REAL-TIME DIGITAL SYSTEMS DESIGN AND VERIFICATION WITH FPGAS
ECE 387 – LECTURE 10

PROF. DAVID ZARETSKY

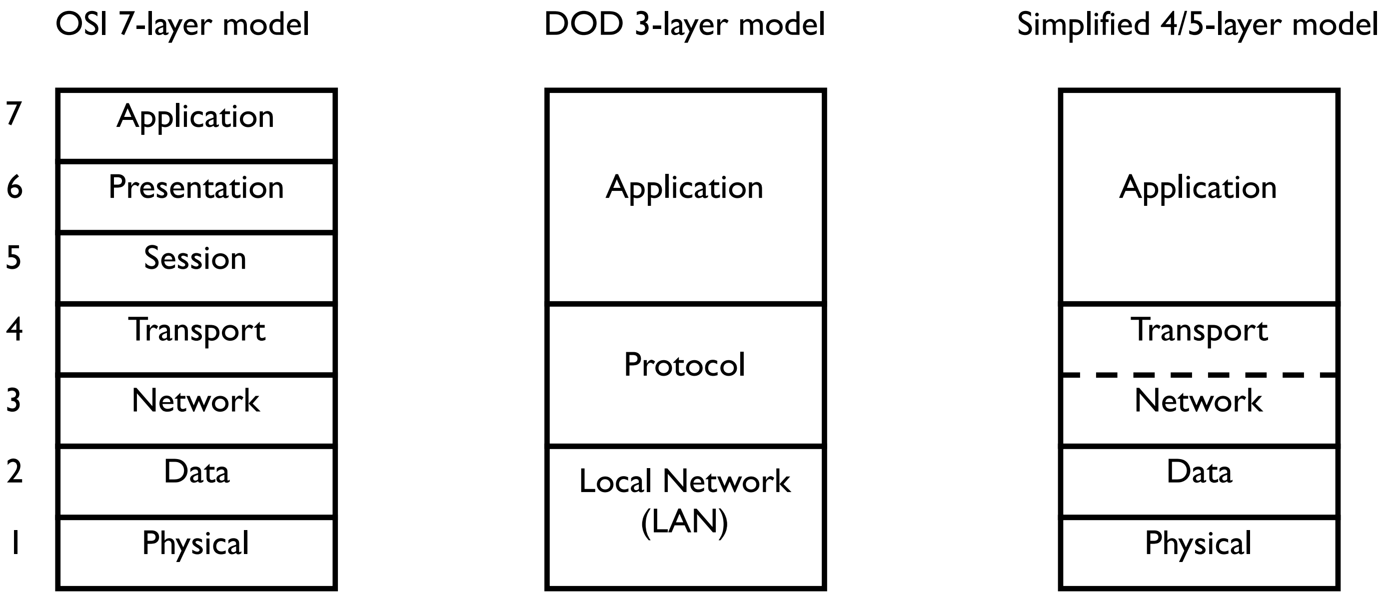
DAVID.ZARETSKY@NORTHWESTERN.EDU

AGENDA

- Network Packet Processing

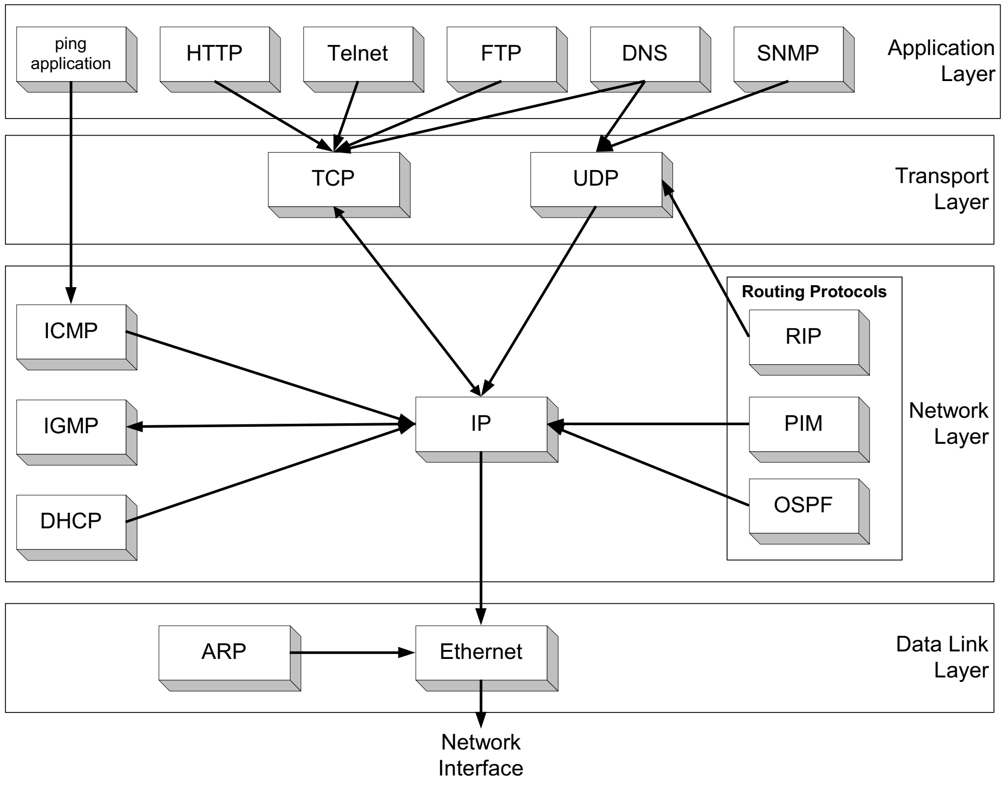
COMPUTER NETWORKING

