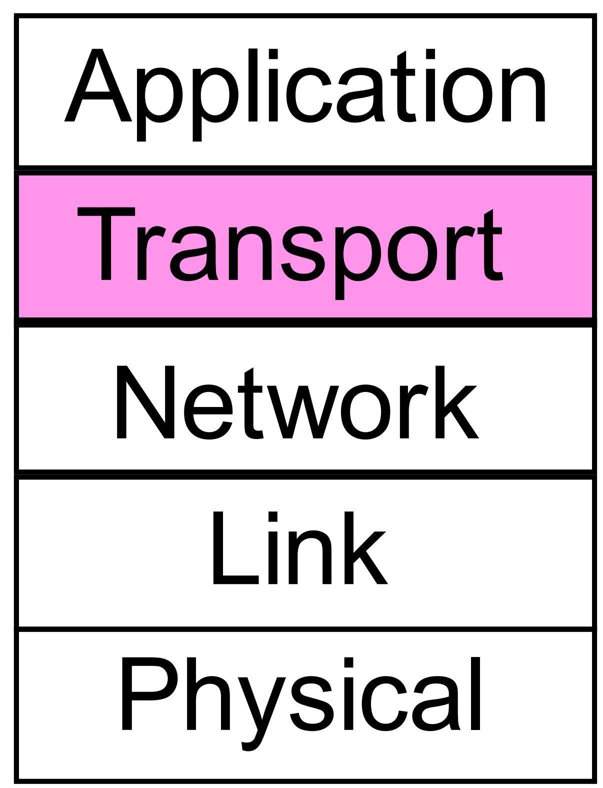
Transport Layer

Thomas Schwarz, SJ

Transport service

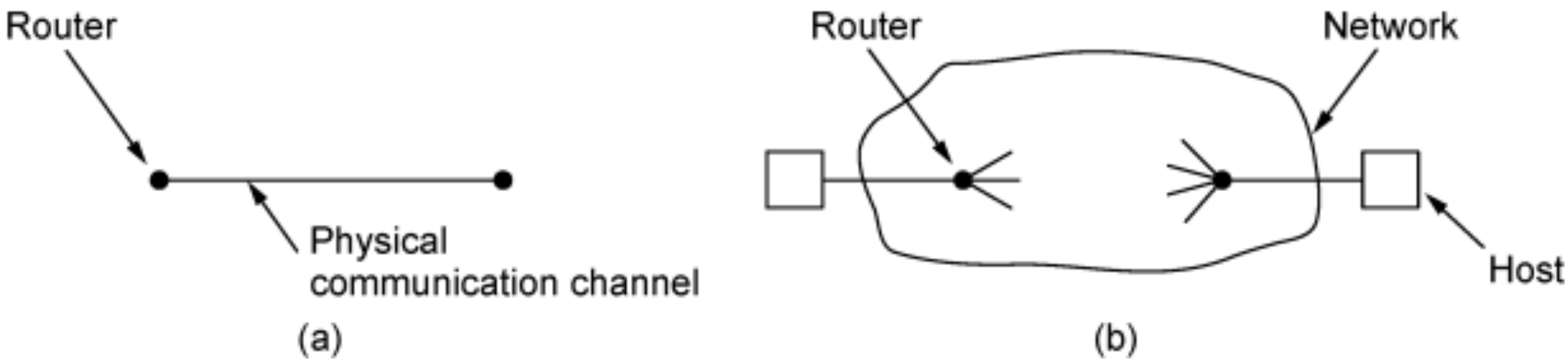
- Responsible for delivering data across networks with the desired reliability or quality



Transport Layer

- Difference to the Network Layer:
  - Transport layer runs at the endpoints only
  - Network layer runs (mainly) at the routers
- Transport layer can make transport service more reliable than the underlying network
- Transport layer primitives are implemented as library procedures
  - Which are independent of network primities

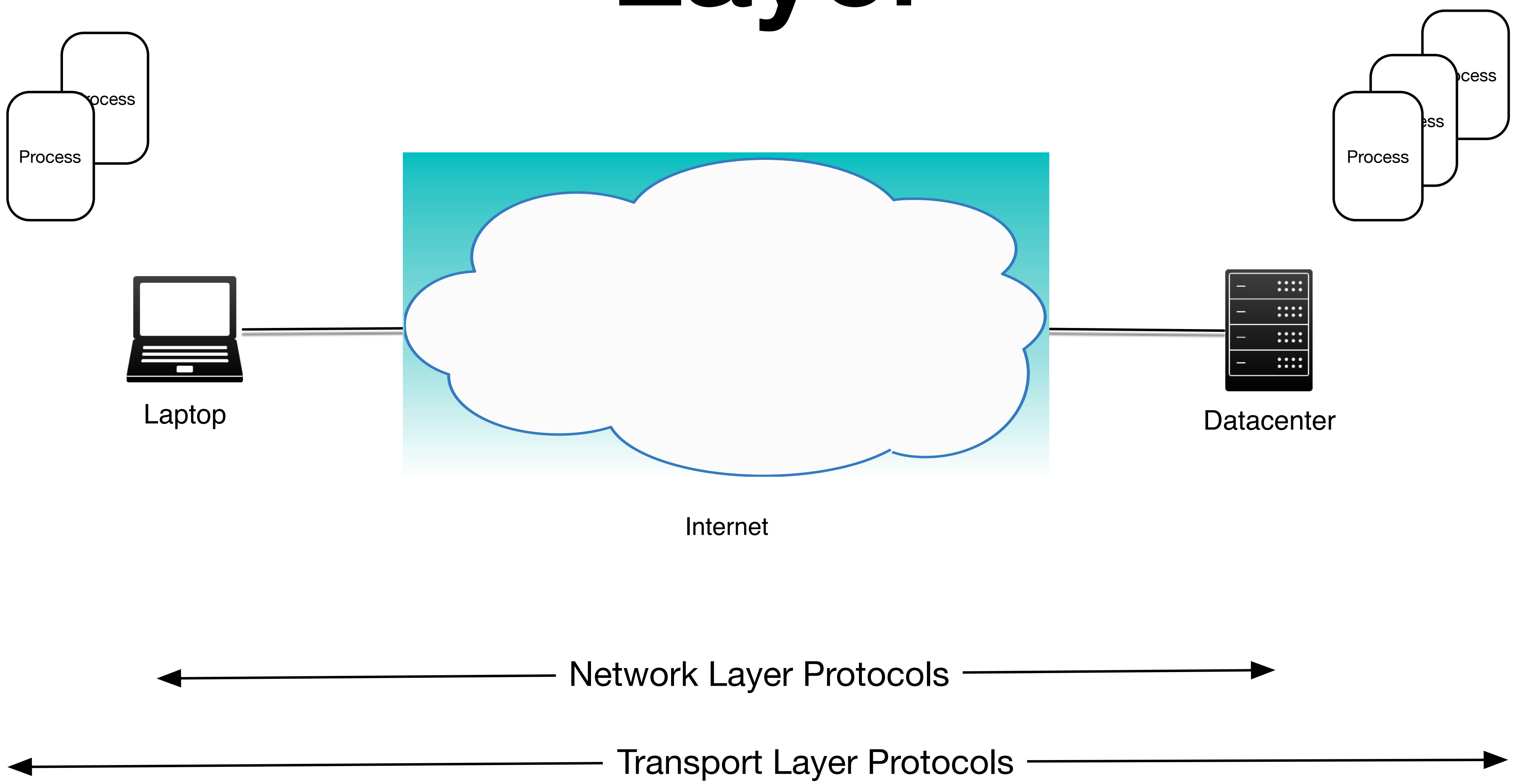
Transport Layer



(a)Environment of the data link layer. (b)Environment of the transport layer.

- Transport layer is between hosts
    - Creates a more reliable means of communication using the network

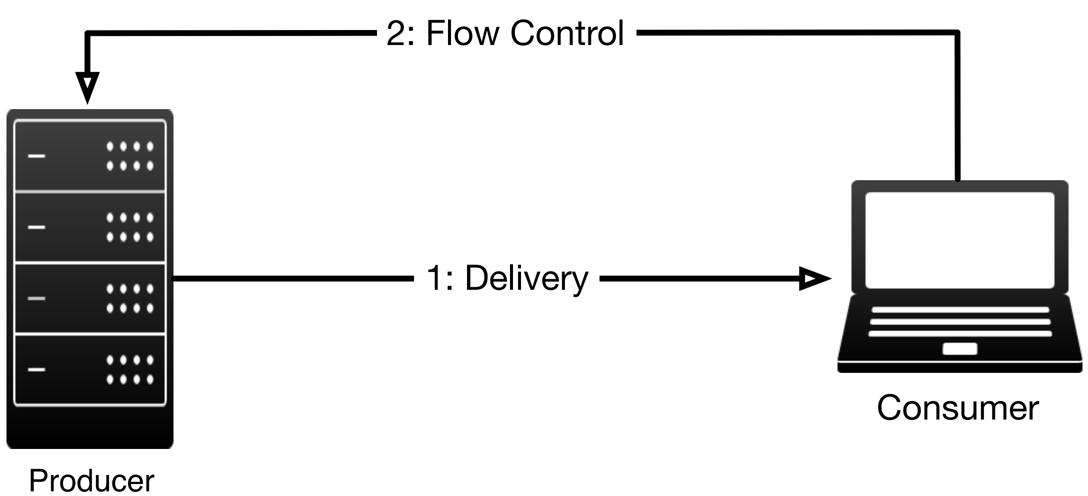
Transport Layer vs Network Layer



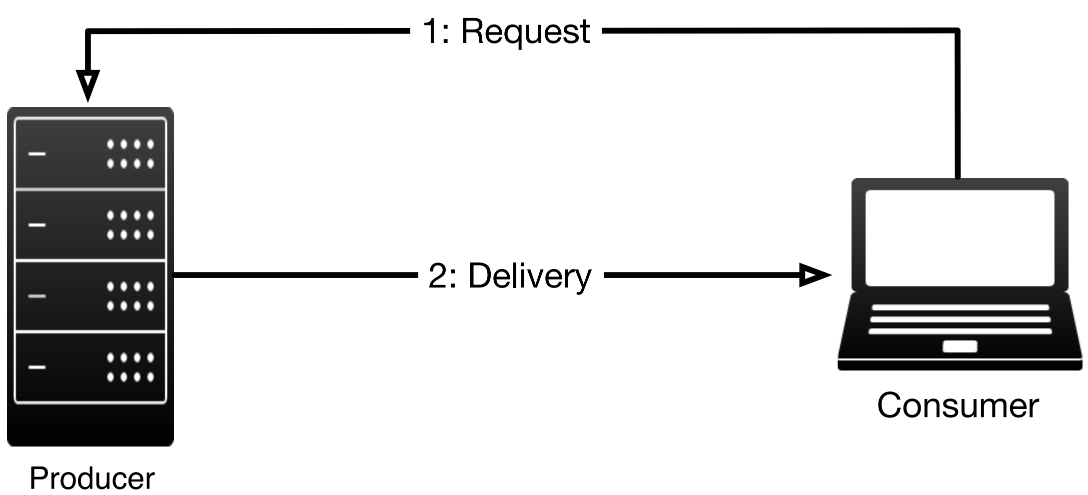
Transport layer protocols provide communic
from process to process.

Transport Layer Duties

- Flow control
  - Can use push or pull paradigm



Pushing



Pulling

Transport Layer Duties

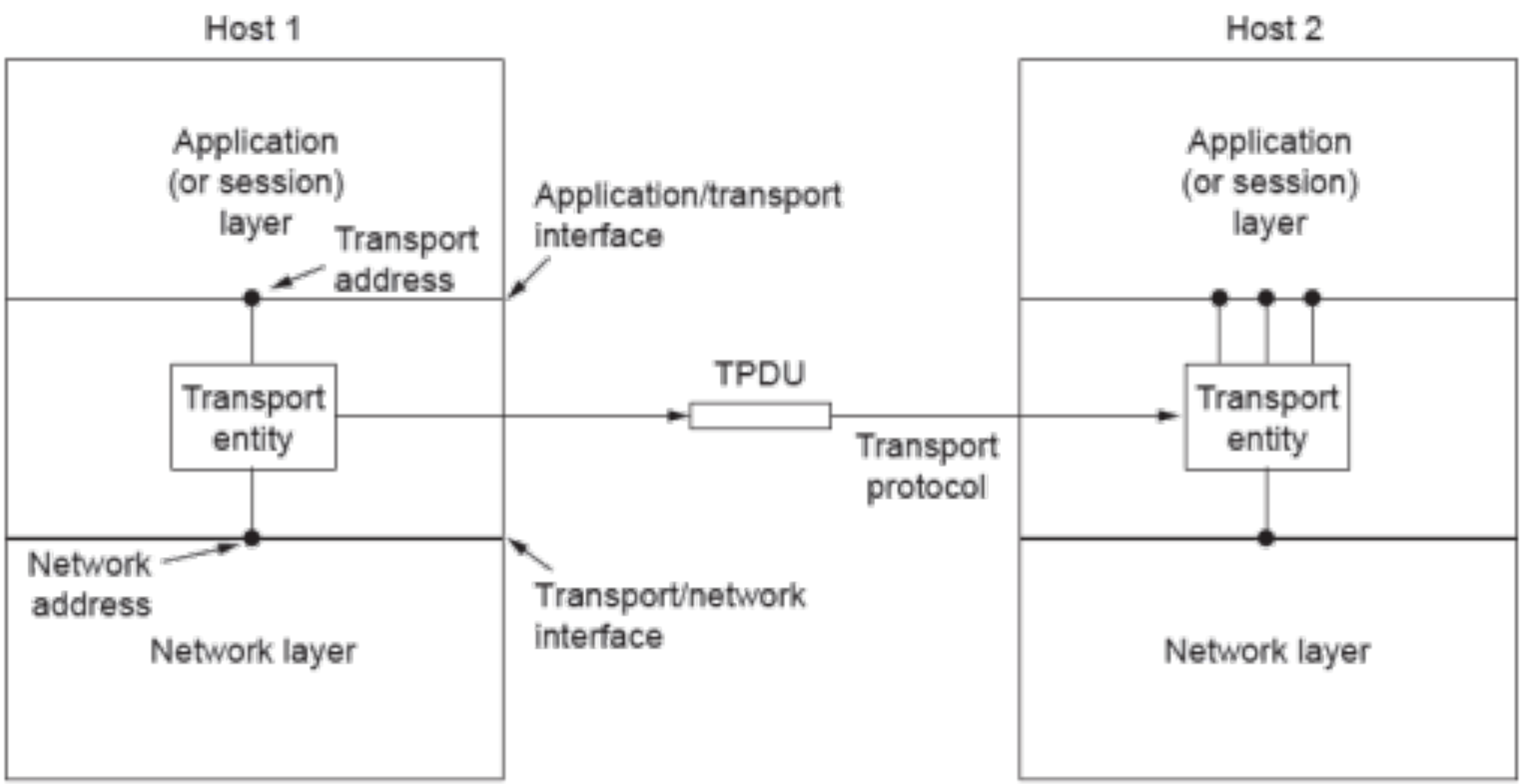
- Error Control
  - Detecting and discarding corrupted packets
  - Keeping track of lost and discarded packets and resending them
  - Recognizing duplicate packets and discarding them
  - Buffering out-of-order packets until the missing packets arrive

