暨南大學
JINAN UNIVERSITY

International School
Jinan University

Computer Networks

L9 – Transport Layer I

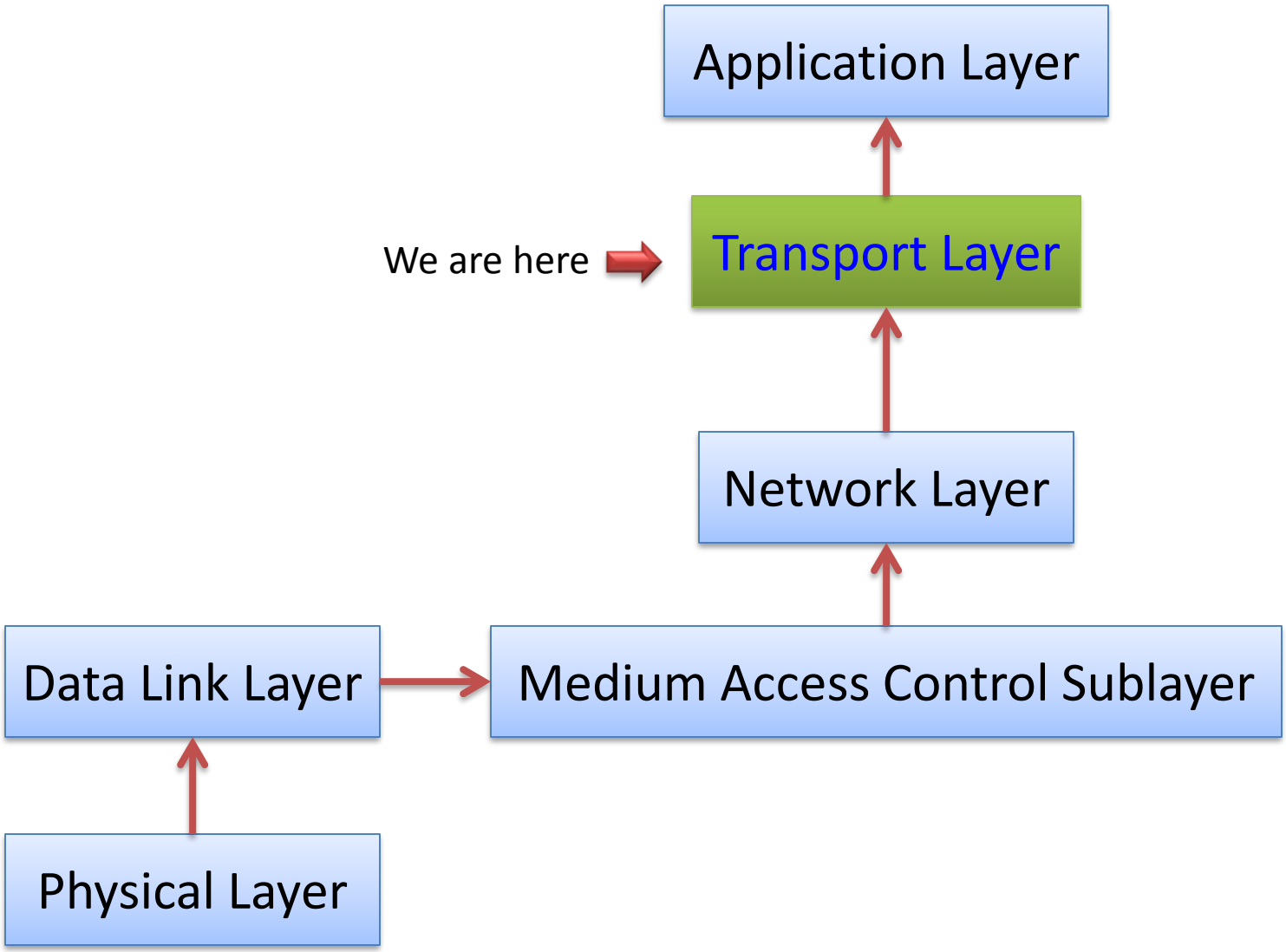
Lecturer: CUI Lin

Department of Computer Science
Jinan University

The Transport Layer

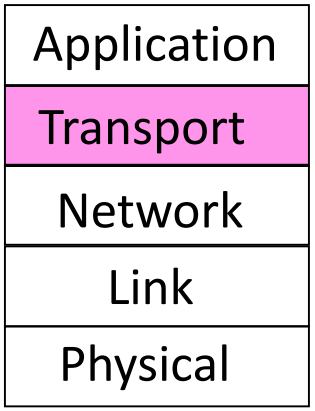
Chapter 6

Roadmap of this course



The Transport Layer

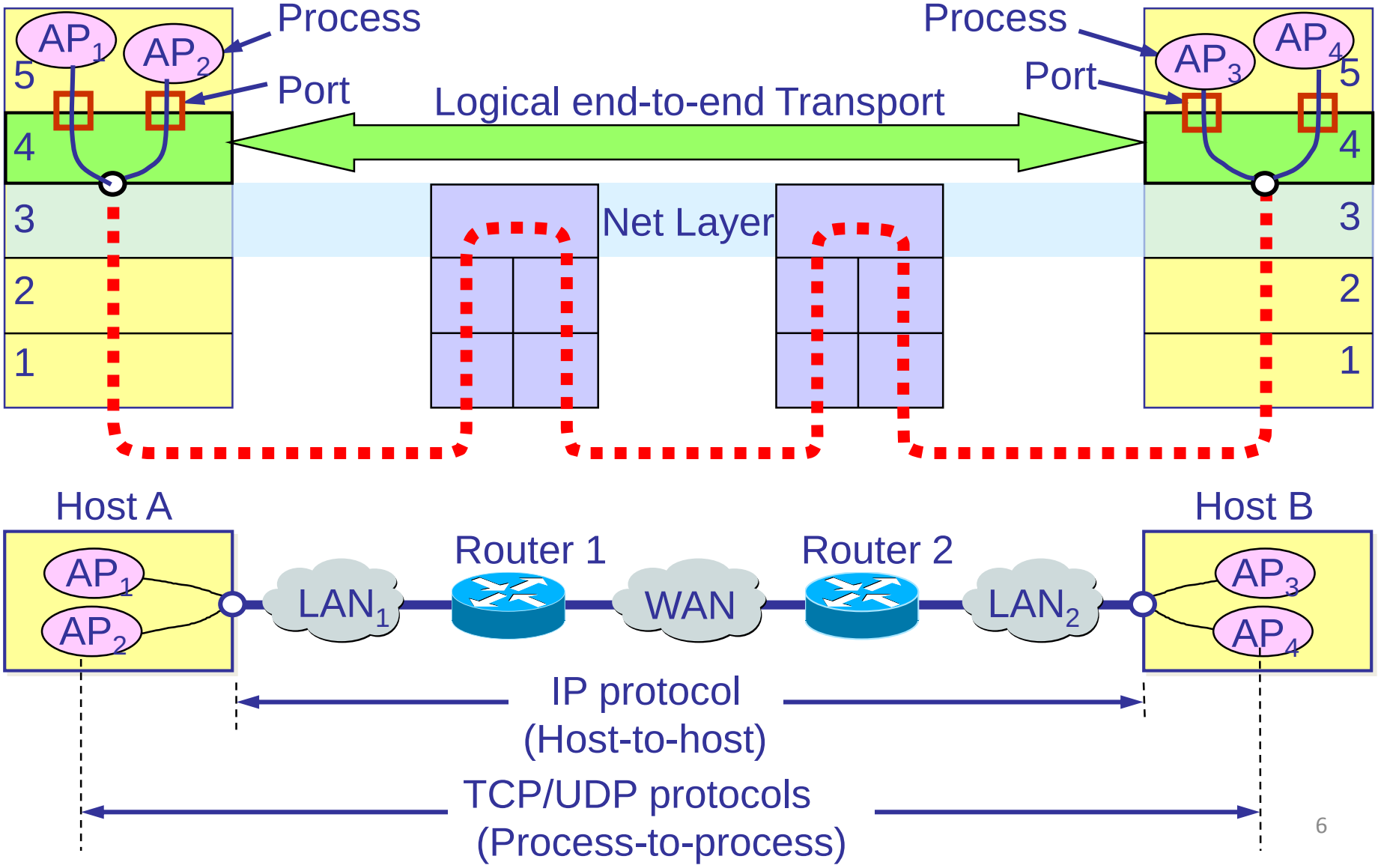
- Responsible for delivering data across networks with the desired reliability or quality



