

## Packet Header Description Example: QUIC

Packet Header Description Language	Type Definitions	Parsing Code
<pre>packet_type := bit[7]; version := bit[32]; cid_len := bit[4]; full_packet_num := bit[62]; frame_type := bit[8];</pre>	<pre>typedef bit[7] packet_type; typedef bit[32] version; typedef bit[4] cid_len; typedef bit[62] full_packet_num; typedef bit[8] frame_type; typedef bit[] cryptobits;</pre>	<pre>parsePacketType :: (bits : bit[]) -&gt; packet_type {     return bits.parseBits(7); }  parseVersion :: (bits : bit[]) -&gt; packet_type {     return bits.parseBits(32); }  parseCidLen :: (bits : bit[]) -&gt; packet_type {     return bits.parseBits(4); }  parseFullPacketNum :: (bits : bit[]) -&gt; packet_type {     return bits.parseBits(62); }  parseFrameType :: (bits : bit[]) -&gt; packet_type {     return bits.parseBits(8); }  parseCryptobits :: (bits : bit[]) -&gt; cryptobits {     return bits.parseBits(len(bits)); }</pre>
<pre>var_enc := {     length : bit[2];     value : bit[]; } where {     value.width = (2^length * 8) - 2; }</pre>	<pre>typedef struct var_enc {     bit    length[2];     bit    value[] } var_enc;</pre>	<pre>parseVarEnc :: (bits : bit[]) -&gt; var_enc {     ve : packet_num;     ve.length = bits.parseBits(2);     ve.value = bits.parseBits(len(bits)-2);     return ve; }</pre>

<pre> packet_num :=   '0' followed by packet_number : bit[7]   '10' followed by packet_number : bit[14]   '11' followed by packet_number : bit[30]; </pre>	<pre> typedef enum packet_num {bit[7],                         bit[14],                         bit[30]} packet_num; </pre>	<pre> parsePacketNum :: (bits : bit[]) -&gt; packet_num {   first_bit : bit;   first_bit = bits.parseBits(1);   if (first_bit == '0') {     return bits.parseBits(7);   } else if (first_bit == '1') {     second_bit : bit;     second_bit = bits.parseBits(1);     if (second_bit == '0') {       return bits.parseBits(14);     } else {       return bits.parseBits(30);     }   } } </pre>
<pre> decrypt :: (enc_payload : cryptobits[],            pn : full_packet_num) -&gt; bit[]; </pre>		<pre> decrypt :: (enc_payload : cryptobits[],            pn : full_packet_num) -&gt; bit[]; </pre>
<pre> long_hdr := {   header      : bit;   type        : packet_type;   ver         : version;   dcid_len    : cid_len;   scid_len    : cid_len;   dcid        : bit[];   scid        : bit[];   payload_length : var_enc;   packet_number : packet_num;   payload     : bit[]; } where {   header_type = 1;   dcid.width = (dcid_len == 0) ? 0 : (dcid_len+3) * 8;   scid.width = (scid_len == 0) ? 0 : (scid_len+3) * 8;   payload.width = 2^payload_length; } onparse {   context.scid_len = scid_len; } </pre>	<pre> typedef struct long_hdr {   bit      header_type;   packet_type type;   version  ver;   cid_len  dcid_len;   cid_len  scid_len;   bit      dcid[];   bit      scid[];   var_enc  payload_length;   packet_num packet_number;   bit      payload[]; } long_hdr; </pre>	<pre> parseLongHdr :: (bits : bit[]) -&gt; long_hdr {   lh : long_hdr;   lh.header_type = bits.parseBits(1);   if (lh.header_type != '1') {     raise ParserException;   }   lh.type = parsePacketType(bits);   lh.ver = parseVersion(bits);   lh.dcid_len = parseCidLen(bits);   lh.scid_len = parseCidLen(bits);   lh.dcid = bits.parseBits(lh.dcid_len == '0' : 0 ?                           (lh.dcid_len+3)*8);   lh.scid = bits.parseBits(lh.scid_len == '0' : 0 ?                           (lh.scid_len+3)*8);   lh.payload_length = parseVarEnc(bits);   lh.packet_number = parsePacketNum(bits);   lh.payload = bits.parseBits(2^lh.payload_length);   context.scid_len = scid_len;   return lh; } </pre>

