OSI 7-Layer Model

Application	
Presentation	semantic
Session	session
Transport	stream
Network	packet
Link	frame
Physical	bit

interpretation of 0/1

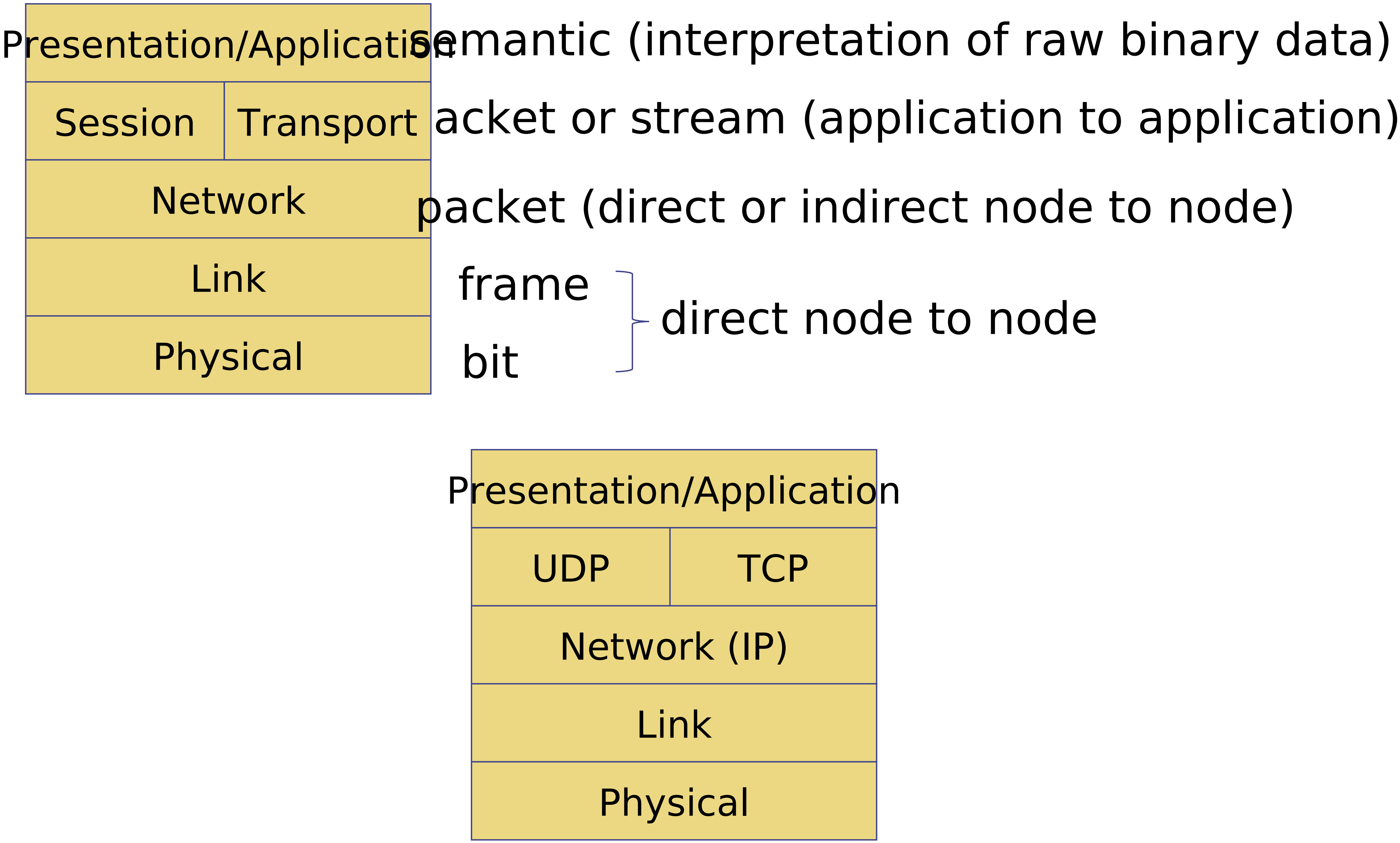
application to application

direct or indirect node to node

raw data just 0/1

direct node to node

OSI 5-Layer Model



IPv4 Header

```
 0               1               3               4
 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  HL   |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Options (length specified by HL)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version: 4
HL: header length, minimum 5 (x32 bits)
Type of Service: priority (real-time voice etc.), but now may be redefined
ID + Fragment offset: the group ID of a datagram, which may be fragmented in transmission
Flags: Specify if fragmentation is allowed
Protocol: the protocol used in the data portion, helping interpret the data portion (e.g., udp or tcp headers)

Internet Protocol (IP)

- Service
  - Send/receive a packet to/from a remote machine
- Interface 1: **`IP_Send(dest, buf)`**
  - Create a packet (**`IP header + buf`**)
  - Find a routing path to host **`dest`**
  - Send the data in **`buf`** to host **`dest`**
- Interface 2: **`IP_Recv(buf)`**
  - Receive an IP packet
  - Deposit the packet into **`buf`**
  - Return the packet size

Problems of IP

- The interface is called by all applications in the same machine
  - How to decide which application gets which packets?
