11. End-to-end Protocols

End-to-end Characteristics

UDP and TCP

Congestion Control

Implementations and Sockets

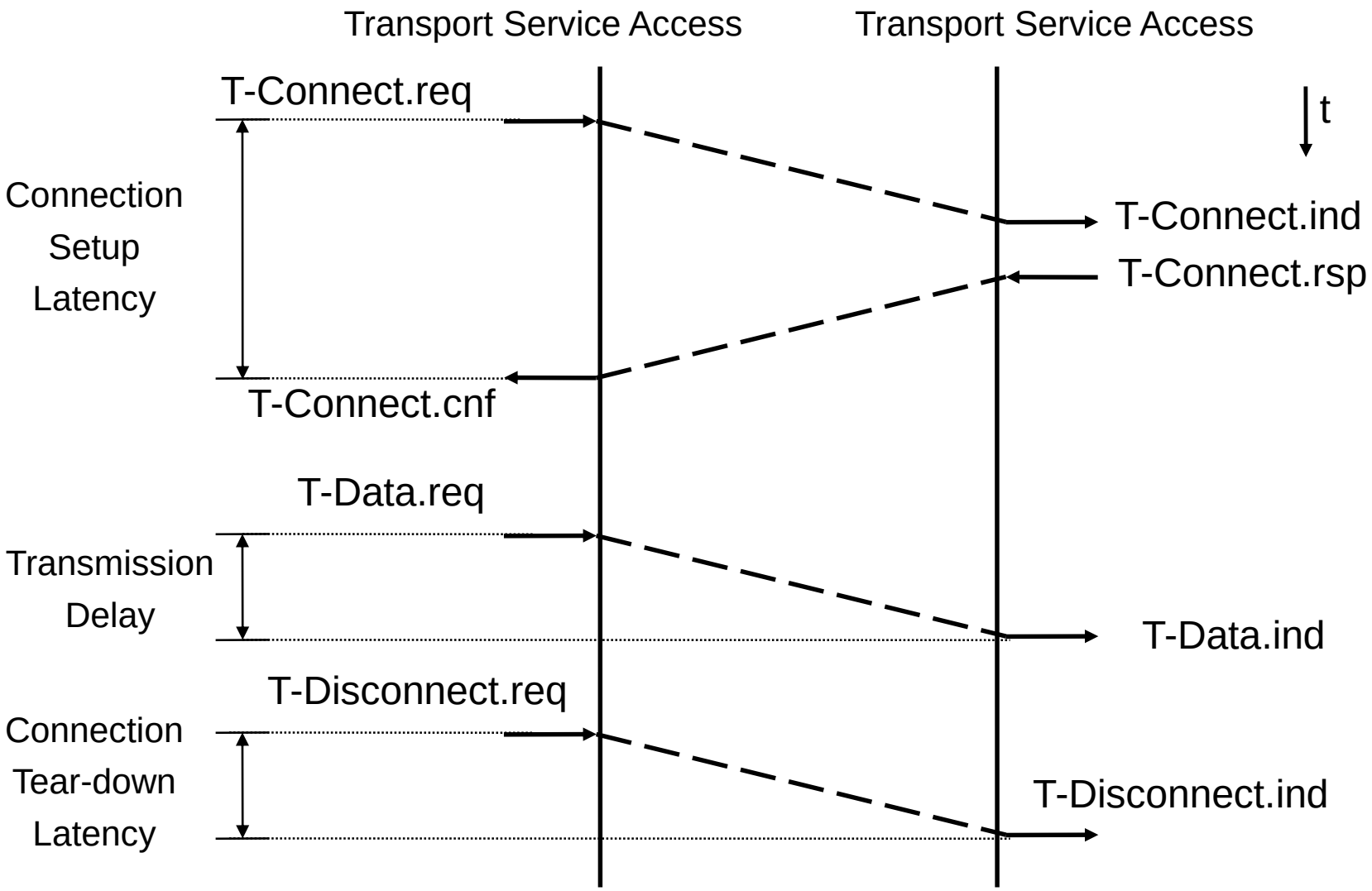
End … Hop-by-hop vs. End-to-end Services

- Underlying best-effort network:
  - Drops messages
  - Re-orders messages
  - Delivers duplicate copies of a given message
  - Limits messages to some finite size
  - Delivers messages after an arbitrarily long delay
- Common end-to-end services:
  - Guaranteed message delivery
  - Deliver messages in the same order they are sent
  - Deliver at most one copy of each message
  - Support arbitrarily large messages
  - Allow the receiver to apply flow control to the sender
  - Support multiple application processes on each host

Transport vs. Link Layer Protocols

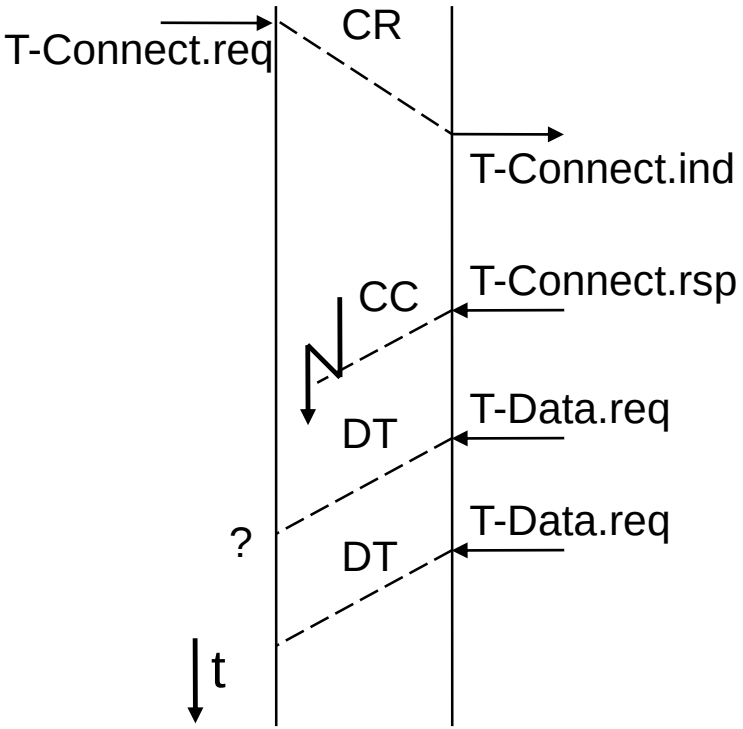
Link Layer Protocols	Transport Protocols
Entities are connected via a single point-to-point link	Entities do not reside in neighbor systems
Mostly fixed and small Round Trip Times (RTT)	Mainly variable and much larger RTTs, leading to a larger bandwidth-delay product
No change of packet sequence	Ordered delivery not guaranteed
Link limits sending rate	Possible bottleneck may exist on any intermediate link

Transport Layer Latencies

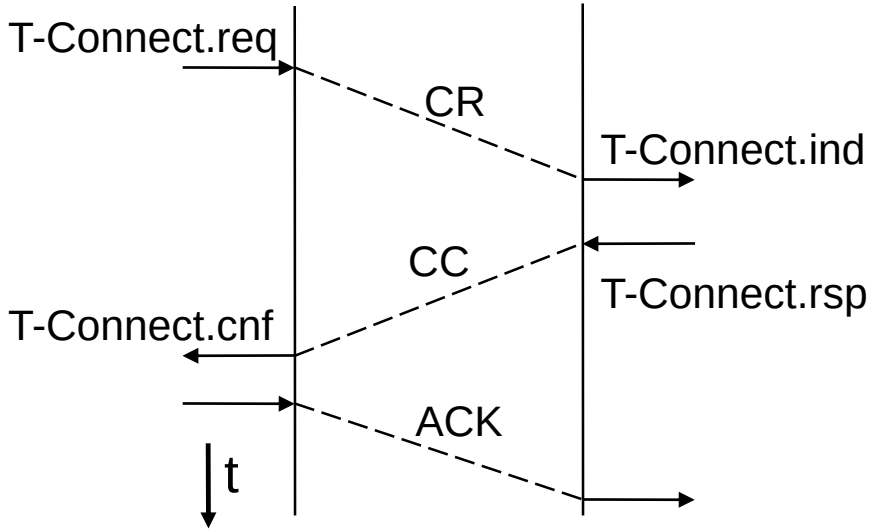


Transport Layer Connection Reliability

- Loss of a connect request:

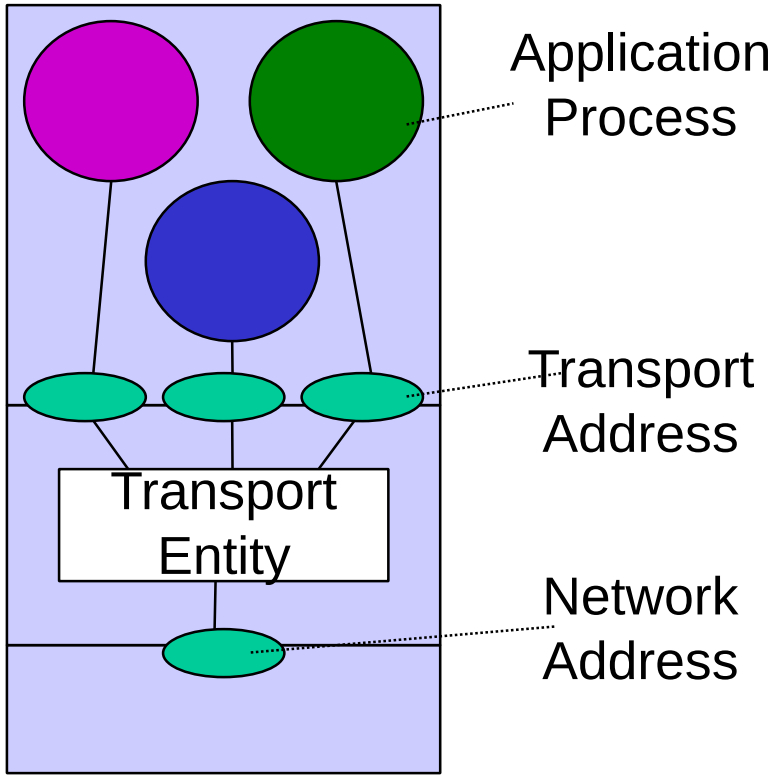


- 3-way-handshake:
  - Connection established as soon as CR and CC have been acknowledged
  - Requires an additional message



Internet Transport Layer Addressing (1)

- Goal: Communication between application processes
- Process ID of operating system can not be used for unique identification of processes
- Full **transport address** equals:
  - IP address ***and*** port number
  - Globally unique
- Port equals an end-point of a communication (OSI CEP)



Internet Transport Layer Addressing (2)

- Identification of transport layer services, in particular the TCP protocol, by ports (cf. SAPs/CEP)
- Reserved port numbers for 1024 frequently used services (well-known ports): http://www.iana.org/assignments/port-numbers
  - 13: NTP (Network Time Protocol)
  - 20: FTP (File Transfer Protocol) Data
  - 21: FTP Control data
  - 25: SMTP (Simple Mail Transfer Protocol)
  - 53: DNS (Domain Name Service)
  - 80: HTTP (Hyper Text Transfer Protocol)
  - 119: NNTP (Network News Transfer Protocol)

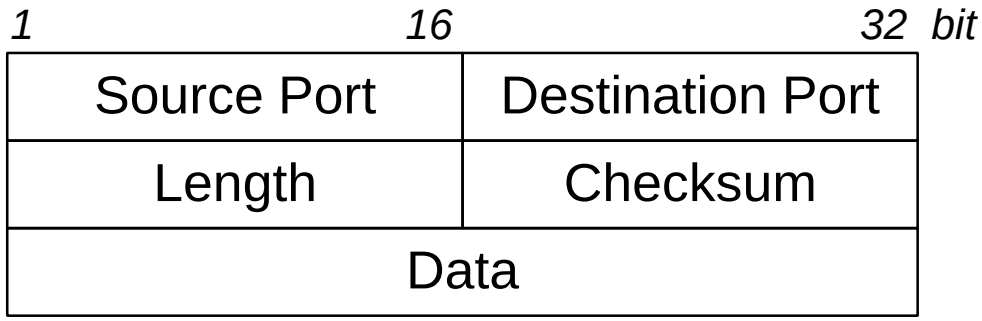
```
> telnet walapai 13
Trying 130.90.1.121...
Connected to host123.
Escape character is '^]'.
Mon Aug  5 16:57:19 2002
Connection closed by foreign
host
```

UDP/TCP ⋯

User Datagram Protocol (UDP)

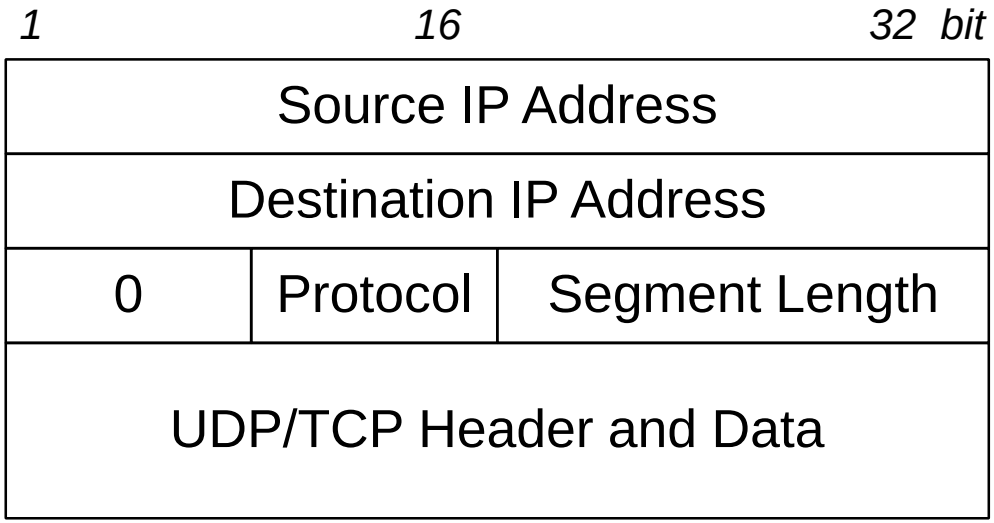
- Connection-less
  - Unreliable and unordered datagram service
- Message-based, including multicasts
  - Adds simple multiplexing
  - No flow control
- Endpoints identified by ports:
  - Servers have *well-known* ports
  - See **`/etc/services`** on Unix
- Optional checksum:
  - Error control
    - Utilizes IPv4 checksum
  - No error correction

Header format



UDP Checksumming

- Definition of a pseudo header (ph), here IPv4:



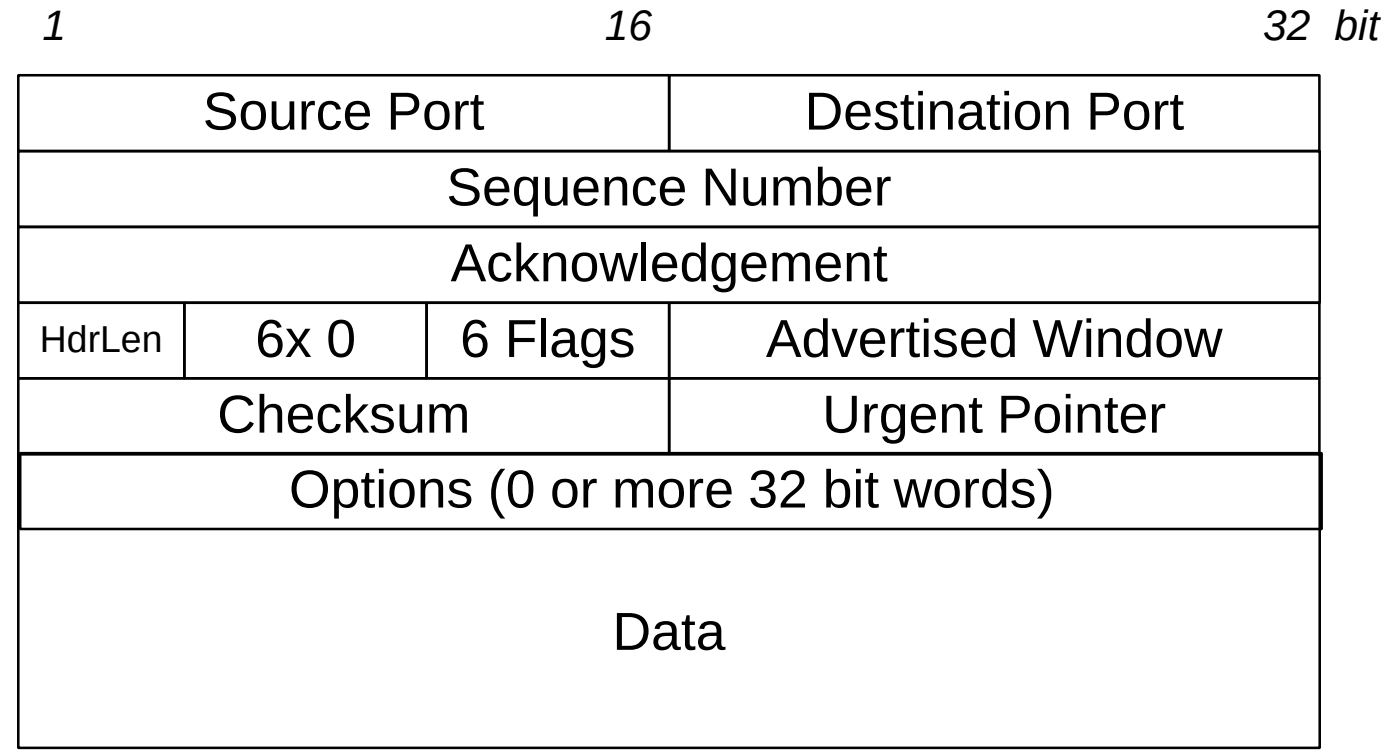
Protocol:
UDP with ID 17

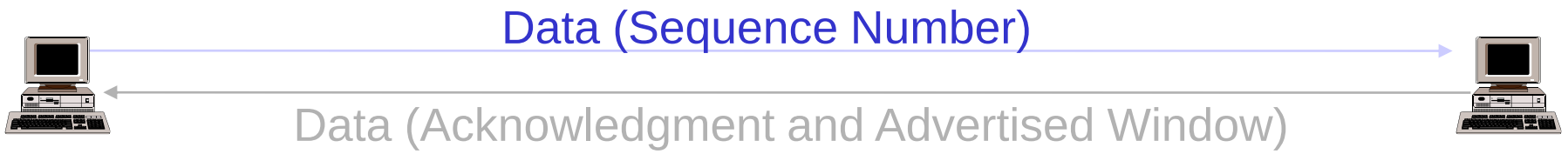
- Calculation of checksum only across:
  - Pseudo header and real UDP header as well as data
- Pseudo header is NOT transmitted, just for calc. only!
- Reliably securing UDP protocol data and detecting wrong destination delivery (due to Destination IP in ph)

Transmission Control Protocol (TCP)

- Connection-oriented and reliable
  - Between two sockets (connection) in full duplex mode
  - Reliable: three way handshake, ordered delivery
  - Error control: Sequence numbering, checksum, acknowledgements and numbers, retransmissions
  - Flow control: keep sender from overrunning receiver
  - Congestion control: keep sender from overrunning network
- Byte-stream driven data transfer
  - Sending process writes some number of bytes
  - TCP breaks into *segments* (TCP packet) and sends via IP
    - Segment size determined according to MTU (Maximum Transmission Unit) of local link
  - Receiving process reads some number of bytes

TCP Header (Segment Format) (1)





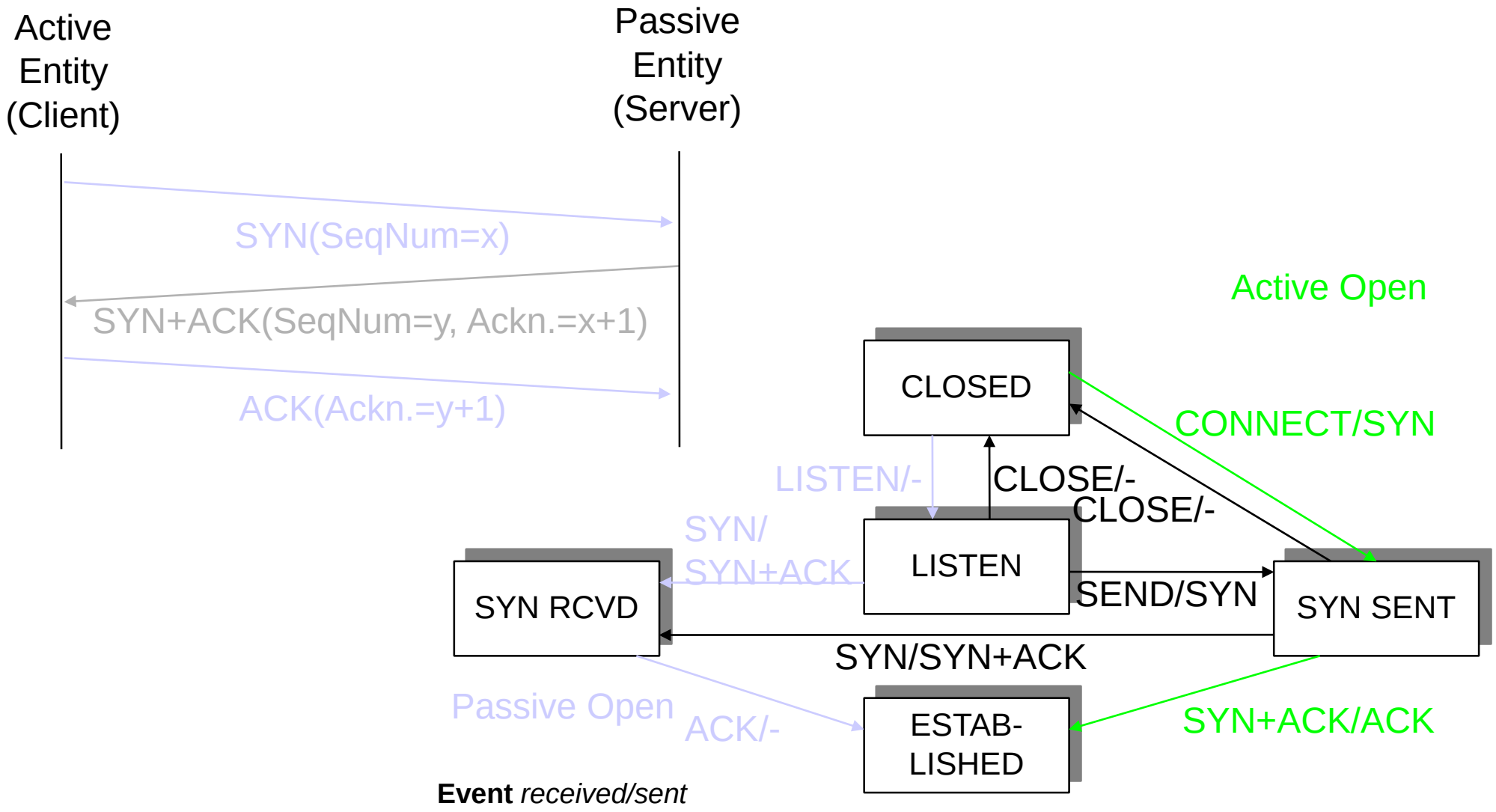
TCP Header (Segment Format) (2)

