-sequence**	**16**	ffffffffffffff	A 128-bit value that will be used for a stateless reset when the associated connection ID is used.
connection_id	**byte-sequence**	**0**	ffffffffffffff	A connection ID of the specified length.
connection_id_length	**uint8**	**1**	ffffffffffffff	An 8-bit unsigned integer containing the length of the connection ID.
retire_prior_to	**byte-sequence**	**0**	ffffffffffffff	An 8-bit unsigned integer containing the length of the connection ID.
path_response_data	**byte-sequence**	**8**	ffffffffffffff	This 8-byte field contains arbitrary data.
path_challenge_data	**byte-sequence**	**8**	ffffffffffffff	This 8-byte field contains arbitrary data.
new_sequence_number	**byte-sequence**	**0**	ffffffffffffff	The sequence number assigned to the connection ID by the sender, encoded as a variable-length integer.
retire_sequence_number	**byte-sequence**	**0**	ffffffffffffff	The sequence number of the connection ID being retired.
allowed_maximum_streams	**byte-sequence**	**0**	ffffffffffffff	A count of the cumulative number of streams of the corresponding type that can be opened over the lifetime of the connection.
cumulative_maximum_streams	**byte-sequence**	**0**	ffffffffffffff	A count of the cumulative number of streams of the corresponding type that can be opened over the lifetime of the connection.
blocked_maximum_stream_data	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the maximum amount of data that can be sent on the identified stream, in units of bytes.
maximum_stream_data	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the maximum amount of data that can be sent on the identified stream, in units of bytes.
blocked_maximum_data	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the connection-level limit at which blocking occurred.

maximum_data	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the maximum amount of data that can be sent on the entire connection, in units of bytes.
crypto_data	**byte-sequence**	**0**	ffffffffffffff	The cryptographic message data.
crypto_data_length	**uint8**	**0**	ffffffffffffff	A variable-length integer specifying the length of the Crypto Data field in this CRYPTO frame.
crypto_offset	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer specifying the byte offset in the stream for the data in this CRYPTO frame.
final_size	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the final size of the stream by the RESET_STREAM sender, in units of bytes.
stop_application_protocol_error_code	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer containing the application-specified reason the sender is ignoring the stream.
reset_application_protocol_error_code	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer containing the application protocol error code that indicates why the stream is being closed.
stream_id	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the stream ID of the stream.
blocked_stream_id	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the stream that is blocked due to flow control.
max_data_stream_id	**byte-sequence**	**0**	ffffffffffffff	The stream ID of the affected stream, encoded as a variable-length integer.
stop_stream_id	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer carrying the stream ID of the stream being ignored.
reset_stream_id	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer encoding of the stream ID of the stream being terminated.

ecn_ce_count	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer representing the total number of packets received with the ECN-CE codepoint in the packet number space of the ACK frame.
ect_1_count	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer representing the total number of packets received with the ECT(1) codepoint in the packet number space of the ACK frame.
ect_0_count	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer representing the total number of packets received with the ECT(0) codepoint in the packet number space of the ACK frame.
ecn_counts	**byte-sequence**	**0**	ffffffffffffff	The three ECN counts. ECN counts are only present when the ACK frame type is 0x03.
ack_range_length	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the number of contiguous acknowledged packets preceding the largest packet number, as determined by the preceding Gap.
gap	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the number of contiguous unacknowledged packets preceding the packet number one lower than the smallest in the preceding ACK Range.
ack_range	**byte-sequence**	**0**	ffffffffffffff	Contains additional ranges of packets that are alternately not acknowledged (Gap) and acknowledged (ACK Range).
first_ack_range	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer indicating the number of contiguous packets preceding the Largest Acknowledged that are being acknowledged.
ack_range_count	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer specifying the number of ACK Range fields in the frame.
ack_delay	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer encoding the acknowledgment delay in microseconds.

largest_acknowledged	**byte-sequence**	**0**	ffffffffffffff	A variable-length integer representing the largest packet number the peer is acknowledging; this is usually the largest packet number that the peer has received prior to generating the ACK frame.
padding_data	**byte-sequence**	**0**	ffffffffffffff	The field contains the bytes of padding frame types. The field exists for brevity purposes to not pollute padding fields.
frame_type	**byte-sequence**	**0**	ffffffffffffff	Frame type.
frame	**byte-sequence**	**0**	ffffffffffffff	Frame section. The payload of QUIC packets, after removing packet protection, consists of a sequence of complete frames.
retry_integrity_tag	**byte-sequence**	**16**	ffffffffffffff	The Retry Integrity Tag is a 128-bit field that is computed as the output of AEAD_AES_128_GCM.
retry_token	**byte-sequence**	**0**	ffffffffffffff	An opaque token that the server can use to validate the client's address.
protected_data	**byte-sequence**	**0**	ffffffffffffff	The "abstract" field which is presented for the data section which cannot be dissected. E.g. when session context or Initial packet of the session are missed.
packet_number	**byte-sequence**	**0**	ffffffffffffff	This field is 1 to 4 bytes long. The field is presented only inside a decrypted layer because the field data is protected.
packet_data	**byte-sequence**	**0**	ffffffffffffff	Packet data section - includes packet number and packet payload fields.
length	**byte-sequence**	**0**	ffffffffffffff	This is the length of the remainder of the packet (that is, the Packet Number and Payload fields) in bytes, encoded as a variable-length integer.

supported_version	**byte-sequence**	**4**	ffffffffffffffff	Supported version.
token	**byte-sequence**	**0**	ffffffffffffffff	The value of the token that was previously provided in a Retry packet or NEW_TOKEN frame.
token_length	**byte-sequence**	**0**	ffffffffffffffff	A variable-length integer specifying the length of the Token field, in bytes. This value is **0** if no token is present.
source_connection_id	**byte-sequence**	**0**	ffffffffffffffff	The source connection id.
source_connection_id_length	**byte-sequence**	**16**	ffffffffffffffff	The length of source connection id field.
destination_connection_id	**byte-sequence**	**0**	ffffffffffffffff	The destination connection id.
destination_connection_id_length	**byte-sequence**	**0**	ffffffffffffffff	The length of destination connection id field.
version	**uint32**	**4**	ffffffffffffffff	The QUIC Version is a 32-bit field that follows the first byte. This field indicates the version of **QUIC** that is in use and determines how the rest of the protocol fields are interpreted.
unprotected_packet_number_length	**uint8**	**1**	3	The field specifies the size of packet number length field. The field is presented only inside a decrypted layer because the field data is protected.
unprotected_key_phase	**uint8**	**1**	4	The field indicates the key phase, which allows a recipient of a packet to identify the packet protection keys that are used to protect the packet. The field is presented only inside a decrypted layer because the field data is protected.
unprotected_1rtt_reserved_bits	**uint8**	**1**	18	Reserved header bits of 1-RTT packet. The field is presented only inside a decrypted layer because the field data is protected.