- IP Packets have limited size
  - Each packet can be no more than 64K bytes
- IP is connectionless and does not guarantee packet delivery
  - Packets can be delayed, dropped, reordered, duplicated
- No congestion control

Implementation of UDP and TCP

CS587x Lecture 2

Department of Computer Science
Iowa State University

User Datagram Protocol (UDP)

- Goal
  - Provide application-to-application communication
- Idea: Concept of port
  - Each connection links to a specific *port*
    - (srcIP, srcPort, dstIP, dstPort) uniquely identifies each connection
- Interface
  - **`UDP_Send(dstIP, buf, port)`**
  - **`UDP_Recv(buf, port)`**
- Service
  - Send datagram from (srcIP, srcPort) to (dstIP, dstPort)
  - Service is unreliable, but error detection possible

UDP Send

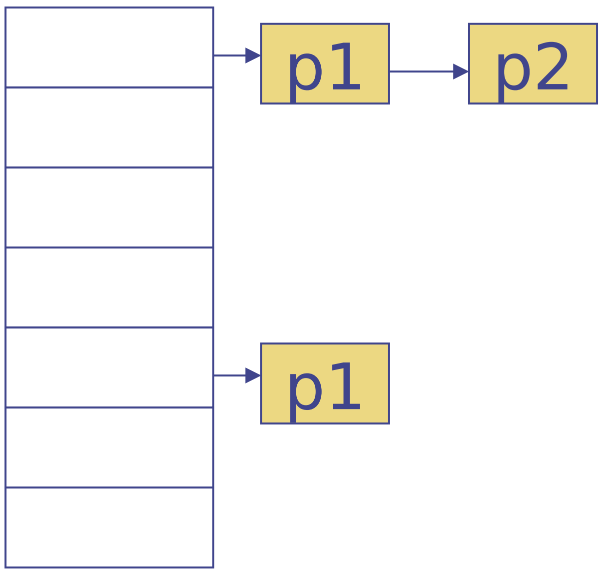
- `UDP_Send(dstIP, buf, port)`
  1. **Prepare header**
     1) **source port (usually any available port)**
     2) **destination port**
     3) **UDP length**
     4) **UDP checksum**
  2. **IP_Send(dstIP, header+buf)**

0 … 15	16 … 31
Source Port	Destination Port
UDP length	UDP checksum
Payload (variable)	

Header: 8 bytes

UDP Receive

Port List



- **`UDP_Recv(buf, port)`**
  1. **`IP_recv(buf)`**
  2. Get **`port`** information from the udp header encoded in **`buf`**