- Source and destination port:
  - Part of the transport layer address
- Sequence number and acknowledgement:
  - Sending and receiving sequence number
- TCP header length (HdrLen):
  - Number of 32 bit words in header
- Advertised window:
  - Sending window of flow control
- Urgent pointer:
  - Pointer to important data
- Flags:
  - URG:
    - Set to 1 in case the Urgent Pointer is used
  - SYN:
    - Used at connection set-up
  - ACK:
    - Signals the validity of the current acknowledgement field in header
  - FIN:
    - Information that sender will not sent further data
  - RST:
    - Restart of a connection
  - EOM or PSH:
    - Information that the end of the message has been reached (End of Message or Push Flag)

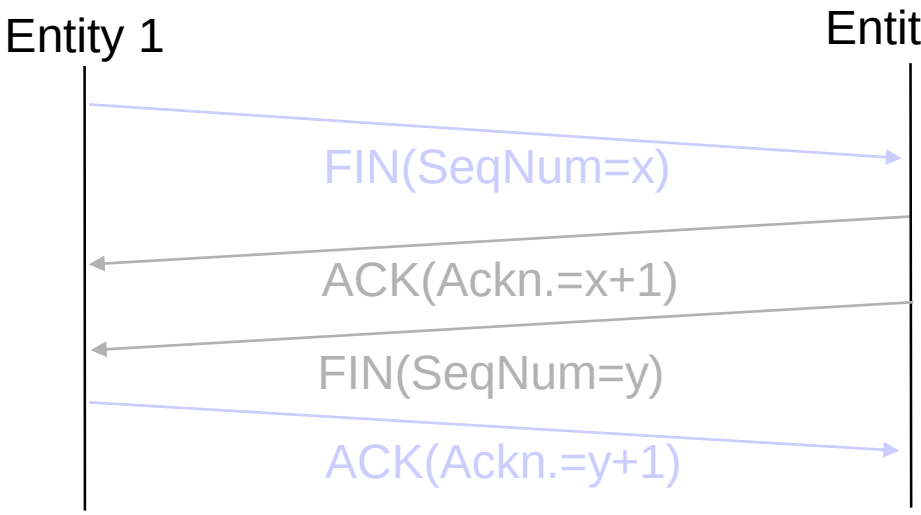
TCP Connection Setup (1)

- Opening of a socket:
  - Active mode:
    - Request of TCP connection from a specific socket (connect)
  - Passive mode:
    - User informs TCP being able to accept an incoming connection (listen/accept)
      - Option (accept):
        - Well-known socket to accept all incoming connections
- Functional view:
  - 3-way-handshake
  - Exchange of initial sequence numbers:
    - Avoids conflicts with ”old“ connections, having used the same port, and remaining ”old“ packets within the network

TCP Connection Setup (2)

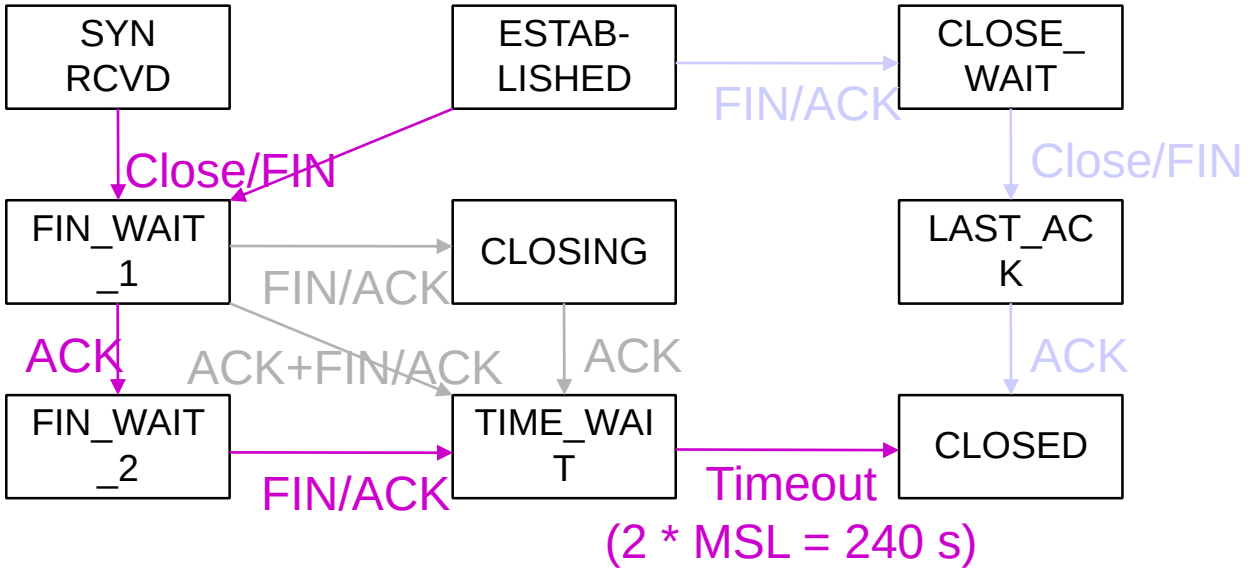


TCP Connection Teardown



- Situations:
  - Receiver closes connection first
  - Sender closes connection first
  - Sender and receiver close connection at the same time

- Both entities close the connection
- All data being sent will reach the receiver



MSL: Maximum Segment Lifetime

TCP Error Detection and Correction

- Piggybacking:
  - 16 bit window size: transmission volume of 2^{16} = 65536 Byte
  - Variable “Receiver Window“: High speed requires 2^{32} Byte
- Checksumming:
  - Reliably securing TCP header, payload (user data), and TCP pseudo header:
    - IP source and destination address, IP protocol field (6), and TCP segment size
- Acknowledgement:
  - As soon as possible upon receipt of segment or delayed acknowledgement
  - Cumulative or selective (optional)

TCP Retransmission (1)

- Retransmission schemes: Lacking acknowledgement
  - Standard: Go-Back-N:
    - Acknowledgement number *n* acks the receipt of all bytes up to sequence number *n-1*
    - Retransmission driven by time out of retransmission timer
    - Negotiation of scheme during connection setup:
      - Selective Repeat (RFC 1106)
        - NAK requests erroneous segment explicitly
  - Optional: Selective ACKs:
      - Selective Acknowledgment (1996, RFC 2018)
        - SACK acknowledges single, well-received segments
  - Use of a retransmission timer:
    - Started upon sending of a segment, retransmission started, if timer timed out and no ACK has been received

TCP Retransmission (2)

- Jacobson/Karels algorithm:
  - New calculation for average RTT:

    Diff = Sample_RTT - Estimated_RTT

    Estimated_RTT = Estimated_RTT +
    (δ x Deviation = Deviation + δ(|Diff|- Deviation)

    where δ is a fraction between 0 and 1

  - Consider variance when setting timeout value:
    - Time_Out = μ x Estimated_RTT + ϕ x Deviation
      where μ = 1 and ϕ = 4

  - Remarks:
    - Algorithm only as good as granularity of clock (500 ms on Unix)
    - Accurate timeout mechanism important for congestion control

