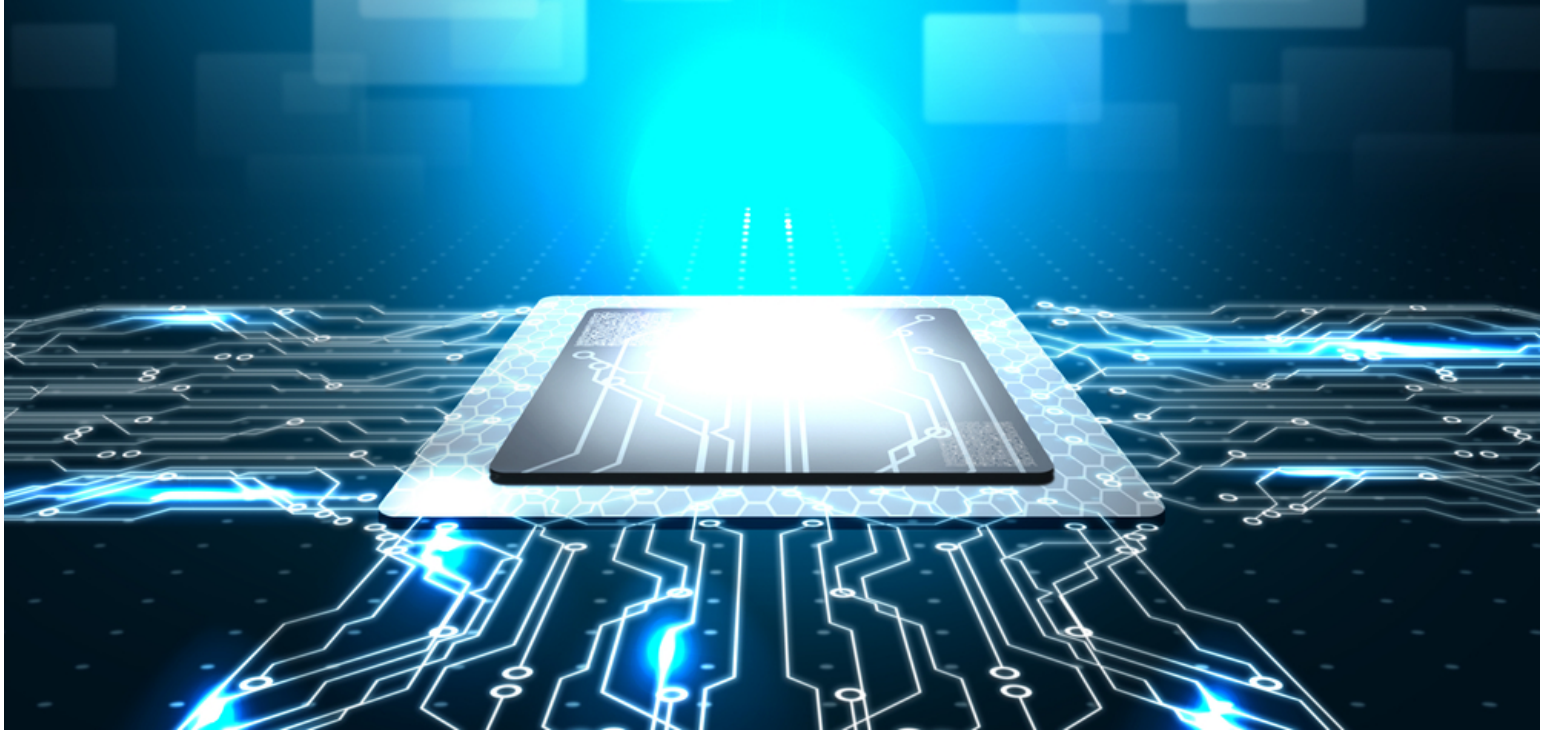
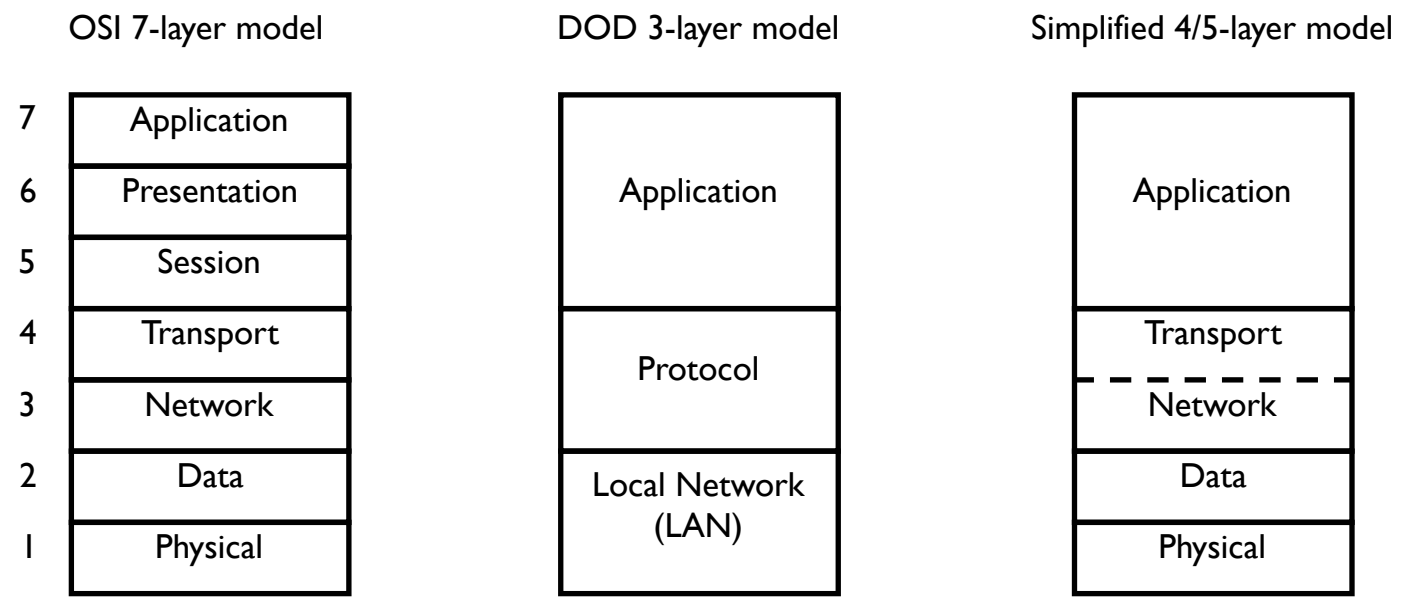