  3. Any listener on **`port`**?
     a) Yes, drop the payload to the message queue of the listener and wake it up (if it is waiting)
     b) No, discard the packet

0 … 15	16 … 31
Source Port	Destination Port
UDP length	UDP checksum
Payload (variable)	

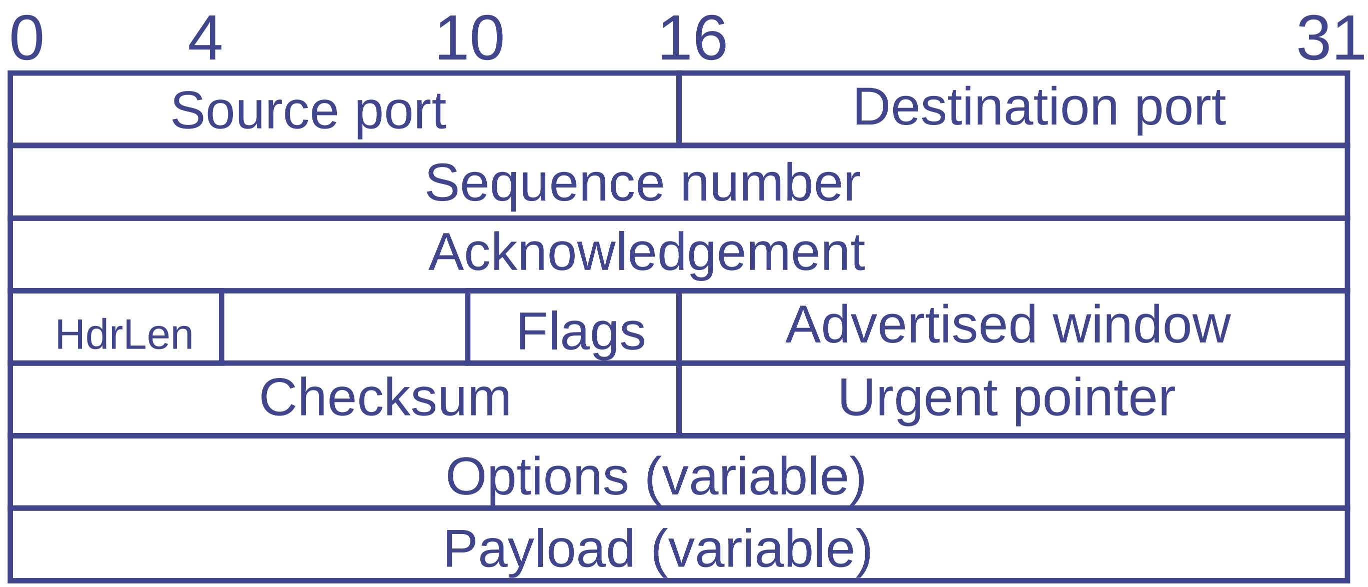
Problems of Port

- Each connection links to a specific *port*
  - (srcIP, srcPort, dstIP, dstPort) uniquely identifies each connection
- Totally there are 65535 ports
  - *Well known ports* (0-1023): everyone agrees which services run on these ports
    - e.g., ssh:22, http:80, snmp: 24
    - Access to these ports needs administrator privilege
  - Ephemeral ports (most 1024-65535): given to clients
    - e.g. chatclient gets one of these
    - Port contention rises

Transmission Control Protocol (TCP)

- Design Goals
  - Provide application-to-application communication
  - Messages can be of arbitrary length
  - Provide reliable and in-order delivery
  - Provide congestion control and avoidance
- Basic idea to address reliability
  - Using timer: sending a packet and waiting for an acknowledgement from the receiver
    - if the ACK is received within a given time period, the packet is received;
    - otherwise, the packet is assumed to be lost

TCP Header