Transport Layer Duties

- Error Control
  - Use error detecting / correcting codes
  - Use sequence numbers to order packets
  - Use acknowledgments as a positive signal for error control

Transport Layer Duties

- Transport layer offers connection-oriented (TCP) and connectionless (UDP) services



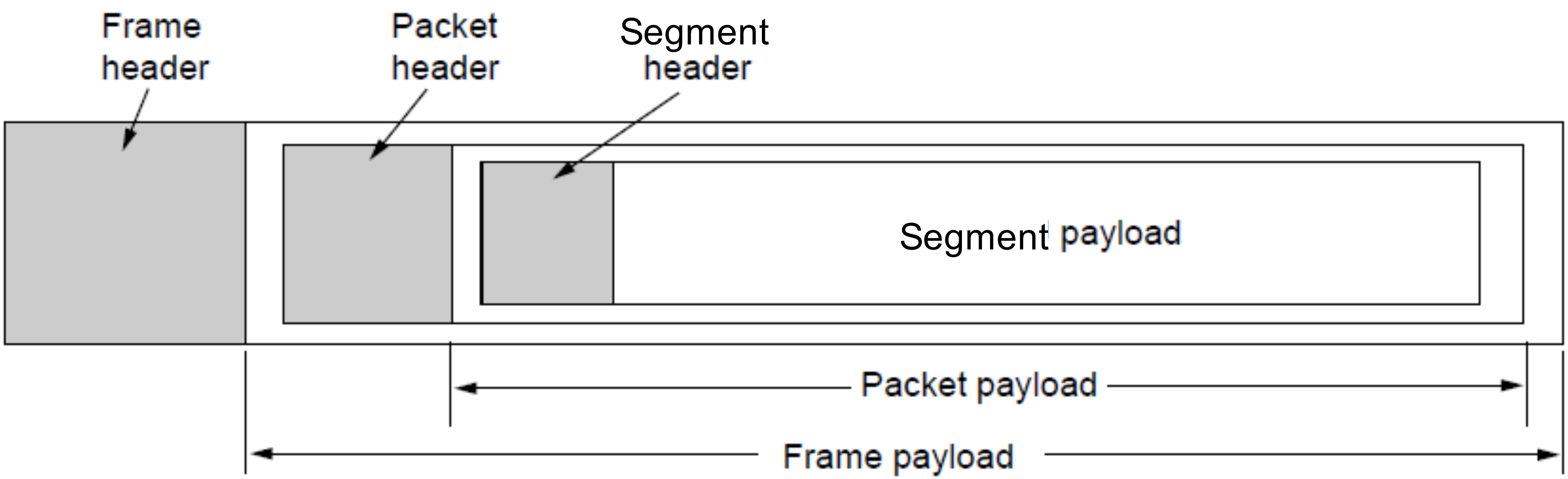
Transport Service Primitives

- Typical primitives provided to application programs

Primitive	Packet sent	Meaning
Listen	-	Block until some process tries to connect
Connect	Connection Request	Actively attempt to establish a connection
Send	Data	Send information
Receive	-	Block until a data packet arrives
Disconnect	Disconnection Request	Request a release of the connection

Transport services provided to application layer

- Transport layer embeds segments in packets that are embedded in frames



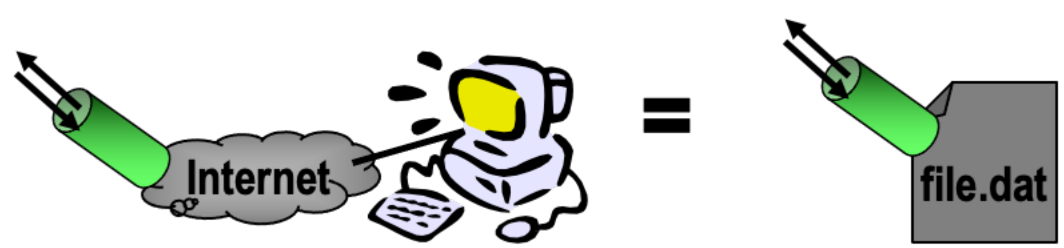
Berkeley Sockets

- Developed for Unix 4.2BSD (1983)
  - Still used for internet programming especially on Unix systems
  - Windows has winsock

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
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

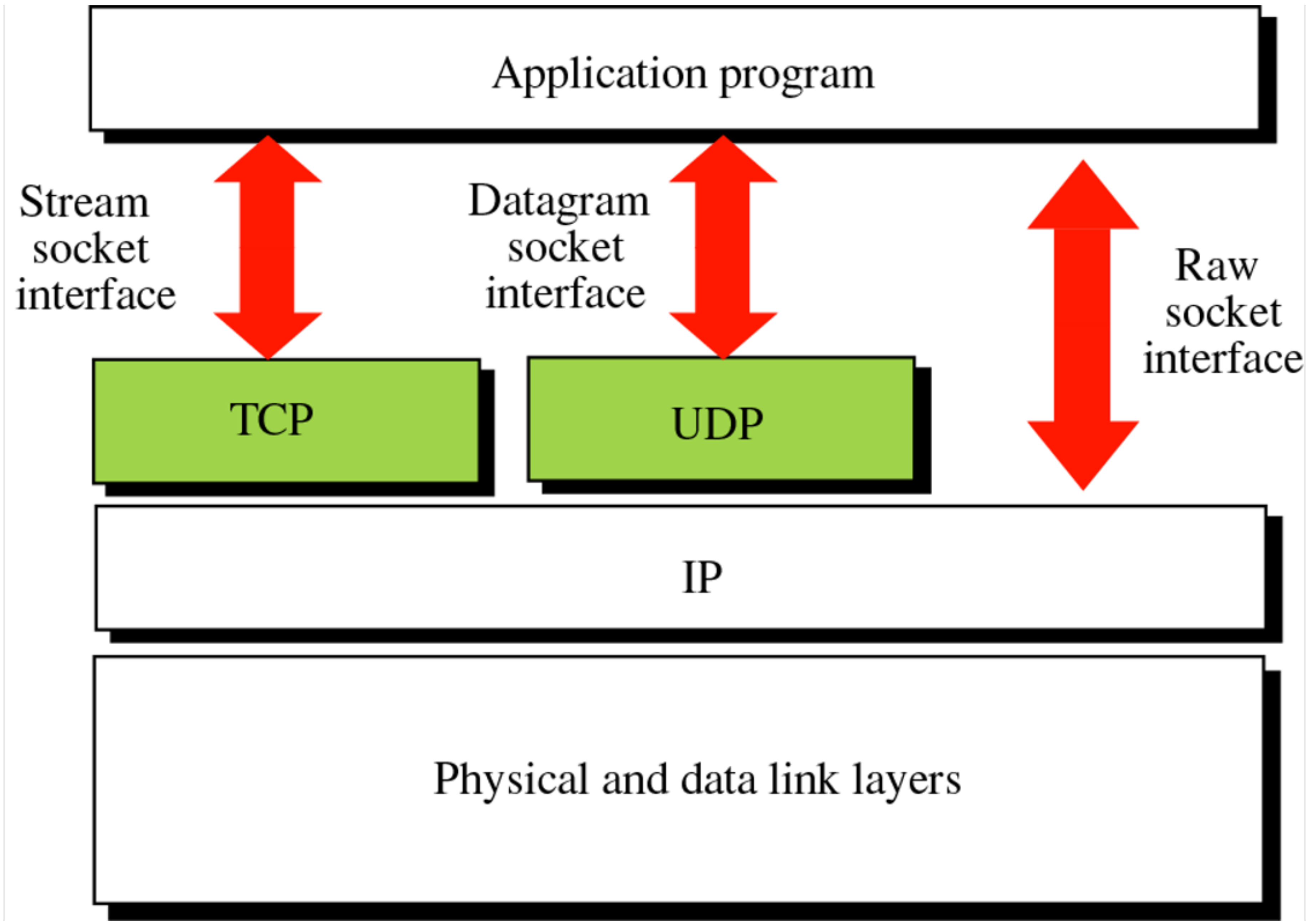
Berkeley Socket

- Basic Idea:
  - Network connection is like a file
    - Read from / Write to like to a file



  - Socket procedures in Unix are systems calls
    - Implemented in the “top half” of the kernel
  - Windows implemented as a library (DLL)

Berkeley Sockets

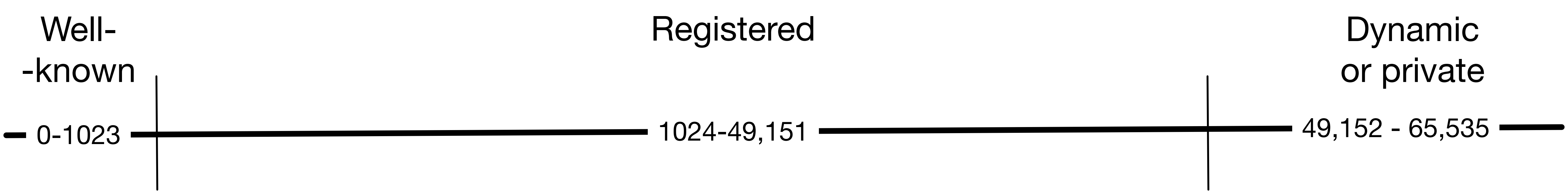


Transport Addresses

- Transport layer allows communications between processes
- Implemented via ports
    - Each host is identified by IP address
    - Each process is identified by a port number

Port Numbers

- Internet Corporation for Assigned Names and Numbers (ICANN)
  - Well-known ports: Assigned by ICANN
  - Registered ports: Neither assigned nor controlled, but can be registered to prevent duplication
  - Dynamic ports: used as temporary or private port numbers



Port Numbers

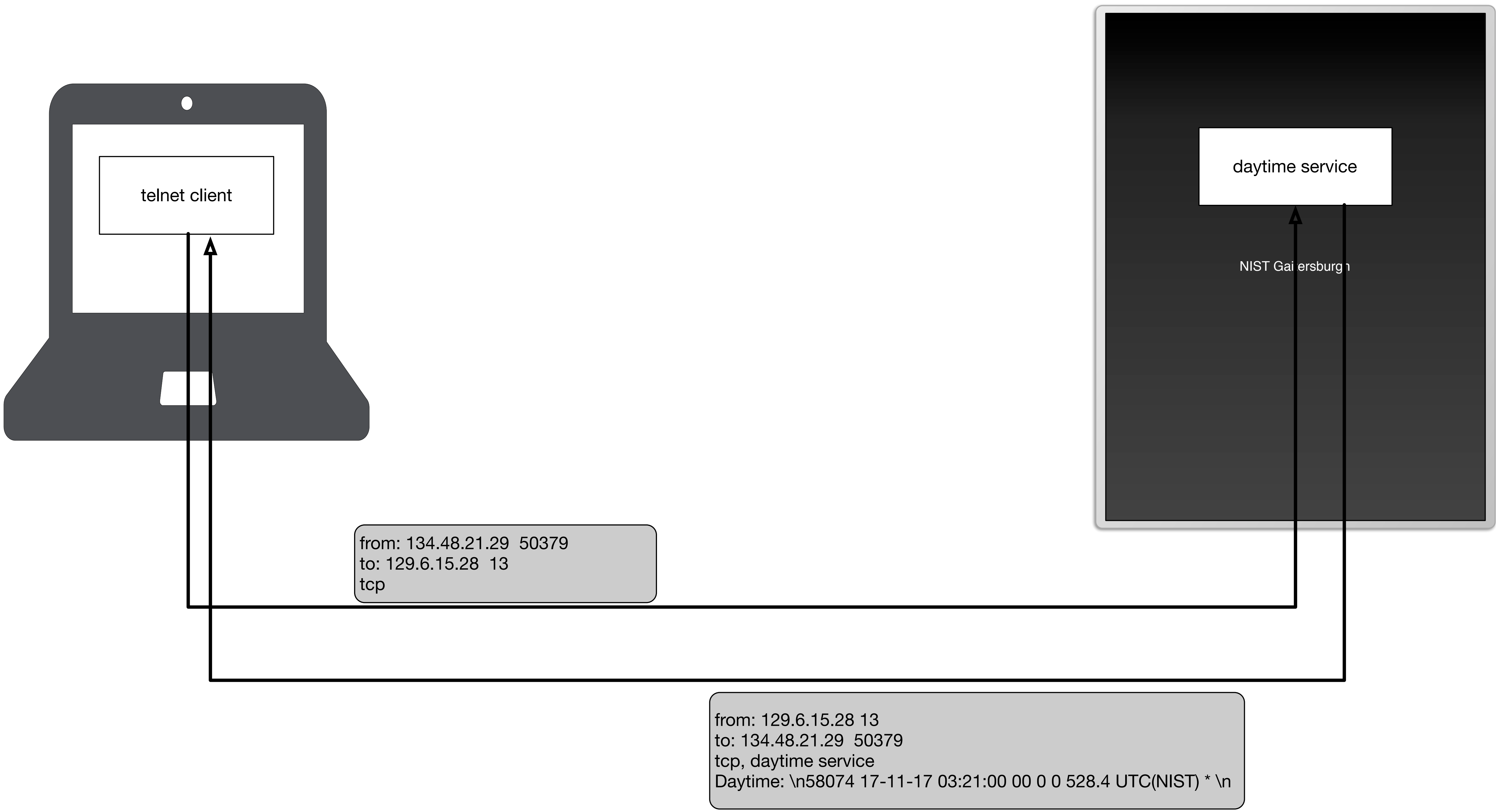
- Example:
  - telnet (needs to be installed on MacOS and Windows OS)
  - telnet 129.6.15.28 13
    - Connects to the daytime service at NIST Gaitersburg on port 13
- In MacOS / UNIX, you can find port assignments in
    - /etc/services