TCP Flow Control

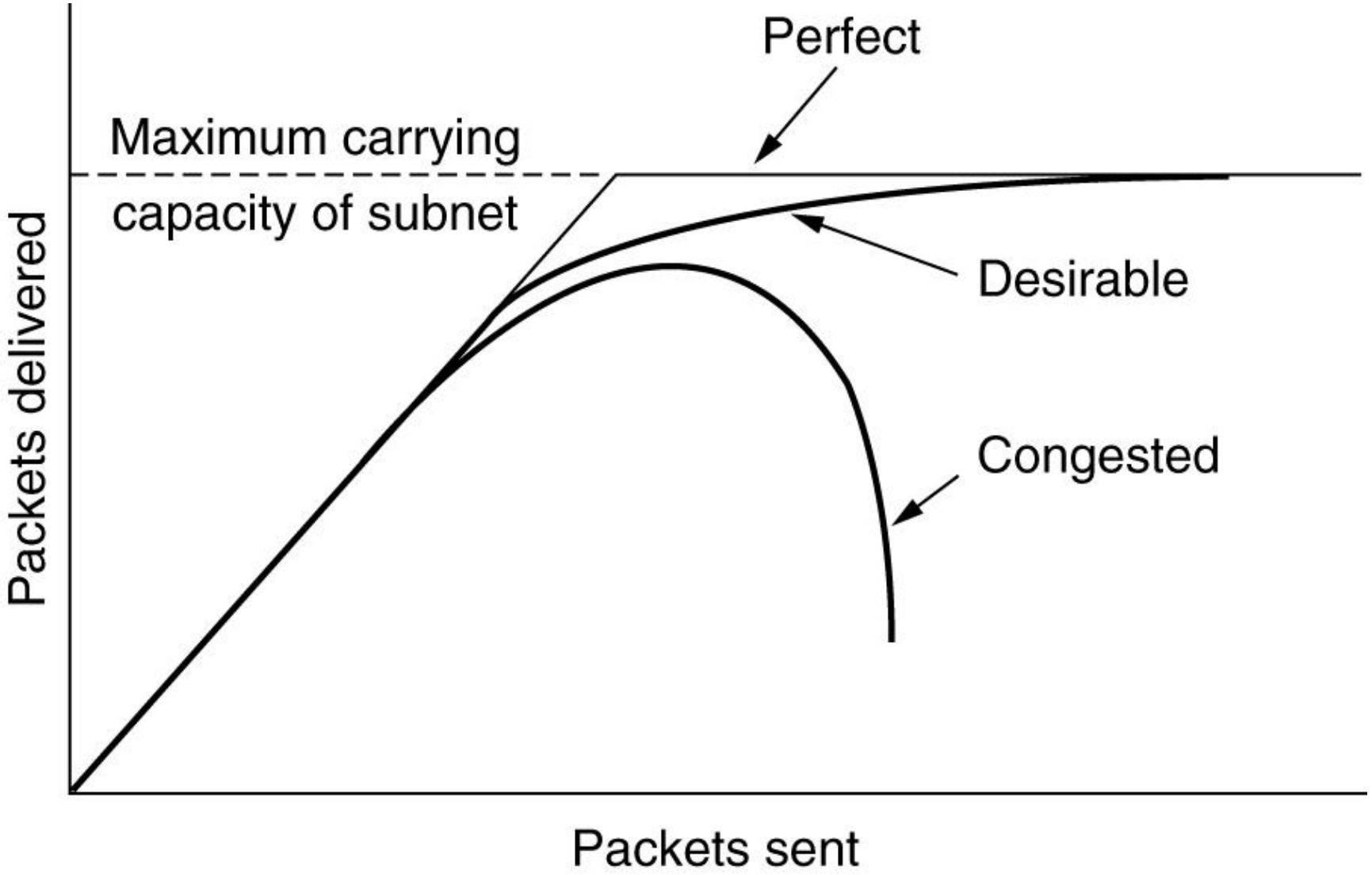
- Cf. slide M08-50 and following
- Flow control maintains the flow of user data
- TCP utilizes the sliding window mechanism:
  - ACK flag acknowledges receipt of all bytes numbered with a smaller number than the sequence number received
  - Advertised Window determines the number of additional bytes, the receiver is able to accept
  - Receiver allows sender to sent data up to the amount of:
    - Acknowledgment + Advertised Window

Congestion …

Internet Traffic Problem

- Internet USA 1987:
  - Congestion Collapse!
  - Network was fully loaded, throughput = 0
  - Only retransmissions observed
- Van Jacobson, LBL Labs, 1988:
  - TCP Congestion Control mechanism
    - Initial version: TCP Tahoe
- Traffic characteristics in the Internet?
  - Extremely difficult to describe mathematically
    - Many Internet flows are TCP-based
- Question: What is the gain, if the performance of a congestion control mechanism is enhanced by 1%?

Congestion Effects



Congestion Control Principles and Prevention

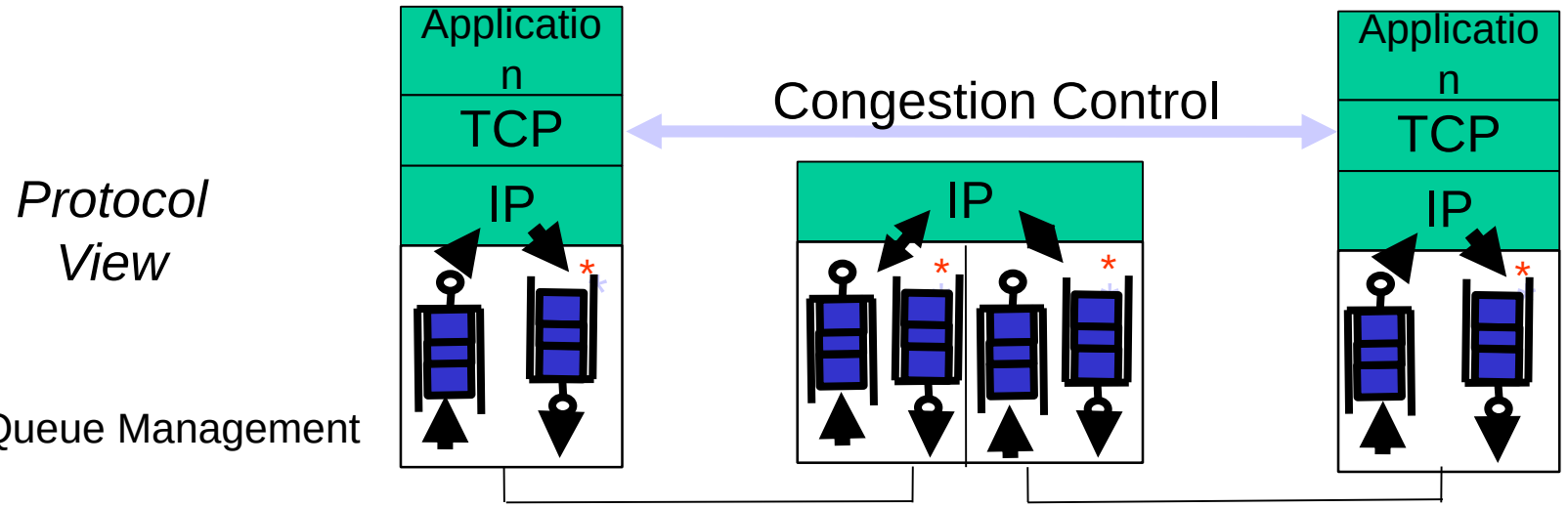
- Monitor the system
  - Detect when and where congestion occurs
- Pass information to where action can be taken
- Adjust system operation to correct the problem.