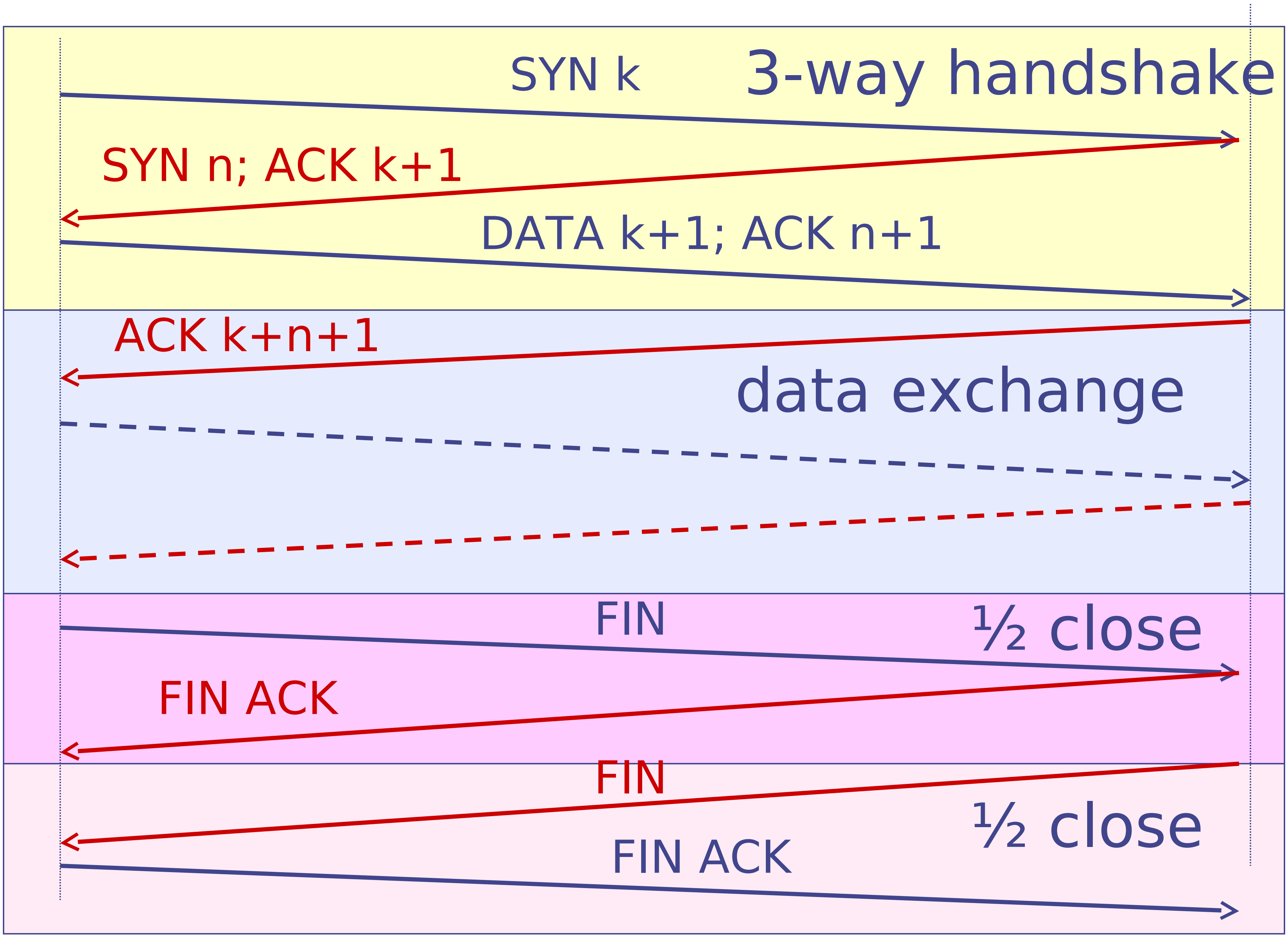


- Sequence number, acknowledgement, and advertised window – used by *sliding-window based flow control*
- Flags:
  - SYN – establishing a TCP connection
  - FIN – terminating a TCP connection
  - ACK – set when Acknowledgement field is valid
  - URG – urgent data; Urgent Pointer says where non-urgent data starts (not defined by standard, but specific to implementation)
  - PUSH – don’t wait to fill segment
  - RESET – abort connection

TCP Implementation

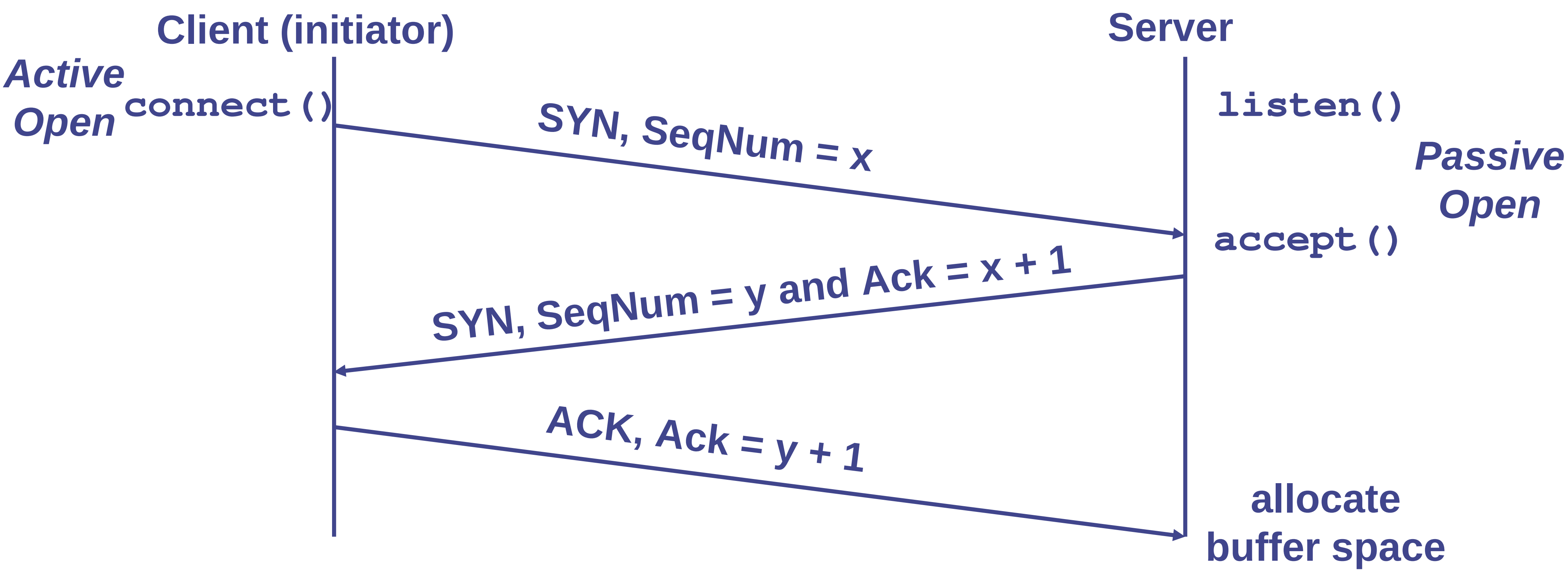
- Start a connection
- Reliable byte stream delivery from (srcIP, srcPort) to (dstIP, dstPort)
  - Indication if connection fails: Reset
- Terminate the connection

Connect/Exchange/ Terminate



Connection: 3-Way Handshake

- Three messages (i.e., syn, syn-ack, ack) are exchanged before data transmission
- Exchange sequence number, total buffer size and the size of the largest segment that can be handled at each side



The initial SeqNums must be randomized!!

Why 3WH?

- Three-way handshake adds 1 RTT delay
  - Expensive for small connections such as RPC
- Why?
  - Congestion control: SYN (40 byte) acts as cheap probe
    - Both sender and receiver has now had one round of trip, which tells some information such as their distance.