Port Numbers

thomasschwarz — -bash — 80×24

```
Connected to time-a-g.nist.gov.
Escape character is '^]'.

58074 17-11-17 02:10:08 00 0 0 532.0 UTC(NIST) *
Connection closed by foreign host.
MSCSs-MacBook-Pro-2:~ thomasschwarz$ clear







MSCSs-MacBook-Pro-2:~ thomasschwarz$ telnet 129.6.15.28 13
Trying 129.6.15.28...
Connected to time-a-g.nist.gov.
Escape character is '^]'.
Connection closed by foreign host.
MSCSs-MacBook-Pro-2:~ thomasschwarz$ telnet 129.6.15.28 13
Trying 129.6.15.28...
Connected to time-a-g.nist.gov.
Escape character is '^]'.
Connection closed by foreign host.
MSCSs-MacBook-Pro-2:~ thomasschwarz$
```

Port Numbers



Port Numbers

from: 134.48.21.29 50379
to: 129.6.15.28 13
tcp

- Destination address selects the server
- Destination port address selects the service (here day-time-server)
- Source address & port are needed to find the destination for the response

from: 129.6.15.28 13
to: 134.48.21.29 50379
tcp, daytime service
Daytime: \n58074 17-11-17 03:21:00 00 0 0 528.4 UTC(NIST) * \n

Port Numbers

thomasschwarz — -zsh — 80×38

```
Last login: Fri Nov  5 02:03:37 on ttys000
thomasschwarz@Peter-Canisius ~ % cat /etc/services
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a
# port number for both TCP and UDP; hence, most entries her
# even if the protocol doesn't support UDP operations.
#
# The latest IANA port assignments can be gotten from
#
#       http://www.iana.org/assignments/port-numbers
#
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 thr
#
# $FreeBSD: src/etc/services,v 1.89 2002/12/17 23:59:10 eri
#       From: @(#)services      5.8 (Berkeley) 5/9/91
#
# WELL KNOWN PORT NUMBERS
#
rtmp              1/ddp    #Routing Table Maintenance Proto
tcpmux            1/udp     # TCP Port Service Multiplexer
tcpmux            1/tcp     # TCP Port Service Multiplexer
#                          Mark Lottor <MKL@nisc.sri.com>
nbp               2/ddp    #Name Binding Protocol
compressnet       2/udp     # Management Utility
compressnet       2/tcp     # Management Utility
compressnet       3/udp     # Compression Process
compressnet       3/tcp     # Compression Process
#                          Bernie Volz <VOLZ@PROCESS.COM>
echo              4/ddp    #AppleTalk Echo Protocol
#                 4/tcp    Unassigned
#                 4/udp    Unassigned
rje               5/udp     # Remote Job Entry
rje               5/tcp     # Remote Job Entry
#                          Jon Postel <postel@isi.edu>
```

thomasschwarz — -zsh — 80×38

```
ibm_wrless_lan  1461/udp    # IBM Wireless LAN
ibm_wrless_lan  1461/tcp    # IBM Wireless LAN
#                          <flanne@vnet.IBM.COM>
world-lm        1462/udp    # World License Manager
world-lm        1462/tcp    # World License Manager
#                          Michael S Amirault <ambi@world.std.com>
nucleus         1463/udp    # Nucleus
nucleus         1463/tcp    # Nucleus
#                          Venky Nagar <venky@fafner.Stanford.EDU>
msl_lmd         1464/udp     # MSL License Manager
msl_lmd         1464/tcp     # MSL License Manager
#                           Matt Timmermans
pipes           1465/udp     # Pipes Platform  mfarlin@peerlogic.com
pipes           1465/tcp     # Pipes Platform
#                           Mark Farlin <mfarlin@peerlogic.com>
oceansoft-lm    1466/udp     # Ocean Software License Manager
oceansoft-lm    1466/tcp     # Ocean Software License Manager
#                           Randy Leonard <randy@oceansoft.com>
csdmbase        1467/udp     # CSDMBASE
csdmbase        1467/tcp     # CSDMBASE
csdm            1468/udp     # CSDM
csdm            1468/tcp     # CSDM
#               Robert Stabl <stabl@informatik.uni-muenchen.de>
aal-lm          1469/udp     # Active Analysis Limited License Manager
aal-lm          1469/tcp     # Active Analysis Limited License Manager
#                           David Snocken  +44 (71)437-7009
uaiact          1470/udp     # Universal Analytics
uaiact          1470/tcp     # Universal Analytics
#                           Mark R. Ludwig <Mark-Ludwig@uai.com>
csdmbase        1471/udp     # csdmbase
csdmbase        1471/tcp     # csdmbase
csdm            1472/udp     # csdm
csdm            1472/tcp     # csdm
#               Robert Stabl <stabl@informatik.uni-muenchen.de>
openmath        1473/udp     # OpenMath
openmath        1473/tcp     # OpenMath
#                           Garth Mayville <mayville@maplesoft.on.ca>
telefinder      1474/udp     # Telefinder
```

Finding Open Ports

- To find open ports:
  - Can use a port scanner over the network that systematically tries out all ports
  - Can use systems tools
    - MacOS:

```
thomasschwarz@Peter-Canisius ~ % lsof -i -P | grep -i "listen"
rapportd    527 thomasschwarz    5u  IPv4 0xc604072814ca13a1      0t0  TCP *:62127 (LISTEN)
rapportd    527 thomasschwarz    9u  IPv6 0xc604072814573699      0t0  TCP *:62127 (LISTEN)
ControlCe  1666 thomasschwarz   12u  IPv4 0xc604072801237e41      0t0  TCP *:7000 (LISTEN)
ControlCe  1666 thomasschwarz   13u  IPv6 0xc6040727f7e8ad79      0t0  TCP *:7000 (LISTEN)
ControlCe  1666 thomasschwarz   16u  IPv4 0xc6040727f968c381      0t0  TCP *:5000 (LISTEN)
ControlCe  1666 thomasschwarz   17u  IPv6 0xc6040728037e4b39      0t0  TCP *:5000 (LISTEN)
mongod     1736 thomasschwarz   10u  IPv4 0xc604072814a513a1      0t0  TCP localhost:27017 (LISTEN)
Google    62219 thomasschwarz  136u  IPv4 0xc604072814c9fe41      0t0  TCP localhost:49787 (LISTEN)
```

Finding Open Ports

- On Windows:
    - netstat
    - nbtstat

Socket Address

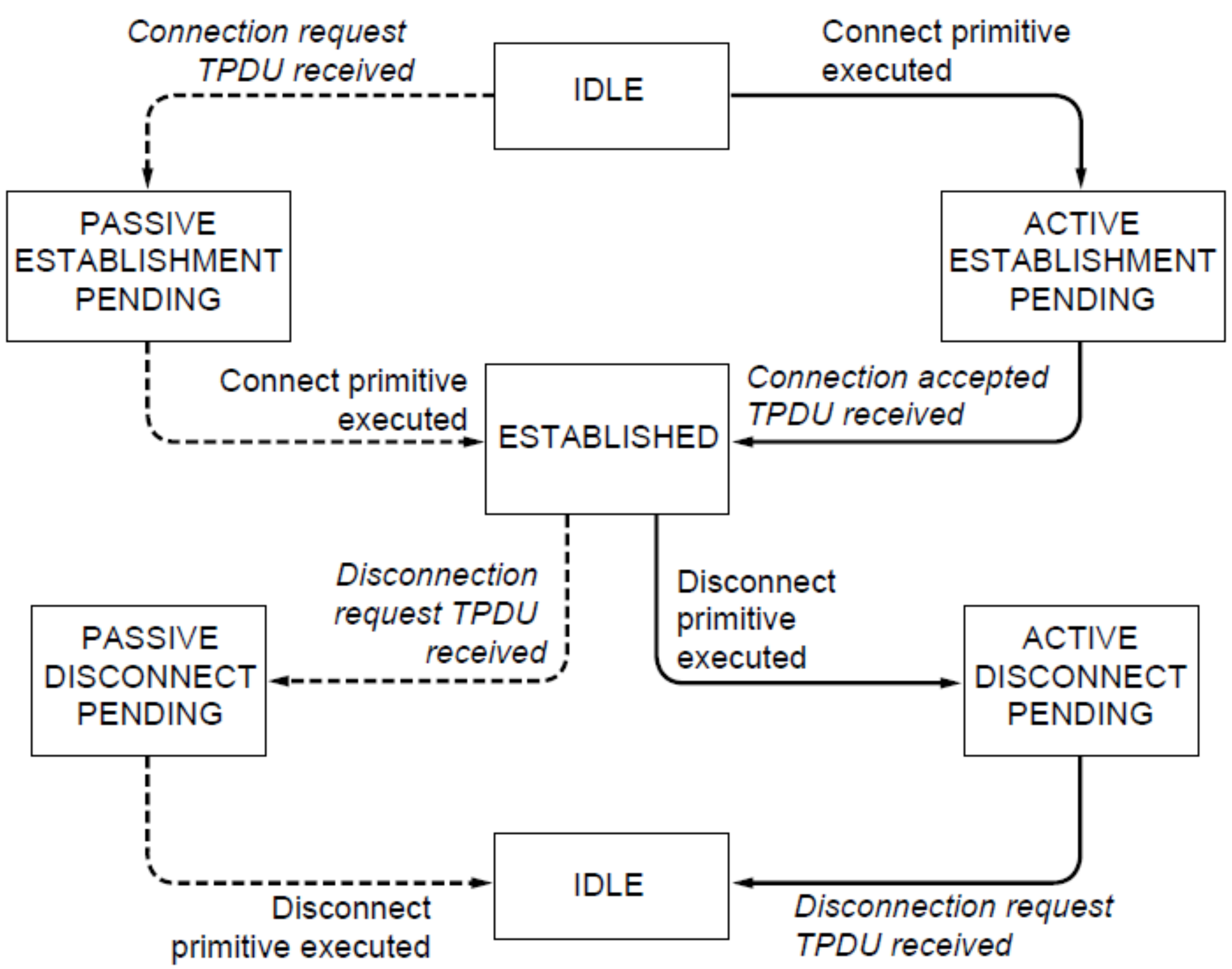
- The combination of IP address and port number is the *socket address*

Transport Service Primitives

- 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

Transport Service Primitives



Solid lines (right) show client state sequence

Dashed lines (left) show server state sequence

Transitions in italics are due to segment arrivals.

Addressing

- How does an application find port numbers?
  - Portmapper (which listens at a well known port)
    - User sends service name and gets port address
    - Services must register with the portmapper
  - Initial connection protocol
    - Each machine with services has a process server that acts as proxy for less heavily used servers
      - inetd on Unix systems
    - Listens to a range of ports waiting for connection requests
    - Process server spawns requested server (if necessary)

Socket Programming In Python

- Python translates the UNIX socket interface
  - IPv4: Use a tuple IP-address, port number
- Sockets go through a life cycle:
  - Creation, Connection, Receiving / Sending, Closing
  - Creation, Binding, Listening, Closing

Socket Programming In Python

- Example:
  - A simple writer to another process
  - Data is send as a byte stream
  - Using local-loop to avoid opening the firewall

Socket Programming In Python

- Server:
  - Create socket

```
import socket

HOST = '127.0.0.1'  #Loopback interface
PORT = 65431  #Silly port

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

Socket Programming In Python

- Bind socket to port and listen

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
```

Socket Programming In Python

- Receive data from client, stop when no data remains

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    print('Connection from:', conn)
    while True:
        data = conn.recv(1024)
        if not data:
            break
        #conn.sendall(data)  to return
        print(data.decode('UTF-8'))
```

- Data is send in binary, as UTF-8

Socket Programming In Python

- Sender / Client:
  - Instead of binding, we directly connect to the socket

```
import socket

HOST = '127.0.0.1'  #Loopback interface
PORT = 65431  #Silly port

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
```

Socket Programming In Python

- Sender / Client:
  - Now we can write to the server

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    myinput = b'?:'
    while myinput:
        s.send(myinput)
        myinput = bytes(input('?:'), 'utf-8')
    s.close() #not necessary
```

UDP

- User Datagram Protocol (UDP)
  - Only adds socket addressing to the networking layer
  - Useful if you do not want the overhead of connection establishment and maintenance

UDP Header

32 Bits

Source port	Destination port
UDP length	UDP checksum

Length: Includes the 8B UDP header

Checksum: Calculated from a pseudo-header (part of IP header)
UDP header (without check-sum)
Data

UDP Applications

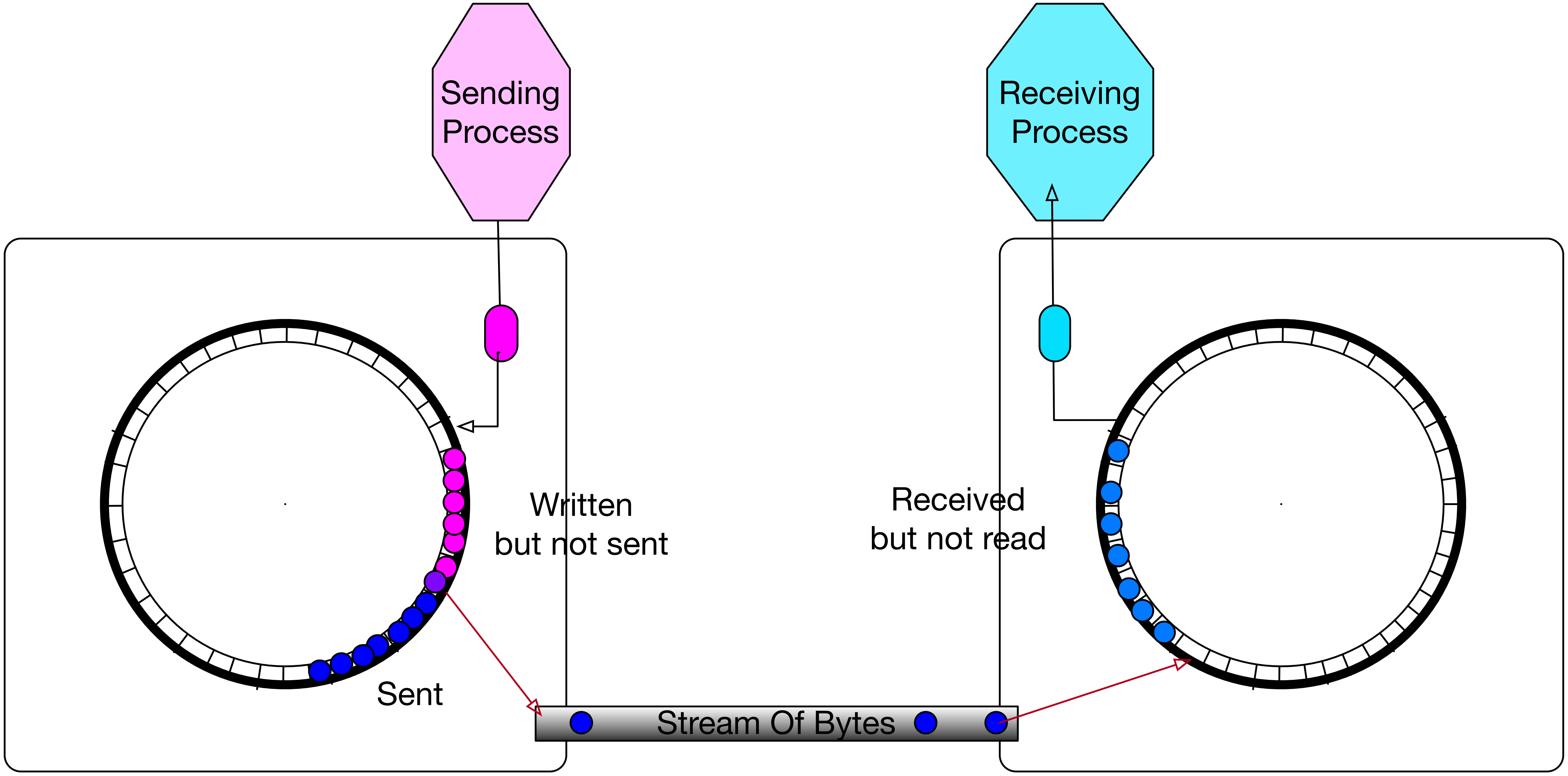
- Replacement for daytime
- Domain Name Service
- Real time services (Phone over IP, Skype, …)
- Congested networks:
  - UDP does not try to control congestion and therefore does not send additional packets
- Trivial File Transfer Protocol: Error- and flow control are built in at the application level
- Multicasting: Built into UDP software, but not TCP
- RIP: Routing Information Protocol