Advanced Digital Systems Design with FPGAs
EECS 395/495 – Lecture 8

Dr. David Zaretsky
david.zaretsky@northwestern.edu

Computer Networking

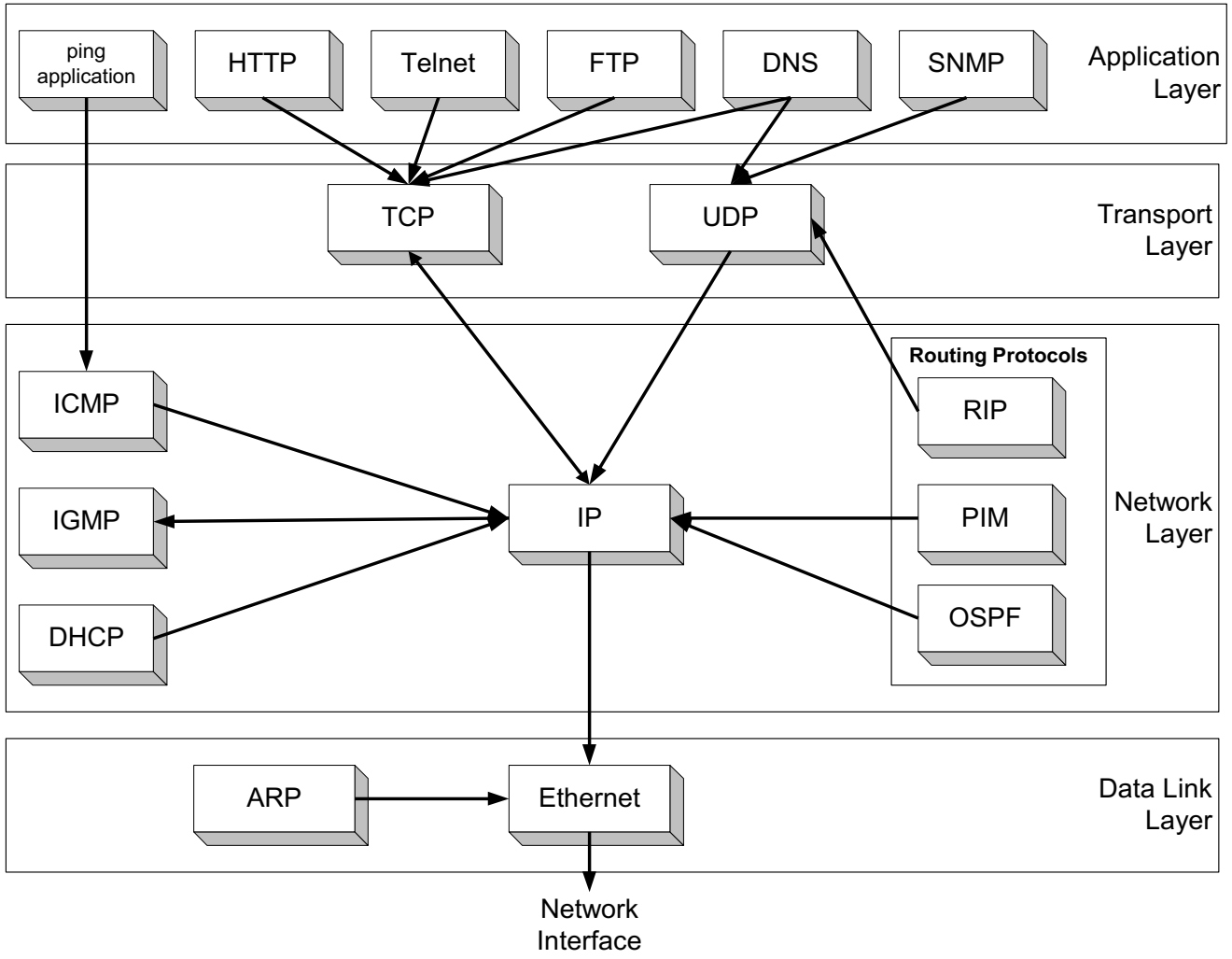
- There are various models of the networking stack, typically arranged in 3-7 layers.



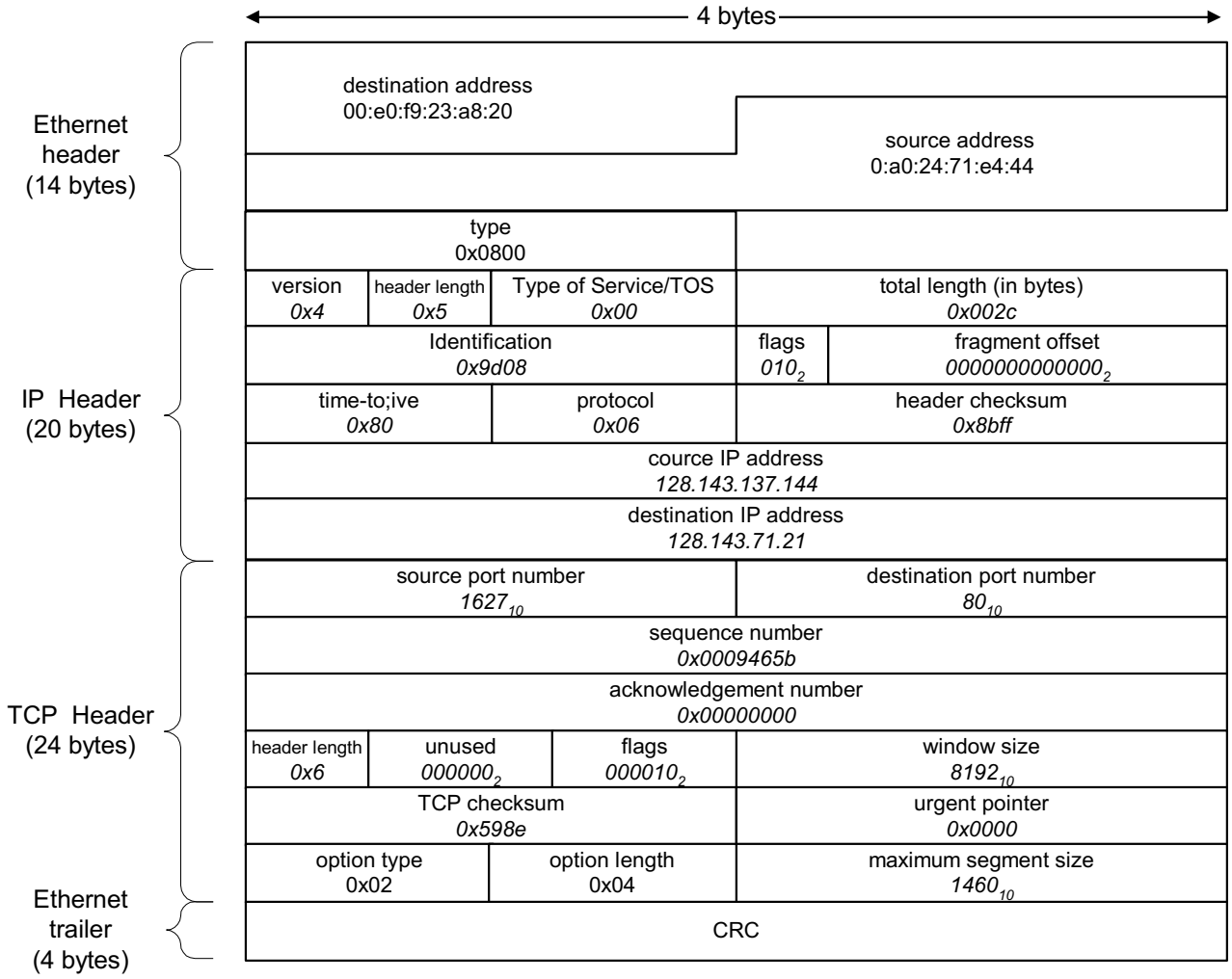
TCP/IP Network Model

- Physical Layer – the physical wires and hardware
- Link Layer - includes device driver and network interface card
- Network Layer - handles the movement of packets, i.e. Routing
- Transport Layer - provides a reliable flow of data between two hosts
- Application Layer - handles the details of the particular application