Outline

- Transport Service
- Elements of Transport Protocols
- Internet Protocols – UDP
- Internet Protocols – TCP

Transport Layer

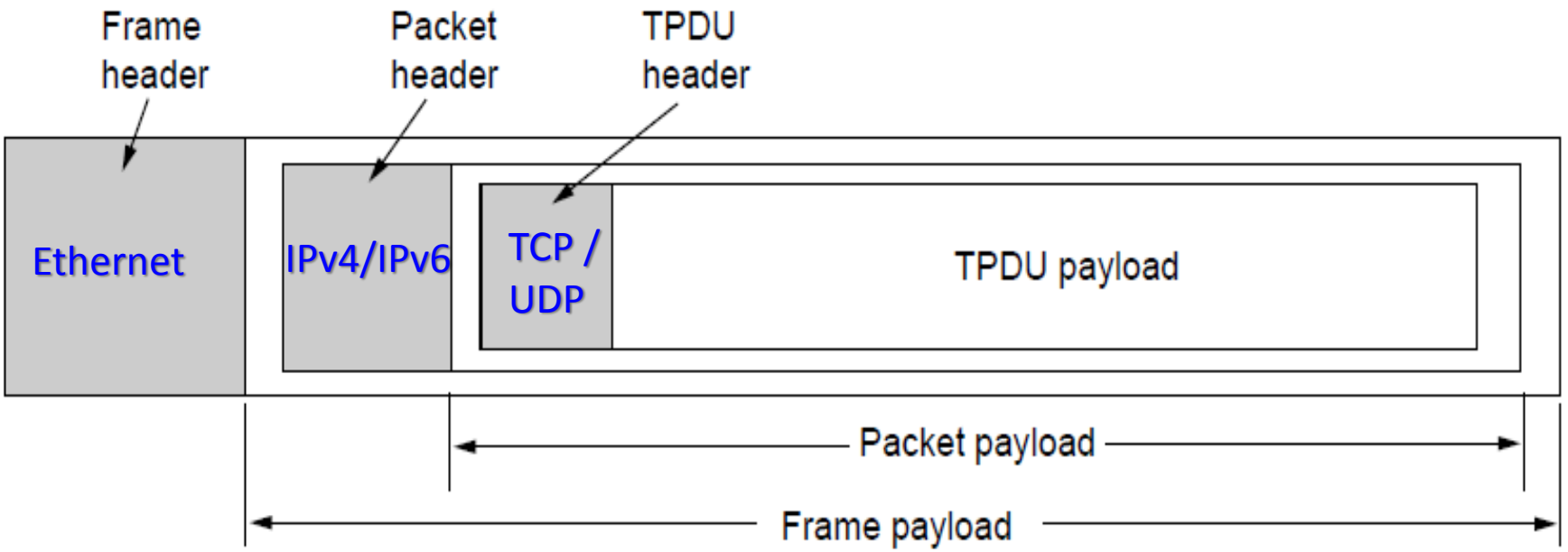


Services Provided to the Upper Layers

- Transport layer adds reliability to the network layer, offers services to applications:
  - Connectionless (e.g., UDP) and
  - Connection-oriented (e.g., TCP) service

Encapsulation

- Transport layer sends segments: TPDU (Transport Protocol Data Unit)



Transport Service Primitives

- Example: primitives that applications might call to transport data for a simple connection-oriented service:
  - Client calls CONNECT, SEND, RECEIVE, DISCONNECT
  - Server calls LISTEN, RECEIVE, SEND, DISCONNECT

Primitive	Segment sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Berkeley Sockets

- Very widely used primitives started with TCP on Unix
  - Notion of "sockets" as transport endpoints
  - Like simple set plus SOCKET, BIND, and ACCEPT

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Socket Example – Internet File Server (1)

Client code:

```
. . .

if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");

. . .
```

Get server's IP address

Make a socket

Try to connect

Socket Example – Internet File Server (2)

Client code (cont.)

```
. . .

    write(s, argv[2], strlen(argv[2])+1);

    while (1) {
        bytes = read(s, buf, BUF_SIZE);
        if (bytes <= 0) exit(0);
        write(1, buf, bytes);
    }
}
```

Write data (equivalent to send)

Loop reading (equivalent to receive) until no more data; exit implicitly calls close

Socket Example – Internet File Server (3)

Server code:

```
. . .

memset(&channel, 0, sizeof(channel));
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s < 0) fatal("socket failed");
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
if (b < 0) fatal("bind failed");

l = listen(s, QUEUE_SIZE);
if (l < 0) fatal("listen failed");

. . .
```

Make a socket

Assign address

Prepare for incoming connections

Socket Example – Internet File Server (4)

Server code

```
. . .
   while (1) {
        sa = accept(s, 0, 0);
        if (sa < 0) fatal("accept failed");

        read(sa, buf, BUF_SIZE);

        /* Get and return the file. */
        fd = open(buf, O_RDONLY);
        if (fd < 0) fatal("open failed");

        while (1) {
              bytes = read(fd, buf, BUF_SIZE);
              if (bytes <= 0) break;
              write(sa, buf, bytes);
        }
        close(fd);
        close(sa);
   }
}
```

Block waiting for the next connection

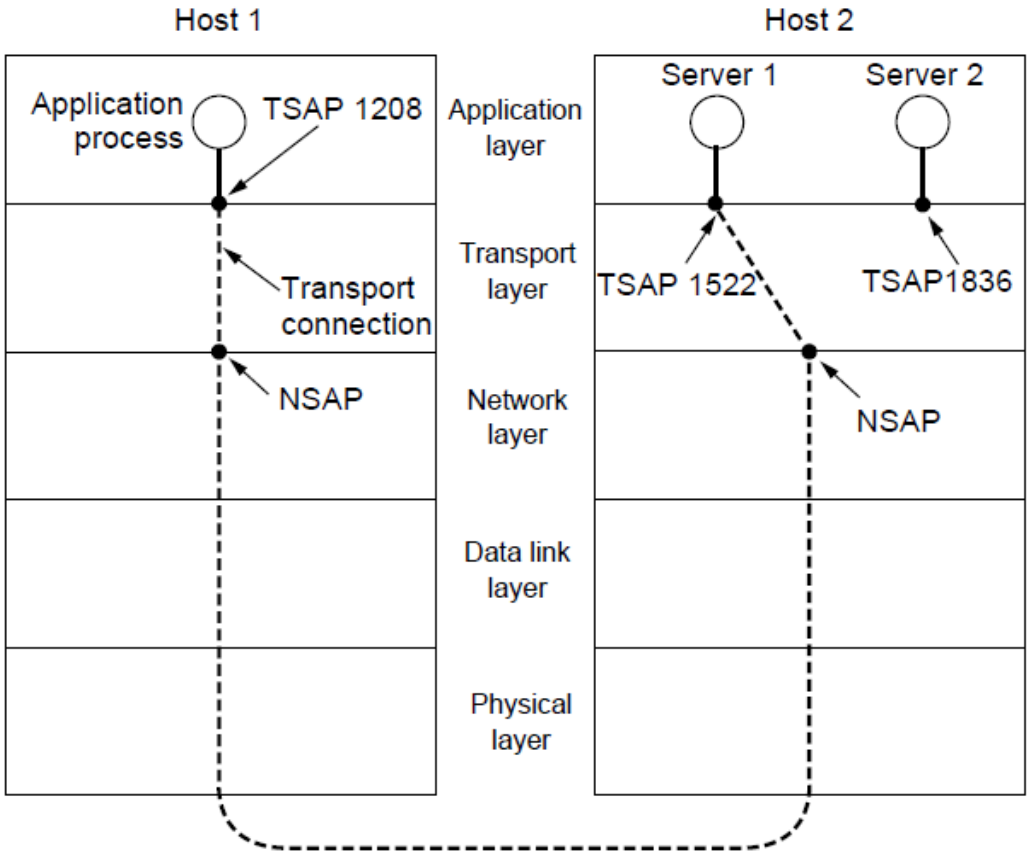
Read (receive) request and treat as file name