Transmission Control Protocol

- Process-to-process communication
- Stream-oriented protocol
- Full duplex communication
- Connection oriented
- Reliable Service

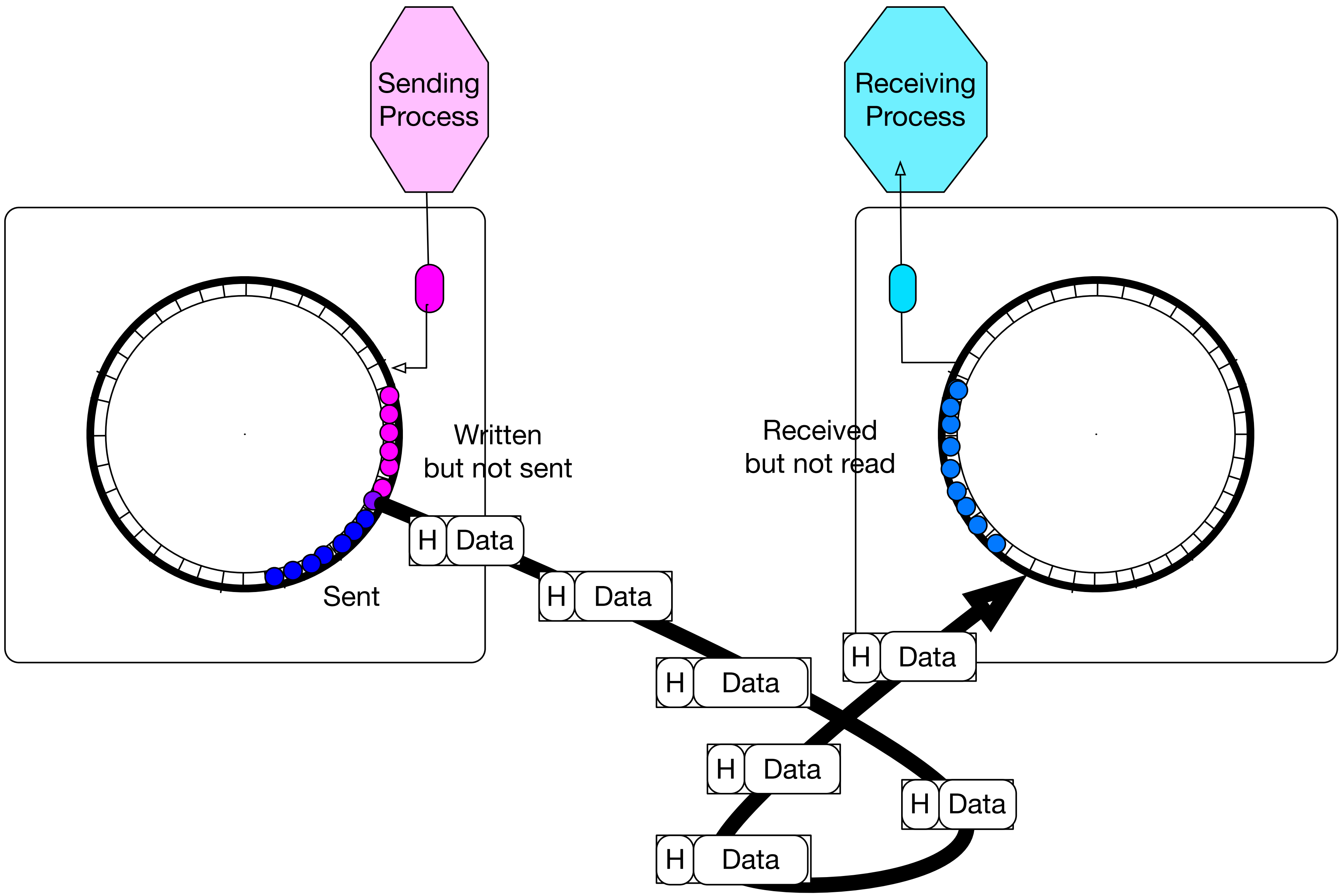
TCP

- Sending and receiving buffers mediate between transport and application layer

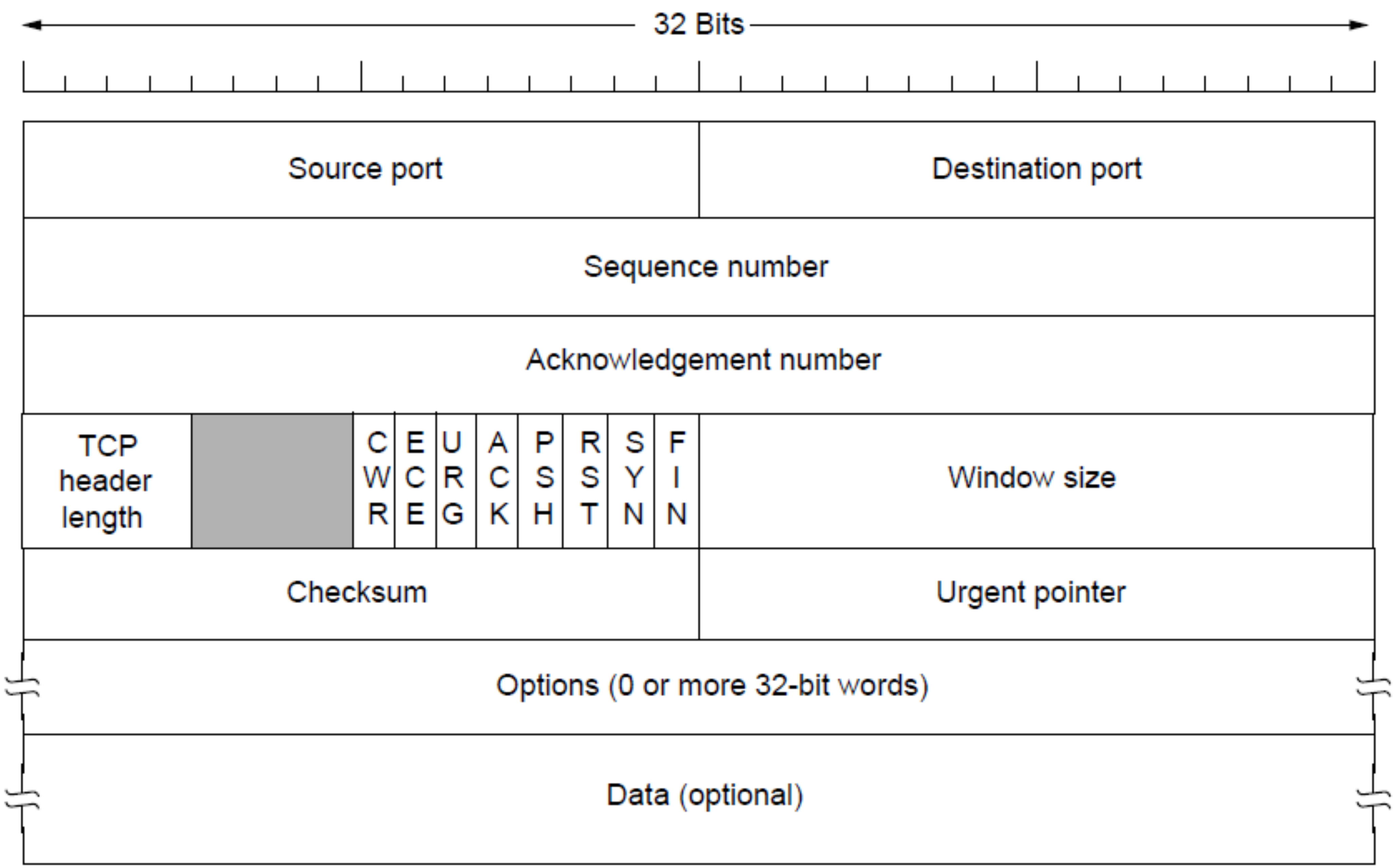


TCP

- Bytes are bundled into segments



TCP Segment Header



TCP Sequence Number

- Refers to a byte count
    - TCP chooses an arbitrary number — Initial Sequence Number (ISN) — between 0 and $2^{32}-1$
    - Sequence number for the first segment is the ISN
    - Sequence number for the next segment is the number of bytes in the first segment added to ISN
    - Sequence number for the next segment is the number of bytes in the previous segment added to previous segment number
    - Sequence numbers wrap around 0

TCP Sequence Number

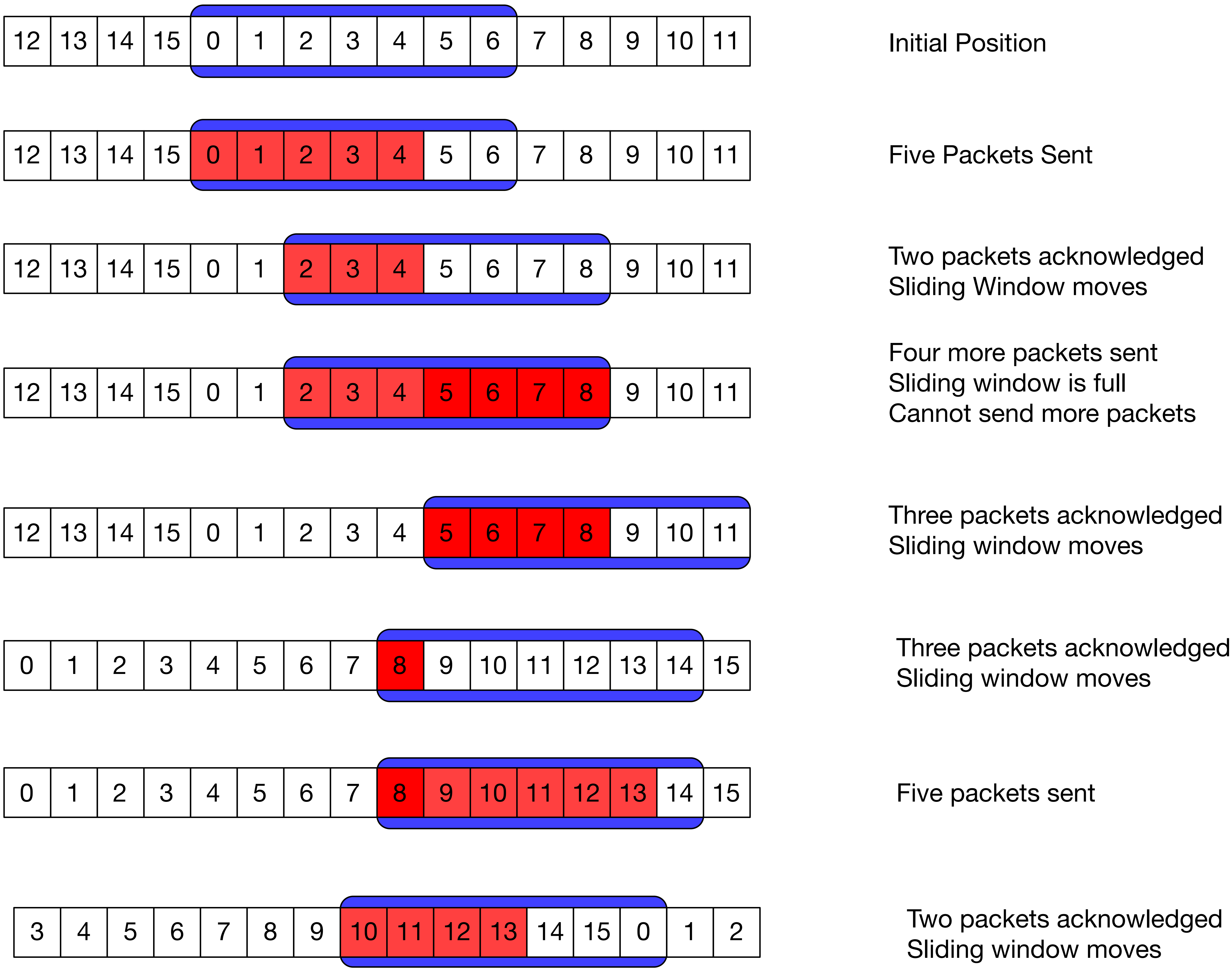
- Actually:
  - Need to keep segments apart in the following scenario:
    - Process makes a TCP connection
    - System fails
    - System and process restarts
    - Process makes the same TCP connection
  - Pre- and post-crash segments need to be distinguished

Acknowledgment Numbers

- A TCP connection is duplex:
    - When a connection is established, both parties send and receive packets.
- Receiver sends acknowledgments embedded in their packets
- Senders use timers to resend un-acknowledged packages
    - Receiver discards corrupted packages
    - Sender realizes that they are lost because of lack of acknowledgment and a timer