unprotected_reserved_bits	**uint8**	**1**	c	Reserved header bits of 0-RTT and Handshake packets. The field is presented only inside a decrypted layer because the field data is protected.
protected_packet_number_length	**uint8**	**1**	3	The field specifies the size of packet number length field. The field is protected. The field is protected.
protected_key_phase	**uint8**	**1**	4	The field indicates the key phase, which allows a recipient of a packet to identify the packet protection keys that are used to protect the packet. The field is protected.
protected_1rtt_reserved_bits	**uint8**	**1**	18	Reserved header bits of 1-RTT packet. The field is protected.
protected_reserved_bits	**uint8**	**1**	c	Reserved header bits of 0-RTT and Handshake packets. The field is protected.
long_packet_type	**uint8**	**1**	30	The field specifies packet type in the long header. Initial (**0**), 0-RTT (**1**), Handshake (**2**), Retry (**3**).
fixed_bit	**uint8**	**1**	40	Packets containing a zero value for this bit are not valid packets in this version and **MUST** be discarded. A value of 1 for this bit allows **QUIC** to coexist with other protocols.
spin_bit	**uint8**	**2**	20	The latency spin bit, which is defined for 1-RTT packets, enables passive latency monitoring from observation points on the network path throughout the duration of a connection.
version_negotiation_unused	**uint8**	**1**	7f	Unused header bits of version negotiation packet.
retry_unused	**uint8**	**1**	f	Unused header bits of retry packet.
header_form	**uint8**	**1**	80	The field specifies header type. It is set to **0** for short headers and is set to **1** for long headers.

header	**uint8**	**1**	ffffffffffffffff	Quic header. The structure of header can be different between different packet types.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

> FIELD TREE

- **Root**

```
.
├── header
├── version
├──destination_connection_id_length
├── destination_connection_id
├── source_connection_id_length
├── source_connection_id
├── token_length
├── token
├── supported_version
├── length
├── packet_data
├── packet_number
├── protected_data
├── retry_token
├── retry_integrity_tag
└── frame
```

- **Header**

```
.
└── header/
    ├── header_form
    ├── retry_unused (Retry Packet)
    ├── version_negotiation_unused (Version Negotiation Packet)
    ├── spin_bit (1-RTT Packet Only)
    ├── fixed_bit (Handshake 0-RTT and 1-RTT)
    ├── long_packet_type (Handshake and 0-RTT)
    ├── protected_reserved_bits  (Handshake 0-RTT)
    ├── protected_1rtt_reserved_bits (1-RTT)
    ├── protected_key_phase (1-RTT Packet Only)
    ├── protected_packet_number_length (Handshake and 0-RTT)
    ├── unprotected_reserved_bits (Handshake and 0-RTT)
    ├── unprotected_1rtt_reserved_bits (1-RTT)
    ├── unprotected_key_phase (1-RTT Packet Only)
    └── unprotected_packet_number_length (Handshake and 0-RTT)
```

- **Frame**

```
.
└── frame/
    ├── frame_type
    ├── padding_data (Padding)
    ├── largest_acknowledged (Ack)
    ├── ack_delay (Ack)
    ├── ack_range_count (Ack)
    ├── first_ack_range (Ack)
    ├── ack_range (Ack)
    ├── ecn_counts (Ack)
    ├── reset_stream_id (ResetStream)
    ├── stop_stream_id (StopSending)
    ├── max_data_stream_id (MaxStreamData)
    ├── blocked_stream_id (StreamDataBlocked)
    ├── stream_id (Stream)
    ├── reset_application_protocol_error_code (ResetStream)
    ├── stop_application_protocol_error_code (StopSending)
    ├── final_size (ResetStream)
    ├── crypto_offset (Crypto)
    ├── crypto_data_length (Crypto)
    ├── crypto_data (Crypto)
    ├── maximum_data (MaxData)
    ├── blocked_maximum_data (DataBlocked)
    ├── maximum_stream_data (MaxStreamData)
    ├── blocked_maximum_stream_data (StreamDataBlocked)
    ├── cumulative_maximum_streams (MaxStreams)
    ├── allowed_maximum_streams (StreamsBlocked)
    ├── retire_sequence_number (RetireConnectionId)
    ├── new_sequence_number (NewConnectionId)
    ├── path_challenge_data (PathChallenge)
    ├── path_response_data (PathResponse)
    ├── retire_prior_to (NewConnectionId)
    ├── connection_id_length (NewConnectionId)
    ├── connection_id (NewConnectionId)
    ├── stateless_reset_token (NewConnectionId)
    ├── stream_offset (Stream)
    ├── stream_data_length (Stream)
    ├── stream_data (Stream)
    ├── error_code (ConnectionClose)
    ├── triggered_frame_type (ConnectionClose)
    └── reason_phrase_length (ConnectionClo
```

- **AckRange**

```
.
└── ack_range (Ack)
    ├── gap (Ack)
    └── ack_range_length (Ack)
```

- **EcnCounts**

```
.
└── ecn_counts (Ack)
    ├── ect_0_count (Ack)
    ├── ect_1_count (Ack)
    └── ecn_ce_count (Ack)
```

> LIMITATION

- Due to specification, Retry Packet contains Retry Token and Retry Integrity Tag. Because of Retry Token length is not specified explicitly, these 2 fields are combined in one field RetryData
- Encoder from human to byte sequence generates sequence accordingly number value. E.g. if number in [0-63] 0-16383 it generates 1 byte sequence, if [0-16383] 0-16383 it generates 2 bytes sequence, and so on.

  Such note is presented here because specification doesn't explain a case when value is less than minimum interval value, e.g. value is 16, but the field length is 2 bytes.
- Quic frames can be dissected only for initial packets (for the rest packets TLS keys are required). Because of that, flow/session cannot be closed when CONNECTION_CLOSE is sent because of it is encrypted. Like that, quic flow/session is closed/released by timeout.