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

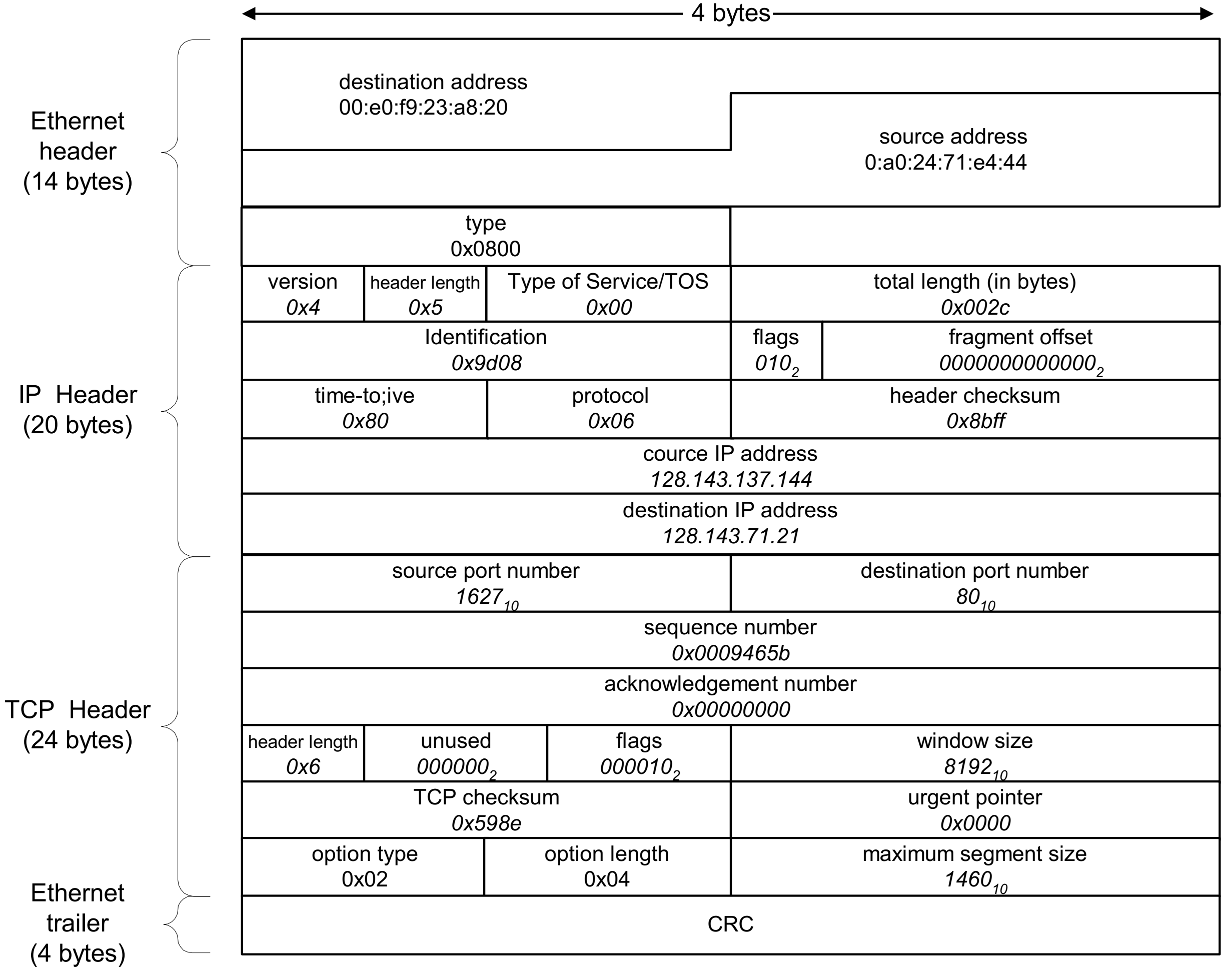


TCP/IP NETWORK MODEL

- 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

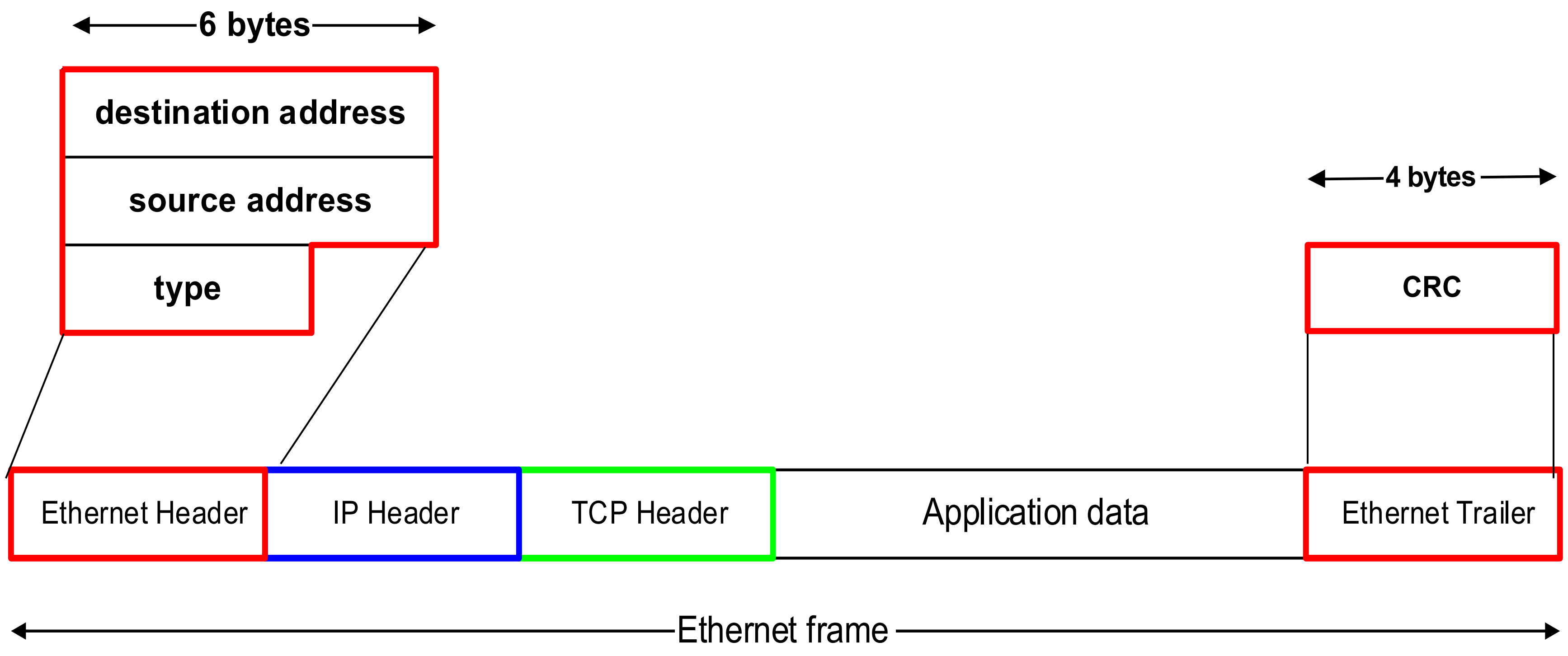


PARSING THE PACKETS



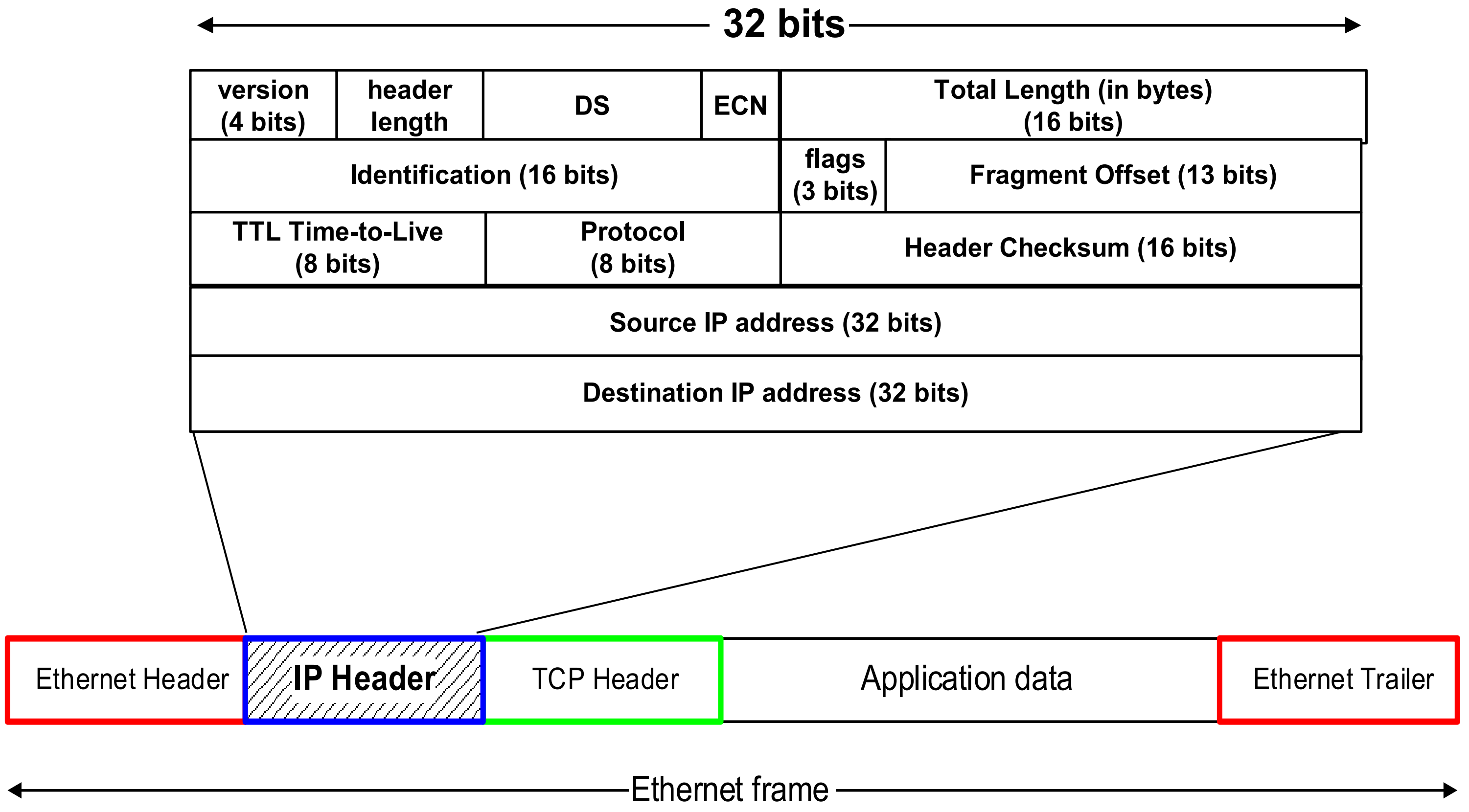