Assignment of Protocols to Layers

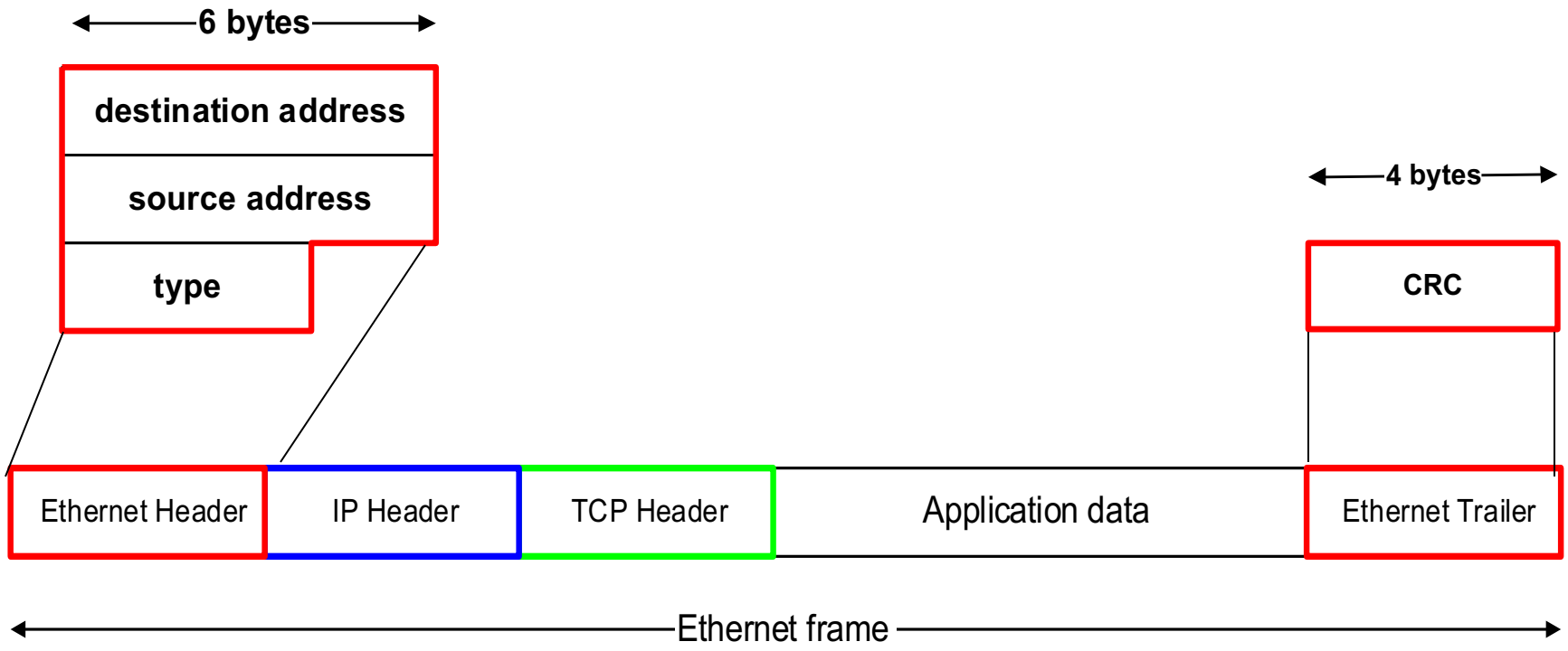


Parsing the Packets



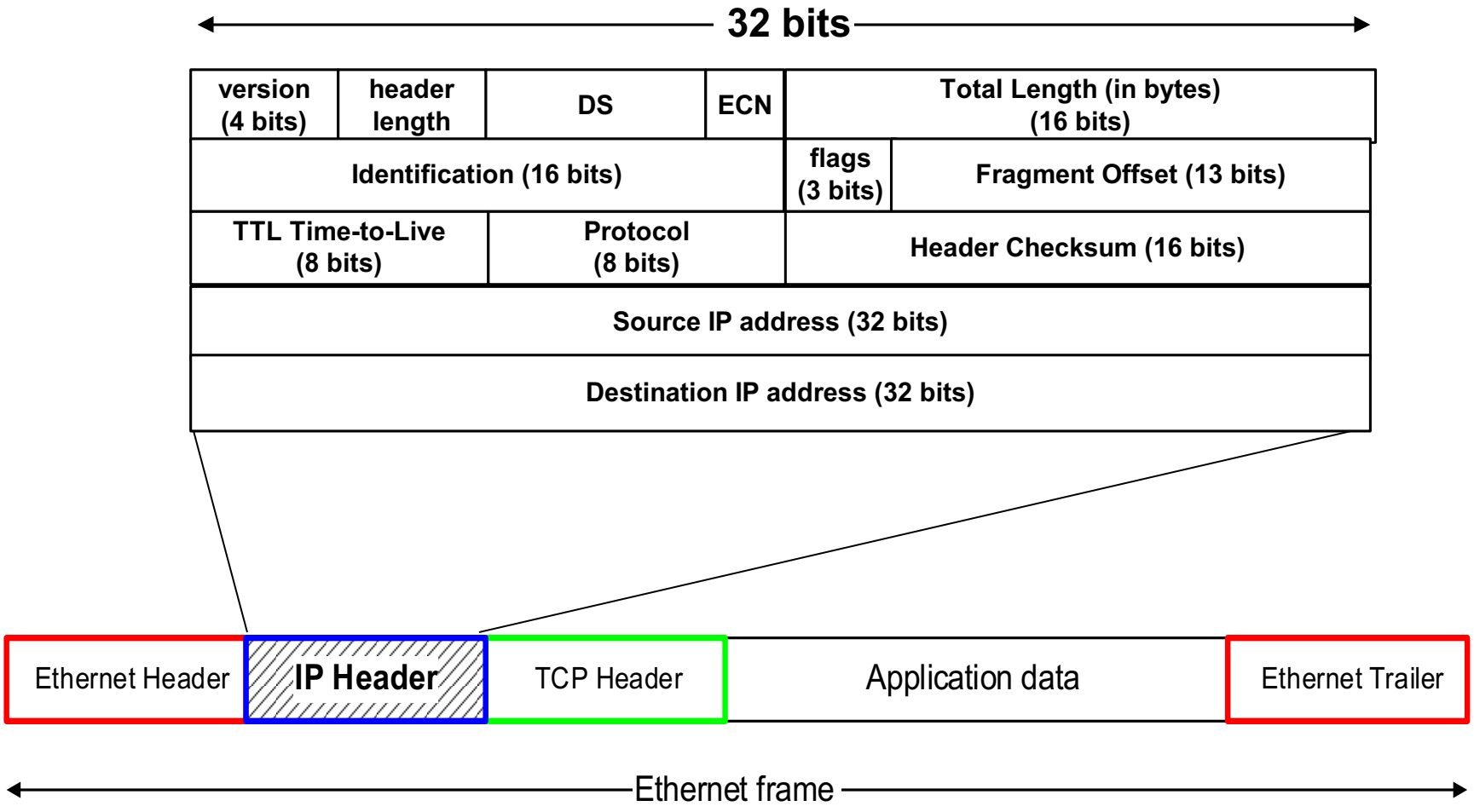
Ethernet Header

- 14 byte header
- 4 byte checksum

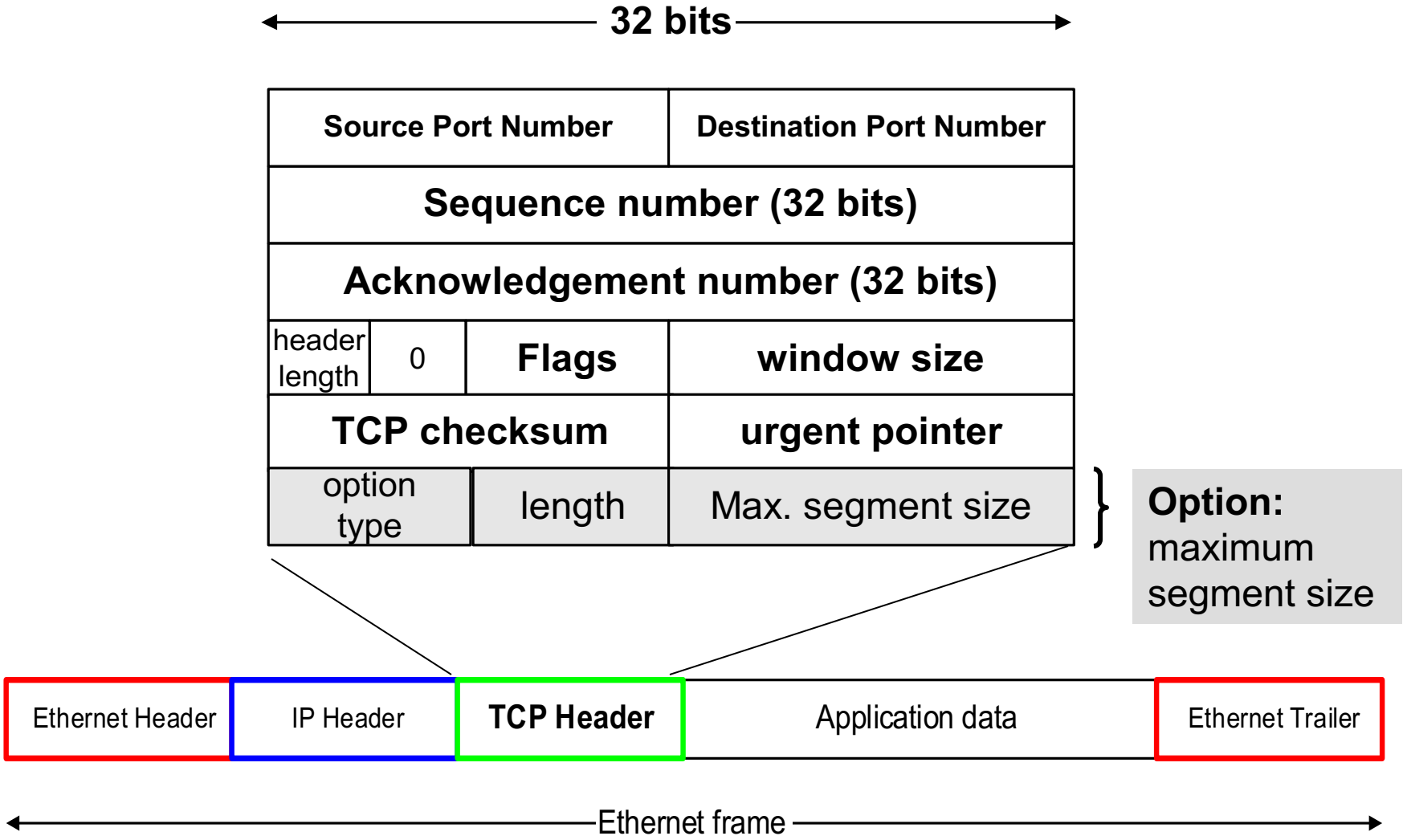


IP Header

- 20 byte header



TCP Header


32 bits
Source Port Number
Destination Port Number
Sequence number (32 bits)
Acknowledgement number (32 bits)
header length
0
Flags
window size
TCP checksum
urgent pointer
option type
length
Max. segment size
Option: maximum segment size
Ethernet Header
IP Header
TCP Header