- Prevention approches

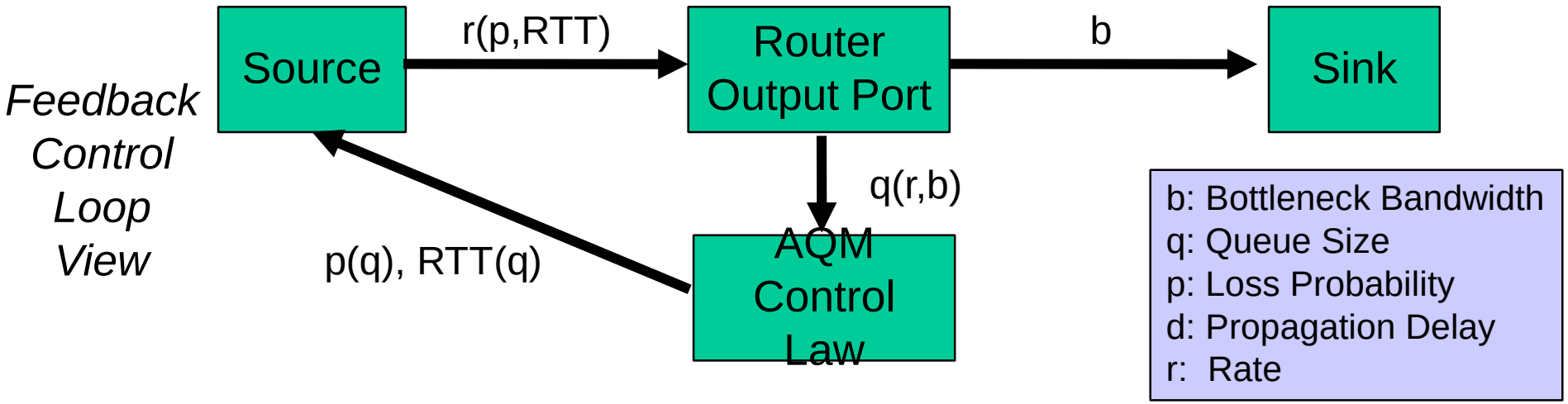
Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

TCP Congestion Control (1)

- Congestion control:
  - Addresses the queuing situation in routers
  - Congestion leads to retransmissions in the transport layer
  - This leads to an increase of the congestion
  - Congestion control needs to adapt sending rate(s)
  - Queue management within routers need to drop packets



TCP Congestion Control (2)

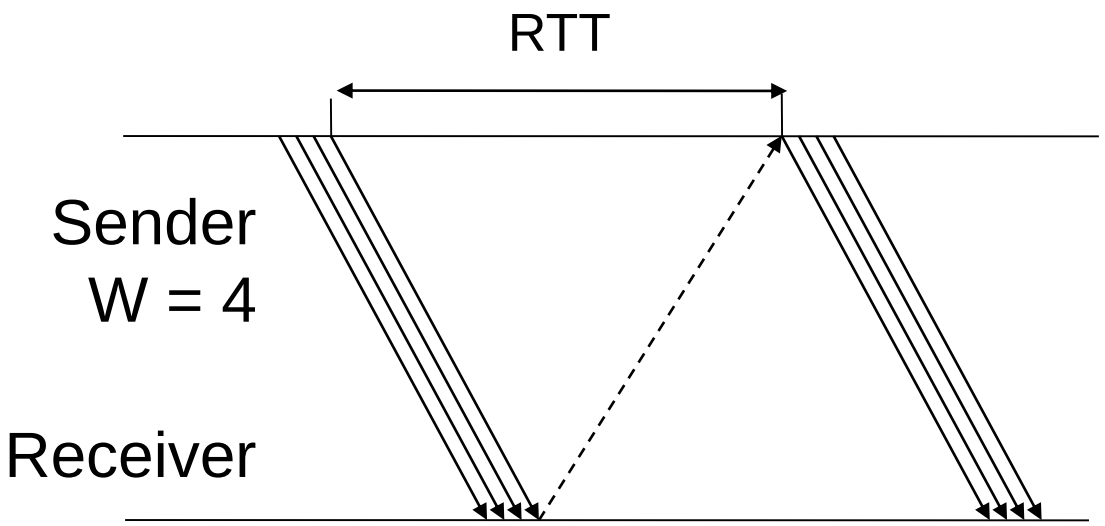


- Approaches for TCP:
  - Implicit mechanisms:
    - Increase of sending credit after receipt of an ACK
    - Decrease of sending credit upon lack of an ACK
  - Explicit mechanisms:
    - Explicit Congestion Notification (RFC 2481)

AQM: Alternate Queue Management

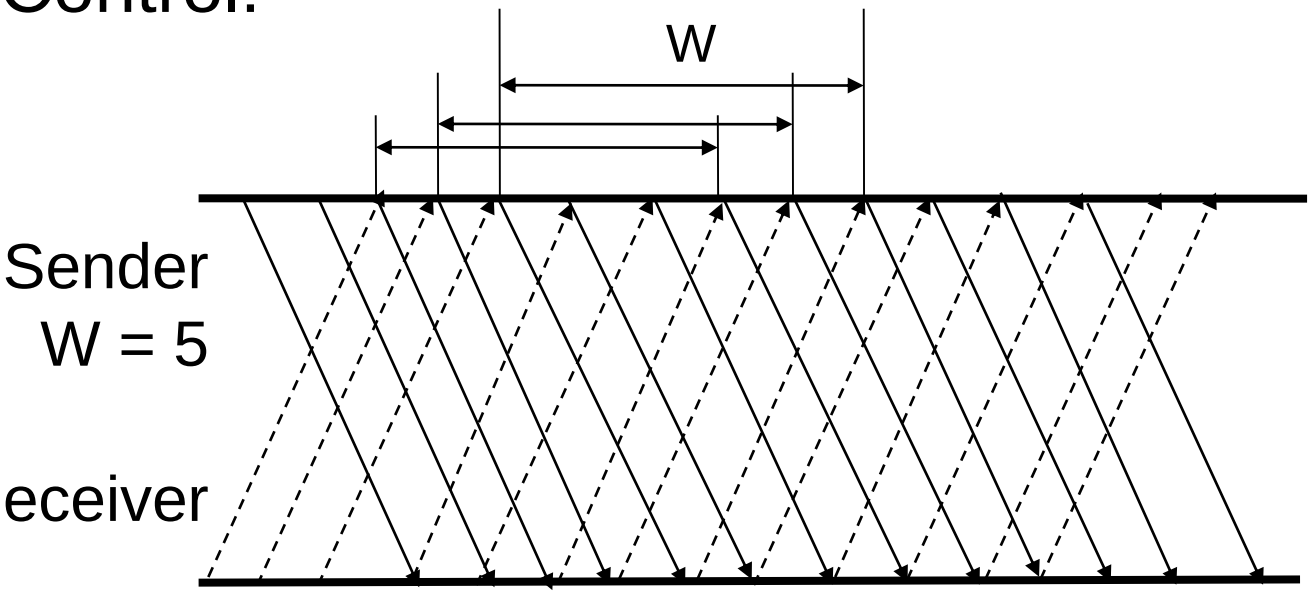
Window-based Congestion Control (1)

- Window W defines the number of packets allowed to be sent upon receipt of an ACK
- Bursty Window Control:
  - ACK step = W
- Advantage:
  - Small overhead due to ACKs
- Drawbacks:
  - Bursty nature
  - Loss of ACKs



Window-based Congestion Control (2)

- Sliding Window Control:
  - ACK step = 1

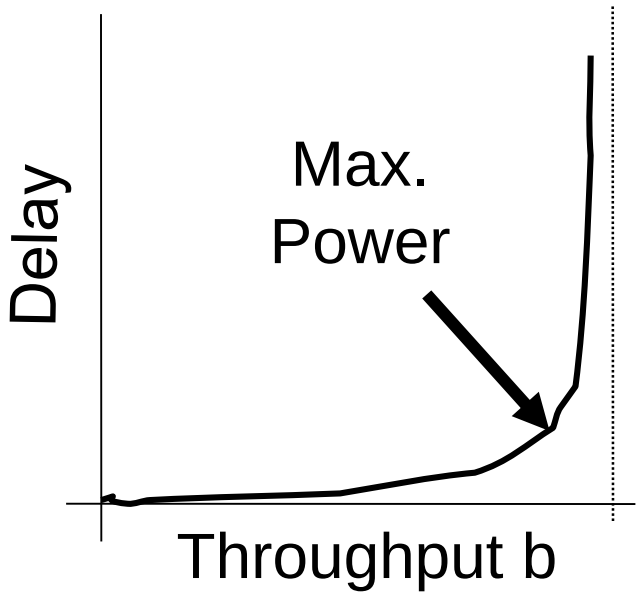


- Advantages:
  - Not bursty
  - Loss of ACKs bearable
- Drawbacks:
  - Large overhead due to many ACKs
  - Interdependency between error control and window size

