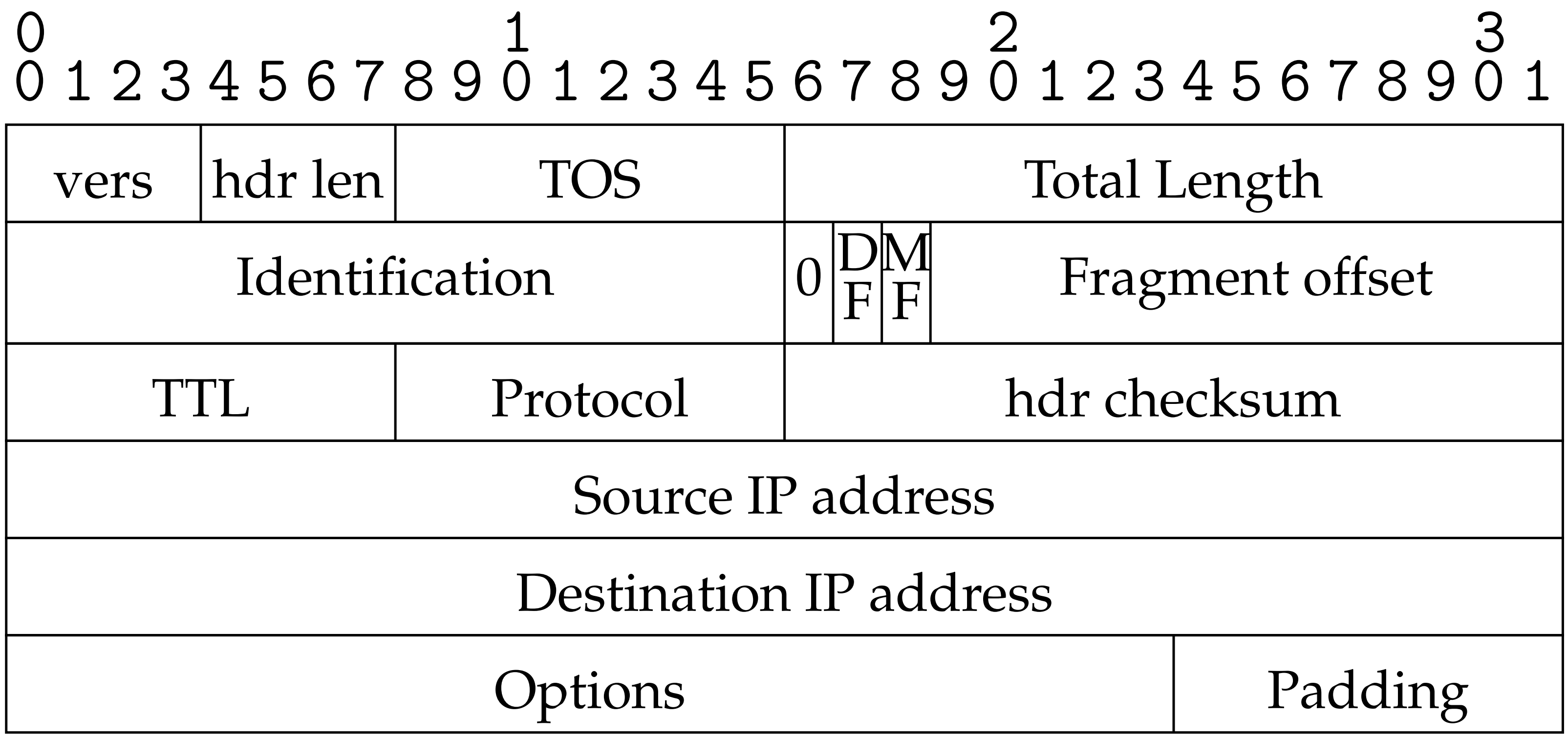
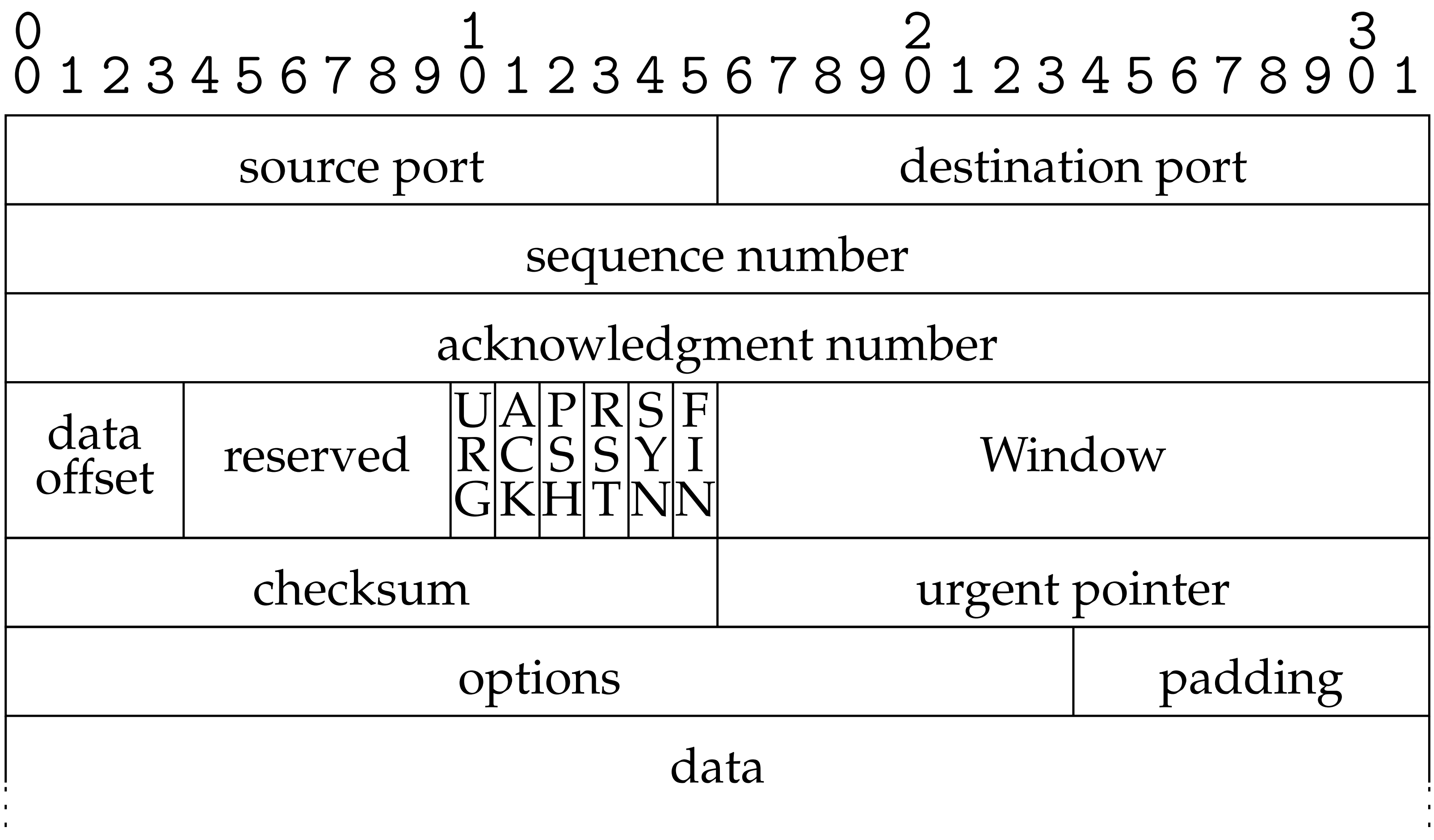
IP header