Write (send) all file data

Done, so close this connection

Addressing

- Transport layer adds TSAP (Transport Service Access Point)
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP (2 bytes)

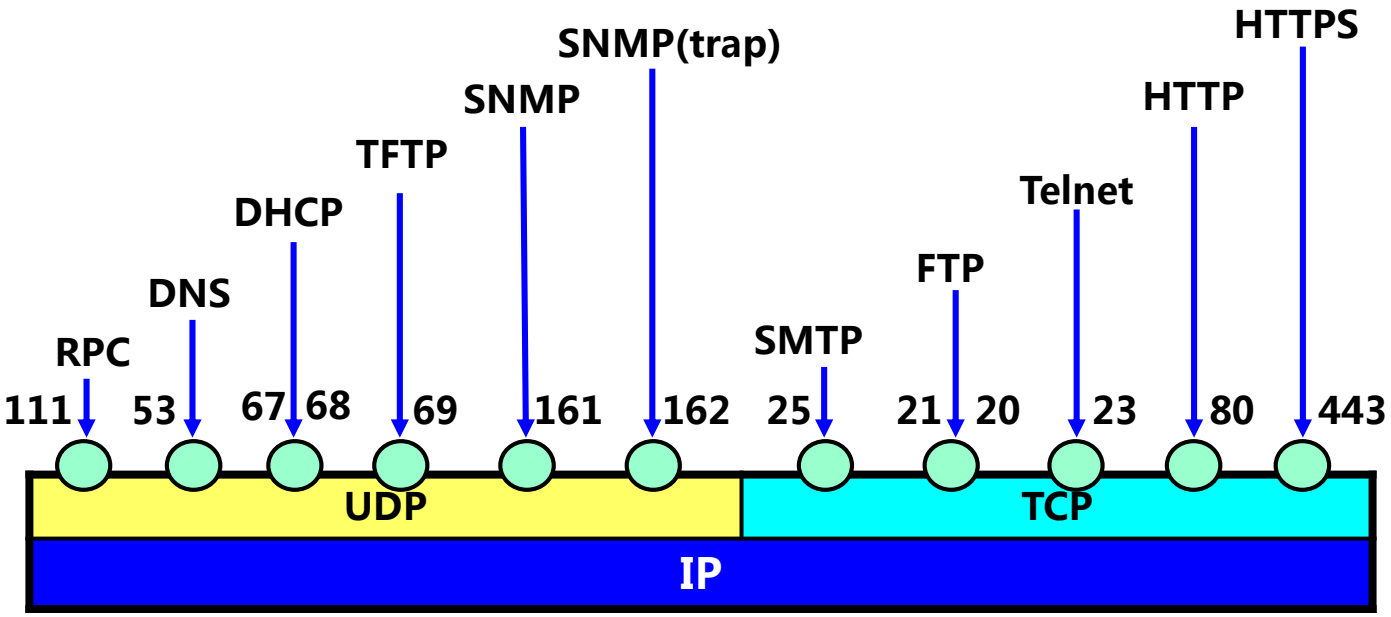


Port Number

- System Ports/Well-known ports (0-1023):
  - used by system processes that provide widely used types of network services, e.g., FTP (20, 21), DNS (53), HTTP (80).
- Registered Ports (1024-49151):
  - assigned by IANA for specific service, e.g., 5004 for RTP
  - On most systems, they can be used by ordinary users.
- Dynamic and/or Private Ports (49152-65535):
  - used for custom or temporary purposes