Window-based Congestion Control (3)

- Control of transmission volume, bytes or packets
- Effective rate = W(t) / RTT(t);
  - W(t) = f(p(t))
  - RTT(t) = q(t) + 2*d
- Optimal Window: W = b * 2d
  - Optimal link utilization:
    - Effective rate = b*(2d/2d) = b
  - No queue at bottleneck:
    - Optimal network performance (power)

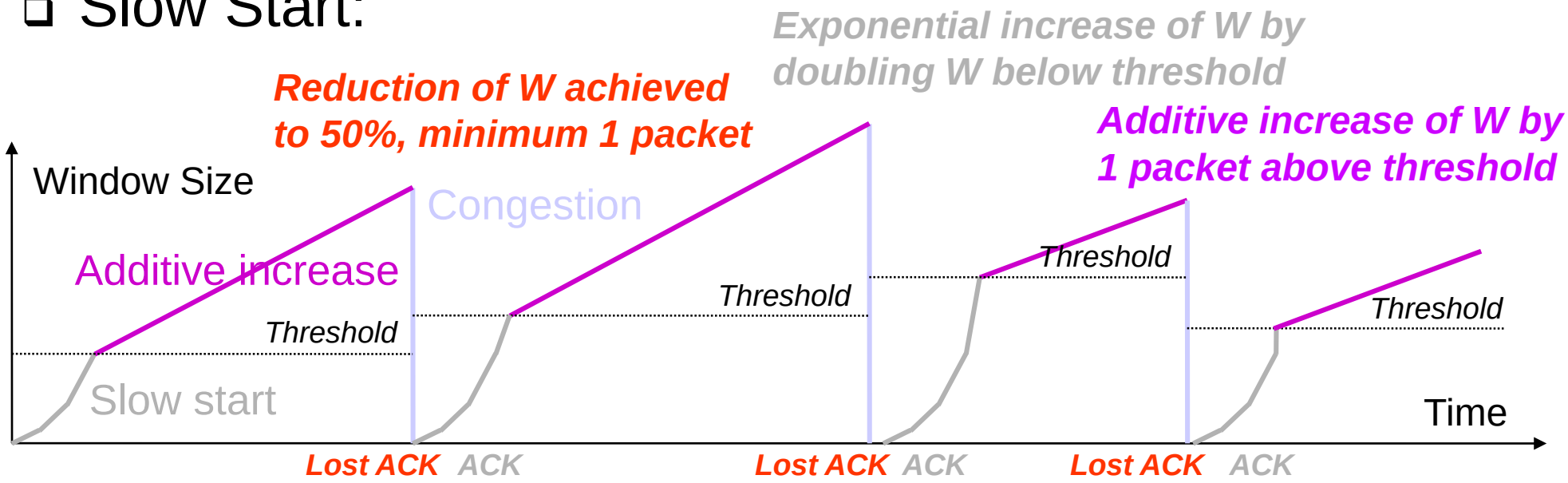
b: Bottleneck Bandwidth
q: Queue Size
p: Loss Probability
d: Propagation Delay



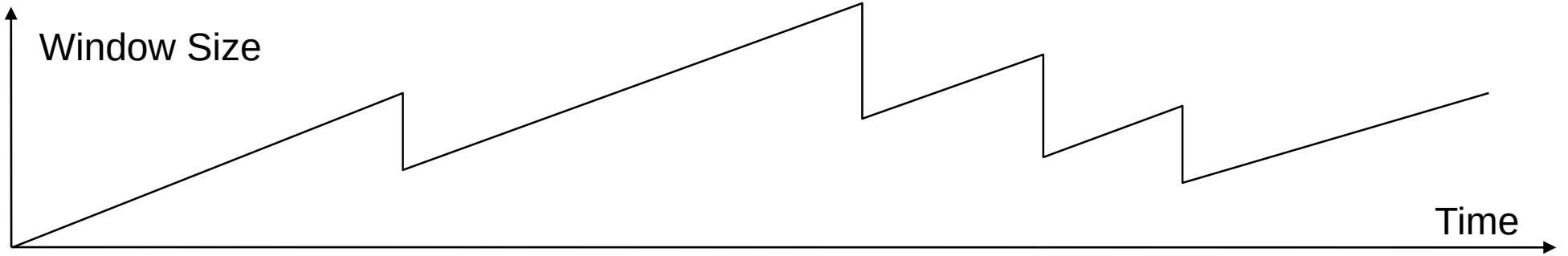
Power = Throughput / Delay:

Adaptation of TCP Window Size

- Slow Start:



- Additive Increase Multiplicative Decrease (AIMD):



Practical TCP Considerations

- Wrap around of TCP sequence numbers:
  - High-speed networks may show an overflow within MSL
- Bandwidth delay product:
  - Advertised Window (16 Bit) defines max window of 64 kByte

Bandwidth	Time until Wrap Around	Bandwidth Delay Product
T1 (1.5 Mbit/s)	6.4 h	18 kByte
Ethernet (10 Mbit/s)	57 min	122 kByte
FDDI (100 Mbit/s)	6 min	549 kByte
STM-1 (155 Mbit/s)	4 min	1200 kByte
STM-3 (622 Mbit/s)	55 s	1800 kByte
STM-16 (2.49 Gbit/s)	14 s	29800 kByte

RTT = 100 ms

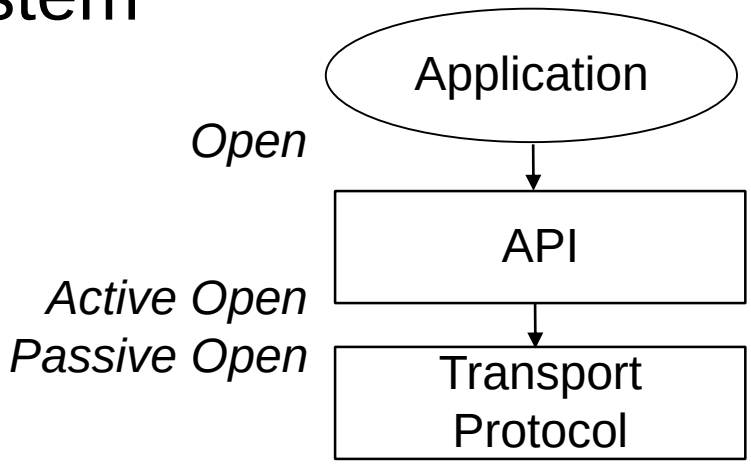
← MSL = 2 min

MSL: Maximum Segment Lifetime

Protocol Implementations and Interfaces

Implementation …

- To separate protocol implementations from protocol interfaces being exported:
  - Importance at transport layer due to application access to the network
  - Interface called Application Programming Interface (API)
- API defined by the operating system
- Specific API: sockets
  - Defined by BSD Unix
  - Ported to many other systems



Socket Calls

socket	Create a new communications end point
bind	Bind server address to socket
listen	Listen to incoming connections
accept	Blocking at passive open, creates a new socket for a server
connect	Active connection setup
send	Send
receive	Receive
close	Connection teardown

Code examples in C may be found at http://cs.baylor.edu/~donahoo/practical/CSockets/textcode.html

```
socket (domain, type, protocol)
bind   (socket, address, addr_len)
listen (socket, backlog)
accept (socket, address, addr_len)
connect(socket, address, addr_len)
send   (socket, message, msg_len, flags)
recv   (socket, buffer, buf_len, flags)
close  (socket)
```