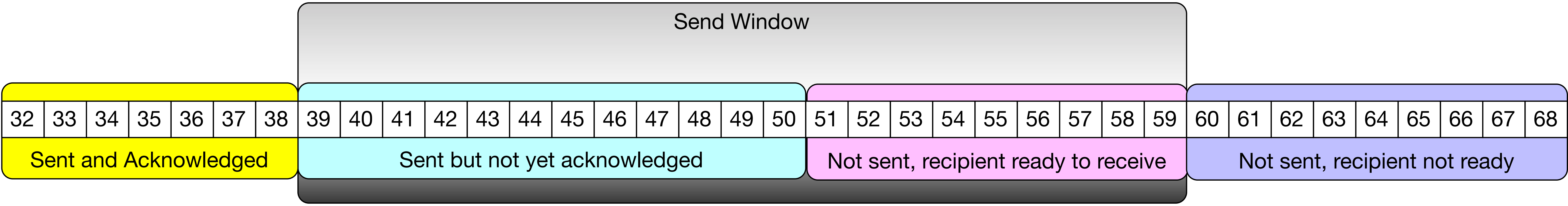
Sliding Window

- Sequence numbers are numbers modulo 2^{32}
- Sliding window is less than half of the sequence number range


12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
Initial Position
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
Five Packets Sent
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
Two packets acknowledged
Sliding Window moves
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
Four more packets sent
Sliding window is full
Cannot send more packets
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
Three packets acknowledged
Sliding window moves
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Three packets acknowledged
Sliding window moves
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Five packets sent
3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2
Two packets acknowledged
Sliding window moves

Categories of a TCP Transmission Stream

- At the sender:
  - Four categories
    1. Bytes sent and acknowledged
    2. Bytes sent but not yet acknowledged
    3. Bytes not yet sent, but the recipient is ready
    4. Bytes not sent and the recipient is not ready

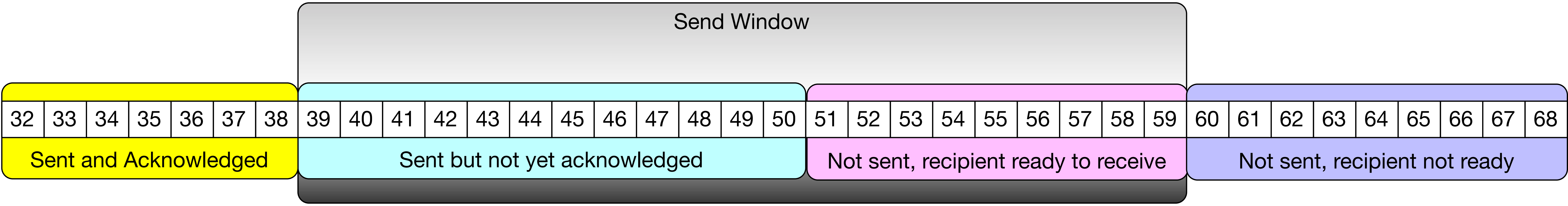


TCP Transmission Stream

- At the receiver:
    1. Bytes received and acknowledged
    2. Bytes received and not acknowledged
    3. Bytes not yet received but ready to receive
    4. Bytes not yet received and not ready to receive

TCP Transmission Stream Send Window

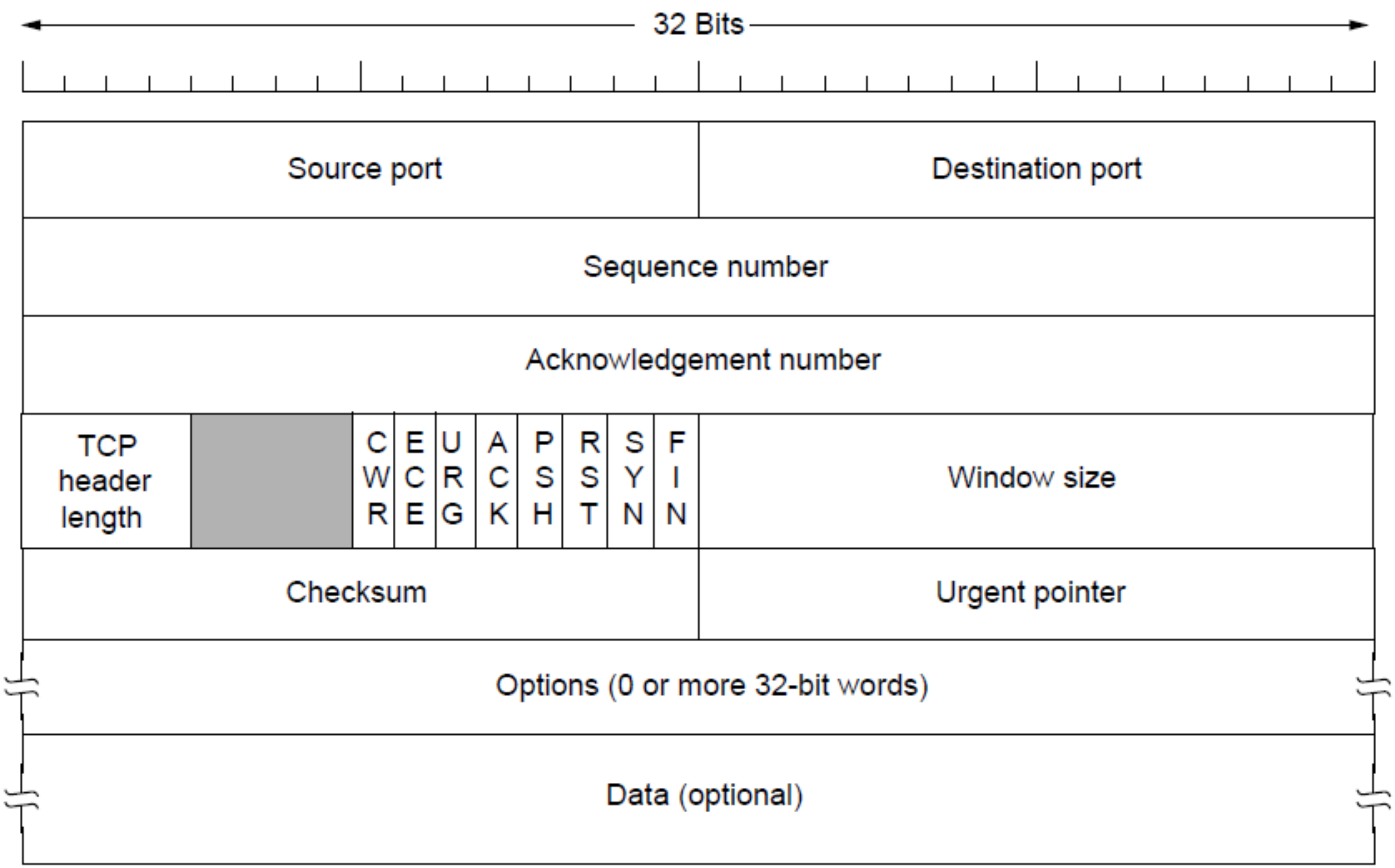
- Send Window:
  - The bytes that the sender is allowed to transmit
    - Category 2 and 3
- Usable Window:
  - The bytes that the sender is still allowed to send
    - Category 3



TCP Transmission Stream

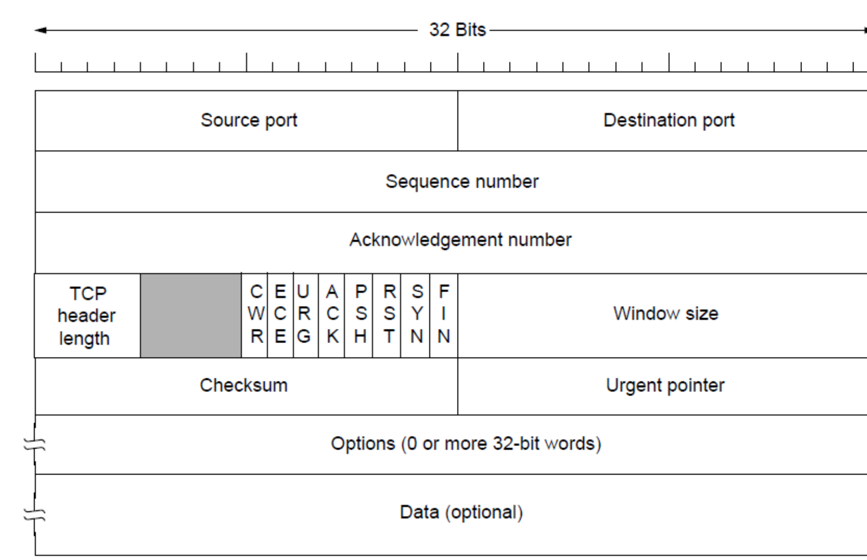
- Lacking acknowledgments:
  - Each segment triggers a timer
    - If the timer expires and the segment is not acknowledged, it is retransmitted
  - This works independently of whether the segment was dropped or the segment with the acknowledgment was dropped

TCP Segment Header



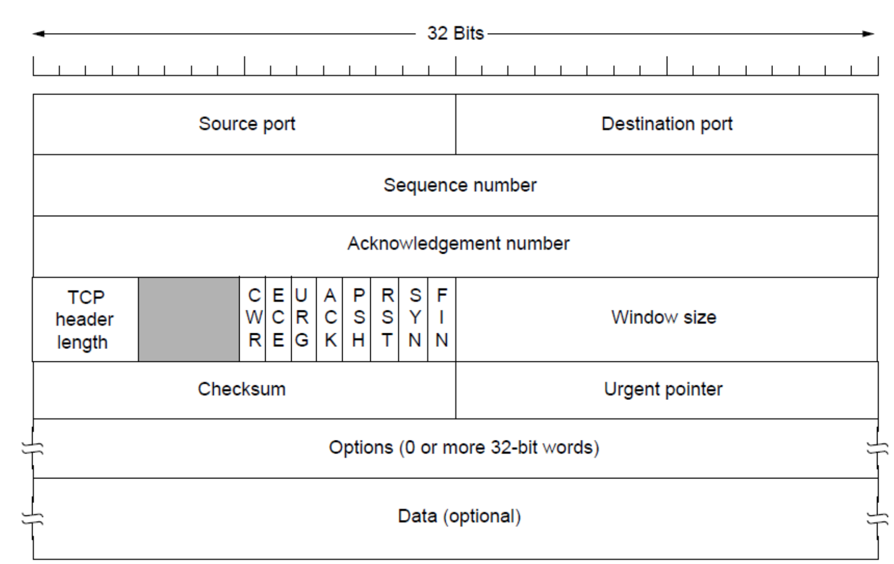
TCP Segments

- Header length
  - 4b field for the number of 4Bs in the header
  - Headers can be between 20 and 60 bytes



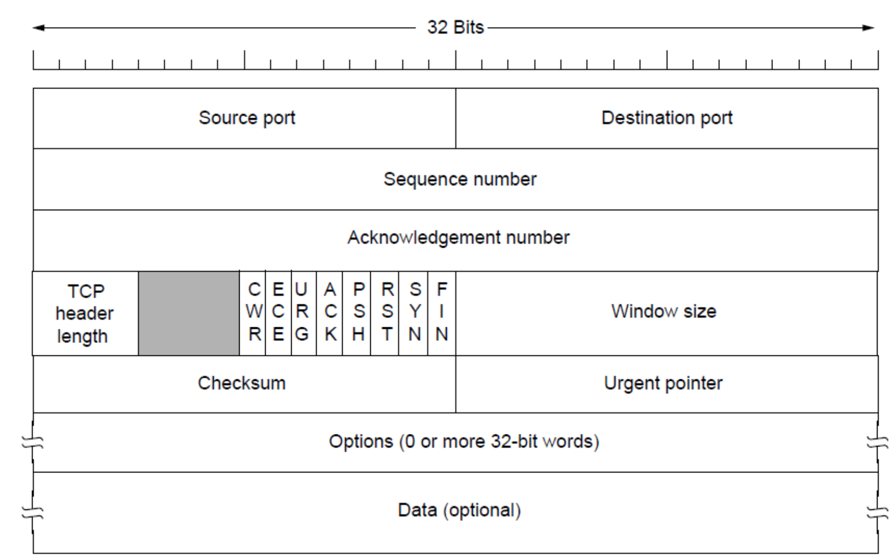
TCP-Segment

- Control flags: Set as bit flags
  - CWR — Congestion window reduced (rare)
  - ECN — Echo. Used by ECN-TCP connections (rare)
  - URG — Urgent: Receiving TCP stack can process the urgent data immediately
  - ACK — Acknowledgment
  - PSH — Push
  - RST — Reset
  - SYN — Synchronize
  - FIN — Terminate connection



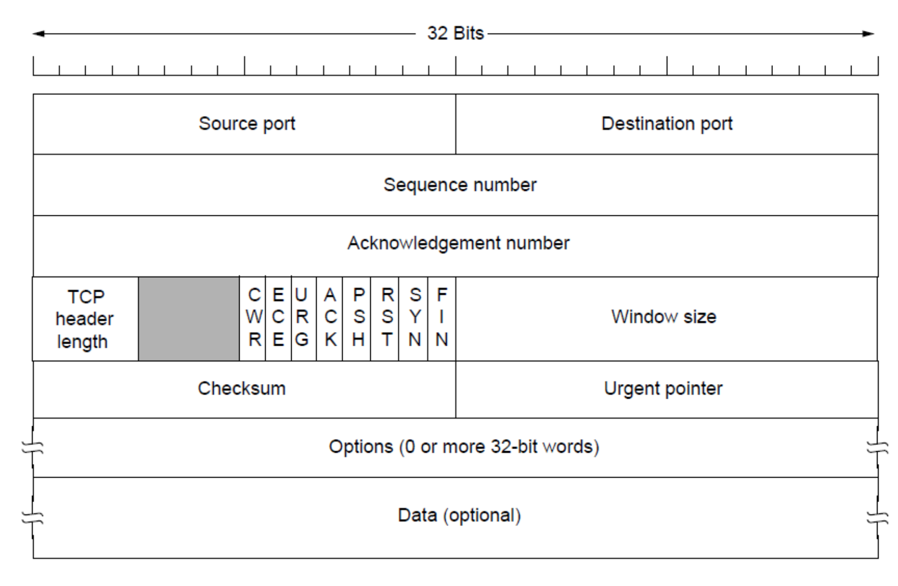
TCP-Segment

- Window size — TCP receiver window size:
    - How much data is the receiving device willing to receive at any moment
    - If the receiver is overwhelmed, will send a zero window size
    - Sender probes with TCP Window Update messages to get flow going again