Application data
Ethernet Trailer
Ethernet frame

Internet Protocol (IP)

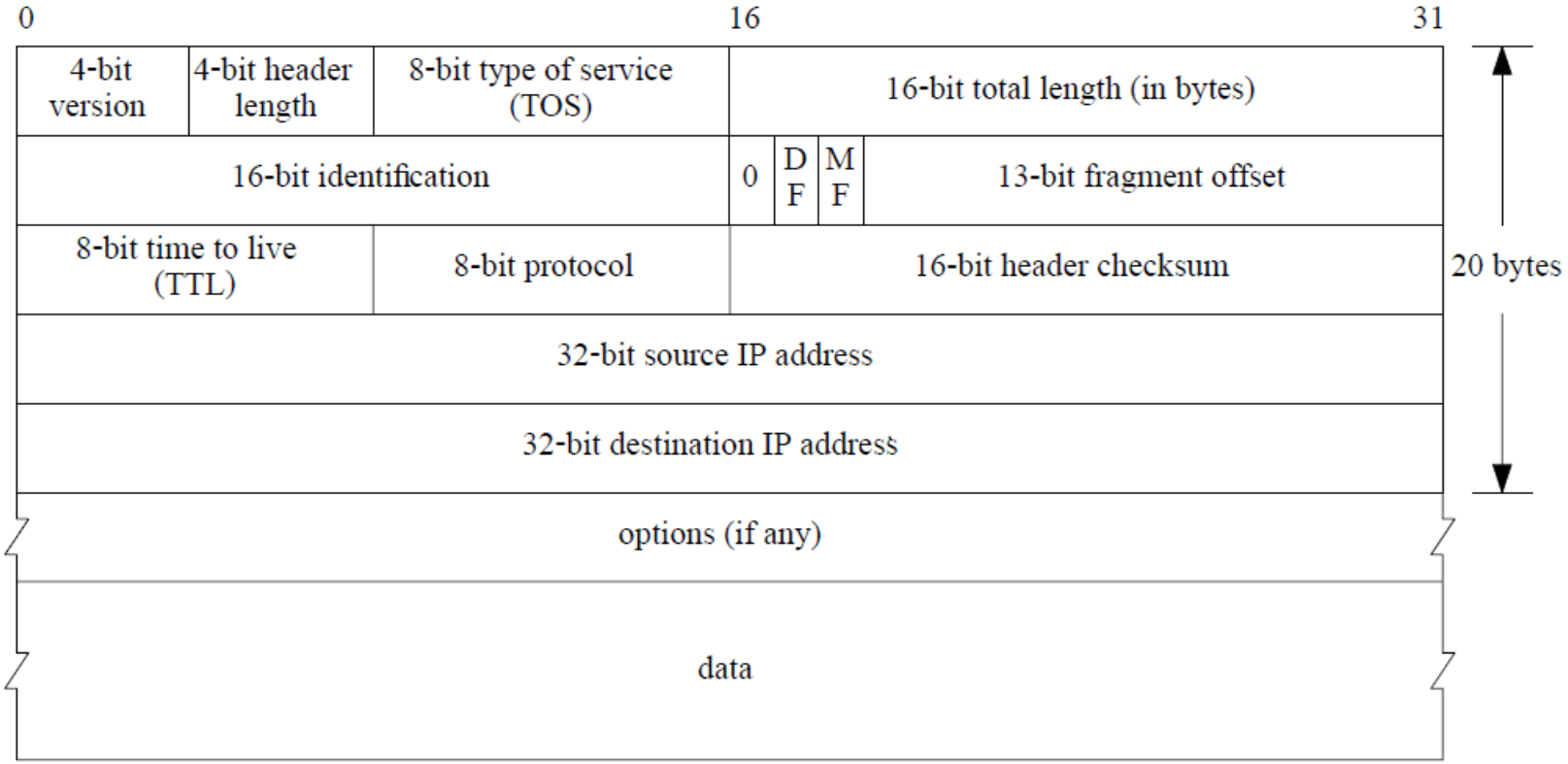
- IP is responsible for addressing and routing of data packets.
- Two versions in current in use: IPv4 & IPv6.
- IPv4: uses a 160 bit (20 byte) header, and 32 bit addresses.
- IPv6 was mainly developed to increase IP address space due to the huge growth in Internet usage during the 1990s.
- IPv6 uses a 320 bit (40 byte) header and 128 bit addresses.
- Header fields include: source and destination addresses, packet length and packet number.