ETHERNET HEADER

- 14 byte header
- 4 byte checksum

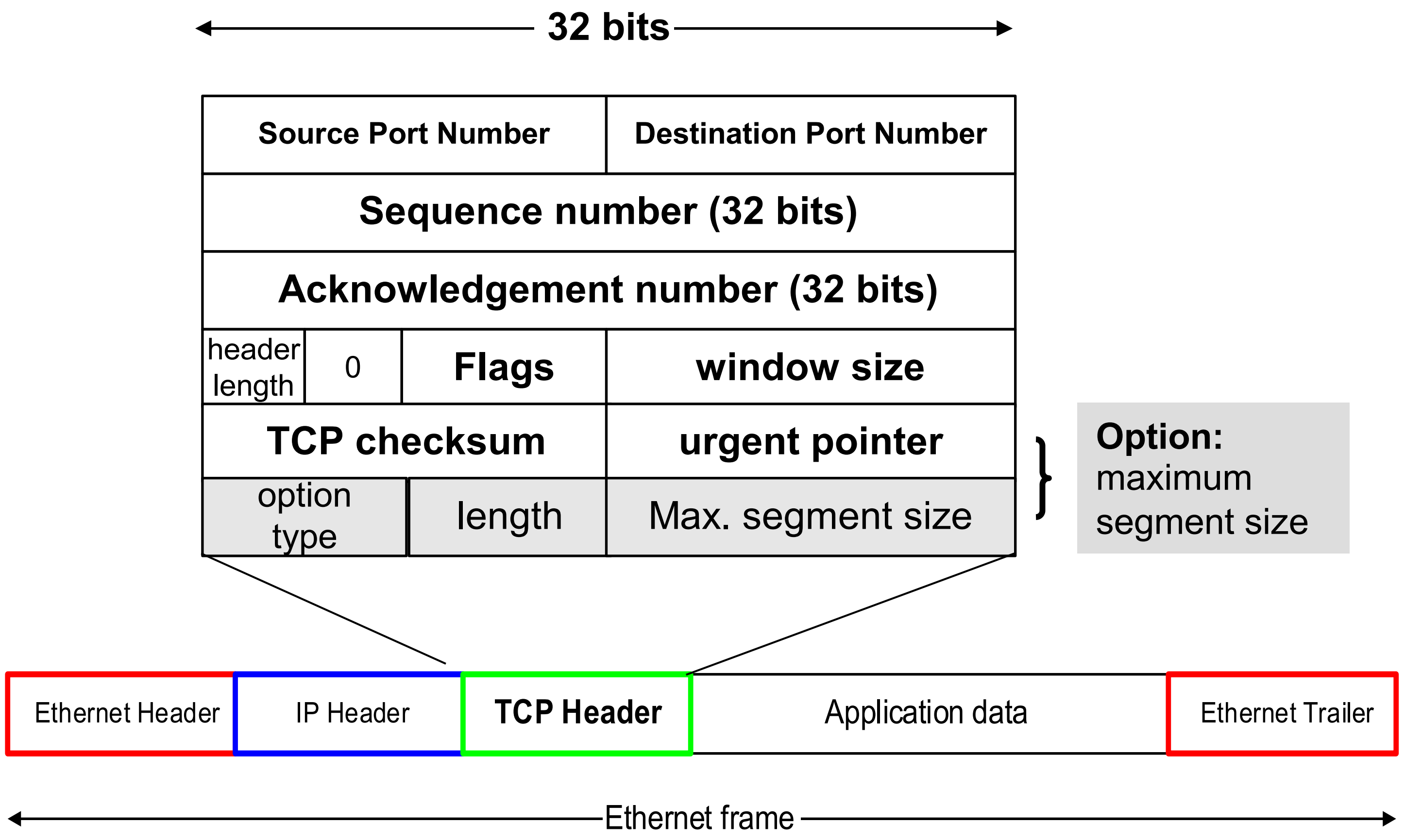


IP HEADER

- 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)

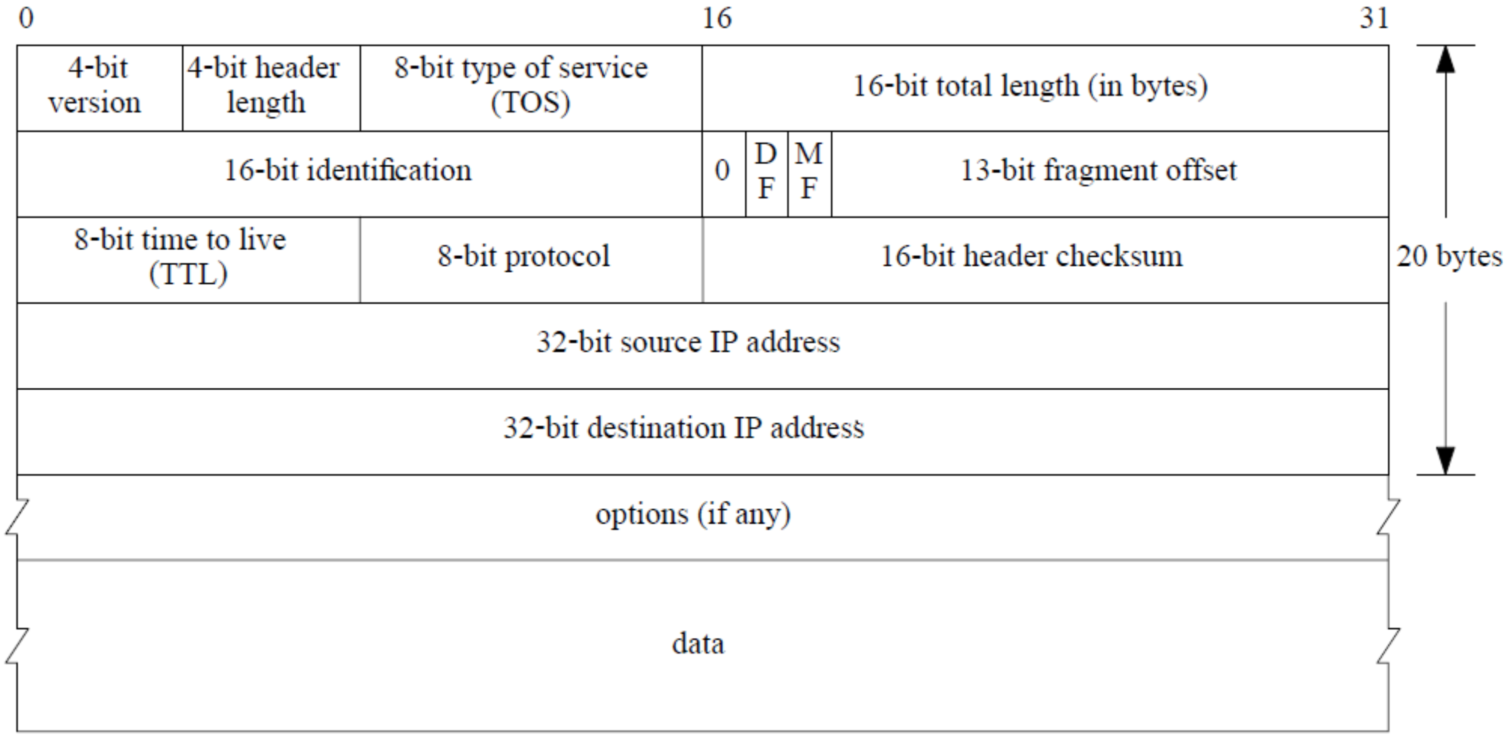
- IP is responsible for addressing and routing of data packets.
- Two versions in current in use: IPv4 & IPv6.
- IPv4: uses a 160 bit (20 byte) header, and 32 bit addresses.
- IPv6 was mainly developed to increase IP address space due to the huge growth in Internet usage during the 1990s.
- IPv6 uses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