  - Protects against delayed packets from other connection (would confuse receiver)

DoS: TCP SYN Flooding

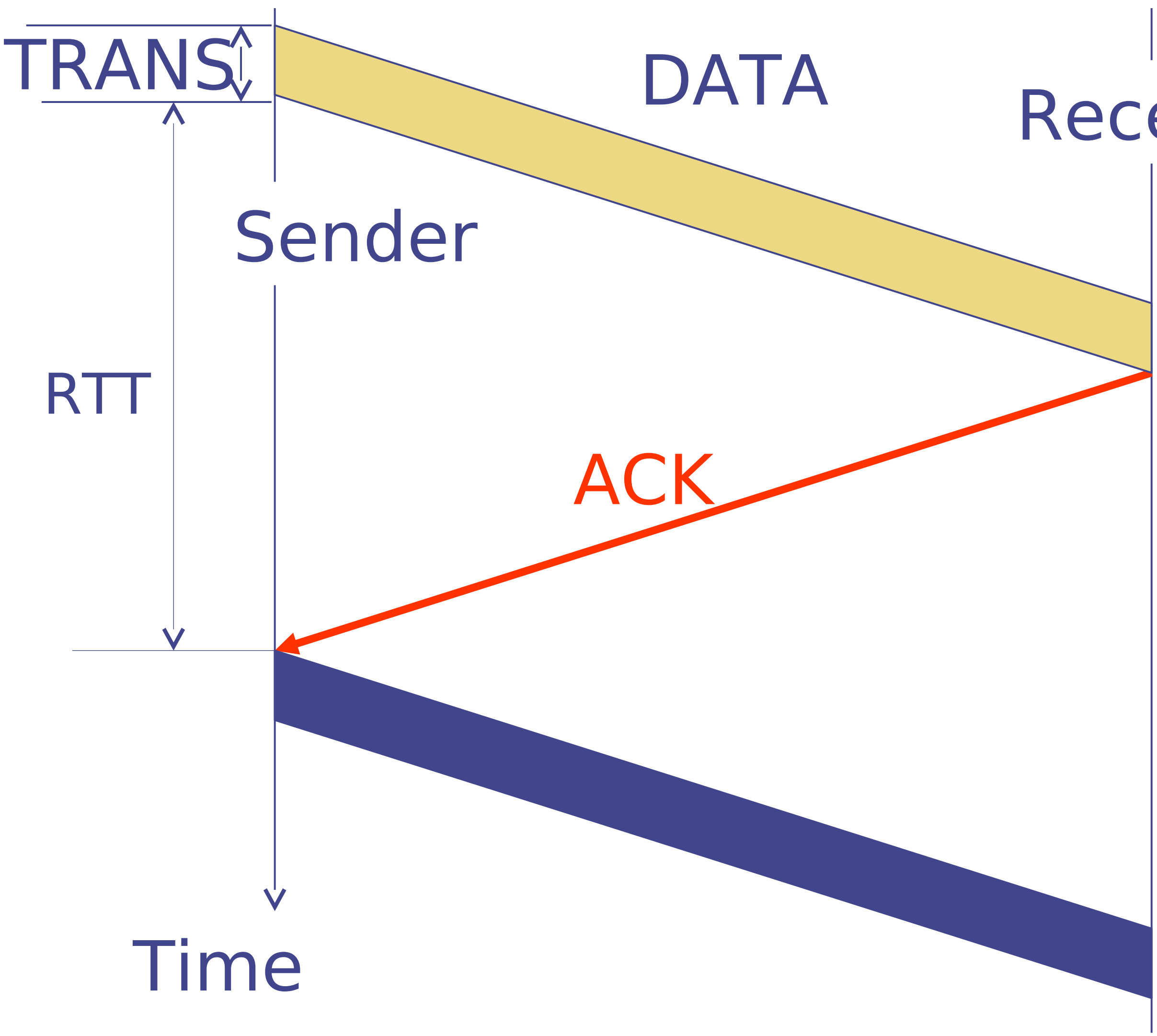
- How it works: exhausting system resources
  - Using a faked IP address
  - Initiates a TCP connection to a server with a faked IP address
    - Sends a SYN message
    - The server responds with a SYN-ACK
    - Since the address does not exist, the server needs to wait until time out
      - The server never receives the ACK (the final stage of the TCP connection)
  - Repeat with a new faked IP address
    - Repeat at a pace faster than the TCP timeouts release the resources
    - All resources will be in use and no more incoming connection requests can be accepted.
- Some common ways to present
  - Install firewall
    - choose deny instead of reject, which sends a message back to the sender
  - Close all ports that are not in using
  - Deny requests from unusual IP addresses
    - Private address
    - Multicast address, etc.

Termination: 4-Way Handshake