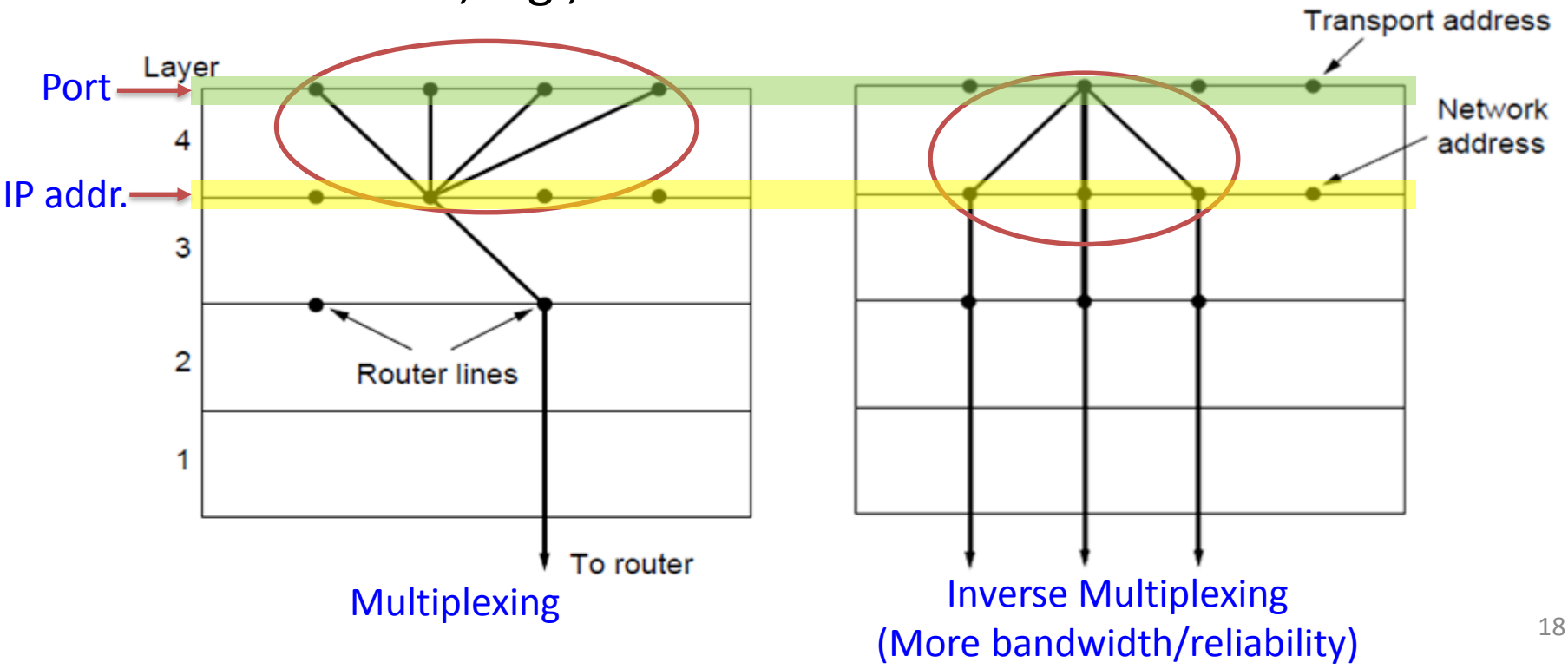
Ref. to Wiki

Several Well-known Ports



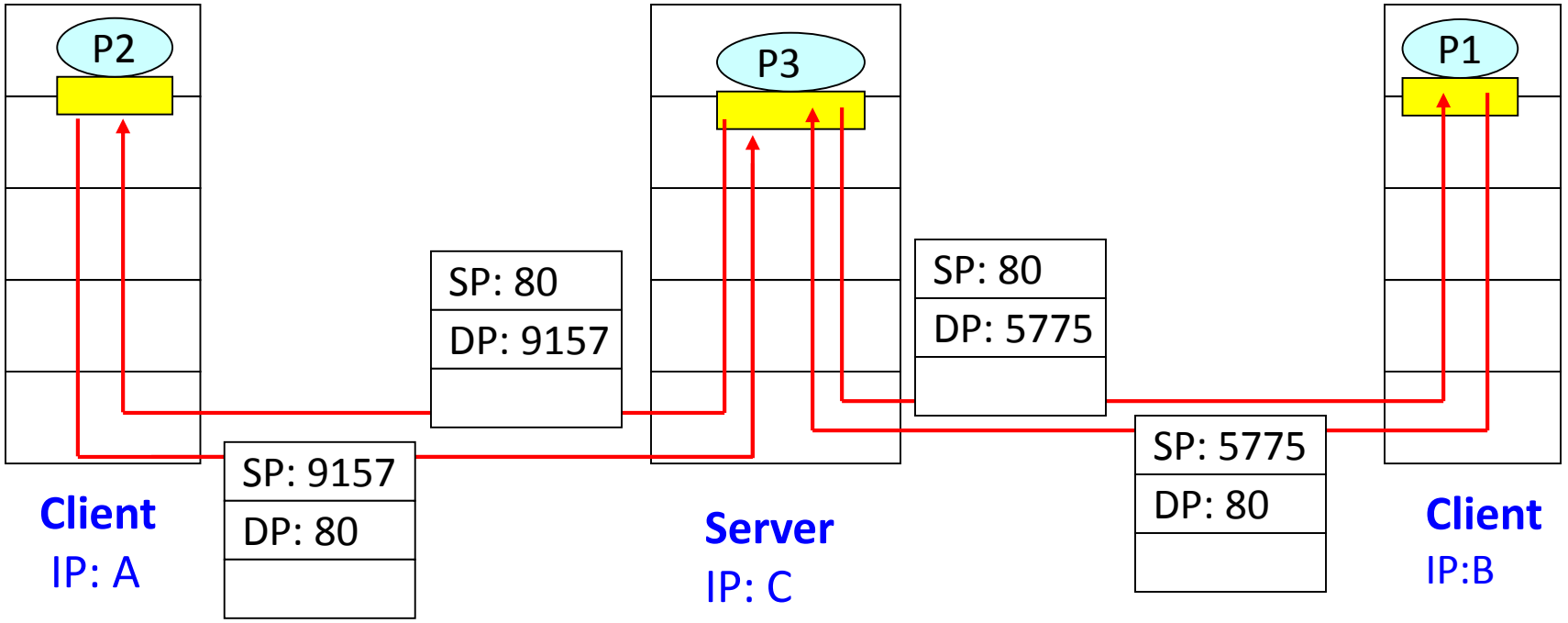
Multiplexing

- Kinds of transport / network sharing that can occur:
  - Multiplexing: connections share one network address (IP addr.)
  - Inverse multiplexing: Multiple network addresses share a connection, e.g., SCTP



Example

SP: Source Port
DP: Dest. Port



SP provides “return address”

Connection Establishment (1)

- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
  - Don't treat an old or duplicate packet as new
  - Use ARQ and checksums for loss/corruption

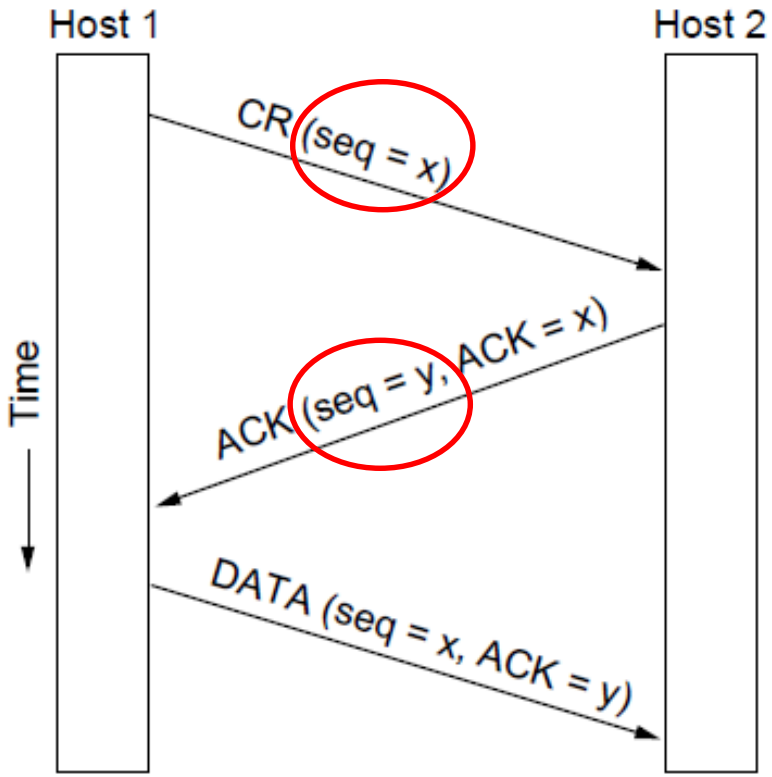
- Approach:
  - Don't reuse sequence numbers within twice the MSL (Maximum Segment Lifetime), e.g., 2MSL = 240s for TCP on the Internet
  - Three-way handshake for establishing connection

Important

Connection Establishment (2)

Three-way handshake
used for initial packet

- Since no state info. from previous connection
- Both hosts contribute fresh seq. numbers



CR = Connect Request

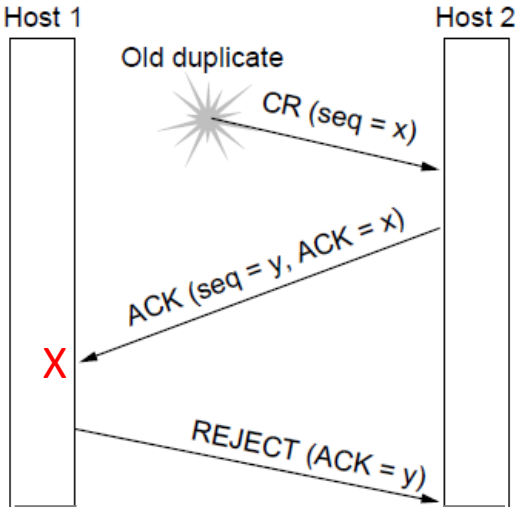
Connection Establishment (3)

Three-way handshake protects against odd cases:

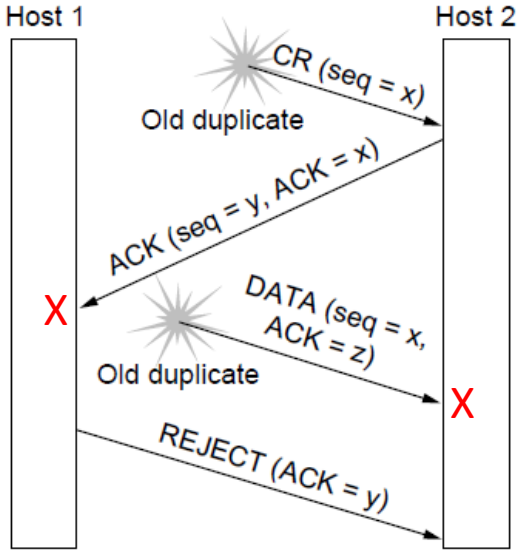
a) Duplicate CR: spurious (假的) ACK does not correct

b) Duplicate CR and DATA: same plus DATA will be rejected (wrong ACK).

a)