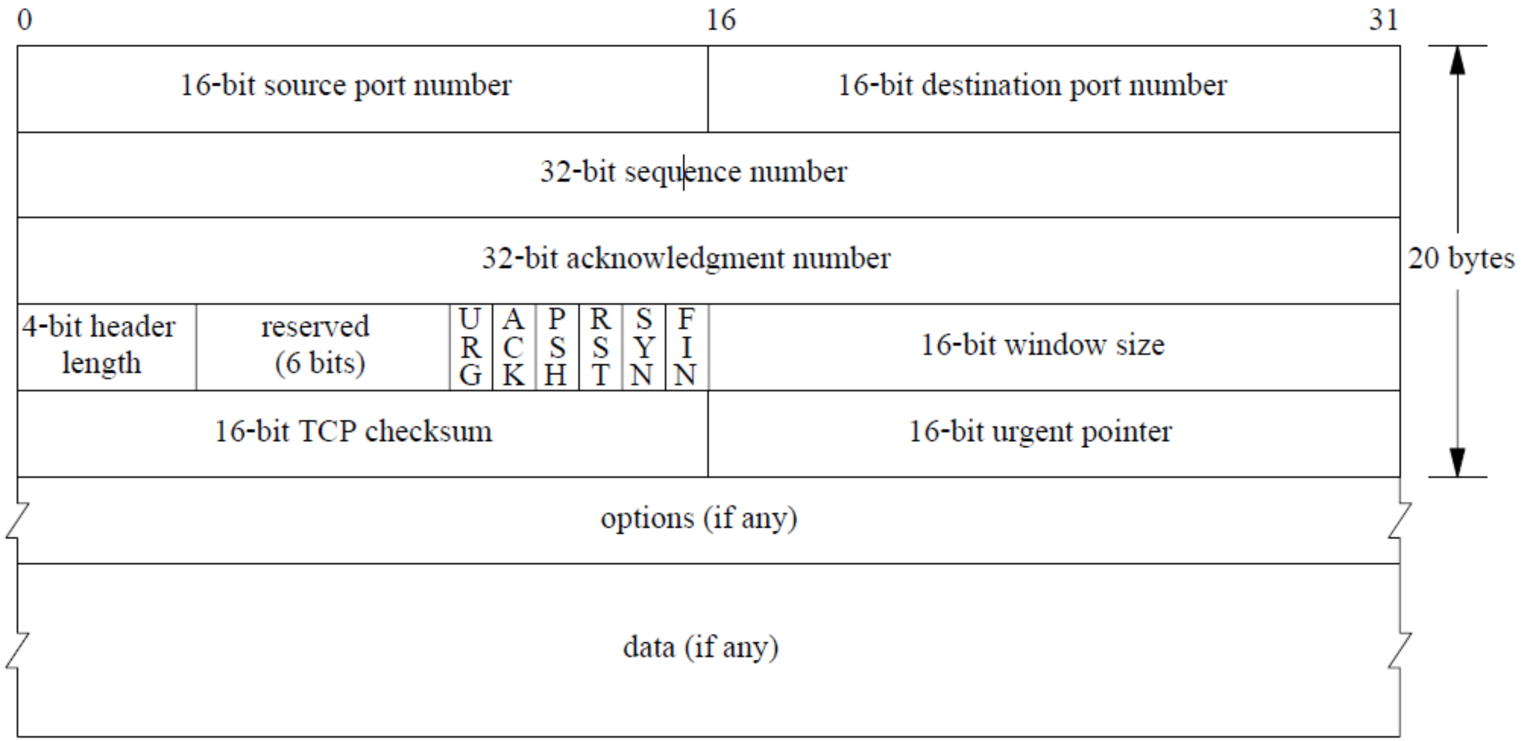


TRANSPORT CONTROL PROTOCOL (TCP)

- 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

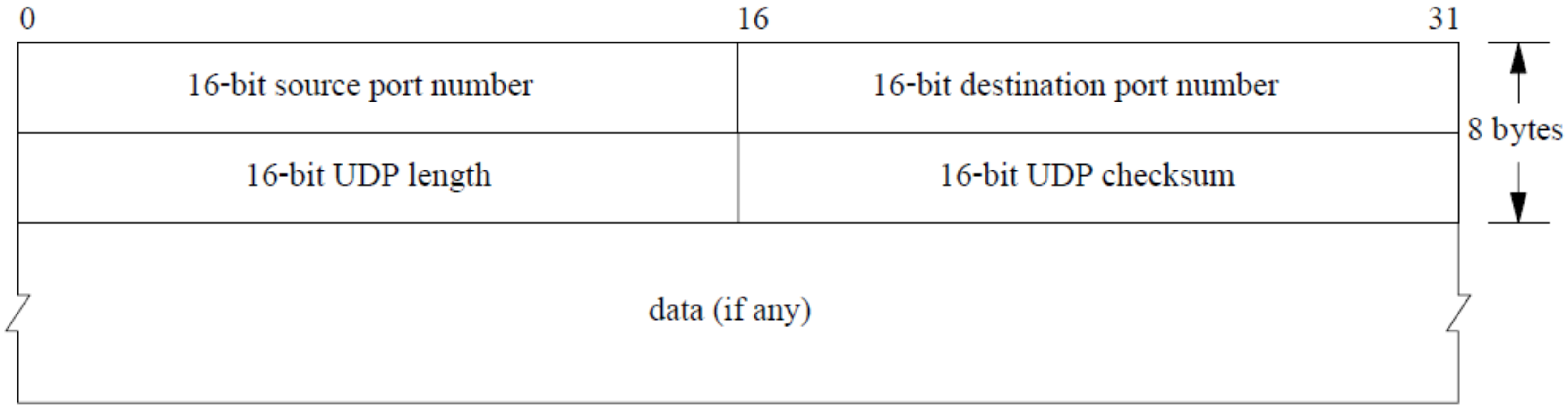


USER DATAGRAM PROTOCOL (UDP)