- Four messages (FIN, ACK, FIN, ACK) are exchanged to terminate a connection
  - FIN from B to A
    - B does not transmit any new data, but is still responsible for any corrupted data
  - ACK from A to B
  - FIN from A to B
    - After reading all of the bytes from B, A sends FIN to B
  - ACK from B to A
    - The connection is formally closed

Exchange: Stop & Wait

- Send; wait for ack
- If timeout, retransmit; else repeat

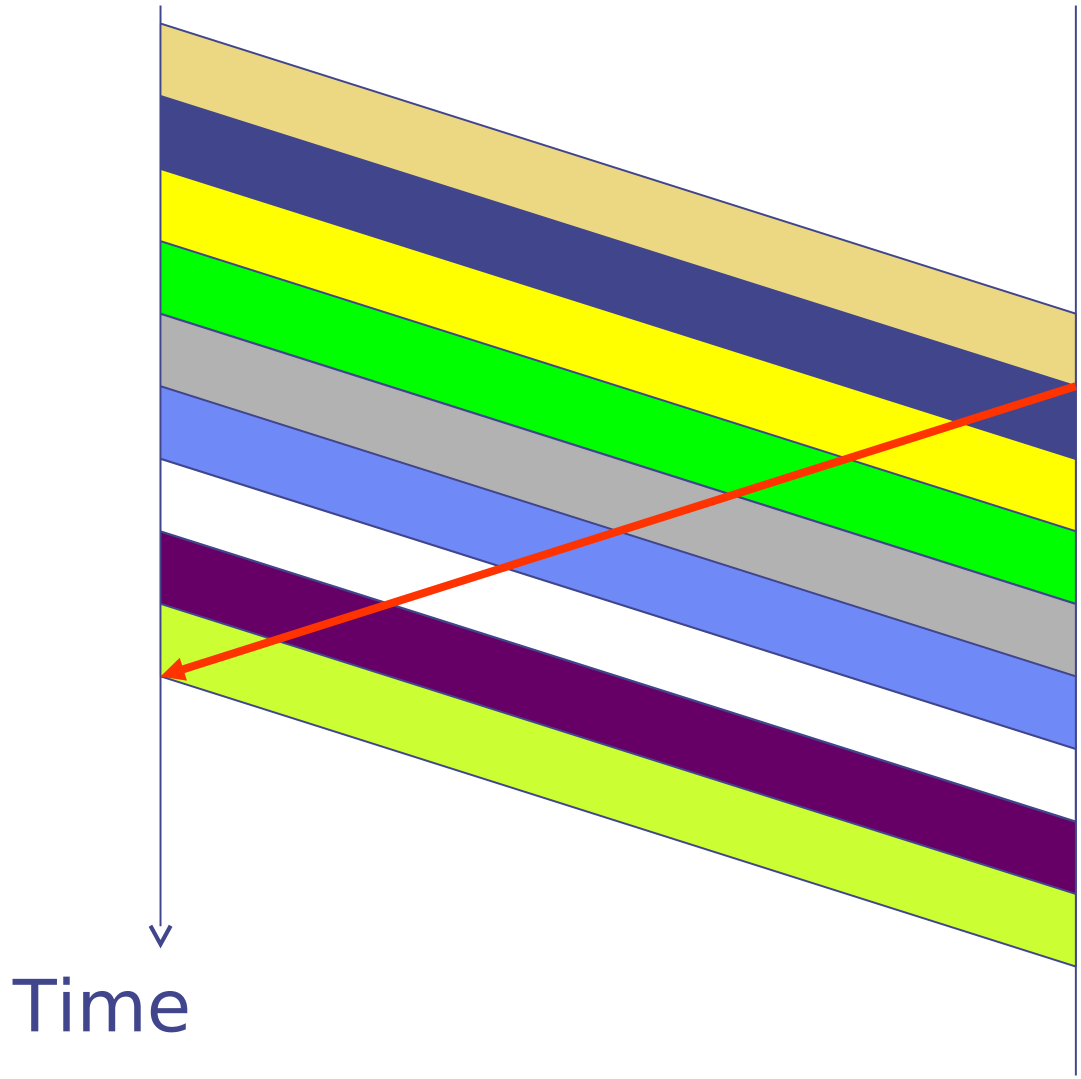


Inefficient if TRANS << RTT

Exchange: Go-Back-n (GBN)

- Sliding Window Protocol
  - Transmit up to n unacknowledged packets/bytes
  - If timeout for ACK(k), retransmit k, k+1, …

Example without errors

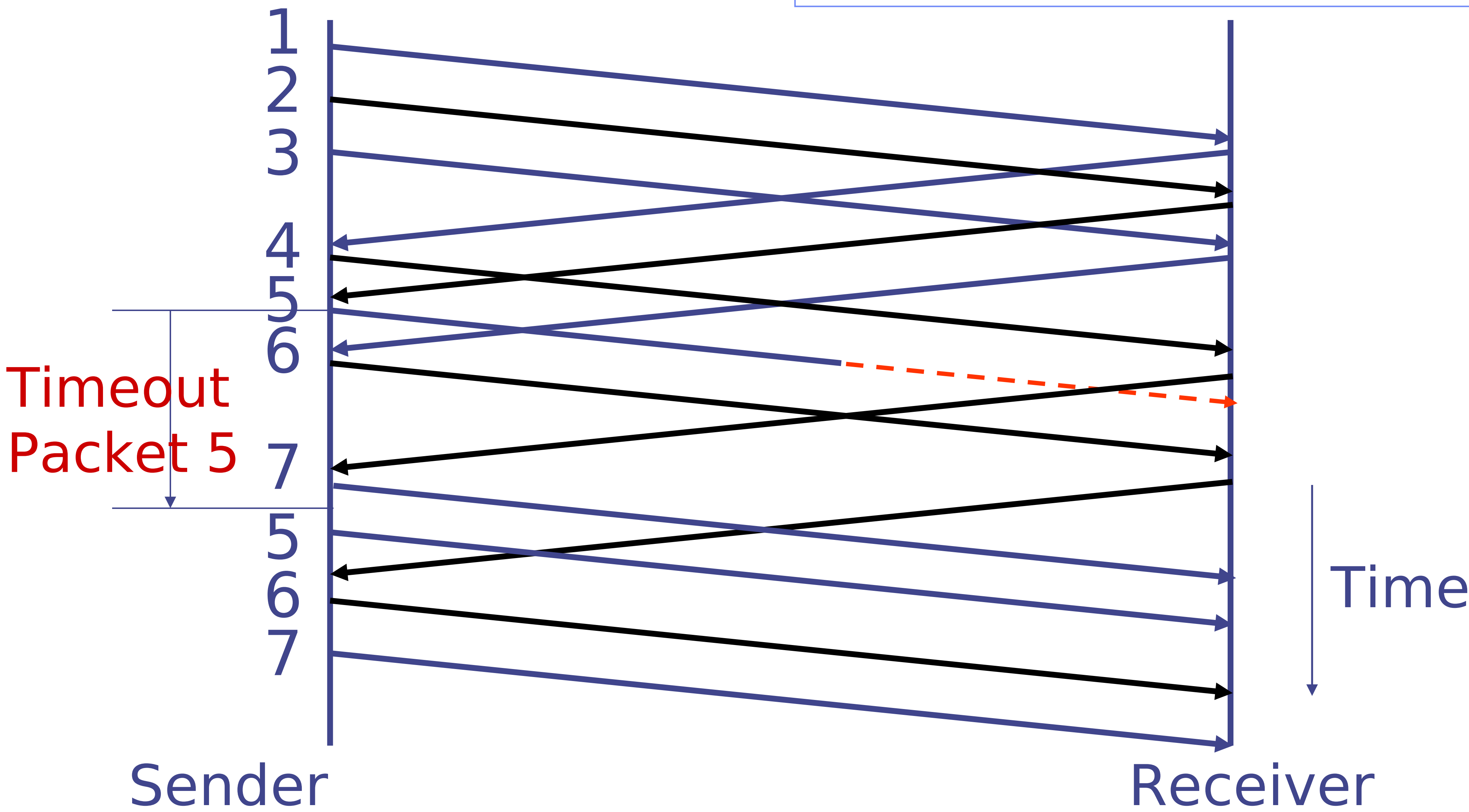


n = 9 packets in one RTT instead of 1

→ Fully efficient

Example with errors

Window size = 3 packets



Observations

- Pros:
  - It is possible to fully utilize a link, provided the sliding window size is large enough. Throughput is ~ (w/RTT)
  - Stop & Wait is like w = 1.