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
vers
hdr len
TOS
Total Length
Identification
0
DF
MF
Fragment offset
TTL
Protocol
hdr checksum
Source IP address
Destination IP address
Options
Padding

IP header details

- **Routing is based on destination address**
- **TTL (time to live) decremented at each hop (avoids loops)**
- **Fragmentation used for large packets**
  - Fragmented in network if links crossed with smaller MTU
  - MF bit means more fragments for this IP packet
  - DF bit says “don’t fragment” (returns error to sender)
- **Almost always want to avoid fragmentation**
  - When fragment is lost, whole packet must be retransmitted
- **Following IP header is “payload” data**
  - Typically beginning with TCP or UDP header

TCP header


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
source port
destination port
sequence number
acknowledgment number
data offset
reserved
URG
ACK
PSH
RST
SYN
FIN
Window
checksum
urgent pointer
options
padding
data

TCP fields

- Ports
- Seq no. – segment position in byte stream
- Ack no. – seq no. sender expects to receive next
- Data offset – # of 4-byte header & option words
- Window – willing to receive (flow control)
- Checksum
- Urgent pointer

TCP Flags

- URG – urgent data present
- ACK – ack no. valid (all but first segment)
- PSH – push data up to application immediately
- RST – reset connection
- SYN – “synchronize” establishes connection
- FIN – close connection

A TCP Connection (no data)

```
orchard.48150 > essex.discard:
        S 1871560457:1871560457(0) win 16384
essex.discard > orchard.48150:
        S 3249357518:3249357518(0) ack 1871560458 win 17376
orchard.48150 > essex.discard: . ack 1 win 17376
orchard.48150 > essex.discard: F 1:1(0) ack 1 win 17376
essex.discard > orchard.48150: . ack 2 win 17376
essex.discard > orchard.48150: F 1:1(0) ack 2 win 17376
orchard.48150 > essex.discard: . ack 2 win 17375
```