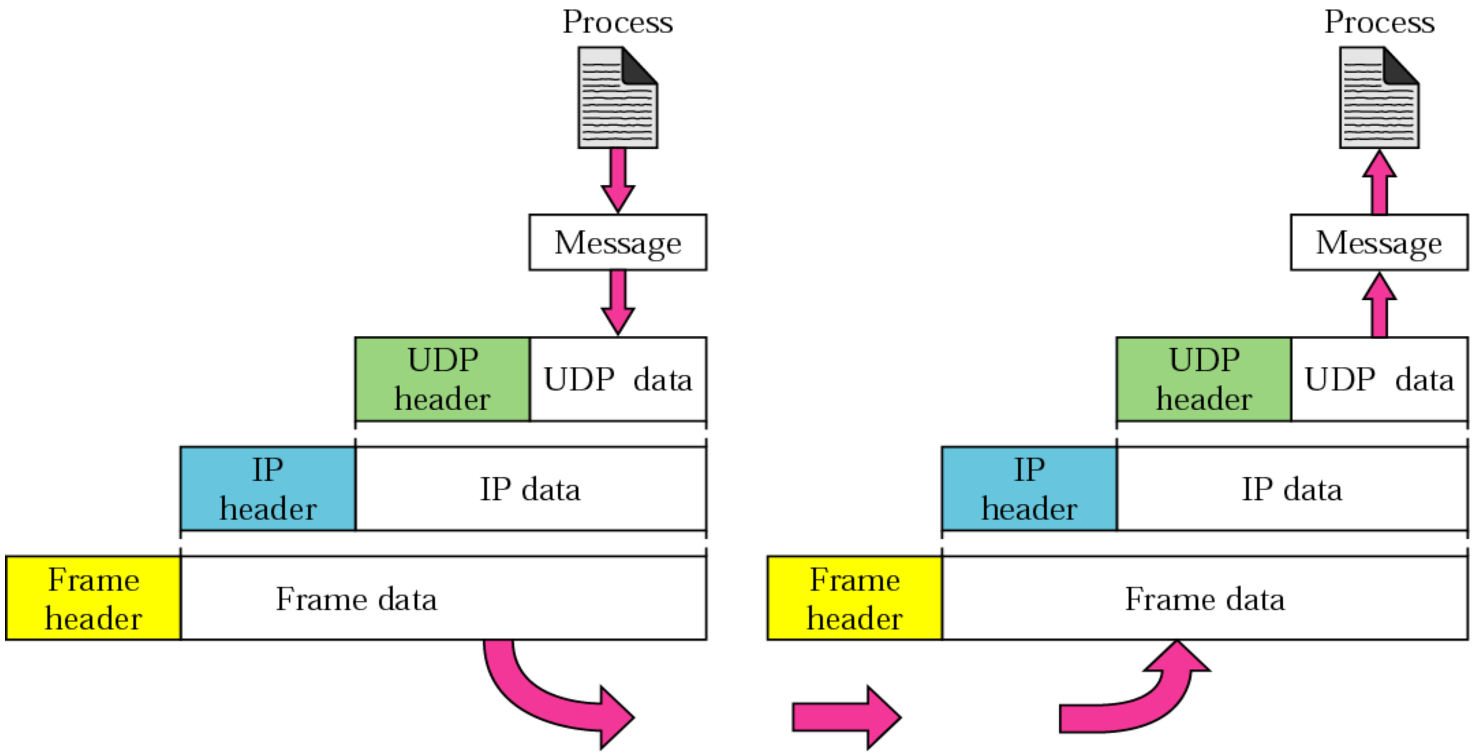
- One-to-one or one-to-many, connectionless and unreliable protocol
- Adds packet length + checksum to guard against corrupted packets
- Source and destination ports are used to associate a packet with a specific application at each end
- Not guaranteed to arrive, in order, or lossless
- Use Cases
  - 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