IP Header Fields

- Ver – version of IP
- IHL – Internet Header Length (32-bit words)
- Service – Precedence/Delay/Throughput/Reliability
- Identification – assistance in reassembling fragments
- CF – control flags:
  - Reserved
  - 1 to prevent fragmentation, else 0
  - 1 if last fragment, else 0
- Fragment Offset – of this fragment in total message, bytes
- TTL – Time to Live, upper limit of life enroute
- Protocol – next higher protocol, e.g., TCP, UDP or ICMP

IP Datagram

IP Header

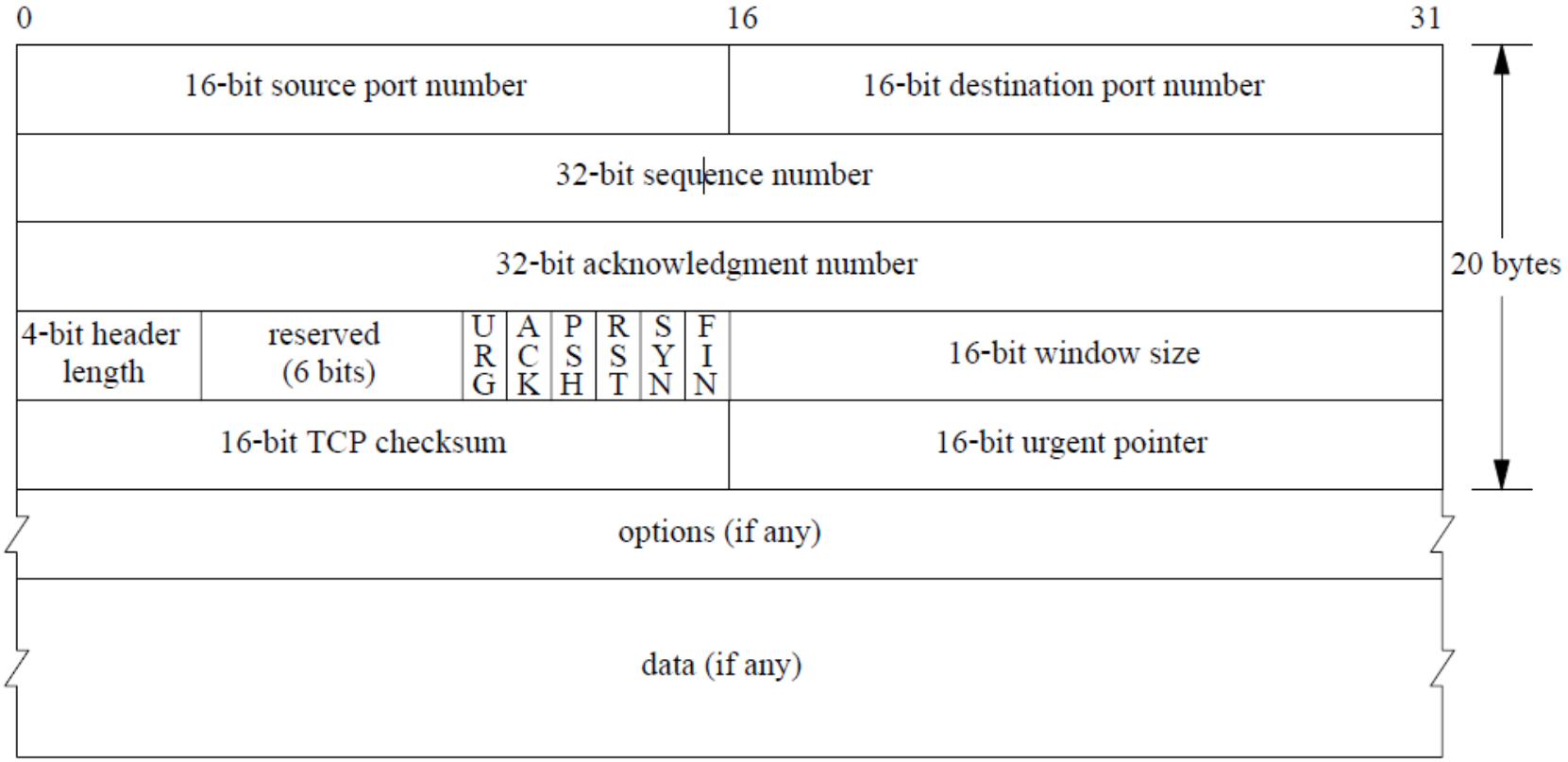


Transport Control Protocol (TCP)

- Reliable, full-duplex, connection-oriented, stream delivery
- Data is guaranteed to arrive, and in the correct order without duplications
- Imposes significant overheads
- Connections are established using a three-way handshake
- Data is divided up into packets by the operating system
- Packets are numbered, and received packets are acknowledged
- Connections are explicitly closed

TCP Datagram

TCP Header

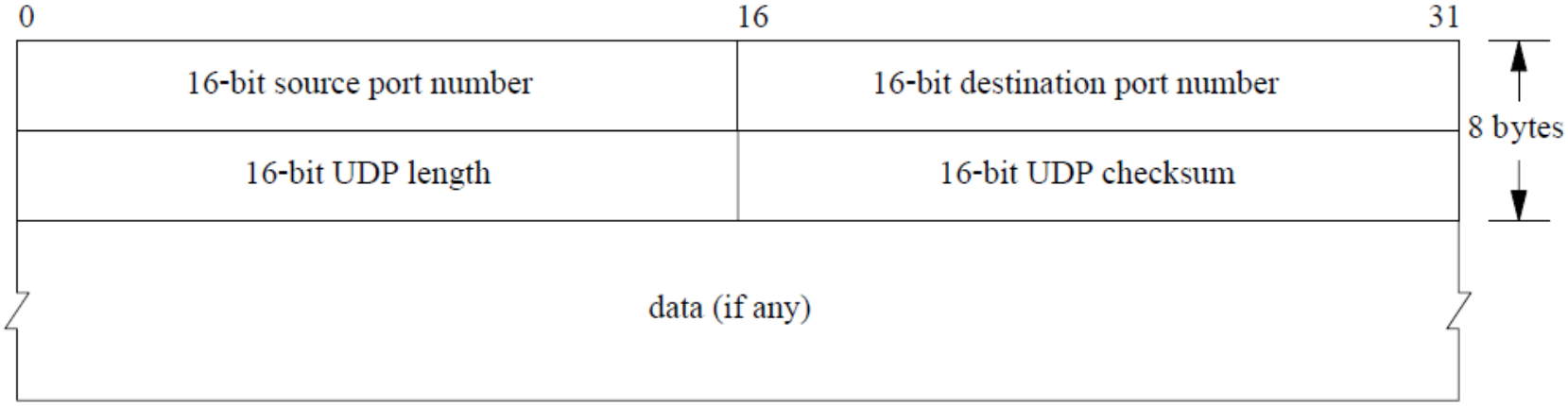


User Datagram Protocol (UDP)