Session

TLS

> STATUS

Protocol	RFC	Status	Tags
TLS	rfc5246 (v1.2) rfc6066 (Extensions) rfc8446 (v1.3) rfc3749 (Compression methods) rfc3943 (LZS compression id) rfc5077 (New Session Ticket) rfc7301 (Application-Layer Protocol Negotiation Extension) rfc4492 (Elliptic Curve Cryptography (ECC)) rfc5289 (Ecdhe Cipher Suites) rfc2246 (v1.0) rfc4346 (v1.1) rfc6101 (ssl v3.0)	Partly	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Layer structure test

> PORTS

- 443 (**tcp**)
- 993 (**tcp**; IMAPS)
- 995 (**tcp**; POP3S)

> FIELDS

Name	Type	Length	Mask	Description
ecdh_cofactor	**byte-sequence**	**0**	ffffffffffffff	The field specifies the cofactor h = #E(Fq)/n, where #E(Fq) represents the number of points on the elliptic curve E defined over the field Fq (either Fp or F2^m). The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_cofactor_length	**uint8**	**1**	ffffffffffffff	The field specifies the cofactor h value length. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_order	**byte-sequence**	**0**	ffffffffffffff	The field specifies the order n of the base point. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_order_length	**uint8**	**1**	ffffffffffffff	The field specifies the order n of the base point value length. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_base	**byte-sequence**	**1**	ffffffffffffff	The field specifies the base point G on the elliptic curve. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_base_length	**uint8**	**1**	ffffffffffffff	The field specifies the base point G value length. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_curve_b	**byte-sequence**	**0**	ffffffffffffff	The b value of the elliptic curve. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_curve_b_length	**uint8**	**1**	ffffffffffffff	The b value of the elliptic curve length. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_curve_a	**byte-sequence**	**0**	ffffffffffffff	The a value of the elliptic curve. The field exists for explicit_prime or explicit_char2 curve_type.

ecdh_curve_a_length	**uint8**	**1**	ffffffffffffff	The a value of the elliptic curve length. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_curve	**byte-sequence**	**0**	ffffffffffffff	The field specifies the coefficients a and b of the elliptic curve E. The field exists for explicit_prime or explicit_char2 curve_type.
ecdh_k3	**byte-sequence**	**0**	ffffffffffffff	The exponents for the pentanomial representation x^m | x^k3 | x^k2 | x^k1 | 1 (such that k3 > k2 > k1). The field exists only for explicit_char2 curve_type and ec_pentanomial basis.
ecdh_k3_length	**uint8**	**1**	ffffffffffffff	The exponent k value length.
ecdh_k2	**byte-sequence**	**0**	ffffffffffffff	The exponents for the pentanomial representation x^m | x^k3 | x^k2 | x^k1 | 1 (such that k3 > k2 > k1). The field exists only for explicit_char2 curve_type and ec_pentanomial basis.
ecdh_k2_length	**uint8**	**1**	ffffffffffffff	The exponent k2 value length.
ecdh_k1	**byte-sequence**	**0**	ffffffffffffff	The exponents for the pentanomial representation x^m | x^k3 | x^k2 | x^k1 | 1 (such that k3 > k2 > k1). The field exists only for explicit_char2 curve_type and ec_pentanomial basis.
ecdh_k1_length	**uint8**	**1**	ffffffffffffff	The exponent k1 value length.
ecdh_k	**byte-sequence**	**0**	ffffffffffffff	The exponent k for the trinomial basis representation x^m | x^k | 1. The field exists for explicit_char2 curve_type and ec_trinomial basis.
ecdh_k_length	**uint8**	**1**	ffffffffffffff	The exponent k value length.
ecdh_basis	**uint8**	**1**	ffffffffffffff	The basis type. Possible values: ec_basis_trinomial (1), ec_basis_pentanomial (2). The field exists only for explicit_char2 curve_type.

ecdh_m	**uint16**	**2**	ffffffffffffff	The degree of the characteristic-2 field F2^m. The field exists only for explicit_char2 curve_type.
ecdh_prime	**byte-sequence**	**0**	ffffffffffffff	The odd prime defining the field Fp. The field exists only for explicit_prime curve_type.
ecdh_prime_length	**uint8**	**1**	ffffffffffffff	The odd prime value length. The field exists only for explicit_prime curve_type.
ecdh_signature	**byte-sequence**	**0**	ffffffffffffff	The ecdh signature.
ecdh_signature_length	**uint16**	**2**	ffffffffffffff	The length of ecdh signature field.
ecdh_signature_and_hash_algorithm	**uint16**	**2**	ffffffffffffff	The ecdh hash and signature algorithm pair.
named_curve	**uint16**	**2**	ffffffffffffff	The field specifies a recommended set of elliptic curve domain parameters. All those values of NamedCurve are allowed that refer to a specific curve.
curve_type	**uint8**	**1**	ffffffffffffff	The field identifies the type of the elliptic curve domain parameters.
fortezza_rs	**byte-sequence**	**128**	ffffffffffffff	Server random number for FORTEZZA KEA (Key Exchange Algorithm).
rsa_exponent	**byte-sequence**	**0**	ffffffffffffff	The public exponent of the server's temporary RSA key.
rsa_exponent_length	**uint16**	**2**	ffffffffffffff	The length of rsa exponent field.
rsa_modulus	**byte-sequence**	**0**	ffffffffffffff	The modulus of the server's temporary RSA key.
rsa_modulus_length	**uint16**	**2**	ffffffffffffff	The length of rsa modulus field.
dh_signature	**byte-sequence**	**0**	ffffffffffffff	The dh signature.
dh_signature_length	**uint16**	**2**	ffffffffffffff	The length of dh signature field.

dh_signature_and_hash_algorithm	**uint16**	**2**	fffffffffffffff	The dh hash and signature algorithm pair.
dh_ys	**byte-sequence**	**0**	fffffffffffffff	The server's Diffie-Hellman public value (g^X mod p).
dh_ys_length	**uint16**	**2**	fffffffffffffff	The server's Diffie-Hellman public value field length.
dh_g	**byte-sequence**	**0**	fffffffffffffff	The generator used for the Diffie-Hellman operation.
dh_g_length	**uint16**	**2**	fffffffffffffff	The generator field length.
fortezza_master_write_iv	**byte-sequence**	**24**	fffffffffffffff	This is the IV for the TEK used to encrypt the premaster secret.
fortezza_server_write_iv	**byte-sequence**	**24**	fffffffffffffff	The IV for the server write key.
fortezza_client_write_iv	**byte-sequence**	**24**	fffffffffffffff	The IV for the client write key.
fortezza_wrapped_server_write_key	**byte-sequence**	**12**	fffffffffffffff	This is the server's write key, wrapped by the TEK.
fortezza_wrapped_client_write_key	**byte-sequence**	**12**	fffffffffffffff	This is the client's write key, wrapped by the TEK.
fortezza_yc_signature	**byte-sequence**	**40**	fffffffffffffff	The tsignature of the KEA public key, signed with the client's DSS private key.
fortezza_rc	**byte-sequence**	**128**	fffffffffffffff	The client's Rc value for the KEA calculation.
fortezza_yc	**byte-sequence**	**0**	fffffffffffffff	The client's Yc value (public key) for the KEA calculation.
fortezza_yc_length	**uint8**	**1**	fffffffffffffff	The client's Yc value (public key) length.
ecdh_public_key	**byte-sequence**	**0**	fffffffffffffff	Client/Server Elliptic Curve Diffie-Hellman public value. For Server Key Exchange maessage that field exists only when curve_type has named_curve (**3**) value.