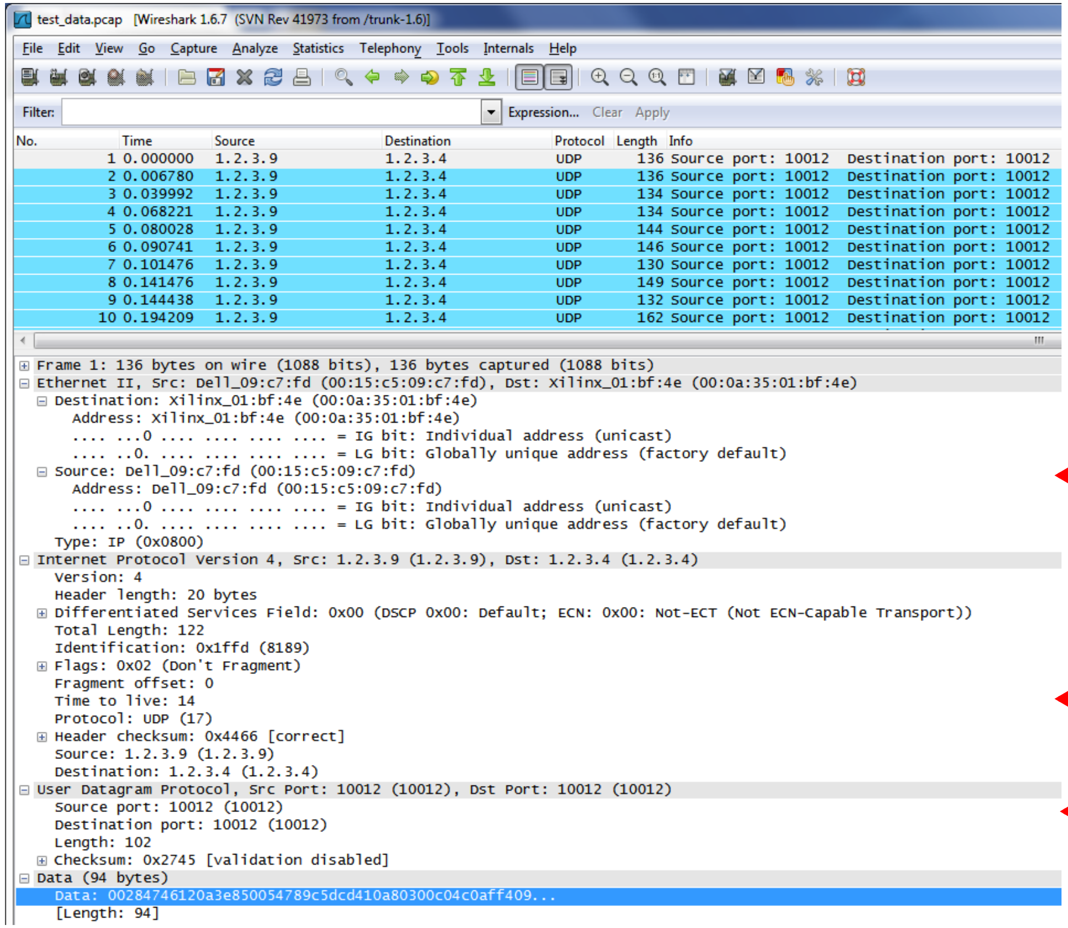


UDP ENCAPSULATION / DECAPSULATION



a. Encapsulation b. Decapsulation

WIRESHARK



Ethernet Frame

IP Frame

UDP Frame

UDP DATAGRAM OUTPUT

```
0000  00 0a 35 01 bf 4e 00 15  c5 09 c7 fd 08 00 45 00   ..5..N.. ......E.
0010  01 77 1f fd 40 00 0e 11  43 69 01 02 03 09 01 02   .w..@... Ci......
0020  03 04 27 1c 27 1c 01 63  10 cf 2f 2f 20 54 68 69   ..'.'..c ..// Thi
0030  73 20 65 78 61 6d 70 6c  65 20 77 69 6c 6c 20 73   s exampl e will s
0040  74 72 69 70 20 74 68 65  20 64 61 74 61 20 73 65   trip the  data se
0050  67 6d 65 6e 74 73 20 66  72 6f 6d 20 65 61 63 68   gments f rom each
0060  20 55 44 50 20 70 61 63  6b 65 74 20 61 6e 64 20    UDP pac ket and
0070  77 72 69 74 65 20 69 74  20 74 6f 20 66 69 6c 65   write it  to file
0080  2e 0a 2f 2f 20 55 73 65  20 57 69 72 65 73 68 61   ..// Use  Wiresha
0090  72 6b 20 28 68 74 74 70  3a 2f 2f 77 77 77 2e 77   rk (http ://www.w
00a0  69 72 65 73 68 61 72 6b  2e 6f 72 67 29 20 74 6f   ireshark .org) to
00b0  20 6f 70 65 6e 20 74 68  65 20 70 63 61 70 20 64    open th e pcap d
00c0  61 74 61 20 66 69 6c 65  2e 0a 0a 2f 2f 20 63 6f   ata file ...// co
00d0  6d 70 69 6c 65 0a 67 2b  2b 20 75 64 70 5f 72 65   mpile.g+ + udp_re
00e0  61 64 65 72 2e 63 70 70  20 2d 6f 20 75 64 70 5f   ader.cpp  -o udp_
00f0  72 65 61 64 65 72 0a 0a  2f 2f 20 72 75 6e 20 74   reader.. // run t
0100  68 65 20 75 64 70 20 74  65 73 74 20 77 69 74 68   he udp t est with
0110  20 74 68 65 20 69 6e 70  75 74 20 70 63 61 70 20    the inp ut pcap
0120  66 69 6c 65 0a 2e 2f 75  64 70 5f 72 65 61 64 65   file../u dp_reade
0130  72 20 3c 20 74 65 73 74  5f 64 61 74 61 2e 70 63   r < test _data.pc
0140  61 70 20 3e 20 74 65 73  74 5f 6f 75 74 70 75 74   ap > tes t_output
0150  2e 74 78 74 0a 0a 2f 2f  20 63 6f 6d 70 61 72 65   .txt..//  compare
0160  20 6f 75 74 70 75 74 0a  64 69 66 66 20 74 65 73    output. diff tes
0170  74 5f 6f 75 74 70 75 74  2e 74 78 74 20 74 65 73   t_output .txt tes
0180  74 2e 74 78 74                                     t.txt
```

```
> Frame 1: 389 bytes on wire (3112 bits), 389 bytes captured (3112 bits)
> Ethernet II, Src: Dell_09:c7:fd (00:15:c5:09:c7:fd), Dst: Xilinx_01:bf:4e (00:0a:35:01:bf:4e)
> Internet Protocol Version 4, Src: 1.2.3.9, Dst: 1.2.3.4
v User Datagram Protocol, Src Port: 10012, Dst Port: 10012
    Source Port: 10012
    Destination Port: 10012
    Length: 355
    Checksum: 0x10cf [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
> Data (347 bytes)
```

```
// This example will strip the data segments from each UDP packet and write it to file.
// Use Wireshark (http://www.wireshark.org) to open the pcap data file.

// compile
g++ udp_reader.cpp -o udp_reader

// run the udp test with the input pcap file
./udp_reader < test_data.pcap > test_output.txt

// compare output
diff test_output.txt test.txt
```

UDP READER IN C

```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);
    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;
}
```