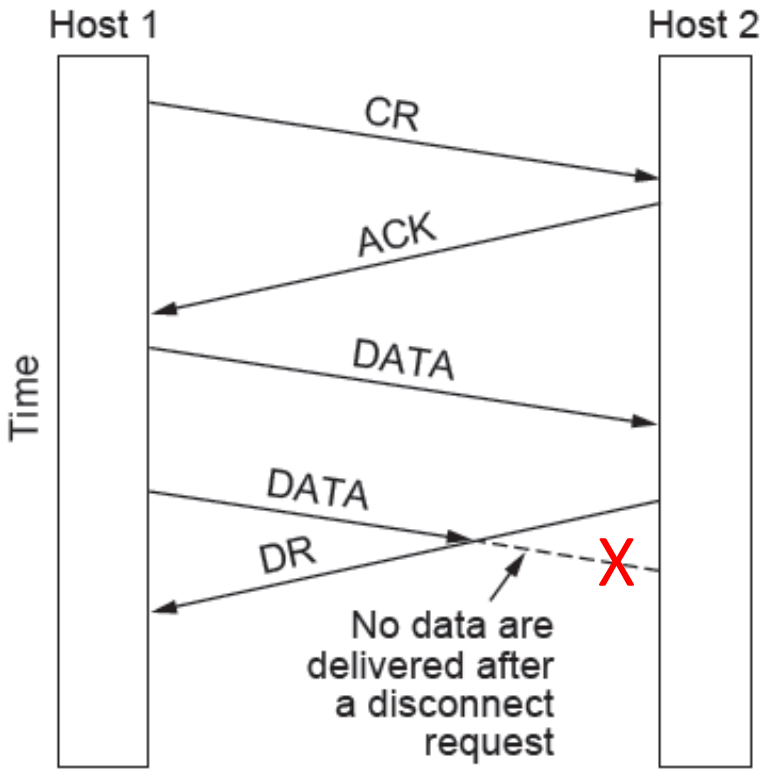
b)



important

Connection Release (1)

- Key problem is to ensure reliability while releasing
- Asymmetric release (when one side breaks connection) is abrupt and may lose data
  - e.g., telephone



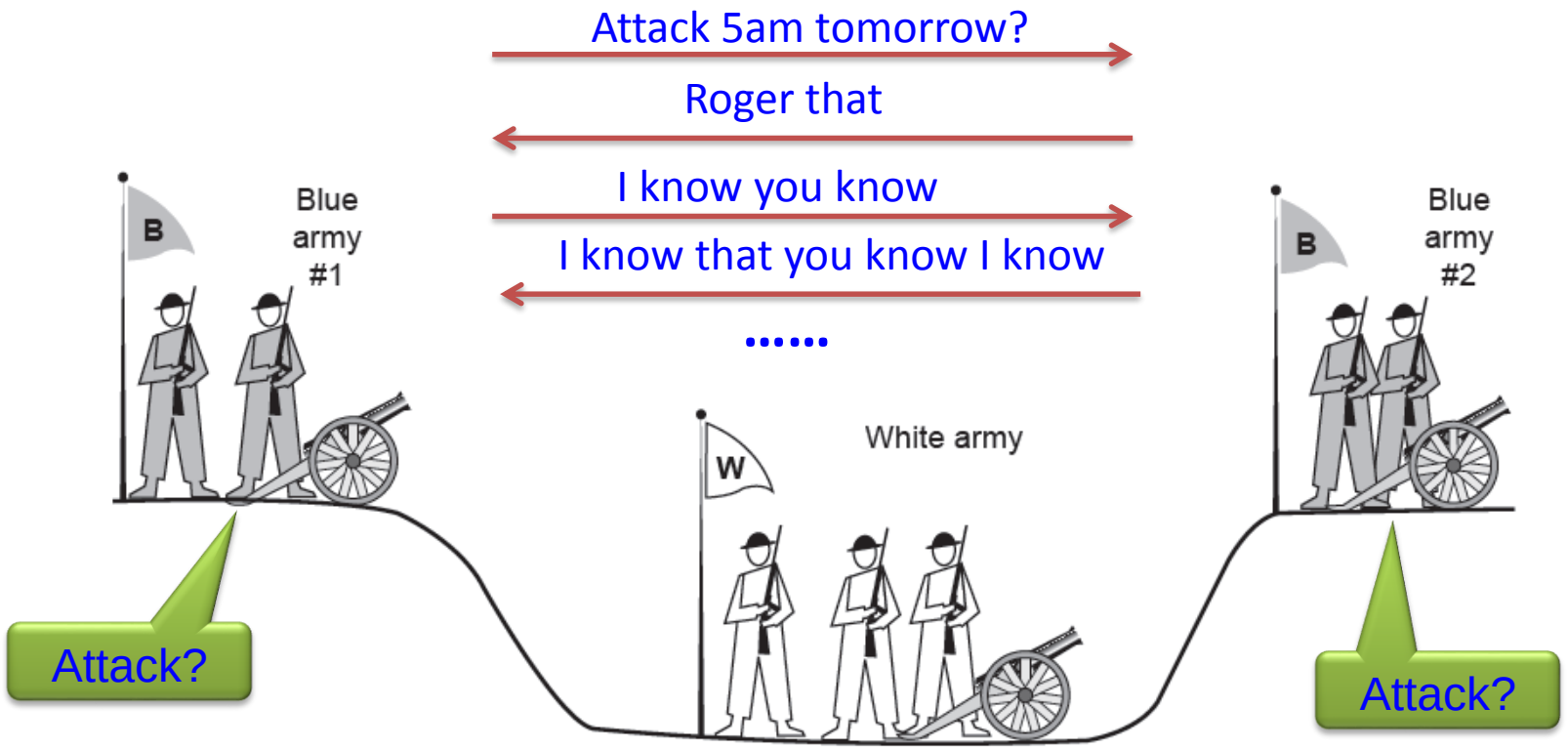
DR = Disconnect Request

Connection Release (2)

- Symmetric release
  - Treat the connection as two separate unidirectional connections
  - Require both sides to release separately
  - Can't be handled solely by the transport layer
  - Two-army problem shows pitfall (陷阱) of agreement

Two-army Problem

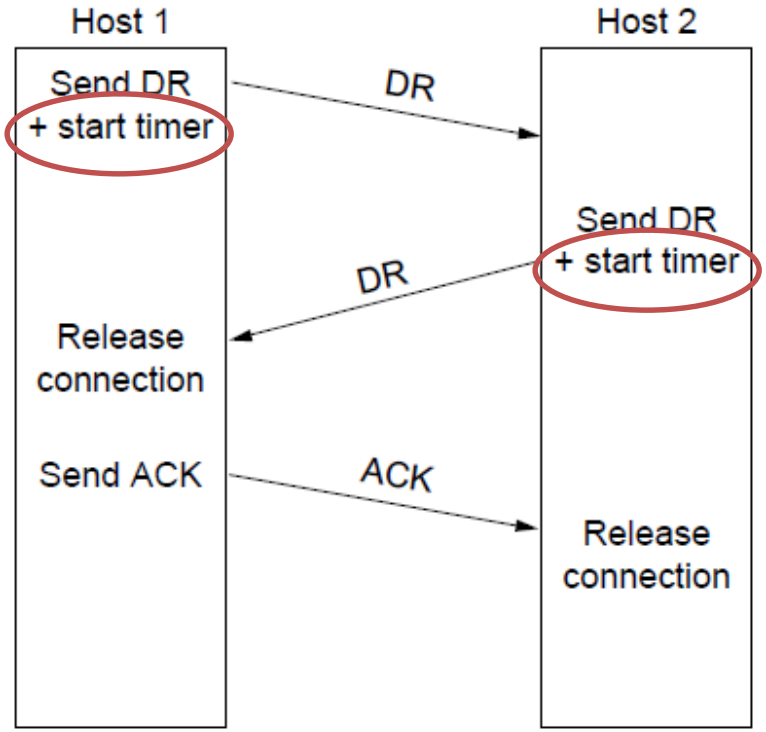
Problem: Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept (拦截) the messengers.