ecdh_public_key_length	**uint8**	**1**	ffffffffffffff	Client/Server Elliptic Curve Diffie-Hellman public value length. For Server Key Exchange maessage that field exists only when curve_type has named_curve(**3**) value.
ecdhe_public_key	**byte-sequence**	**0**	ffffffffffffff	Client Elliptic Curve Ephemeral Diffie-Hellman public value.
ecdhe_public_key_length	**uint8**	**1**	ffffffffffffff	Client Elliptic Curve Ephemeral Diffie-Hellman public value length.
dhe_public_key	**byte-sequence**	**0**	ffffffffffffff	Client Ephemeral Diffie-Hellman public value.
dhe_public_key_length	**uint16**	**2**	ffffffffffffff	Client Ephemeral Diffie-Hellman public value length
dh_public_key	**byte-sequence**	**0**	ffffffffffffff	Client Diffie-Hellman public value.
dh_public_key_length	**uint16**	**2**	ffffffffffffff	Client Diffie-Hellman public value length.
premaster_key	**byte-sequence**	**0**	ffffffffffffff	The value which client generates and sends as encrypted premaster secret message. The field exists only for RSA key agreement.
premaster_key_length	**uint16**	**2**	ffffffffffffff	The length of premaster key.
certificate	**byte-sequence**	**0**	ffffffffffffff	The certificate data.
certificate_length	**byte-sequence**	**3**	ffffffffffffff	The length of certificate.
certificate_list	**byte-sequence**	**0**	ffffffffffffff	The certificate list data. The certificate list can contain more than one certificate.
rsa_modulus_length	**uint16**	**2**	ffffffffffffff	The length of rsa modulus field.
dh_signature	**byte-sequence**	**0**	ffffffffffffff	The dh signature.
certificate_list_length	**byte-sequence**	**3**	ffffffffffffff	The length of certificate list field.
message_hash_data	**byte-sequence**	**0**	ffffffffffffff	The data section of message hash handshake protocol.

session_ticket	**byte-sequence**	**0**	ffffffffffffff	The session ticket field.
session_ticket_length	**uint16**	**2**	ffffffffffffff	The length of session ticket field.
session_ticket_lifetime	**uint32**	**4**	ffffffffffffff	Indicates the lifetime in seconds as a 32-bit unsigned integer in network byte order from the time of ticket issuance.
sha_hash	**byte-sequence**	**20**	ffffffffffffff	The part of finished message. The field can exist only for ssl v3.0 sessions. The length is fixed.
md5_hash	**byte-sequence**	**16**	ffffffffffffff	The part of finished message. The field can exist only for ssl v3.0 sessions. The length is fixed.
verify_data	**byte-sequence**	**0**	ffffffffffffff	The part of finished message. For tls v1.0 the length is fixed.
request_update	**uint8**	**1**	ffffffffffffff	If the request_update field is set to update_requested (0), then the receiver MUST send a KeyUpdate of its own with request_update set to update_not_requested (1) prior to sending its next Application Data record.
signature	**byte-sequence**	**0**	ffffffffffffff	A digital signature using algorithms over the contents of the element.
signature_length	**uint16**	**2**	ffffffffffffff	The length of signature field.
signature_and_hash_algorithm	**uint16**	**2**	ffffffffffffff	The hash and signature algorithm pair.
signature_and_hash_algorithms	**byte-sequence**	**0**	ffffffffffffff	Signature and hash algorithm elements.
signature_and_hash_algorithms_length	**uint16**	**2**	ffffffffffffff	The length of signature and hash algorithms field.
signature_scheme	**uint16**	**2**	ffffffffffffff	The field specifies hash and signature algorithm. The field can exist only for tls v1.3 sessions.
quic_transport_parameter_value	**byte-sequence**	**0**	ffffffffffffff	The quic transport parameter value.

quic_transport_parameter_length	**byte-sequence**	**0**	fffffffffffff	The field contains the length of the Transport Parameter Value field in bytes.
quic_transport_parameter_id	**byte-sequence**	**0**	fffffffffffff	The identificator of quic transport parameter.
quic_transport_parameter	**byte-sequence**	**0**	fffffffffffff	The quic transport parameter section.
supported_version	**uint16**	**2**	fffffffffffff	A supported version.
supported_versions	**byte-sequence**	**0**	fffffffffffff	The list of supported versions in preference order, with the most preferred version first.
supported_versions_length	**uint8**	**1**	fffffffffffff	The length of supported versions
protocol_name	**ascii-string**	**0**	fffffffffffff	The protocol name string.
protocol_name_length	**uint8**	**1**	fffffffffffff	The length of protocol name.
protocol_name_list	**byte-sequence**	**0**	fffffffffffff	The list contains the list of protocols advertised by the client, in descending order of preference.
server_name	**ascii-string**	**0**	fffffffffffff	The server name string.
server_name_length	**uint16**	**2**	fffffffffffff	The length of server name.
server_name_type	**uint8**	**1**	fffffffffffff	The type of server name: **0** (host_name), **255**.
server_name_list	**byte-sequence**	**0**	fffffffffffff	The list of server name elements.
server_name_list_length	**uint16**	**2**	fffffffffffff	The length of server name list
extension_type	**uint16**	**2**	fffffffffffff	The field identifies the particular extension type. A part of extension header. (Client/Server handshake header field)
extension_length	**uint16**	**2**	fffffffffffff	The length of extension data. (Client/Server handshake header field)