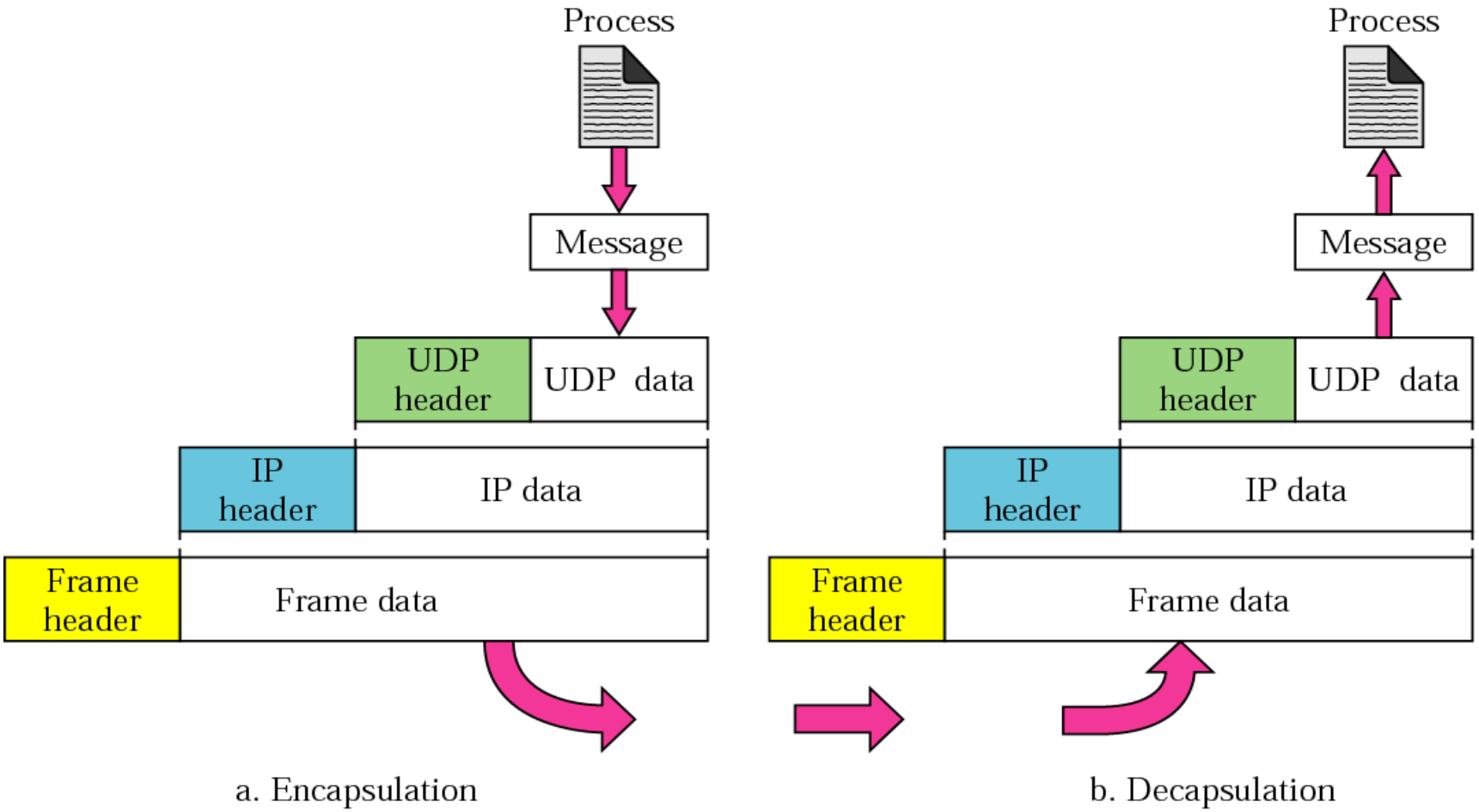
- One-to-one or one-to-many, connectionless and unreliable protocol
- Adds packet length + checksum to guard against corrupted packets
- Source and destination ports are used to associate a packet with a specific application at each end
- Not guaranteed to arrive, in order, or lossless
- Use Cases
  - Where packet loss is better handled by the application than the network stack
  - Where the overhead of setting up a connection isn't wanted
- Typical Applications
  - VOIP
  - Audio / video simulcasting