Sample Socket Calls

- Creating a socket:

```
int socket(int domain, int type, int protocol)
domain=PF_INET, PF_UNIX
type=SOCK_STREAM, SOCK_DGRAM
```

- Passive open on server:

```
int bind(int socket, struct sockaddr *address,
         int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address,
           int *addr_len)
```

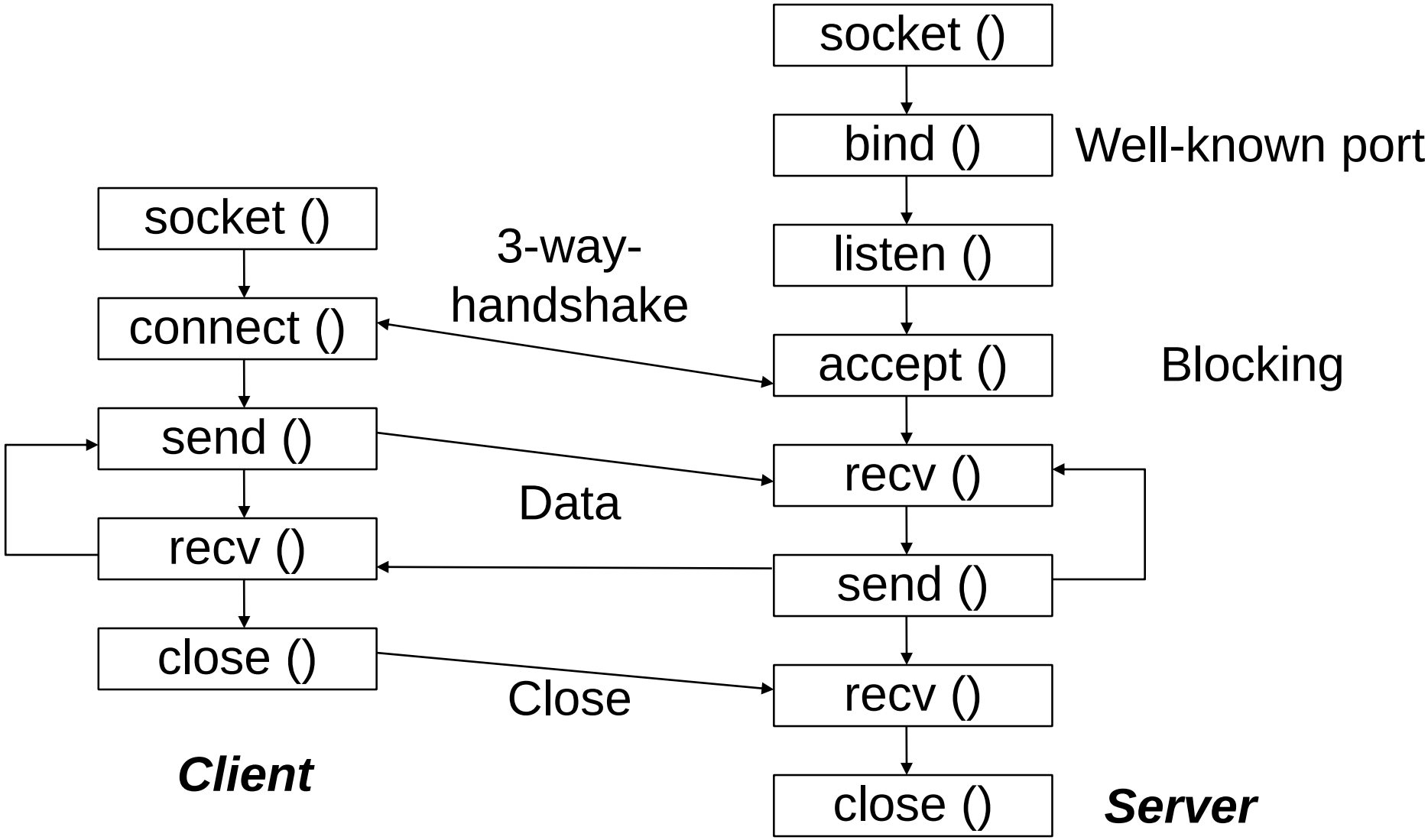
- Active open on client:

```
int connect(int socket, struct sockaddr *address,
            int addr_len)
```

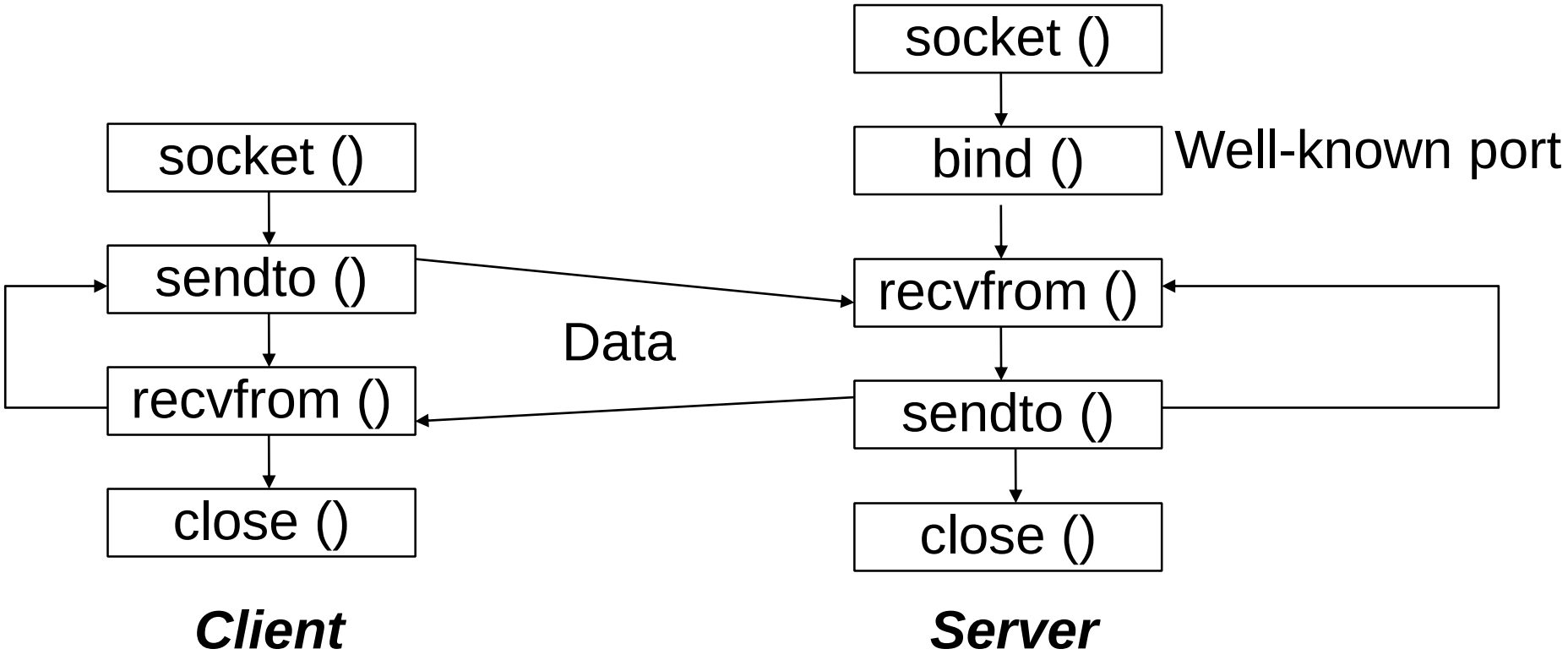
- Sending and receiving messages:

```
int write(int socket, char *message, int msg_len, int flags)
int read(int socket, char *buffer, int buf_len, int flags)
```

TCP Client and Server



UDP Client and Server




```
recvfrom (socket, buffer, buf_len, flags, from, from_len)
sendto (socket, message, msg_len, flags, to, to_len)
```