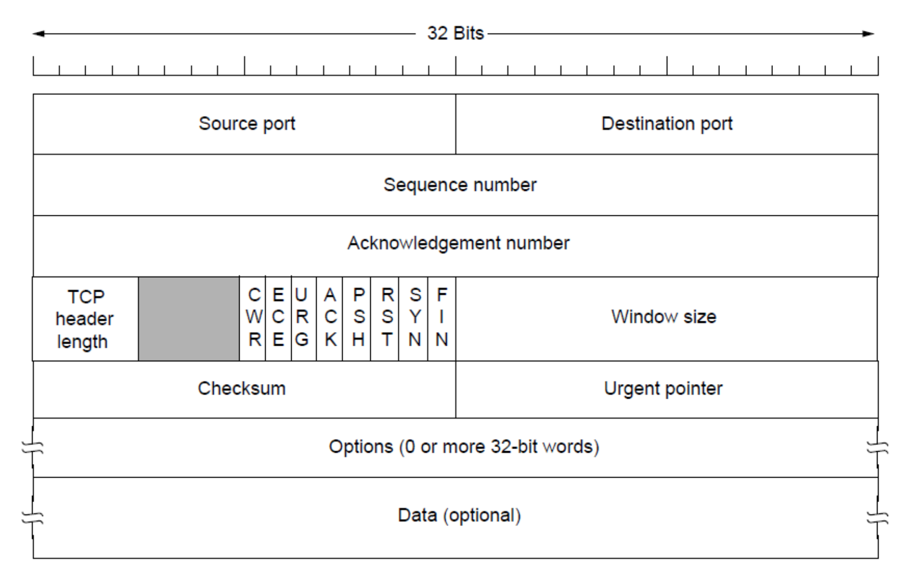
TCP-Segment

- Checksum
  - Includes segment and an IP pseudo-header
  - Use is mandatory



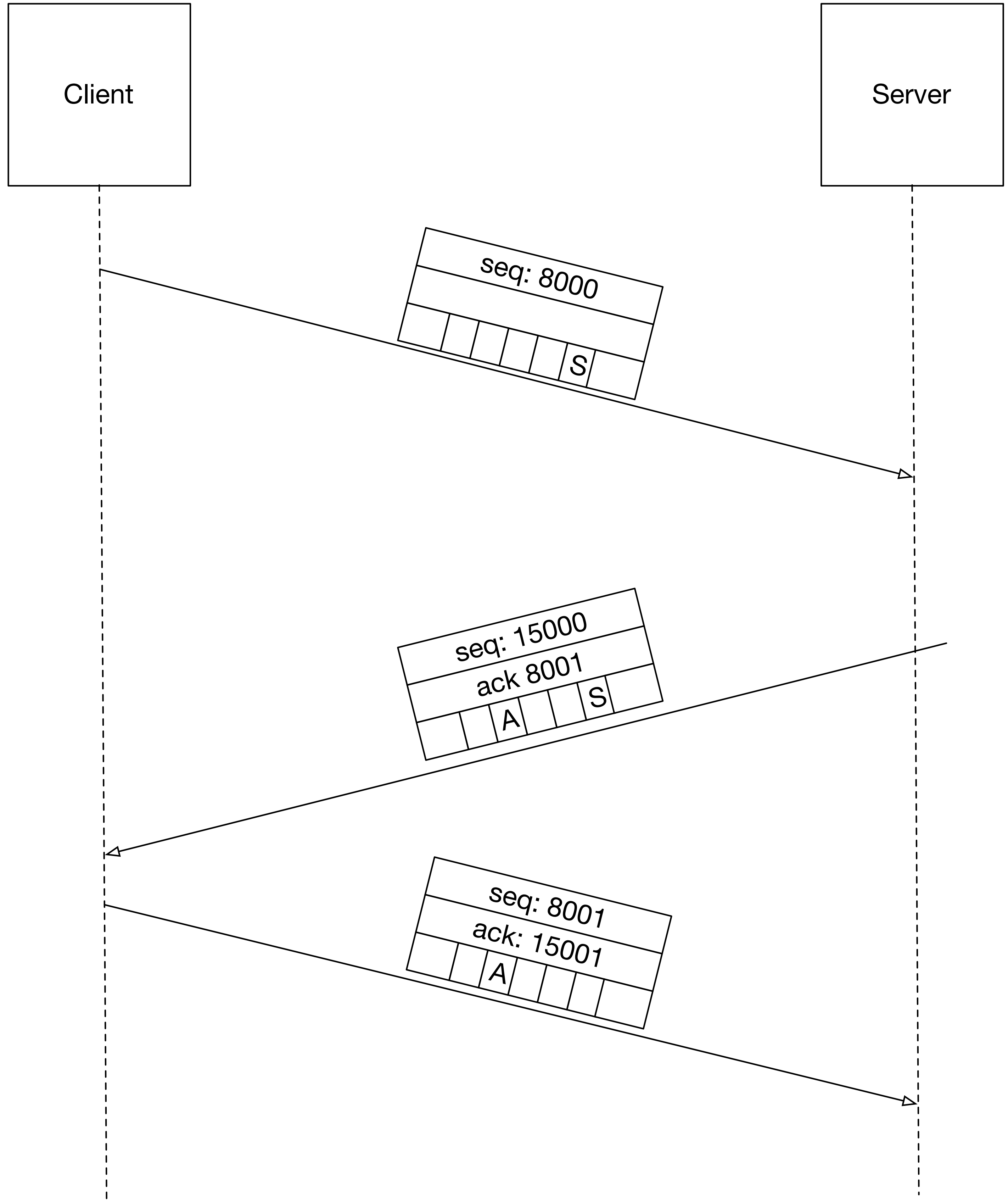
TCP-Segment

- Urgent pointer
  - Used only when URG flag is set
  - Defines a value that needs to be added to the sequence number
  - This defines the number of the last urgent byte

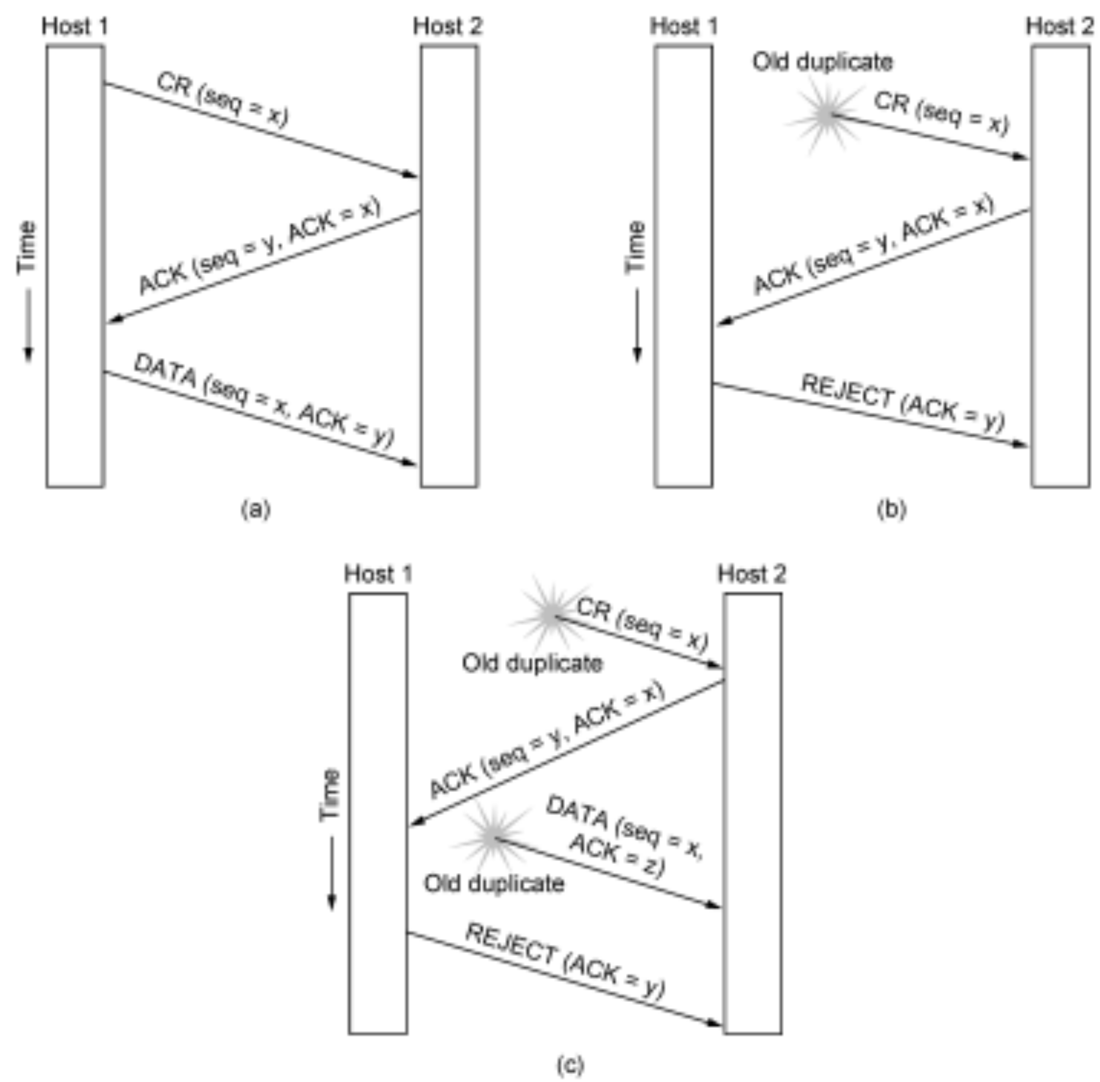


TCP Connections

- TCP transmits data in full-duplex mode
- Three way handshake:
  - Server sends a SYN packet
    - with the Syn bit set
    - with a starting syn-number
  - Receivers sends a SYN-ACK packet
    - with a starting syn-number for the other direction
  - Server sends an ACK packet


Client
Server
seq: 8000
S
seq: 15000
ack 8001
A
S
seq: 8001
ack: 15001
A

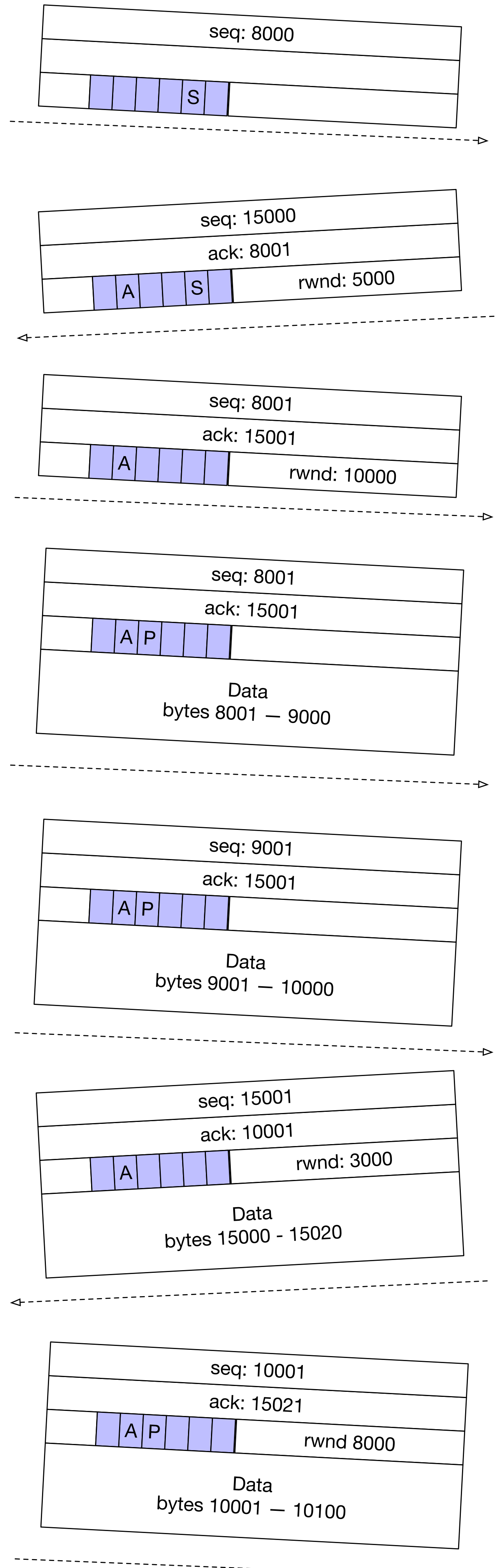
TCP Connection Setup



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. (a)Normal operation. (b)Old duplicate CONNECTION REQUEST appearing out of nowhere. (c)Duplicate CONNECTION REQUEST and duplicate ACK

TCP Connections

- SYN carries no data, but is counted as one byte in a stream
- SYN-ACK carries no data, but is counted as one byte in a stream
- If ACK carries no data, it is **not** counted as a byte



Syn

Syn Ack

Ack

Data sent

Data sent

Data received

Data sent


seq: 10101
ack: 15021
A
P
rwnd 8000
Data
bytes 10101 — 10200
Data sent
seq:10201
ack: 15021
A
F
rwnd: 8000
Fin sent
seq: 15001
ack: 10201
A
Fin acknowledged
seq: 15001
ack: 10201
A
Data
bytes 15021 - 16021
Data received
seq: 10201
ack: 16021
A
rwnd: 8000
Data acknowledged
seq: 16021
ack: 10201
A
F
Fin Received
seq:10201
ack: 16021
A
Fin acknowledged

TCP Connections

- To tear down a connection
  - Three-way handshake
    - One party sends a FIN message
      - Counts as one byte
    - Other party responds with a FIN-ACK message
    - First party acknowledges

TCP-Connection

- Half-close
  - Used when one sides does not want to send any more data
  - Initiator sends a Fin message
  - Receiver acknowledges
  - Receiver can still send segments to the initiator
  - Initiator only sends acknowledgments
  - Eventually, receiver sends a Fin message
  - Initiator acknowledges

Syn Flood

- Sending many syn requests forces receiver to spend resources
  - Because receiver needs to remember its syn number set in the syn-ack packet
- Kevin Mitnick used it to bring down machines that he was incorporating

TCP - Windows

- TCP uses two windows:
  - The send window (swin)
  - The receive window (rwin)