- Cons:
  - Sender has to buffer all unacknowledged packets, because they may require retransmission
  - Receiver may be able to accept out-of-order packets, but only up to its buffer limits

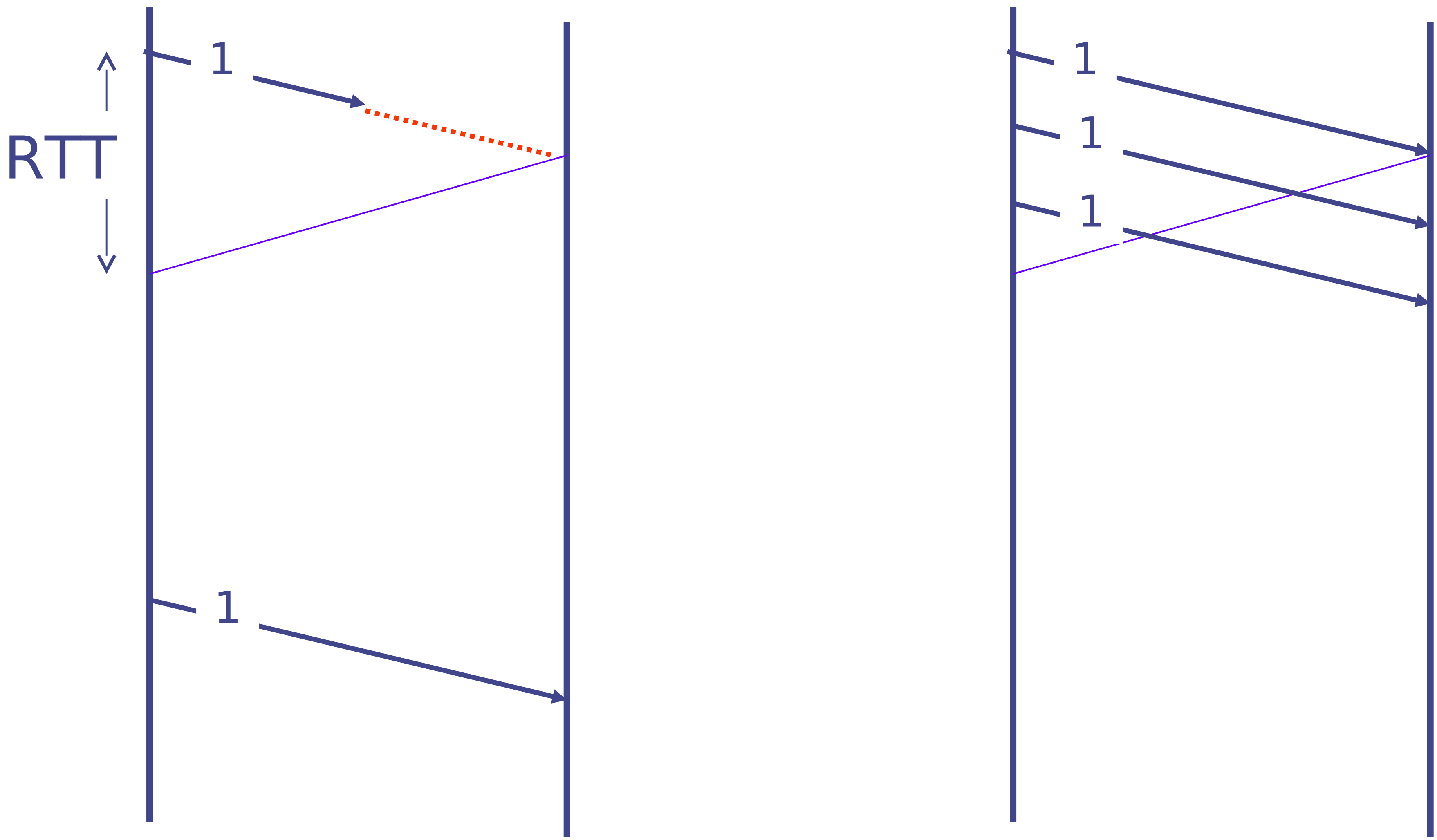
Sliding Window Size

- What size should the window be?