Connection Release (3)

Normal release sequence: initiated by transport user on Host 1

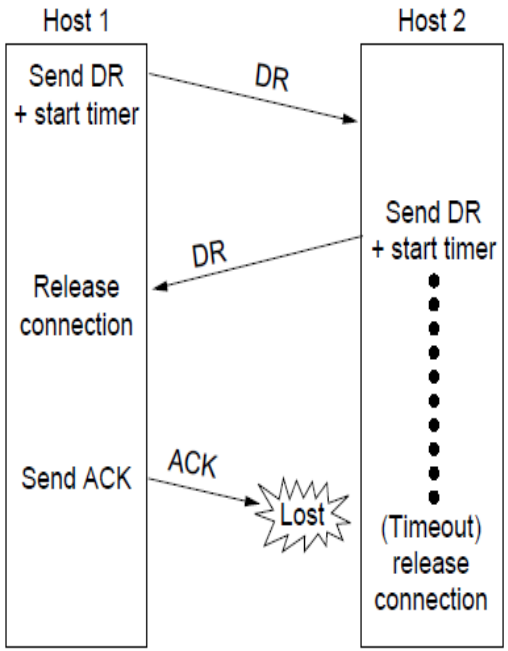
- Both DRs are ACKed by the other side



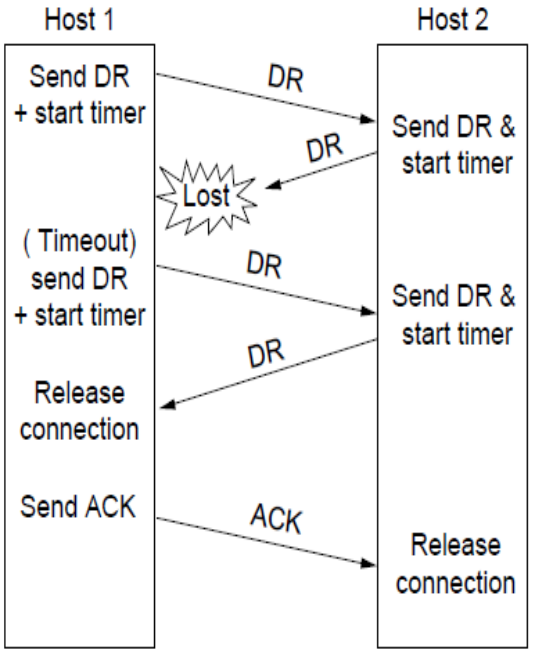
DR=Disconnect Request

Connection Release (4)

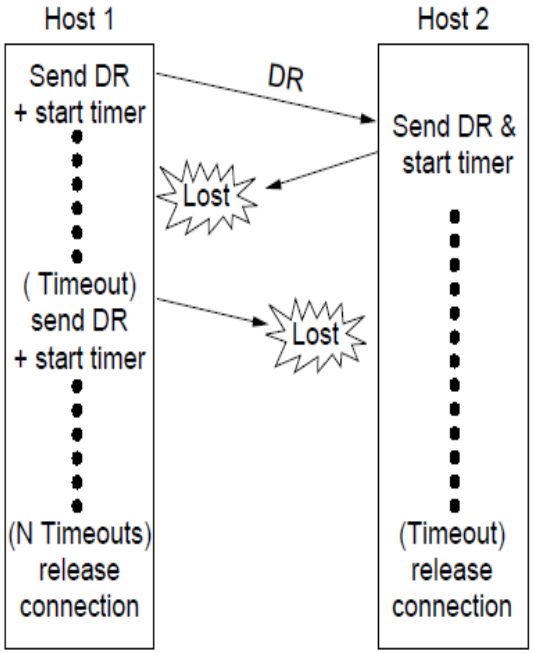
- Error cases are handled with timer and retransmission



Final ACK lost,
Host 2 times out



Lost DR causes
retransmissions



Extreme: Many lost
DRs cause both hosts
to timeout

Error Control and Flow Control (1)

- Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions
- Flow control manages buffering at sender/receiver
  - Problem: data goes to/from the network and applications at different rates
  - Sliding window tells sender available buffering at receiver
  - Makes a variable-size sliding window

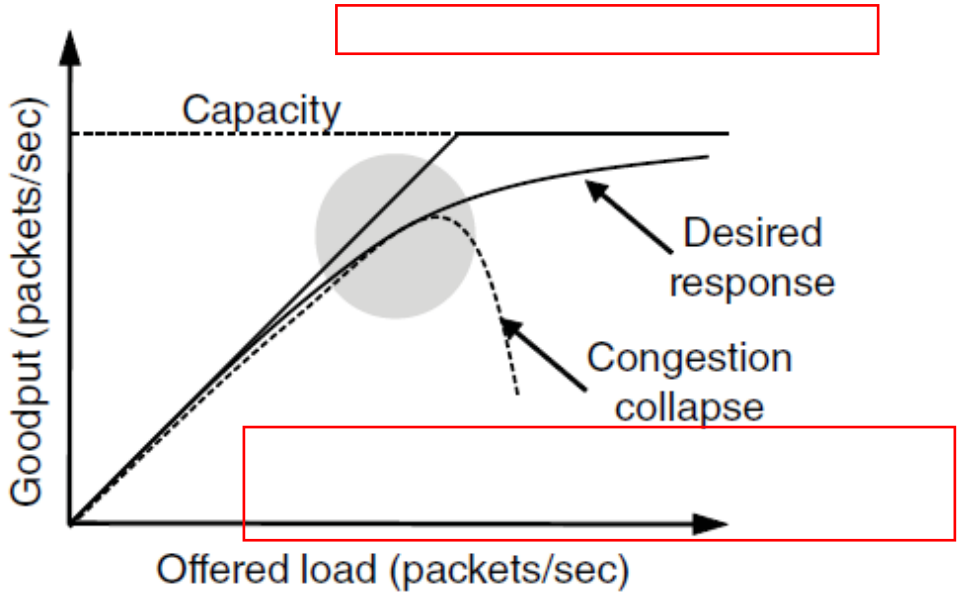
Congestion Control

- Congestion (拥塞): Too many packets in the network causes packet delay and loss that degrades performance.
  - Occurs at routers
- Two layers are responsible for congestion control:
  - Transport layer: controls the offered load
  - Network layer: experiences and detects congestion

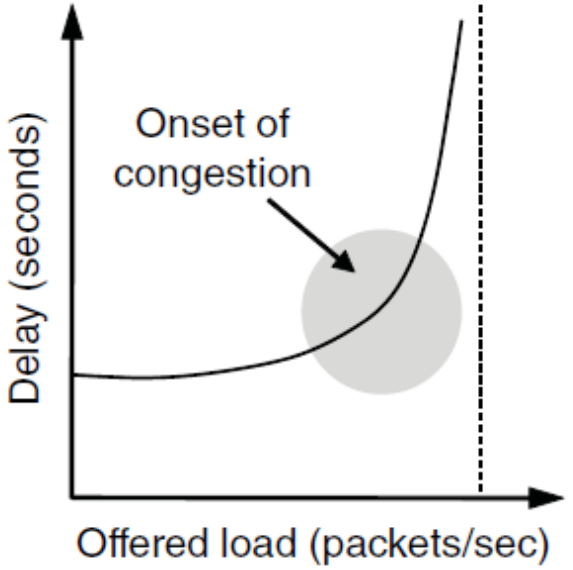
explain detailedly

Desirable Bandwidth Allocation

- Efficient use of bandwidth gives high goodput, low delay



Goodput rises more slowly than load when congestion sets in

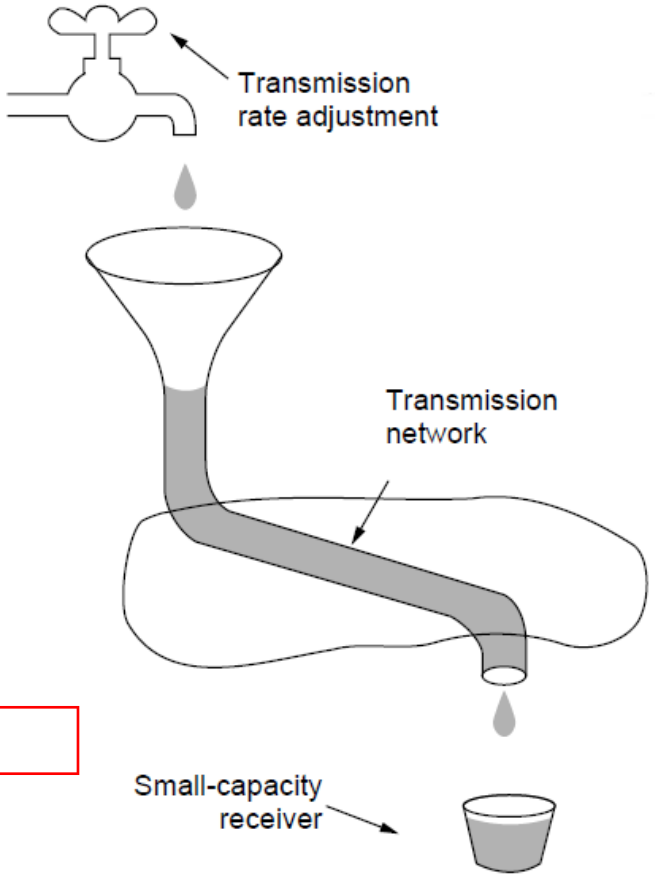


Delay begins to rise sharply when congestion sets in

Regulating the Sending Rate (1)