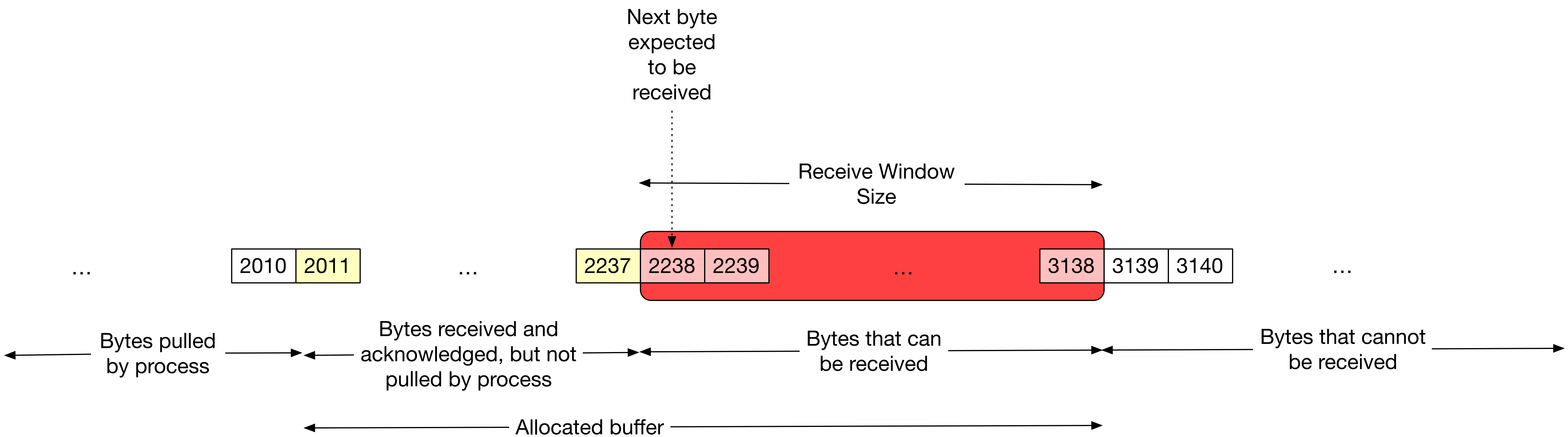
TCP - Send Window

- *Sliding Window:* Maximum number of unacknowledged bytes that a device is allowed to have outstanding
- *Usable Window*: Amount of the send window that the device is still allowed to send
  - Window size in bytes
    - Sliding window algorithms
      - Window size cannot be more than half the number of segment numbers
    - Window slides with acknowledgments from receiver

TCP - Receive Window

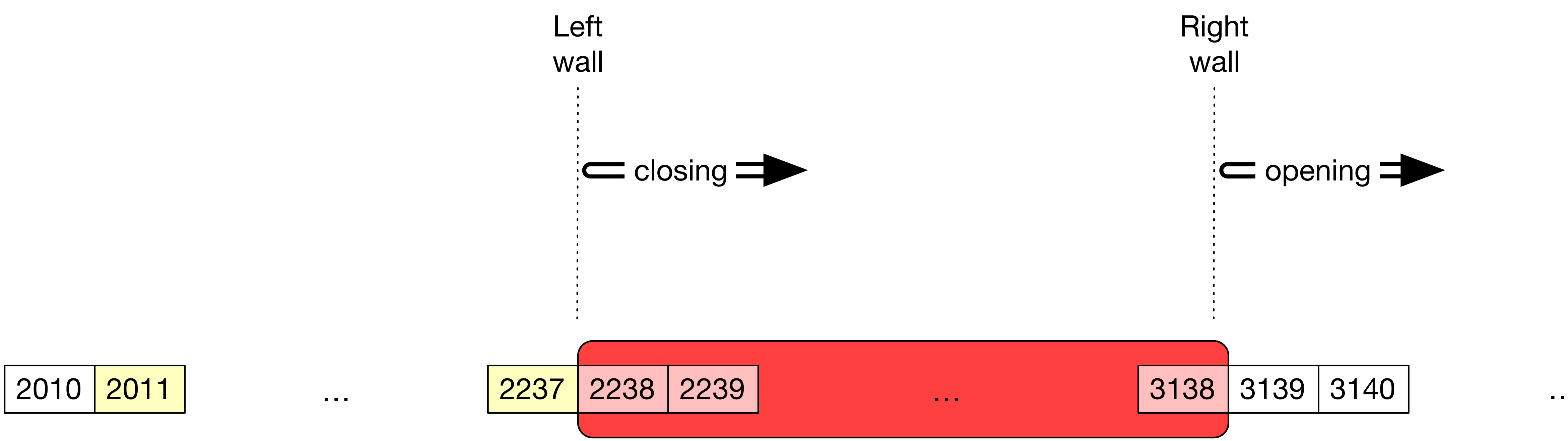
- Necessary because segments can arrive out of order
  - Receive window defines the byte numbers that can be accepted.
  - Bytes outside of the receive window are not accepted.
  - The receiver publishes ***rwnd***, the difference between buffer size and the number of bytes to be pulled by the process

TCP - Receive Window

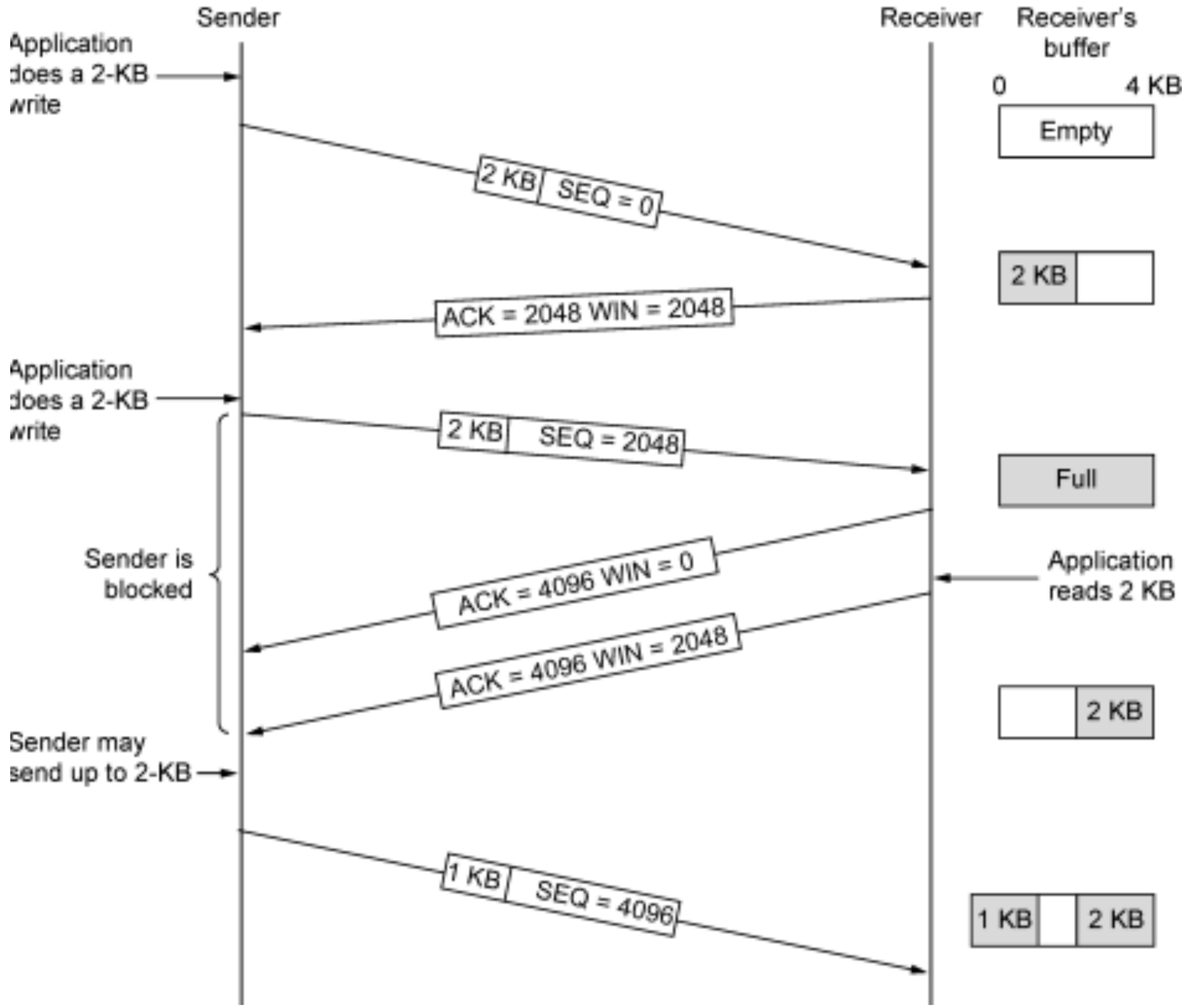


TCP-Receive Window

- Receive window closes by receiving segments
- Receive window opens by process consuming bytes



Window Management in TCP



TCP - Flow Control

- Flow control balances
  - rate at which a producer can produce
  - rate at which a receiver can consume
- TCP forces sender and receiver to adjust their flow control

TCP - Flow Control

- Send window changes controlled by receiver
  - Closes when receiver sends an ack
    - Left wall is moved to the right
  - Opens, when the receive window size (*rwnd*) allows it:
    - `new AckNr + new rwnd > last AckNr + last rwnd`
  - If this is violated, then the window shrinks
    - which can cause problems, because sender might already have sent data

TCP - Flow Control

- Window shut-down
  - Receiver sends a rwnd of zero
    - Means receiver does not want any data
  - Sender can *probe* by sending segments with a single byte
  - The acknowledgment by receiver can reset the rwnd if so desired

Silly Window Syndrome 1

- If the send window is very small
  - Sender can send segments with only few bytes
  - TCP packets have an overhead of 40B
    - 41B to send 1B is a lot of overhead
    - But it is worse, when we take layers 1 and 2 into account

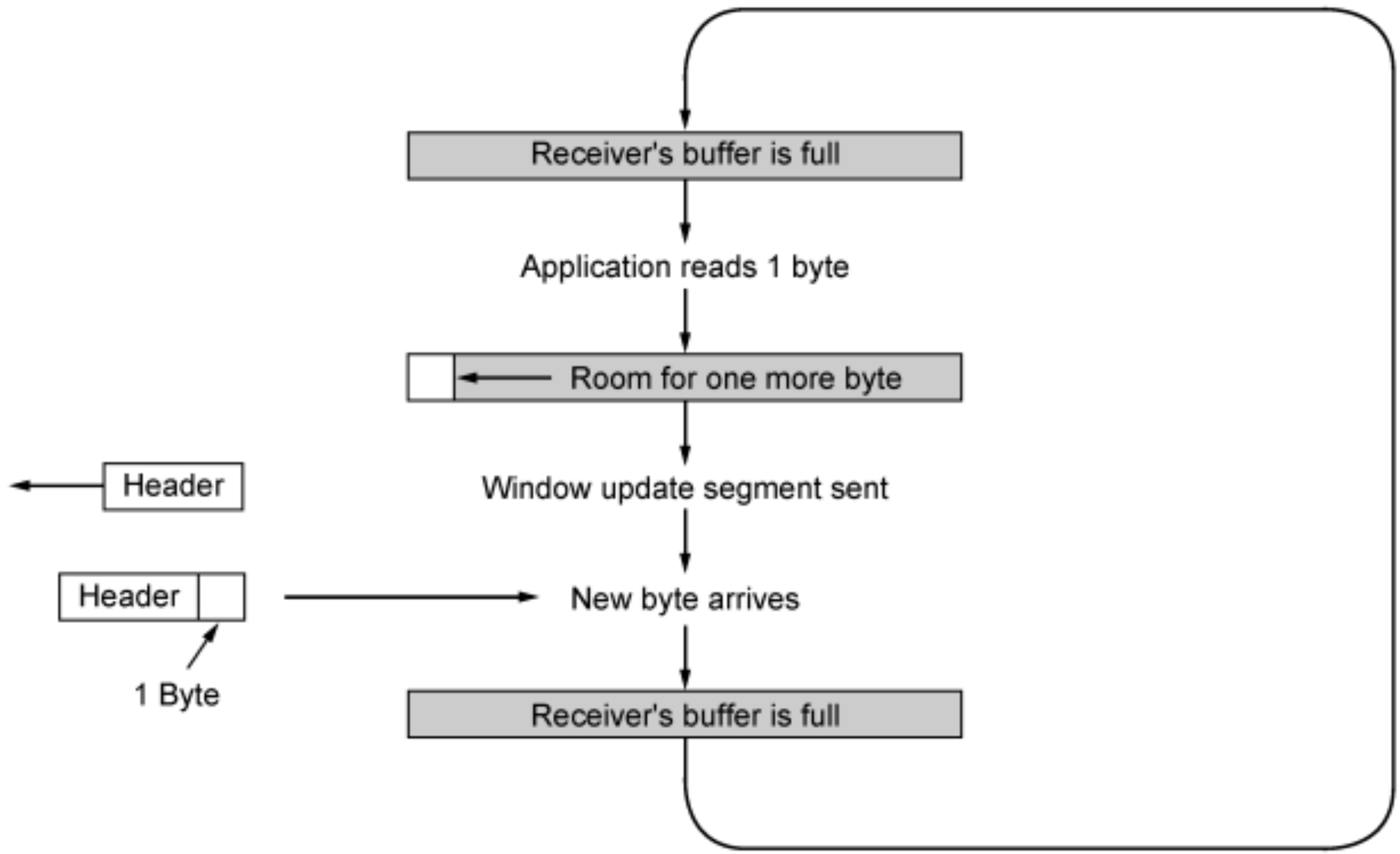
Nagle’s algorithm

- Sender sends the first piece of data it receives from process
  - Even if it is only one byte
- Sender afterwards accumulates data
- Data is sent if
  - Enough data has accumulated for a maximum sized segment
  - An acknowledgment has been received

Silly Window Syndrome 2

- If the receiver has a process that consumes bytes slowly:
  - Sender fills buffer
  - Receiver advertises a very small rwnd
  - Sender sends accordingly a very small segment

Silly Window Syndrome



Clark's Solution

- Send an acknowledgment as soon as data arrives
- But announce a window size of zero
  - Until there is enough space to accommodate a maximum-sized segment

Delayed Acknowledgments

- Only acknowledge segments when there is enough space for a maximum-sized segment
- In order to not cause the sender to resend segments, do not delay acknowledgment by more than 500 msec.