  - Too small:
    - Inefficient, degenerated to S&W when w=1
  - Too large:
    - more buffer required for both sender and receiver
    - Transmitting too fast results in network congestion and packet lost
- Congestion control
  - Slow-start phase
    - Initially set to be 1 or 2
    - Increase the window by 1 for each ACK received (this results in multiplicatively increase)
  - Congestion-avoidance phase
    - The window is increased by only 1 at a time after it is larger than the slow-start threshold (i.e., half of the size that causes congestion)
  - In the case a packet is lost, the window is decreased by half (window / 2).

Timer

- The sender needs to set timers in order to know when to retransmit a packet that may have been lost
- How long to set the timer for?
  - Too short: may retransmit before data or ACK has arrived, creating duplicates
  - Too long: if a packet is lost, will take a long time to recover (inefficient)

Illustrations



Timer too long

Timer too short

Adaptation

- The amount of time the sender should wait is about the round-trip time (RTT) between the sender and receiver
  - For link-layer networks (LANs), this value is essentially known
  - For multi-hop WANS, rarely known
- Must work in both environments, so protocol should adapt to the path behavior
- Measure successive ack delays T(n)
  Set timeout = average + 4 deviations

Questions of ACKs

- What exactly should the receiver ACK?
- Some possibilities:
  - ACK every packet, giving its sequence number
  - use *cumulative ACK*, where an ACK for number n implies ACKS for all $k < n$
  - use *negative ACKs* (NACKs), indicating which packet did not arrive
  - use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order

Parallel/Concurrent FTP?

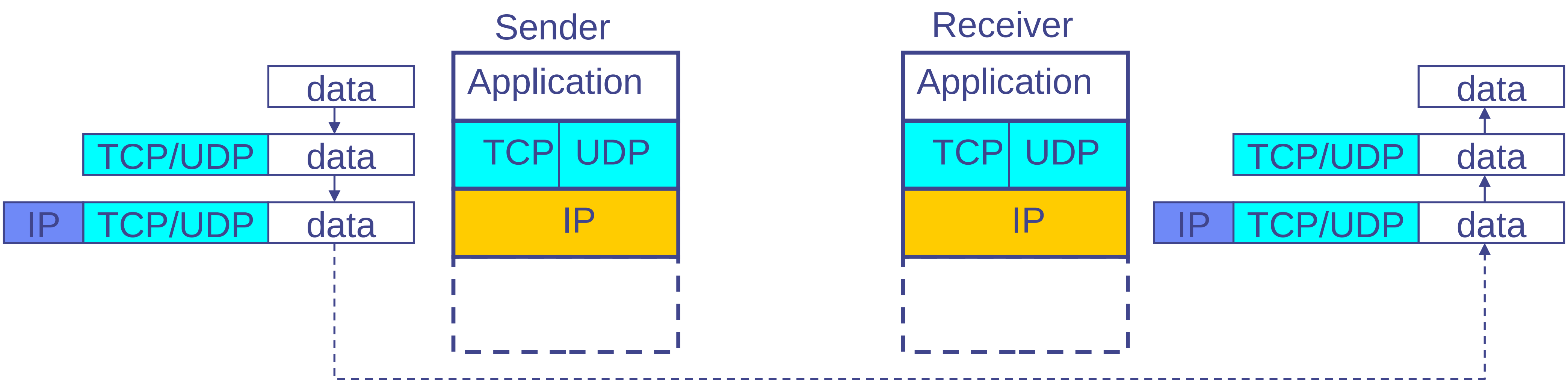
- Multi-Source Downloading
  - A large file may be available on multiple servers
    - The connections to these servers may not be reliable
  - To speed up downloading, a client may download the file from several servers
    - Subject to the limitation of client download bandwidth
  - What factors to consider?
    - Which part of the file should be downloaded from a server
    - What happens if some server is down?
    - How about disk I/O cost?

Summary

- UDP: application-to-application
- TCP: Reliable Byte Stream
  - Connect (3WH); Exchange; Close (4WH)
  - Reliable transmissions: ACKs…
  - S&W not efficient → Go-Back-n
  - What to ACK? (cumulative, …)
  - Timer Value: based on measured RTT

Review of TCP/IP Suite

- IP header → used for IP routing, fragmentation, error detection, etc.
- UDP header → used for multiplexing/demultiplexing, error detection
- TCP header → used for multiplexing/demultiplexing, data streaming, flow and congestion control



Reading Materials

1. https://en.wikipedia.org/wiki/Transmission_Control_Protocol
2. https://simple.wikipedia.org/wiki/User_Datagram_Protocol

Review Questions

1. How can the unreliable IP be leveraged to implement reliable TCP?
2. Explain the three stages of TCP transmission: handshake, data exchange, and termination.
3. In handshake, it is crucial to set the sequential number to be a random number. Why? If the number is set to be a constant, say 0, what could go wrong?
4. There are two ways to implement data exchange: Stop and Wait, and Go-back-n. Explain their pros and cons.
5. Explain the concept of sliding windows. What make(s) it difficult to set an appropriate sliding window?
6. Explain one kind of deny-of-service (DOS) attack on TCP.