extension	**byte-sequence**	**0**	ffffffffffffff	Extension record/unit.
extensions_length	**uint16**	**2**	ffffffffffffff	The length of extensions field.
extensions	**byte-sequence**	**0**	ffffffffffffff	A list of extensions. Clients MAY request extended functionality from servers by sending data in the extensions field. Note that only extensions offered by the client can appear in the server's list. (Client/Server handshake header field)
compression_method	**uint8**	**1**	ffffffffffffff	For client: an element of compression methods. For server: the single compression algorithm selected by the server from the client compression method list. (Client/Server handshake header field)
compression_methods	**uint-8-arraysequence**	**0**	ffffffffffffff	This is a list of the compression methods supported by the client, sorted by client preference. (Client handshake header field)
compression_methods_length	**uint8**	**1**	ffffffffffffff	The length of compression methods field. (Client handshake header field)
cipher_suite	**uint16**	**2**	ffffffffffffff	For client: an element of cipher suites. For server: the single cipher suite selected by the server from the client cipher suite list. (Client/Server handshake header field)
cipher_suites	**uint-16-array**	**0**	ffffffffffffff	This is a list of the cryptographic options supported by the client, with the client's first preference first. (Client handshake header field)
cipher_suites_length	**uint16**	**2**	ffffffffffffff	The length of cipher suites field. (Client handshake header field)
session_id	**byte-sequence**	**0**	ffffffffffffff	Id of the session corresponding to this connection. (Client/Server handshake header field)
session_id_length	**uint8**	**1**	ffffffffffffff	The length of session id. (Client/Server handshake header field)

random_bytes	**byte-sequence**	**28**	ffffffffffffffff	28 bytes generated by a secure random number generator. (Client/Server handshake header field)
random_gmt_unix_time	**uint32**	**4**	ffffffffffffffff	The current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, UTC, ignoring leap seconds) according to the sender's internal clock. (Client/Server handshake header field)
random	**byte-sequence**	**32**	ffffffffffffffff	A client/server generated random structure. The structure which is generated by the server MUST be independently generated from the ClientHello.random. (Client/Server handshake header field)
server_minor_version	**uint8**	**1**	ffffffffffffffff	The minor number of server TLS protocol.
server_major_version	**uint8**	**1**	ffffffffffffffff	The major number of server TLS protocol.
client_minor_version	**uint8**	**1**	ffffffffffffffff	The minor number of client TLS client protocol.
client_major_version	**uint8**	**1**	ffffffffffffffff	The major number of client TLS protocol.
client_version	**uint16**	**2**	ffffffffffffffff	The version of the TLS protocol by which the client wishes to communicate during this session.
handshake_message	**byte-sequence**	**0**	ffffffffffffffff	The handshake message.
handshake_message_length	**32-bit-field**	**4**	ffffff	The length of handshake message.
handshake_type	**32-bit-field**	**4**	ff000000	The handshake message type: **0** (hello_request), **1** (client_hello), **2** (server_hello), **11** (certificate), **12** (server_key_exchange), **13** (certificate_request), **14** (server_hello_done), **15** (certificate_verify), **16** (client_key_exchange), **20** (finished), **255**.

handshake_header	**byte-sequence**	**4**	fffffffffffffff	The header of handshake protocol. The TLS Handshake Protocol is one of the defined higher-level clients of the TLS Record Protocol. This protocol is used to neg otiate the secure attributes of a session. Handshake messages are supplied to the TLS record layer, where they are encapsulated within one or more TLSPlaintext structures, which are processed and transmitted as specified by the current active session state.
change_cipher_spec_type	**uint8**	**1**	fffffffffffffff	The change cipher spec protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) connection state. The message consists of a single byte of value 1.
alert_description	**uint8**	**1**	fffffffffffffff	Alert message description.
alert_level	**uint8**	**1**	fffffffffffffff	Alert message level.
heartbeat_padding	**byte-sequence**	**0**	fffffffffffffff	The padding is random content that MUST be ignored by the receiver. The padding_length MUST be at least 16.
heartbeat_payload	**byte-sequence**	**0**	fffffffffffffff	The payload consists of arbitrary content.
heartbeat_payload_length	**uint16**	**2**	fffffffffffffff	The length of the payload.
heartbeat_message_type	**uint8**	**1**	fffffffffffffff	The message type, either heartbeat_request (**1**) or heartbeat_response (**2**).
record_message	**byte-sequence**	**0**	fffffffffffffff	The application data. This data is transparent and treated as an independent block to be dealt with by the higher-level protocol specified by the type field.

record_message_length	**uint16**	**2**	ffffffffffffff	The length (in bytes) of the following TLSPlaintext.fragment. The length MUST NOT exceed 2^14.
record_protocol_minor_version	**uint8**	**1**	ffffffffffffff	The minor number of protocol version.
record_protocol_major_version	**uint8**	**1**	ffffffffffffff	The major number of protocol version.
record_protocol_version	**uint16**	**2**	ffffffffffffff	The version of the protocol being employed.
dh_p	**byte-sequence**	**0**	ffffffffffffff	The prime modulus used for the Diffie-Hellman operation.
record_content_type	**uint8**	**1**	ffffffffffffff	The higher-level protocol used to process the enclosed fragment/ message.
dh_p_length	**uint16**	**2**	ffffffffffffff	The prime modulus field length.
record	**byte-sequence**	**0**	ffffffffffffff	Record layer.
server_version	**uint16**	**2**	ffffffffffffff	This field will contain the lower of that suggested by the client in the clienthello and the highest supported by the server.
protocol_name_list_length	**uint16**	**2**	ffffffffffffff	The length of handshake message.
fortezza_encrypted_pre_master_secret	**byte-sequence**	**48**	ffffffffffffff	A random value, generated by the client and used to generate the master secret.
root	**uint8**	**1**	ffffffffffffff	Payload data.

> FIELD TREE

- **Root**

```
.
└── record/
    ├── record_content_type
    ├── record_protocol_version
    ├── record_protocol_major_version
    ├── record_protocol_minor_version
    ├── record_message_length
    └── record_message/
        ├── [Heartbeat]
        ├── heartbeat_message_type
        ├── heartbeat_payload_length
        ├── heartbeat_payload
        ├── heartbeat_padding
        ├── [Certificate Verify]
        ├── signature_scheme
        ├── [Change Cipher Spec Type]
        ├── change_cipher_spec_type
        ├── [Alert]
        ├── alert_level
        ├── alert_description
        ├── [Handshake]
        ├── handshake_header
        ├── handshake_type
        ├── handshake_message_length
        └── handshake_message
```

- **Handshake Message**

```
.
└── handshake_message/
    ├── [ClientHello only]
    ├── client_version
    ├── client_major_version
    ├── client_minor_version
    ├── [ServerHello only]
    ├── server_version
    ├── server_major_version
    ├── server_minor_version
    ├── [ClientHello and ServerHello]
    ├── random
    ├── random_gmt_unix_time
    ├── random_bytes
    ├── session_id_length
```

```
├── session_id
├── cipher_suites_length
├── cipher_suites
├── cipher_suite
├── compression_methods_length
├── compression_methods
├── compression_method
├── extensions (also can belong Handshake Enctypted Extensions message)
└── extensions_length
```

- **Extensions**

```
.
└── extensions/
    ├── extensions_length
    └── extension/
        ├── extension_type
        ├── extension_length
        └── ... (extension related fields)
```

or

```
.
└── extensions/
    ├── extensions_length
    ├── extension_type
    └── extension_length
```

when extension size cannot be defined.