Connection establishment

- **Three-way handshake:**
  - $C \rightarrow S$: SYN, seq S_C
  - $S \rightarrow C$: SYN, seq S_S, ack $S_C + 1$
  - $C \rightarrow S$: ack $S_S + 1$
- **If no program listening: server sends RST**
- **If server backlog exceeded: ignore SYN**
- **If no SYN-ACK received: retry, timeout**

Connection termination

- **FIN bit says no more data to send**
  - Caused by close or shutdown on sending end
  - Both sides must send FIN to close a connection
- **Typical close:**
  - $A \rightarrow B$: FIN, seq S_A, ack S_B
  - $B \rightarrow A$: ack $S_A + 1$
  - $B \rightarrow A$: FIN, seq S_B, ack $S_A + 1$
  - $A \rightarrow B$: ack $S_B + 1$
- **Can also have simultaneous close**
- **After last message, can A and B forget about closed socket?**

TIME_WAIT

- **Problems with closed socket**
  - What if final ack is lost in the network?
  - What if the same port pair is immediately reused for a new connection? (Old packets might still be floating around.)
- **Solution: “active” closer goes into TIME_WAIT**
  - Active close is sending FIN before receiving one
  - After receiving ACK and FIN, keep socket around for 2MSL (twice the “maximum segment lifetime”)
- **Can pose problems with servers**
  - OS has too many sockets in TIME_WAIT, slows things down
  - Hack: Can send RST and delete socket, set SO_LINGER socket option to time 0 (useful for benchmark programs)

Sending data

- **Data sent in MSS-sized segments**
  - Chosen to avoid fragmentation (e.g., 1460 on ethernet LAN)
  - Write of 8K might use 6 segments—PSH set on last one
  - PSH avoids unnecessary context switches on receiver
- **Sender's OS can delay sends to get full segments**
  - Nagle algorithm: Only one unacknowledged short segment
  - TCP_NODELAY option avoids this behavior
- **Segments may arrive out of order**
  - Sequence number used to reassemble in order
- **Window achieves flow control**
  - If window 0 and sender's buffer full, write will block or return EAGAIN

A TCP connection (3 byte echo)

```
orchard.38497 > essex.echo:
        S 1968414760:1968414760(0) win 16384
essex.echo > orchard.38497:
        S 3349542637:3349542637(0) ack 1968414761 win 17376
orchard.38497 > essex.echo: . ack 1 win 17376
orchard.38497 > essex.echo: P 1:4(3) ack 1 win 17376
essex.echo > orchard.38497: . ack 4 win 17376
essex.echo > orchard.38497: P 1:4(3) ack 4 win 17376
orchard.38497 > essex.echo: . ack 4 win 17376
orchard.38497 > essex.echo: F 4:4(0) ack 4 win 17376
essex.echo > orchard.38497: . ack 5 win 17376
essex.echo > orchard.38497: F 4:4(0) ack 5 win 17376
orchard.38497 > essex.echo: . ack 5 win 17375
```

Delayed ACKs

- **Goal: Piggy-back ACKs on data**
  - Echo server just echoes, why send separate ack first?
  - Delay ACKs for 200 msec in case application sends data
  - If more data received, immediately ACK second segment
  - Note: Never delay duplicate ACKs (if segment out of order)
- **Warning: Can interact badly with Nagle**
  - "My login has 200 msec delays"
  - Set TCP_NODELAY
  - In `libasync` library, call `tcp_nodelay (fd);`

Retransmission

- TCP dynamically estimates round trip time
- If segment goes unacknowledged, must retransmit
- Use exponential backoff (in case loss from congestion)
- After $\sim$10 minutes, give up and reset connection
- Problem: Don't necessarily want to halt everything for one lost packet

Congestion avoidance

- **Transmit at just the right rate to avoid congestion**
  - Slowly increase transmission rate to find maximum
  - One lost packet means too fast, cut rate
  - Use additive increase, multiplicative decrease
- **Sender-maintained congestion window limits rate**
  - Maximum amount of outstanding data:
    min(congestion-window, flow-control-window)
- **Cut rate in half after 3 duplicate ACKs**
  - Fewer duplicates may just have resulted from reordering
  - Fast retransmit: resend only lost packet
- **If timeout, cut cong. window back to 1 segment**
  - Slow start – exponentially increase to ss thresh

Other details

- **Persist timer**
  - Sender can block because of 0-sized receive window
  - Receiver may open window, but ACK message lost
  - Sender keeps probing (sending one byte beyond window)
- **Path MTU discovery (optional)**
  - Dynamically discover appropriate MSS
  - Set don't fragment bit in IP, and binary search on known sizes