UDP Packets

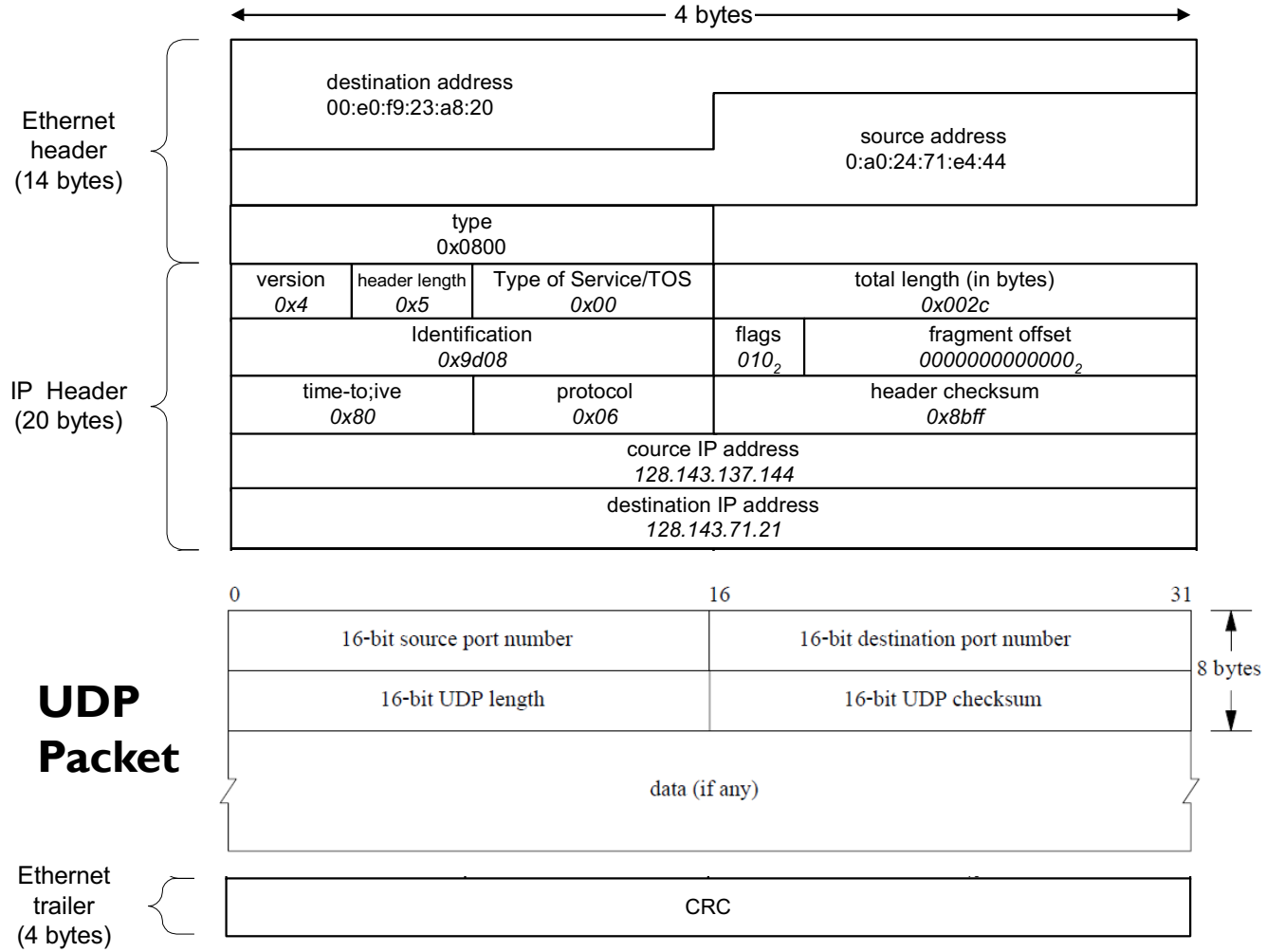
UDP Header



UDP Encapsulation / Decapsulation



Parsing UDP Packets



UDP Reader in C


```
#define ETH_DST_ADDR_BYTES      6
#define ETH_SRC_ADDR_BYTES      6
#define ETH_PROTOCOL_BYTES      2
#define IP_VERSION_BYTES        1
#define IP_HEADER_BYTES         1
#define IP_TYPE_BYTES           1
#define IP_LENGTH_BYTES         2
#define IP_ID_BYTES             2
#define IP_FLAG_BYTES           2
#define IP_TIME_BYTES           1
#define IP_PROTOCOL_BYTES       1
#define IP_CHECKSUM_BYTES       2
#define IP_SRC_ADDR_BYTES       4
#define IP_DST_ADDR_BYTES       4
#define UDP_DST_PORT_BYTES      2
#define UDP_SRC_PORT_BYTES      2
#define UDP_LENGTH_BYTES        2
#define UDP_CHECKSUM_BYTES      2
#define IP_PROTOCOL_DEF         0x0800
#define IP_VERSION_DEF          0x4
#define IP_HEADER_LENGTH_DEF    0x5
#define IP_TYPE_DEF             0x0
#define IP_FLAGS_DEF            0x4
#define TIME_TO_LIVE            0xe
#define UDP_PROTOCOL_DEF        0x11


int read_udp_packet(FILE *source, unsigned char *packet_data)
{
    unsigned char eth_dst_addr[ETH_DST_ADDR_BYTES];
    unsigned char eth_src_addr[ETH_SRC_ADDR_BYTES];
    unsigned char eth_protocol[ETH_PROTOCOL_BYTES];
    unsigned char ip_version[IP_VERSION_BYTES];
    unsigned char ip_header[IP_HEADER_BYTES];
    unsigned char ip_type[IP_TYPE_BYTES];
    unsigned char ip_length[IP_LENGTH_BYTES];
    unsigned char ip_id[IP_ID_BYTES];
    unsigned char ip_flag[IP_FLAG_BYTES];
    unsigned char ip_time[IP_TIME_BYTES];
    unsigned char ip_protocol[IP_PROTOCOL_BYTES];
    unsigned char ip_checksum[IP_CHECKSUM_BYTES];
    unsigned char ip_dst_addr[IP_SRC_ADDR_BYTES];
    unsigned char ip_src_addr[IP_DST_ADDR_BYTES];
    unsigned char udp_dst_port[UDP_DST_PORT_BYTES];
    unsigned char udp_src_port[UDP_SRC_PORT_BYTES];
    unsigned char udp_length[UDP_LENGTH_BYTES];
    unsigned char udp_checksum[UDP_CHECKSUM_BYTES];
    unsigned char udp_data[1024];
    unsigned short udp_data_length = 0, crc = 0, checksum = 0;
    int p = 0;

    if ( feof(source) ) return 0;

    fread(eth_dst_addr, 1, ETH_DST_ADDR_BYTES, source);
    fread(eth_src_addr, 1, ETH_SRC_ADDR_BYTES, source);
    fread(eth_protocol, 1, ETH_PROTOCOL_BYTES, source);
    if ( (((unsigned int)eth_protocol[0] << 8) | (unsigned
int)eth_protocol[1]) != IP_PROTOCOL_DEF )
        return 0;

    fread(ip_version, 1, IP_VERSION_BYTES, source);
    if ( (ip_version[0] >> 4) != IP_VERSION_DEF )
        return 0;
    ip_header[0] = ip_version[0] & 0xF;

    fread(ip_type, 1, IP_TYPE_BYTES, source);
    fread(ip_length, 1, IP_LENGTH_BYTES, source);
    fread(ip_id, 1, IP_ID_BYTES, source);
    fread(ip_flag, 1, IP_FLAG_BYTES, source);
    fread(ip_time, 1, IP_TIME_BYTES, source);
    fread(ip_protocol, 1, IP_PROTOCOL_BYTES, source);
    if ( ip_protocol[0] != UDP_PROTOCOL_DEF )
        return 0;
    fread(ip_checksum, 1, IP_CHECKSUM_BYTES, source);
    fread(ip_src_addr, 1, IP_SRC_ADDR_BYTES, source);
    fread(ip_dst_addr, 1, IP_DST_ADDR_BYTES, source);
    fread(udp_dst_port, 1, UDP_DST_PORT_BYTES, source);
    fread(udp_src_port, 1, UDP_SRC_PORT_BYTES, source);
    fread(udp_length, 1, UDP_LENGTH_BYTES, source);
    fread(udp_checksum, 1, UDP_CHECKSUM_BYTES, source);

    // get the UDP data
    udp_data_length = (((unsigned int)udp_length[0] << 8) |
(unsigned int)udp_length[1]);
    udp_data_length -= (UDP_CHECKSUM_BYTES +
UDP_LENGTH_BYTES + UDP_DST_PORT_BYTES +
UDP_SRC_PORT_BYTES); // + CME_HEADER);
    fread(udp_data, 1, udp_data_length, source);

    // calculate the checksum
    crc = udp_sum_calc( ip_src_addr, ip_dst_addr, ip_protocol,
ip_length, udp_src_port, udp_dst_port, udp_length, udp_data
);
    checksum = ((unsigned int)udp_checksum[0] << 8) |
(unsigned int)udp_checksum[1];
    if ( checksum != crc ) {
        fprintf( stderr, "ERROR: Checksum mismatch -- %04x !=
%04x\n", crc, checksum);
        return 0;
    }

    for ( int i = 0; i < udp_data_length; i++ )  {
        packet_data[i] = udp_data[i];
    }

    return udp_data_length;
}
```

UDP Checksum

```
unsigned short udp_sum_calc(
      unsigned char *ip_src_addr,
      unsigned char *ip_dst_addr,
      unsigned char *ip_protocol,
      unsigned char *ip_length,
      unsigned char *udp_src_port,
      unsigned char *udp_dst_port,
      unsigned char *udp_length,
      unsigned char *udp_data)
{
   unsigned short padd = 0;
   unsigned int sum = 0;
   int udp_len = ((udp_length[0] << 8) | udp_length[1]);
   int data_length = udp_len - (UDP_CHECKSUM_BYTES +
                  UDP_LENGTH_BYTES + UDP_DST_PORT_BYTES +
                  UDP_SRC_PORT_BYTES);

  // Find out if the length of data is even or odd number. If odd,
  // add a padding byte = 0 at the end of packet
  if ( (data_length & 1) == 1 )  {
     padd=1;
     udp_data[data_length]=0;
  }

  // add the UDP pseudo header
  for (int i=0; i<4; i=i+2)  {
     sum += ((ip_src_addr[i]<<8)&0xFF00)+(ip_src_addr[i+1]&0xFF);
  }

   for (int i=0; i<4; i=i+2)   {
     sum += ((ip_dst_addr[i]<<8)&0xFF00)+(ip_dst_addr[i+1]&0xFF);
  }

   sum += ip_protocol[0];
   sum += (((unsigned short)ip_length[0]<<8)&0xFF00)+(ip_length[1]&0xFF) - 20;
   sum += (((unsigned short)udp_dst_port[0]<<8)&0xFF00) +
(udp_dst_port[1]&0xFF);
   sum += (((unsigned short)udp_src_port[0]<<8)&0xFF00) +
(udp_src_port[1]&0xFF);
   sum += (((unsigned short)udp_length[0]<<8)&0xFF00) +
(udp_length[1]&0xFF);

   // make 16 bit words out of every two adjacent 8 bit words and
   // calculate the sum of all 16 bit words
   for (int i=0; i < data_length; i += 2)
   {
      sum += (((unsigned short)udp_data[i]<<8)&0xFF00);
      sum += (udp_data[i+1]&0xFF);
   }

   // keep only the last 16 bits of the 32 bit calculated sum and add the carries
   while (sum >> 16 != 0)
   {
     sum = (sum & 0xFFFF) + (sum >> 16);
   }

   // Take the one's complement of sum
   sum = ~sum;
    //printf("sum: %08x\n", sum );

    return sum;
}
```