- **Quic Transport Parameter extension**

```
.
└── quic_transport_parameter/
    ├── quic_transport_parameter_id
    ├── quic_transport_parameter_length
    └── quic_transport_parameter_value
```

- **EcDhCurve**

```
.
└── ecdh_curve/
    ├── ecdh_curve_a_length
    ├── ecdh_curve_a
    ├── ecdh_curve_b_length
    └── ecdh_curve_b
```

> SUB-PROTOCOL SUPPORT LIST

- [x] Alert protocol
- [x] Application Data protocol
- [x] Heartbeat protocol
- [ ] Handshake protocol
  - [x] HelloRequest
  - [x] ClientHello
  - [x] ServerHello
  - [x] NewSessionTicket (rfc5077)
  - [x] EndOfEarlyData (rfc8446)
  - [x] EncryptedExtensions (rfc8446)
  - [ ] Certificate
  - [x] ServerKeyExchange
  - [ ] CertificateRequest (1.2 and 1.3 have differences)
  - [x] ServerHelloDone
  - [x] CertificateVerify
  - [x] ClientKeyExchange
  - [x] Finished
  - [x] KeyUpdate (rfc8446)
  - [x] MessageHash (rfc8446)
- [x] Change Cipher Spec protocol
- [ ] Extensions
  - [x] ServerName
  - [ ] MaxFragmentLength
  - [ ] ClientCertificateUrl
  - [ ] TrustedCaKeys
  - [ ] TrustedHmac
  - [ ] StatusRequest
  - [ ] SupportedGroups

- [ ] SignatureAlgorithms
- [ ] UseSrtp
- [ ] Heartbeat
- [ ] ApplicationLayerProtocolNegotiation
- [ ] SignedCertificateTimestamp
- [ ] ClientCertificateType
- [ ] ServerCertificateType
- [ ] Padding
- [ ] Reserved0
- [ ] PreSharedKey
- [ ] EarlyData
- [x] SupportedVersions
- [ ] Cookie
- [ ] PskKeyExchangeModes
- [ ] Reserved1
- [ ] CertificateAuthorities
- [ ] OidFilters
- [ ] PostHandshakeAuth
- [ ] SignatureAlgorithmsCert
- [ ] KeyShare

- [ ] Not enumerated extensions (have to be found in rfc):
  - [ ] EcPointFormats
  - [ ] SignedCertificateTimestamp
  - [ ] RenegotiationInfo
  - [ ] SessionTicket
  - [ ] NextProtocolNegotiation
  - [ ] ExtendMasterSecret

> LIMITATION

- TLSPGP is not supported (rfc50810)
- New Session Ticket is implemented related to rfc5077 (rfc8446 has a different structure)

> NOTES

- CertificateVerify message contains SignatureAndHashAlgorithm (rfc5246) field which is the same with SignatureScheme (rfc8446). Since that field is different between tls1.2 and tls1.3 - it has to be interpreted depending on flow context (tls version). SignatureAndHashAlgorithm::HashAlgorithm and SignatureAndHashAlgorithm::SignatureAlgorithm are not used in dissection to save that field universal for 1.2 and 1.3 versions.

- CertificateRequest and Certificate message dissection are not supported. For extracting base fields of certificates - use tls_certificate in-built extension.

Presentation

None

Application

Telnet

> STATUS

Protocol	RFC	Status	Tags
Telnet	rfc854	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based

> PORTS

- 23 (tcp)

> FIELDS

Name	Type	Length	Mask	Description
data	**byte-sequence**	**0**	ffffffffffffffff	Stream data.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

DNS

> STATUS

Protocol	RFC	Status	Tags
DNS	rfc1035 rfc3596 rfc2874	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Layer structure test

> PORTS

- 53 (udp/tcp)

> FIELDS

Name	Type	Length	Mask	Description
qclass	**uint16**	**2**	ffffffffffffffff	A two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.
rdata_class	**uint16**	**2**	ffffffffffffffff	Specifies the class of the data in the rdata.
prefix_name	**byte-sequence**	**0**	ffffffffffffffff	The name of the prefix, encoded as a domain name.
address_suffix	**byte-sequence**	**16**	ffffffffffffffff	An IPv6 address suffix, encoded in network order (high-order octet first).
prefix_length	**uint8**	**1**	ffffffffffffffff	A prefix length, encoded as an eight-bit unsigned integer with value between 0 and 128 inclusive.
aaaa	**byte-sequence**	**16**	ffffffffffffffff	A **128** bit IPv6 address in network byte order (high-order byte first).

bit_mask	**byte-sequence**	**0**	ffffffffffffffff	Bit map has one bit per port of the specified protocol.
protocol	**uint8**	**1**	ffffffffffffffff	IP protocol number.
wks_address	**uint32**	**4**	ffffffffffffffff	Internet address.
os	**ascii-string**	**0**	ffffffffffffffff	A character-string which specifies the operating system type.
os_length	**uint8**	**1**	ffffffffffffffff	OS string length.
cpu	**ascii-string**	**0**	ffffffffffffffff	A character-string which specifies the cpu type.
cpu_length	**uint8**	**1**	ffffffffffffffff	CPU string length.
null_data	**byte-sequence**	**0**	ffffffffffffffff	Any data. Null section.
address	**uint32**	**4**	ffffffffffffffff	Internet address.
minimum	**uint32**	**4**	ffffffffffffffff	Minimum ttl field that should be exported with any RR from this zone.
expire	**uint32**	**4**	ffffffffffffffff	Time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.
retry	**uint32**	**4**	ffffffffffffffff	Time interval that should elapse before a failed refresh should be retried.
refresh	**uint32**	**4**	ffffffffffffffff	Time interval before the zone should be refreshed.
serial	**uint32**	**4**	ffffffffffffffff	Version number of the original copy of the zone. zone transfers preserve this value.
rname	**byte-sequence**	**0**	ffffffffffffffff	A domain name which specifies the mailbox of the person responsible for this zone.
rmailbx	**byte-sequence**	**0**	ffffffffffffffff	A domain name which specifies a mailbox which is responsible for the mailing list or mailbox.