<pre> short_hdr := {   header_type      : bit;   key_phase        : bit;   third_bit        : bit;   forth_bit        : bit;   google_demux     : bit;   reserved         : bit[3];   dcid             : bit[];   packet_number    : packet_num;   protected_payload : cryptobit[] -&gt; (payload : frame[]); } where {   header_type = 0;   third_bit = 1;   forth_bit = 1;   google_demux = 0;   dcid.width = (context.scid_len == 0) ? 0 :                (context.scid_len+3) * 8; } onparse {   payload = decrypt(protected_payload, packet_number); } </pre>	<pre> typedef struct short_hdr {   bit      header_type;   bit      key_phase;   bit      third_bit;   bit      forth_bit;   bit      google_demux;   bit      reserved[3];   bit      dcid[];   packet_num packet_number;   cryptobit protected_payload[];   frame      payload[]; } short_hdr; </pre>	<pre> parseShortHdr :: (bits : bit[]) -&gt; short_hdr {   sh : short_hdr;   sh.header_type = bits.parseBits(1);   if (sh.header_type != '0') {     raise ParserException;   }   sh.key_phase = bits.parseBits(1);   sh.third_bit = bits.parseBits(1);   if (sh.third_bit != '1') {     raise ParserException;   }   sh.forth_bit = bits.parseBits(1);   if (sh.forth_bit != '1') {     raise ParserException;   }   sh.google_demux = bits.parseBits(1);   if (sh.google_demux != '0') {     raise ParserException;   }   sh.reserved = bits.parseBits(3);   sh.dcid = bits.parseBits(context.scid_len == '0' : 0 ?                           (context.scid_len+3)*8);   sh.packet_number = parsePacketNum(bits);   sh.protected_payload = parseCryptobits(bits);   unprotected_payload : bit[];   while (len(unprotected_payload) &gt; 0) {     sh.payload += parseFrame(unprotected_payload);   }   return sh; } </pre>
<pre> version_negotiation := {   header_type      : bit;   unused          : bit[7];   ver              : version;   dcid_len        : cid_len;   scid_len        : cid_len;   dcid            : bit[];   scid            : bit[];   supported_versions : version[]; } where {   header_type = 1;   ver = 0;   dcid.width = dcid_len == 0 ? 0 : (dcid_len+3) * 8;   scid.width = scid_len == 0 ? 0 : (scid_len+3) * 8; } </pre>	<pre> typedef struct version_negotiation {   bit      header_type;   bit      unused[7];   version  ver;   cid_len  dcid_len;   cid_len  scid_len;   bit      dcid[];   bit      scid[];   version  supported_versions[]; } version_negotiation; </pre>	<pre> parseVersionNegotiation :: (bits : bit[]) -&gt; version_negotiation {   vn : version_negotiation;   vn.header_type = bits.parseBits(1);   vn.unused = bits.parseBits(7);   vn.version = parseVersion(bits);   if (vn.header_type != '1' and vn.version != '0') {     raise ParserException;   }   vn.dcid_len = parseCidLen(bits);   vn.scid_len = parseCidLen(bits);   vn.dcid = bits.parseBits(vn.dcid_len == '0' : 0 ?                           (vn.dcid_len+3)*8);   vn.scid = bits.parseBits(vn.scid_len == '0' : 0 ?                           (vn.scid_len+3)*8);   return vn; } </pre>

<pre>quic_pdu :=   long_hdr   short_hdr   version_negotiation;</pre>	<pre>typedef enum quic_pdu {long_hdr,                       short_hdr,                       version_negotiation} quic_pdu;</pre>	<pre>parseQUICPDU :: (bit[] : bits) -&gt; quic_pdu {   try:     return parseLongHdr(bits);   except ParserException:     continue;   try:     return parseShortHdr(bits);   except ParserException:     continue;   return parseVersionNegotiation(bits); }</pre>
<pre>padding_frame := {   type : frame_type; } where {   type = 0; };</pre>	<pre>struct padding_frame {   frame_type type; };</pre>	<pre>parsePaddingFrame :: (bits : bit[]) -&gt; padding_frame {   pf : padding_frame;   pf.type = parseFrameType(bits);   if (pf.type != '0') {     raise ParserException;   } }</pre>
<pre>frame := {   padding_frame   rst_stream_frame   connection_close_frame   ..};</pre>	<pre>typedef enum frame {padding_frame,                     rst_stream_frame,                     connection_close_frame,                     ..} enum_frame;</pre>	<pre>parseFrame :: (bits : bit[]) -&gt; frame {   try:     return parsePaddingFrame(bits);   except ParserException:     continue;   try:     return parseRstStreamFrame(bits);   except ParserException:     continue;   return ConnectionCloseFrame(bits); }</pre>