Sender may need to slow down for different reasons:

- Flow control, when the receiver is not fast enough

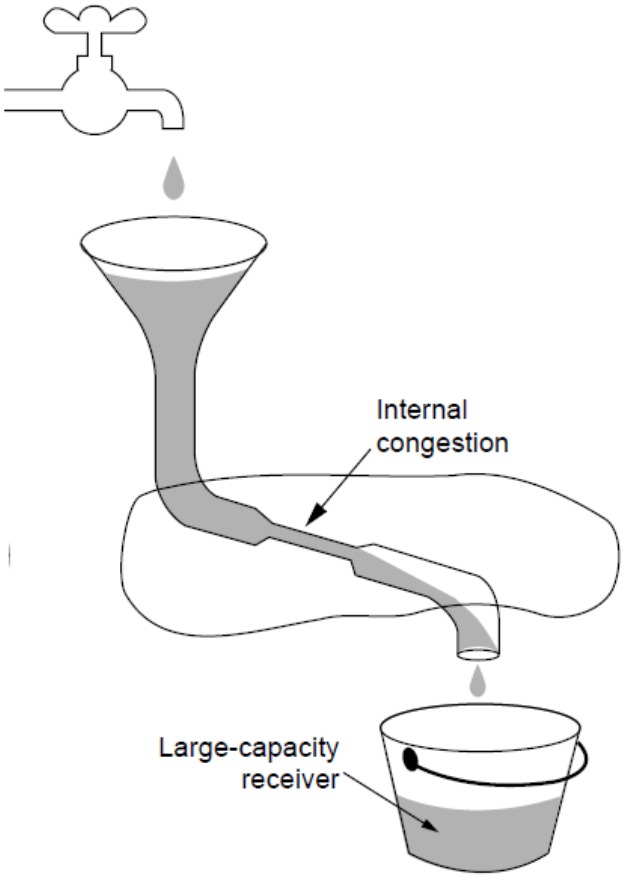


A fast network feeding a low-capacity receiver
→ flow control is needed

Regulating the Sending Rate (2)

Congestion focuses on dealing with the problem:

- when the network is not fast enough



A slow network feeding a high-capacity receiver
→ congestion control is needed

Transport Layer

- Transport Service
- Elements of Transport Protocols
- Congestion Control
- Internet Protocols – UDP
- Internet Protocols – TCP

UDP and TCP

Two main transport protocols in Internet:

- UDP (User Datagram Protocol)
  - Connectionless, unreliable service ("best effort"), datagrams
  - Simple and efficient
- TCP (Transmission Control Protocol)
  - Connection-oriented, reliable service (byte-stream), segment
  - More complex than UDP

Internet Protocols: UDP

- UDP: User Datagram Protocol [RFC 768]
- "Best effort" service, UDP datagrams may be:
  - lost, no retransmission
  - May be delivered out of order to applications
- Connectionless:
  - No handshaking between UDP sender, receiver
  - Each UDP datagram is handled independently
  - No congestion control/flow control

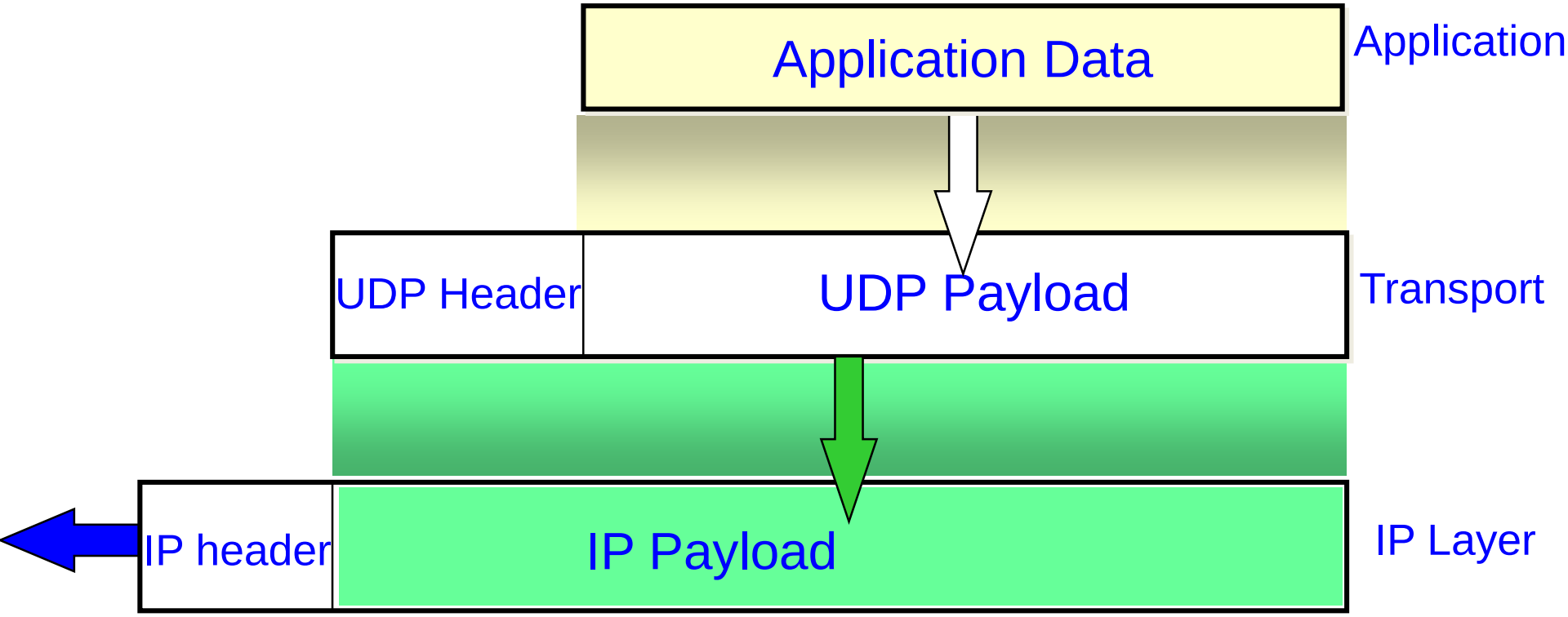
Why do we need UDP?