exchange	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a host willing to act as mail exchange for the owner name.
preference	**uint16**	**2**	ffffffffffffff	Specifies the preference given to this RR among others at the same owner.
ptrdname	**byte-sequence**	**0**	ffffffffffffff	A domain name which points to some location in the domain name space.
newname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a mailbox which is the proper rename of the specified mailbox.
mgname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a mailbox which is a member of the mail group specified by the domain name.
cname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies the canonical or primaryname for the owner.
mf_madname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a host which has a mail agent for the domain which will accept mail for forwarding to the domain.
md_madname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a host which has a mail agent for the domain which should be able to deliver mail for the domain.
mb_madname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a host which has the specified mailbox.
nsdname	**byte-sequence**	**0**	ffffffffffffff	A domain name which specifies a host which should be authoritative for the specified class and domain.
rdata	**byte-sequence**	**0**	ffffffffffffff	A variable length string of octets that describes the resource.
rd_length	**uint16**	**2**	ffffffffffffff	Specifies the length in octets of the rdata.
ttl	**uint32**	**4**	ffffffffffffff	Specifies the time interval (in seconds) that the resource record may be cached before it should be discarded.

rdata_type	**uint16**	**2**	ffffffffffffffff	Specifies the meaning of the data in the rdata.
domain_name _offset	**uint16**	**2**	3fff	A domain name offset.
domain_name _pointer	**uint16**	**2**	ffffffffffffffff	A domain name pointer.
domain_name _label	**ascii-string**	**0**	ffffffffffffffff	A domain name label.
domain_name _label_length	**uint8**	**1**	ffffffffffffffff	A domain name label length.
domain_name	**byte-sequence**	**0**	ffffffffffffffff	A domain name to which this resource record pertains.
authority_records	**byte-sequence**	**0**	ffffffffffffffff	Authority records section.
answers	**byte-sequence**	**0**	ffffffffffffffff	Answer section.
qtype	**uint16**	**2**	ffffffffffffffff	A two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.
qname	**byte-sequence**	**0**	ffffffffffffffff	A domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.
query	**byte-sequence**	**0**	ffffffffffffffff	Query record.
queries	**byte-sequence**	**0**	ffffffffffffffff	Question section.
arcount	**uint16**	**2**	ffffffffffffffff	An unsigned 16 bit integer specifying the number of resource records in the additional records section.

nscount	**uint16**	**2**	ffffffffffffffff	An unsigned 16 bit integer specifying the number of name server resource records in the authority records section.
ancount	**uint16**	**2**	ffffffffffffffff	An unsigned 16 bit integer specifying the number of resource records in the answer section.
qdcount	**uint16**	**2**	ffffffffffffffff	An unsigned 16 bit integer specifying the number of entries in the question section.
rcode	**uint16**	**2**	f	Response code - this 4 bit field is set as part of responses. The values have the following interpretation: 0 (No error condition), 1 (Format error - The name server was unable to interpret the query), 2 (Server failure - The name server was unable to process this query due to a problem with the name server), 3 (Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist), 4 (Not Implemented - The name server does not support the requested kind of query), 5 (Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g. zone transfer) for particular data), 6-15 (Reserved for future use).
z	**uint16**	**2**	70	Reserved for future use. Must be zero in all queries and responses.
ra	**uint16**	**2**	80	Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.

rd	**uint16**	**2**	100	Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.
tc	**uint16**	**2**	200	TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
aa	**uint16**	**2**	400	Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.
opcode	**uint16**	**2**	7800	A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are: **0** (standard query), **1** (inverse query), **2** (server status request), **3-15** (reserved for future use).
mname	**byte-sequence**	**0**	ffffffffffffffff	A domain name of the name server that was the original or primary source of data for this zone.
qr	**uint16**	**2**	8000	Bit specifies message type. Query (**0**), response (**1**).
txt	**ascii-string**	**0**	ffffffffffffffff	One or more character string(s). are used to hold descriptive text. the semantics of the text depends on the domain where it is found.
id	**uint16**	**2**	ffffffffffffffff	A **16** bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.
record	**byte-sequence**	**0**	ffffffffffffffff	Resource record
txt_length	**uint8**	**1**	ffffffffffffffff	Txt string length.

dns_message_length	**uint16**	**2**	ffffffffffffffff	Dns message length - is presented only for tcp transport.
additional_records	**byte-sequence**	**0**	ffffffffffffffff	Additional records section.
emailbx	**byte-sequence**	**0**	ffffffffffffffff	A domain name which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR.
root	**uint8**	**1**	ffffffffffffffff	Payload data.

> FIELD TREE

- **Queries**

```
.
└── queries/
    ├── qname/
    |   ├── label-0
    |   ├── label-1
    |   ├── ...
    |   ├── qtype
    |   └── qclass
    └── qname/
        ├── label-0
        ├── ...
        ├── qtype
        └── qclass
```

- **Resource Records**

```
.
└── answers, authority_records, additional_records,
    └── record/
        ├── domain_name/
        |   ├── domain_name_label_length
        |   ├── domain_name_label
        |   ├── domain_name_pointer
        |   └── domain_name_offset
        ├── rdata_type
        ├── rdata_class
        ├── ttl
        ├── rd_length
        └── rdata
```

- **Queries**

RData (Mb, Md, Mf, CName, Mg, Mr, Ptr)

```
.
└── rdata/
    └── mb_madname, md_madname, mf_madname, cname, mgname, newname, ptrdname/
        ├── domain_name_label_length
        ├── domain_name_label
        ├── domain_name_pointer
        └── domain_name_offset
```

RData Mx

```
.
└── rdata/
    ├── preference
    └── exchange/
        ├── domain_name_label_length
        ├── domain_name_label
        ├── domain_name_pointer
        └── domain_name_offset
```

RData MInfo

```
.
└── rdata/
    ├── rmailbx/
    |   ├── domain_name_label_length
    |   ├── domain_name_label
    |   ├── domain_name_pointer
    |   └── domain_name_offset
    ├── emailbx/
    |   ├── domain_name_label_length
    |   ├── domain_name_label
    |   ├── domain_name_pointer
    |   └── domain_name_offset
```

RData Txt

```
.
└── rdata/
    ├── txt_length
    └── txt
```

RData Soa

```
.
└── rdata/
    ├── mname
    ├── rname
    ├── serial
    ├── refresh
    ├── Retry
    ├── expire
    └── minimum
```

RData A

```
.
└── rdata/
    └── address
```

RData Null

```
.
└── rdata/
    └── null_data
```

RData HInfo

```
.
└── rdata/
    ├── cpu_length
    ├── cpu
    ├── os_length
    └── os
```

RData Wks

```
.
└── rdata/
    ├── wks_address
    ├── protocol
    └── bit_mask
```

AckRange

```
.
└── ack_range (Ack)
    ├── gap (Ack)
    └── ack_range_length (Ack)
```

EcnCounts

```
.
└── ecn_counts (Ack)
    ├── ect_0_count (Ack)
    ├── ect_1_count (Ack)
    └── ecn_ce_count (Ack)
```