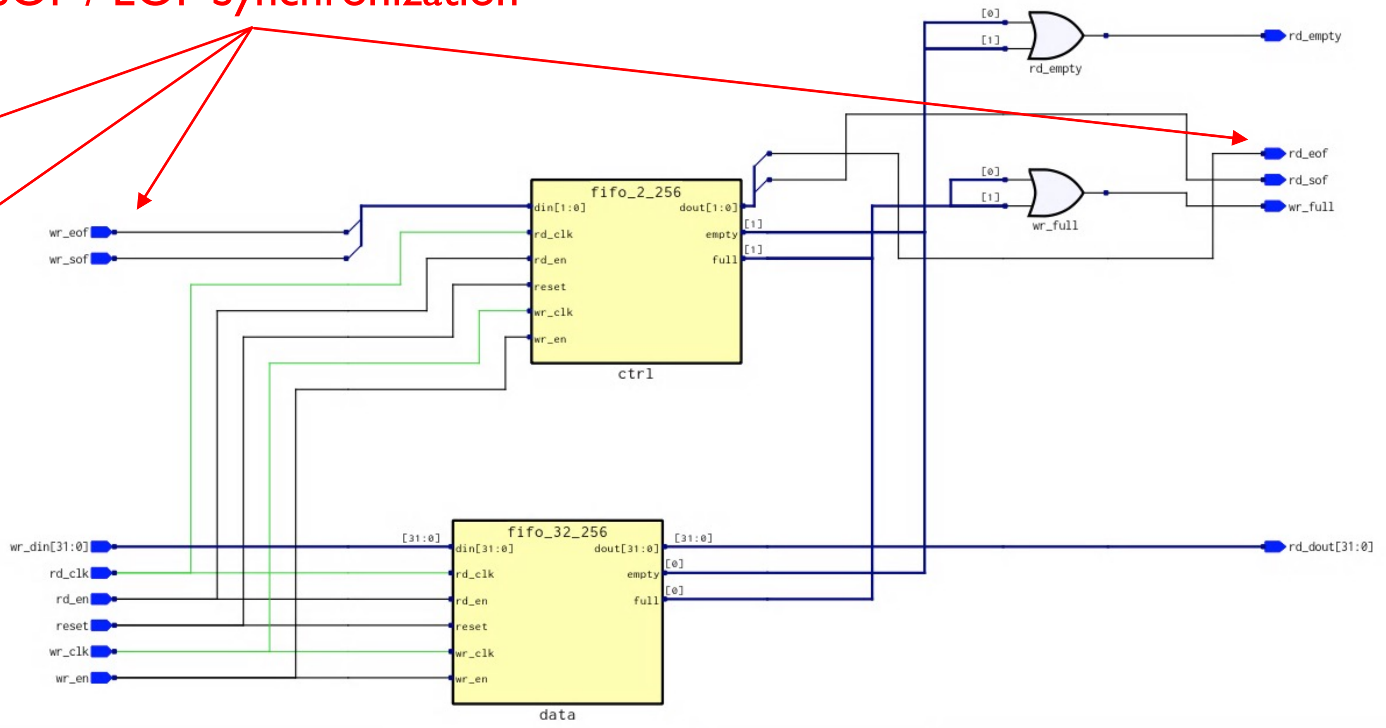
FIFO CONTROL

```
module fifo_ctrl #(
  parameter FIFO_DATA_WIDTH = 32,
  parameter FIFO_BUFFER_SIZE = 1024)
(
  input logic reset,

  input logic wr_clk,
  input logic wr_en,
  input logic wr_sof,
  input logic wr_eof,
  input logic [FIFO_DATA_WIDTH-1:0] din,
  output logic full,

  input logic rd_clk,
  input logic rd_en,
  output logic rd_sof,
  output logic rd_eof,
  output logic [FIFO_DATA_WIDTH-1:0] dout,
  output logic empty
);
```

SOF / EOF Synchronization



UDP READER IN RTL

```
WAIT_FOR_SOF_STATE: begin
    // wait for start-of-frame
    if ( (in_rd_sof == 1'b1) && (in_empty == 1'b0) ) begin
        next_state = ETH_DST_ADDR_STATE;
    end else if ( in_empty == 1'b0 ) begin
        in_rd_en = 1'b1;
    end
end

ETH_DST_ADDR_STATE: begin
    if ( in_empty == 1'b0 ) begin
        // concatenate new input to bottom 8-bits of previous value
        eth_dst_addr_c = ($unsigned(eth_dst_addr) << 8) | (ETH_DST_ADDR_BYTES*8)'($unsigned(in_dout));
        num_bytes_c = (num_bytes + 1) % ETH_DST_ADDR_BYTES;
        in_rd_en = 1’b1;
        if ( num_bytes == ETH_DST_ADDR_BYTES-1 ) begin
            next_state = ETH_SRC_ADDR_STATE;
        end
    end
end
```

READING PCAP DATA IN SIMULATION

```
initial begin : pcap_read_process
  int i, j;
  int packet_size;
  int in_file;
  logic [0:PCAP_FILE_HEADER_SIZE-1] [7:0] file_header;
  logic [0:PCAP_PACKET_HEADER_SIZE-1] [7:0] packet_header;

  @(negedge reset);
  $display("@ %0t: Loading file %s...", $time, PCAP_IN_NAME);

  in_file = $fopen(PCAP_IN_NAME, "rb");
  in_wr_en = 1'b0;
  in_wr_sof = 1'b0;
  in_wr_eof = 1'b0;

  // Skip PCAP Global header
  i = $fread(file_header, in_file, 0, PCAP_FILE_HEADER_SIZE);

  // Read data from image file
  while ( !$feof(in_file) ) begin
    // read pcap packet header & get packet length
    packet_header = {(PCAP_PACKET_HEADER_SIZE){8'h00}};
    i += $fread(packet_header, in_file, i, PCAP_PACKET_HEADER_SIZE);
    packet_size = {<<8{packet_header[8:11]}};
    $display("Packet size: %d", packet_size);

    // iterate through packet length
    j = 0;
    while ( j < packet_size ) begin
      @(negedge clock);
      if (in_full == 1'b0) begin
        i += $fread(in_din, in_file, i, 1);
        in_wr_en = 1'b1;
        in_wr_sof = j == 0 ? 1'b1 : 1'b0;
        in_wr_eof = j == packet_size-1 ? 1'b1 : 1'b0;
        j++;
      end else begin
        in_wr_en = 1'b0;
        in_wr_sof = 1'b0;
        in_wr_eof = 1'b0;
      end
    end
  end

  @(negedge clock);
  in_wr_en = 1'b0;
  in_wr_sof = 1'b0;
  in_wr_eof = 1'b0;
  $fclose(in_file);
  in_write_done = 1'b1;
end
```

IMPLEMENTATION OF UDP READER

- Synchronize packets
  - Augment FIFO architecture to include start-of-frame and end-of-frame signals
  - use start of frame and end of frame to delineate start/end of packets
- 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
- Error checking
  - Use the DISPLAY command in SystemVerilog to display data in the log window
  - Compare against C code output

NEXT…

- Homework 4: UDP Packer Parser