- No connection establishment (which can add delay)
- Simple: no connection status at sender and receiver
- Small packet header (8 bytes)
- No congestion control: UDP can send datagrams as fast as desired

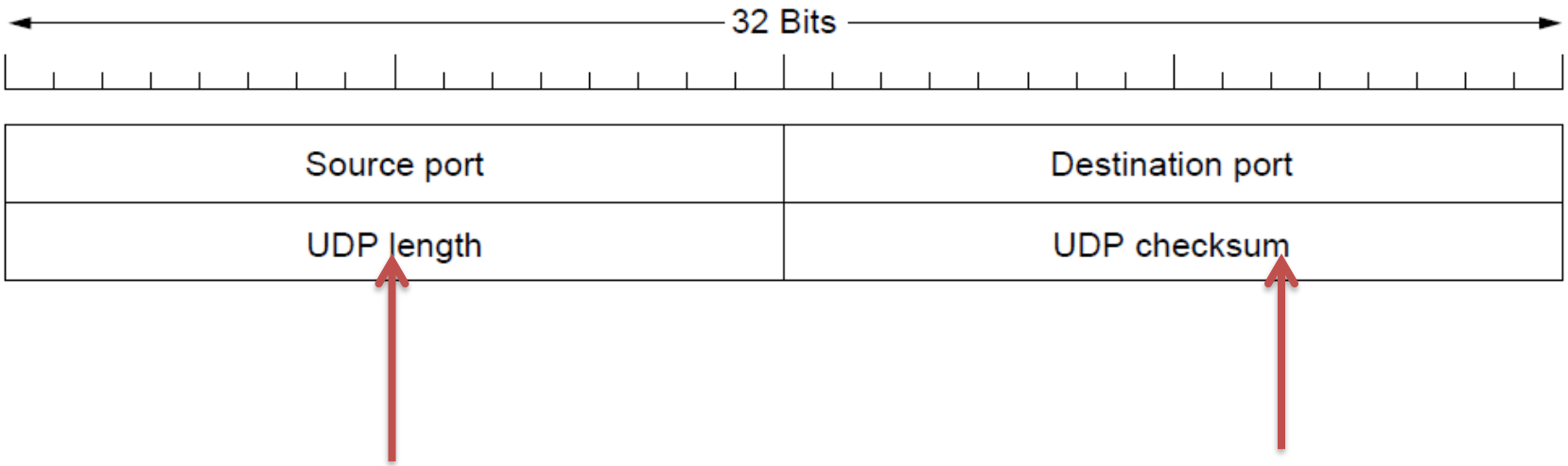
UDP Applications

- Often used for streaming multimedia applications
  - Loss tolerant (可容忍丢包)
  - Rate sensitive (对速率敏感)
- UDP are used in:
  - RIP: To send the route information periodically
  - DNS: Avoid the delay to setup the TCP connection
  - Other protocols: TFTP, DHCP, etc.
  - QUIC and HTTP3
- Add reliability at **application layer** if necessary

UDP Encapsulation



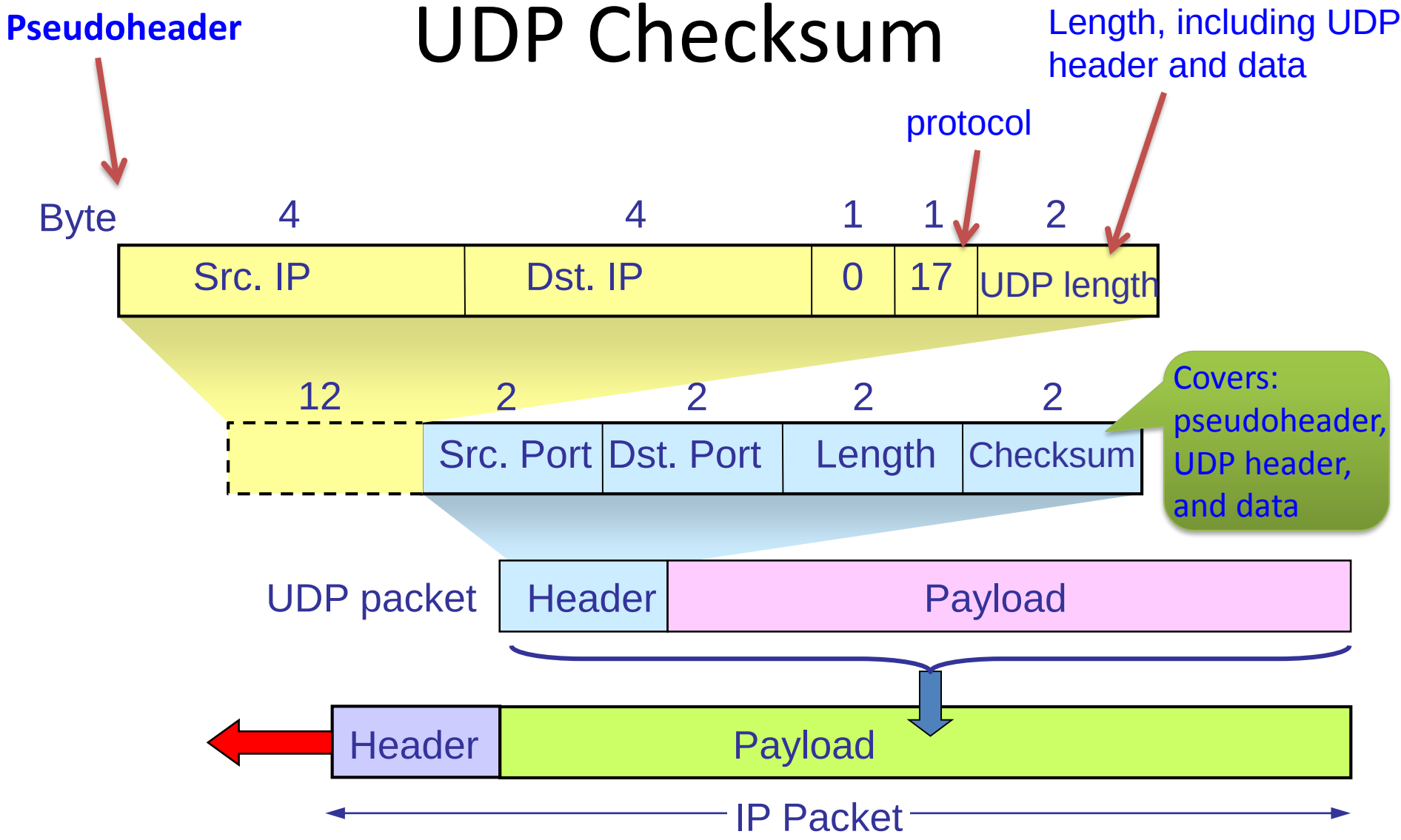
UDP Header



Length of the entire UDP datagram, including header and data

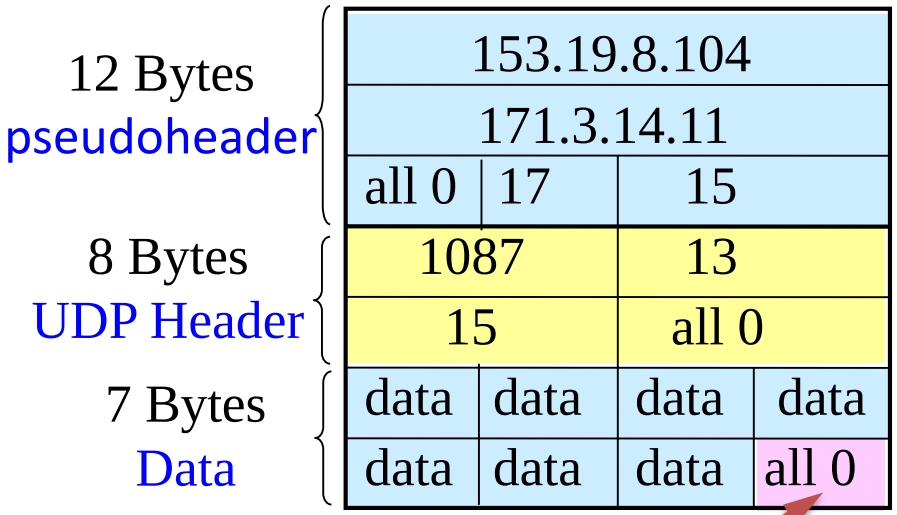
- Optional checksum of entire datagram (including data and pseudo header).
- all-0: source doesn't calculate the checksum

UDP Checksum



Checksum covers UDP segment and IP pseudoheader, providing an end-to-end delivery check. **Pseudoheader is only used for checksum calculation**.

UDP Checksum Calculation



Padding additional zero byte
if length is odd number

10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 and 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0 (Checksum)
01010100 01000101 → data
01010011 01010100 → data
01001001 01001110 → data
01000111 00000000 → data and padding

summed up using 1's complement addition: 10010110 11101101 → summed result

1's complement: 01101001 00010010 → Checksum

Thank you!

Q & A