> NOTES

- Resource Records have **Name** field in rfc1035. Our engine uses **DomainName** field.
- **NsDname**, **MadName**, **CName**, **MgName**, **NewName**, **PtrDname** have domain name structure. It means that fields have children fields such as **DomainNameLabelLength**, **DomainNameLabel**, **DomainNamePointer**, **DomainNameOffset**.
- Due to rfc1035, MadName field is presented in the following RData sections: **Mb**, **Md**, **Mf**. To don't use the same name for different RData sections and don't check the parent objects to detect a type, our engine uses **MbMadName**, **MdMadName**, **MfMadName** field names.
- The following malformed reasons are suitable for all fields which have **domain-name** structure:
  - DnsDomainNameHasInvalidFormat
  - DnsDomainNameFirstOctetCannotBeDissected
  - DnsDomainNamePointerCannotBeDissected
  - DnsResourceRecordHeaderCannotBeDissected
  - DnsResourceDataLengthExceedDataLength
  - DnsDomainNameLabelLengthExceedDataLength
  - DnsDomainNameOffsetExceedDataLength

MDNS

> STATUS

Protocol	RFC	Status	Tags
MDNS	rfc6762	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Layer structure test

> PORTS

- 5353 (udp/tcp)

> FIELDS

All **DNS fields** are valid for **MDNS** as well. But there are new fields which are presented below.

Name	Type	Length	Mask	Description
unicast_response	**16-bit-field**	**2**	8000	1 bit unicast response flag. When this bit is set in a question, it indicates that the querier is willing to accept unicast replies in response to this specific query, as well as the usual multicast responses.
cache_flush	**16-bit-field**	**2**	8000	Announcements to flush outdated cache entries.

> FIELD TREE

All **DNS field tree structures** are also valid for **MDNS**. The main difference is unicast_response is used instead of qclass and cache_flush is used instead of qclass

> NOTES

- From the dissection point of view, **MDNS** is almost the same as **DNS**. All **DNS** notes belong to **MDNS** as well.

HTTP

> STATUS

Protocol	RFC	Status	Tags
HTTP	rfc1945(HTTP/1.0) rfc2626(HTTP/1.1) rfc7231(HTTP/1.1)	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Patterns
- Layer structure test

> PORTS

- 80 (tcp)

> PATTERNS

HTTP request methods are used for protocol pattern detection:

- GET
- POST
- HEAD
- PUT
- DELETE
- CONNECT
- OPTIONS
- TRACE
- COPY
- LOCK
- MKCOL
- MOVE
- PROPFIND
- PROPPATCH
- SEARCH
- UNLOCK
- BIND
- REBIND
- UNBIND
- ACL
- REPORT
- MKACTIVITY
- CHECKOUT
- MERGE
- PATCH
- PURGE
- MKCALENDAR
- LINK
- UNLINK
- SOURCE

> FIELDS

Name	Type	Length	Mask	Description
trailer	**ascii-string**	**0**	ffffffffffffff	The trailer field allows the sender to include additional HTTP header fields at the end of the message.
chunk_data	**ascii-string**	**0**	ffffffffffffff	The data part of chunk.
chunk_extension	**ascii-string**	**0**	ffffffffffffff	The part of chunk size line. Optional field.
chunk_size	**ascii-string**	**0**	ffffffffffffff	The string of hex digits indicating the size of the chunk.
chunk	**ascii-string**	**0**	ffffffffffffff	The part of HTTP body. The field is presented when Transfer-Encoding header has 'chunked' value.
body	**ascii-string**	**0**	ffffffffffffff	HTTP message body.
header	**ascii-string**	**0**	ffffffffffffff	An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. The fields pass additional context and metadata about the request or response.
version	**ascii-string**	**0**	ffffffffffffff	The version of an HTTP message.
reason_phrase	**ascii-string**	**0**	ffffffffffffff	The part of HTTP response status line which describes status code.
status_code	**ascii-string**	**0**	ffffffffffffff	The part of HTTP response status line which is presented as a 3-digit integer number of the attempt to understand and satisfy the request.
uri	**ascii-string**	**0**	ffffffffffffff	The part of HTTP request line which describes the exact location of a page, post, file, or other asset.
method	**ascii-string**	**0**	ffffffffffffff	The method token indicates the method to be performed on the resource identified by the Request-URI.
root	**uint8**	**1**	ffffffffffffff	Payload data.

> LAYER DETECTION METHODS

- HTTP **1.0** RFC suggests few default request methods. HTTP **1.1** RFC has a bit more default methods. Since each of specification allows to extend HTTP method - dissection process doesn't validate method name is suitable for specific HTTP version.
- HTTP **1.0** has Simple-Response format of response. It doesn't have any HTTP RFC patterns and requires to cache HTTP request. If client sends Simple-Request server must to reply with Simple-Response. Packet library doesn't cache any data and because of that such answers cannot be properly dissected.
- HTTP **1.1** requires to have Host header in requests. Packet library doesn't check that. It dissects just a structure of HTTP message.
- Packet library strongly follows RFC. If RFC describes only 1 **SP** char between tokens - dissection process will expect only 1 **SP**. Not duplicated, not any count of **LWS**.
- If Content-Length has invalid value - the rest of data is dissected as HTTP Body.
- Calling GetNextLayer of HTTP layer doesn't dissect a layer, but it doesn't validate chunk-extension and trailer parts of chunked body. First HTTP line and HTTP header are validating.
- Dissect expects only one space separator in HTTP first line, but GetNextLayer allows any space char count between tokens.
- HTTP Body decode is not supported.

SSDP

> STATUS

Protocol	RFC	Status	Tags
Ssdp	draft-cai-ssdp-v1-02 draft-cai-ssdp-v1-03	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based
- Patterns
- Layer structure test

> PORTS

- 1900 (udp)

> PATTERNS

SSDP request methods are used for protocol pattern detection:

- M-SEARCH
- NOTIFY
- SUBSCRIBE
- SSDPC

> FIELDS

All **HTTP fields** are valid for SSDP as well.

Dropbox

> STATUS

Protocol	RFC	Status	Tags
Dropbox Lan Sync Dropbox Lan Sync Discovery	Absent	Fully	basic, network, internet

> LAYER DETECTION METHODS

- Port-based

> PORTS

- 17500 (udp/tcp)

> FIELDS

Name	Type	Length	Mask	Description
data	**byte-sequence**	**0**	ffffffffffffffff	Data payload (it should have json format with 'host_int', 'version', 'displayname', 'port', 'namespaces' fields).
root	**uint8**	**1**	ffffffffffffffff	Payload data.

slinkin.tech