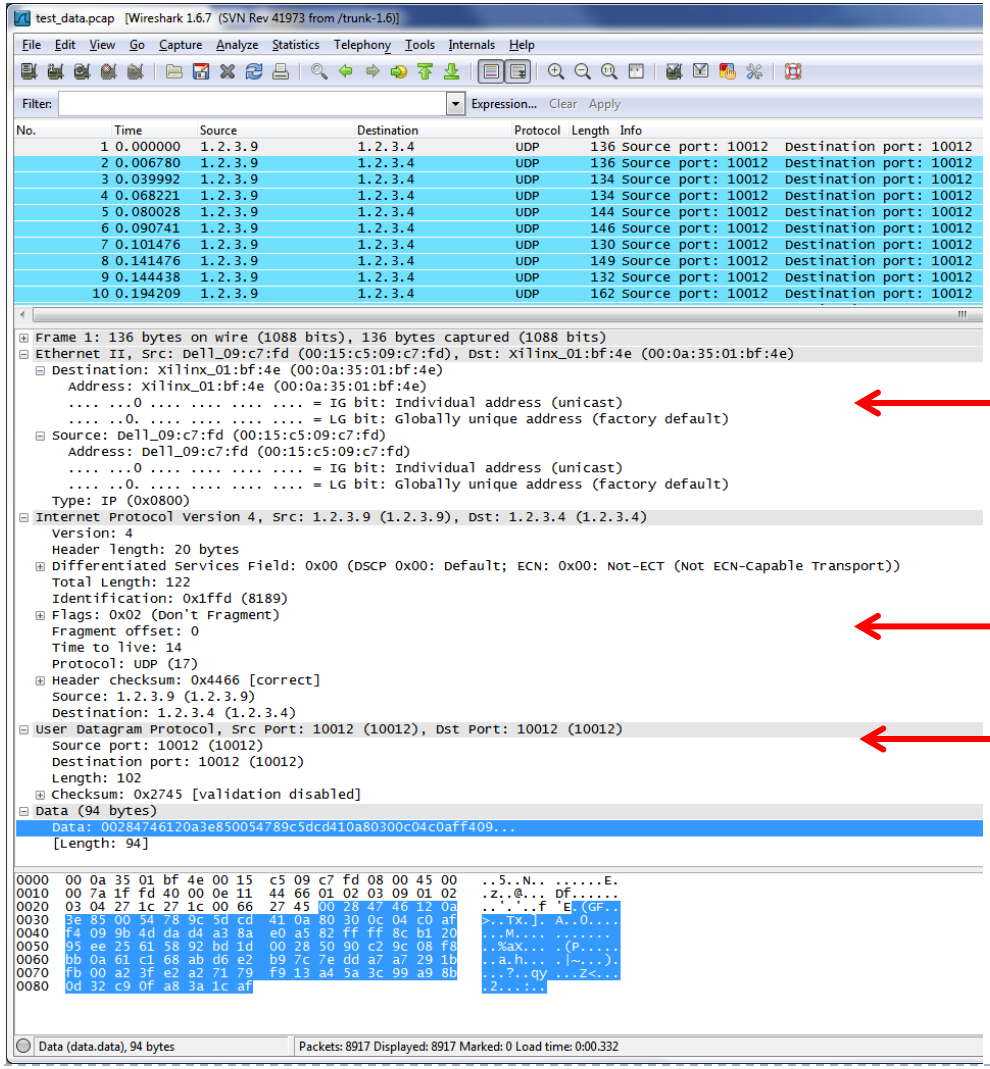
Programming Assignment #3: Streaming UDP Reader

- Streaming UDP process
  - Read packets from an input FIFO
  - Extract datagrams and write to output FIFO
  - Use 0x02 and 0x03 values to delineate start and end of packets, respectively.
- Simulate & Synthesize
  - Use pcap file as input (will need to remove pcap header)
  - Simulate to get cycle count and verify correctness
  - Synthesize to get resource utilization

Implementation of UDP Reader

- Synchronize packets using start of frame (0x02) and end of frame (0x03) delineators
- Checksum can be calculated in parallel as each data point is acquired, instead of doing it at the end
- Data needs to be validated before it goes out
  - Store in temporary fifo buffer
  - Clear the fifo if any checksum errors are found
  - Burst out packets after checksum is validated
- Use the REPORT command in VHDL to display data in the log window

Wireshark



Ethernet Frame

IP Frame

UDP Frame

UDP Datagram Output

```
00 28 47 46 12 0a 3e 85  00 54 78 9c 5d cd 41 0a
80 30 0c 04 c0 af f4 09  9b 4d da d4 a3 8a e0 a5
82 ff ff 8c b1 20 95 ee  25 61 58 92 bd 1d 00 28
50 90 c2 9c 08 f8 bb 0a  61 c1 68 ab d6 e2 b9 7c
7e dd a7 a7 29 1b fb 00  a2 3f e2 a2 71 79 f9 13
a4 5a 3c 99 a9 8b 0d 32  c9 0f a8 3a 1c af

00 28 47 47 12 0a 3e 8c  00 54 78 9c 5d cc bb 0d
80 30 10 83 e1 55 32 82  ed dc e5 92 12 10 12 4d
2a f6 df 85 87 14 21 f2  37 96 be c2 5b df 71 47
64 77 51 9e 04 04 24 52  30 11 e8 8b 85 1a 31 7c
3f 8f 48 53 ab de 79 6e  be 68 16 40 ad c0 0f 0b
1c 33 31 1c 2c 65 48 ae  d1 2e a5 c3 1c c5

00 28 47 48 12 0a 3e ae  00 52 78 9c 5d 8b c1 0d
80 20 14 43 57 61 84 b6  f2 f9 72 54 43 e2 85 93
fb ef 22 e0 41 e5 25 4d  d3 a6 3d 6a 41 83 58 cc
44 e5 20 c0 21 91 42 14  81 ba 45 57 ee f1 e9 cb
75 7a 98 d8 35 8c f8 61  6a 9f ae 0f 8c 69 5a 8d
4a d6 96 e9 7d da 7a 03  a2 d1 1c a0

00 28 47 49 12 0a 3e ca  00 52 78 9c 65 8d 41 0a
80 40 0c 03 bf e2 13 d2  b4 da f6 a8 b2 e0 65 2f
fe ff 31 ae 0b a2 8b b9  84 0c 03 d9 6b 01 22 08
59 92 a2 74 02 0e 52 84  30 0a 50 57 4d 18 e6 87
97 93 87 4f 63 36 f6 6a  f6 37 d2 77 8c 30 d8 be
7e 48 6f dd 5e 64 79 01  bc a3 1c d2

00 28 47 4a 12 0a 3e d5  00 5c 78 9c 65 8d c1 0d
80 30 0c 03 57 e9 08 b6  53 9a 7e 01 f5 d9 0f fb
0f 43 41 20 35 aa 3f 56  2e 89 7d f6 06 d4 2a d0
5d 34 73 01 0e 89 14 b2  08 f4 9d e0 56 ec e7 ed
52 f7 14 75 e8 35 8e f3  59 23 02 c8 91 55 db 0c
0b 2a 8f 4f 9c 48 69 9e  e2 f6 7b d2 9a e3 88 8d
d4 0d bc 37 2a 56

00 28 47 4b 12 0a 3e e0  00 5e 78 9c 65 8c 3b 0e
c0 20 0c 43 af c2 11 6c  f3 09 6b 5b 31 b2 f4 fe
87 29 a9 54 09 8a 87 58  7a 79 c9 d5 1b 50 ab 40
33 31 26 13 60 90 48 21  89 40 3f 08 e6 92 3e de
6e 75 0b 6b 4e bd c5 a1  cf 61 19 23 af ac c6 1c
b1 21 17 31 71 22 84 c9  f0 b5 b6 a3 1f 71 64 de
69 fa a3 07 c1 3a 2a 61
```

UDP Parser in VHDL

```
when WAIT_FOR_SOF_STATE =>
  if ( input_empty = '0' ) then
    input_rd_en <= '1';
  end if;
  if ( (input_empty = '0') and (input_dout = X"02") ) then
    next_state <= ETH_DST_ADDR_STATE;
  end if;

when ETH_DST_ADDR_STATE =>
  if ( input_empty = '0' ) then
    input_rd_en <= '1';
    eth_dst_addr_t := std_logic_vector((unsigned(eth_dst_addr) sll 8) or resize(unsigned(input_dout),ETH_DST_ADDR_BYTES*8));
    eth_dst_addr_c := eth_dst_addr_t;
    num_bytes_c <= (num_bytes + 1) mod ETH_DST_ADDR_BYTES;
    if ( num_bytes = ETH_DST_ADDR_BYTES-1 ) then
      next_state <= ETH_SRC_ADDR_STATE;
    end if;
  end if;

  :

 when IP_SRC_ADDR_STATE =>
  if ( input_empty = '0' ) then
    input_rd_en <= '1';
    ip_src_addr_t := std_logic_vector((unsigned(ip_src_addr) sll 8) or resize(unsigned(input_dout),IP_SRC_ADDR_BYTES*8));
    ip_src_addr_c <= ip_src_addr_t;
    num_bytes_c <= (num_bytes + 1) mod IP_SRC_ADDR_BYTES;
    if ( num_bytes = IP_SRC_ADDR_BYTES-1 ) then
      checksum_c <= checksum + std_logic_vector(resize(unsigned(ip_src_addr_t(31 downto 16)),32) + resize(unsigned(ip_src_addr_t(15 downto 0)),32));
      next_state <= IP_DST_ADDR_STATE;
    end if;
  end if;
```