Error Control

- Checksum
  - Each segment has a checksum
  - Corrupted packets are detected and not acknowledged

Acknowledgment Types

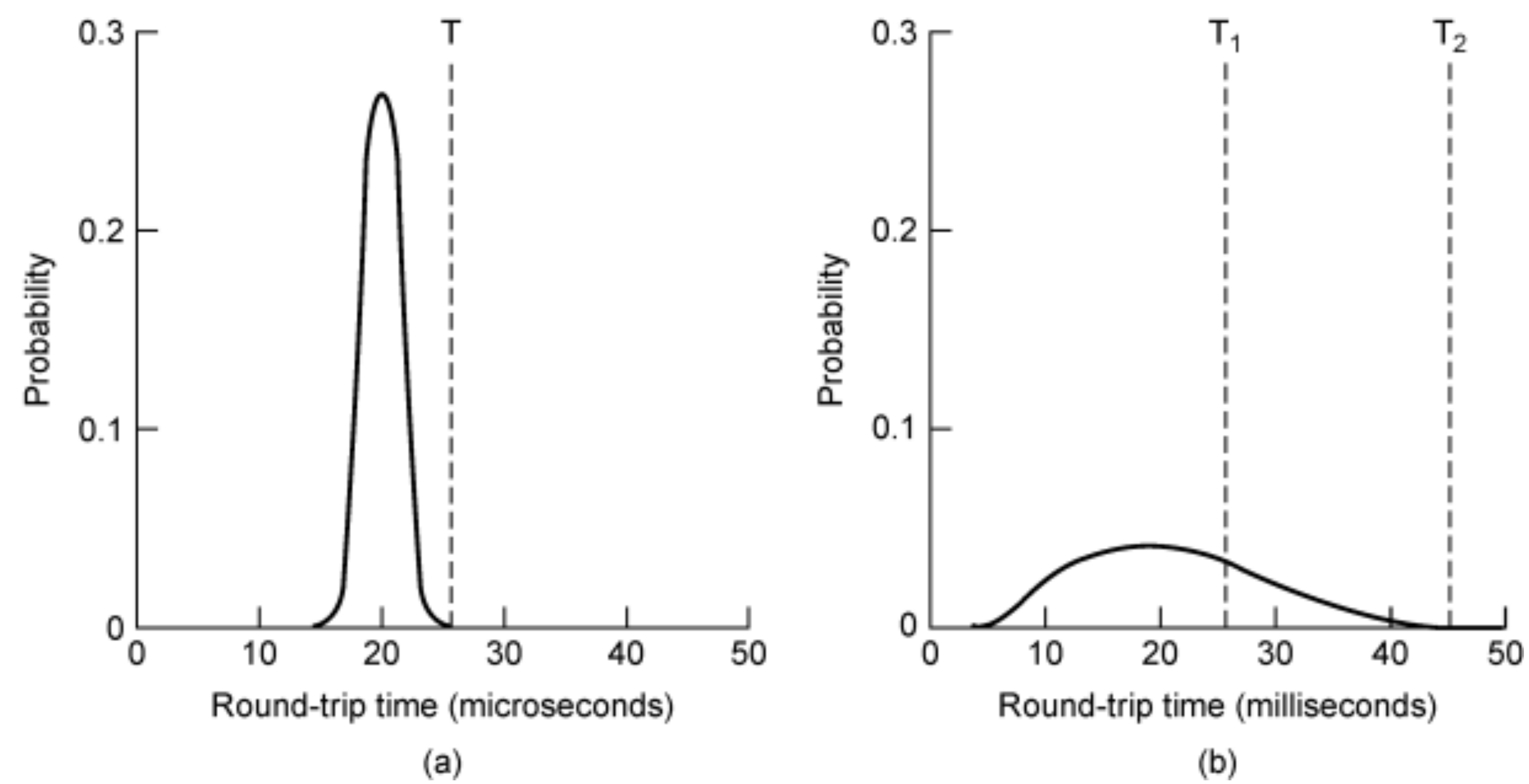
- Original: Cumulative acknowledgment
  - Receiver advertises the next byte it expects
  - Indicated by Ack bit set
- Selective Acknowledgments (SACK)
  - SACK reports
    - a block of bytes that is out of order
    - a block of bytes duplicated
  - Implemented as an option in the TCP header

Generating Acknowledgment

- Rules for generating acknowledgments:
  1. When you send a packet: piggy-backing
  2. Don’t send an ack if you are only acknowledging a single segment or if 500 msec have passed
  3. If the second unacknowledged segment arrives
  4. If segments with out-of-order numbers arrives, immediately ack with the sequence number of the next expected segment
     - Rapid retransmission
  5. When missing segments arrive, ack immediately
  6. If duplicate segments arrive, immediately send an ack indicating the next in-order segment.

TCP Timer Management

- Timers are more difficult at the transport layers



(a) Probability density of acknowledgement arrival times in the data link layer. (b) Probability density of acknowledgement arrival times for TCP.

TCP Timer Management

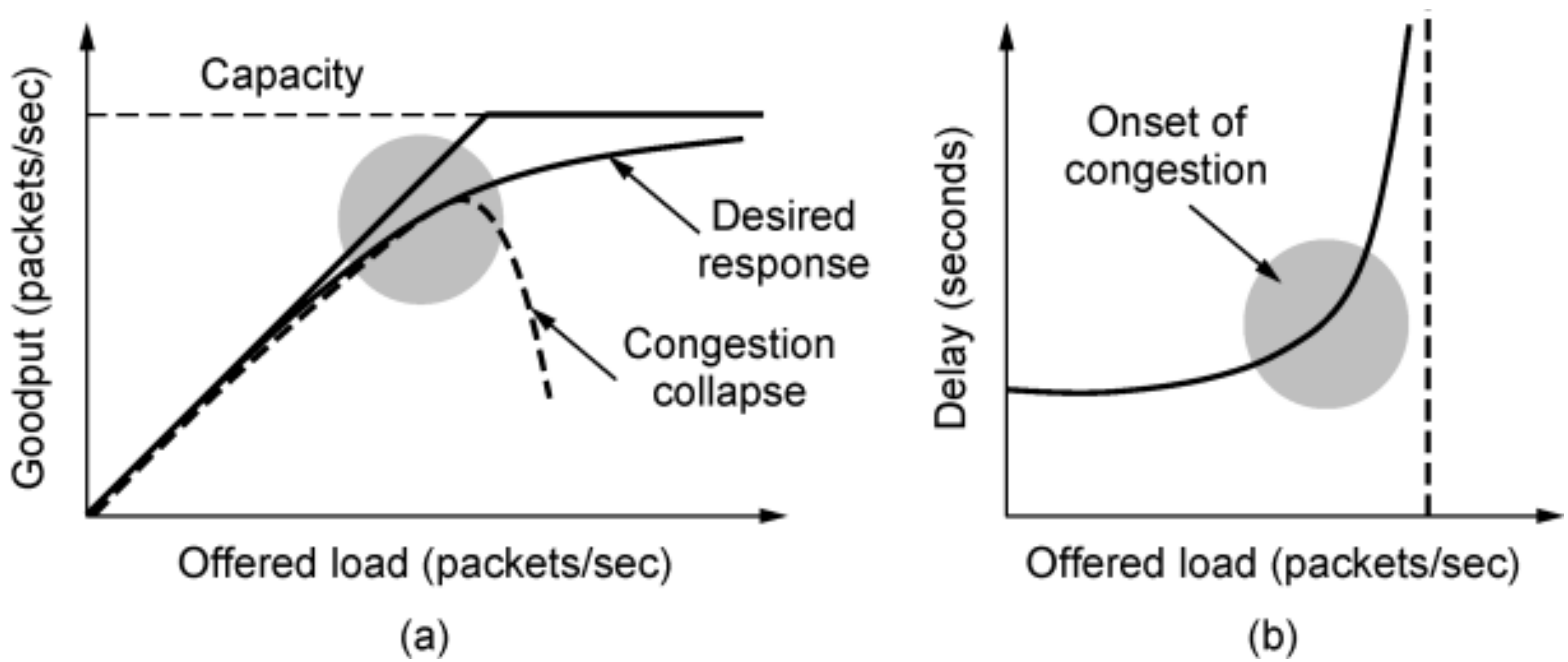
- TCP needs a dynamic algorithm
  - For each connection, maintain Smoothed Round-Trip Time (SRTT)
  - Use exponentially weighted moving average to adjust

$$\mathrm{SRTT} = \frac{7}{8}\mathrm{SRTT} + \frac{1}{8}\mathrm{RoundtripTime}$$

  - Jacobson: Maintain and update also roundtrip time variation
  - Karn: There are problems if the medium is unreliable. Only update estimates with non-retransmitted segments

TCP Congestion Control

- Goodput is a function of offered load



TCP Congestion Control

- Load with highest power represents an efficient load

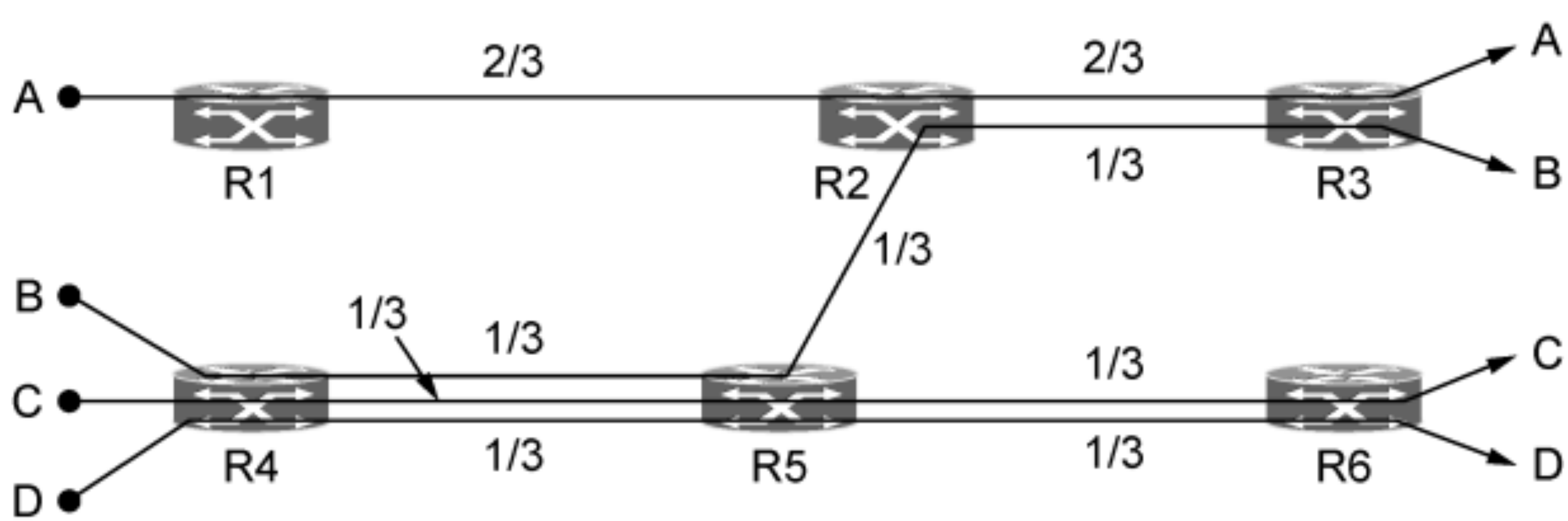
$$\text{power} = \frac{\text{load}}{\text{delay}}$$

TCP - Congestion Control

- Fairness
  - What does it mean to allocate a scarce resource (congested network connections) **fairly**
  - Complicated by flows sharing different links
  - **Max-Min fairness**
    - Bandwidth of one flow cannot be increased without decreasing the bandwidth of another flow with an allocation that is not larger

TCP - Congestion Control

- All routes have the same capacity 1
- Four flows: A, B, C, D
- B, C, D compete for the link between R4 and R5
- B and A compete for the link between R2 and R3



Max-min bandwidth allocation for four flows.

TCP - Congestion Control

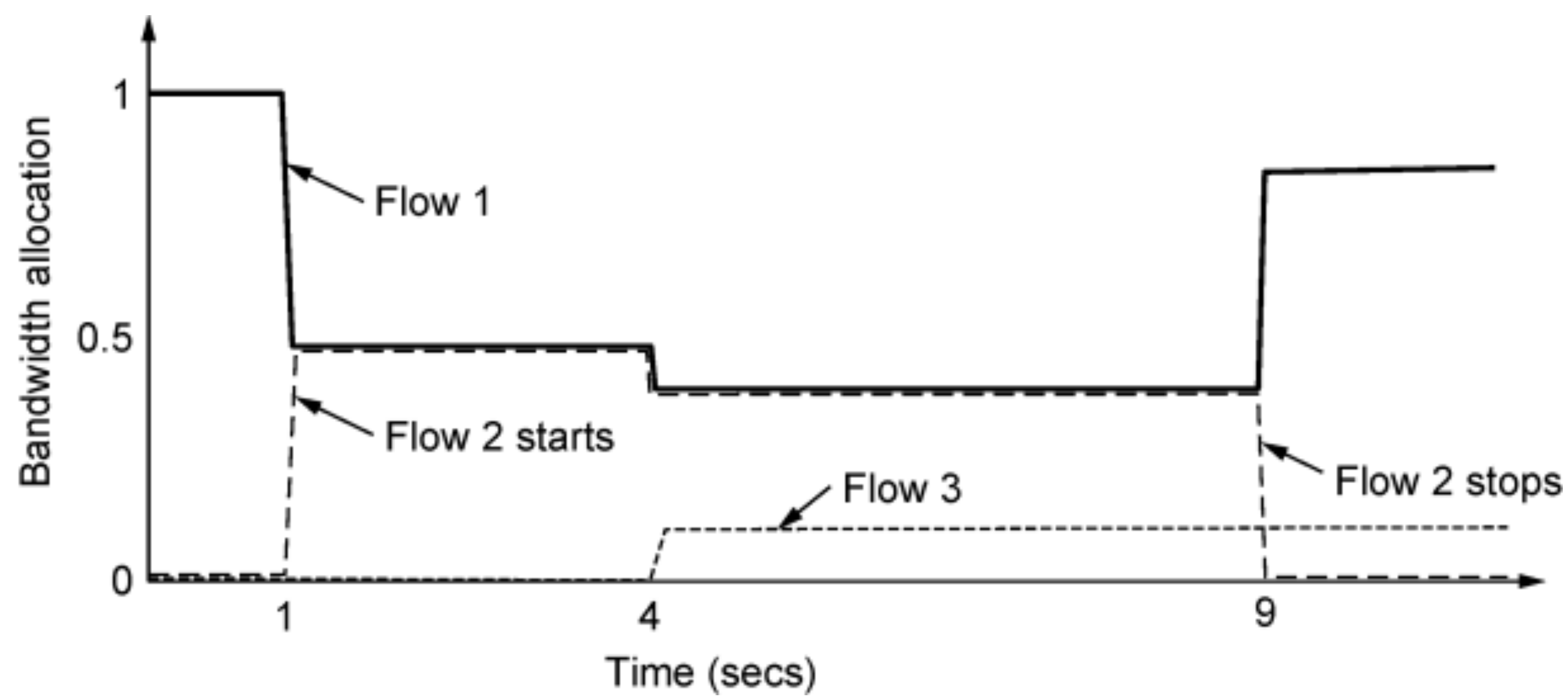
- Max Min fair allocations
  - Can be calculated with complete knowledge of net
    - Can start with flows at zero
    - Increase flows slowly until they are limited by a bottleneck

TCP - Congestion Control

- Max-Min fairness
  - Can be easily manipulated
    - BitTorrent (in P2P systems) opens many different connections
      - All of which get their share

TCP - Congestion Control

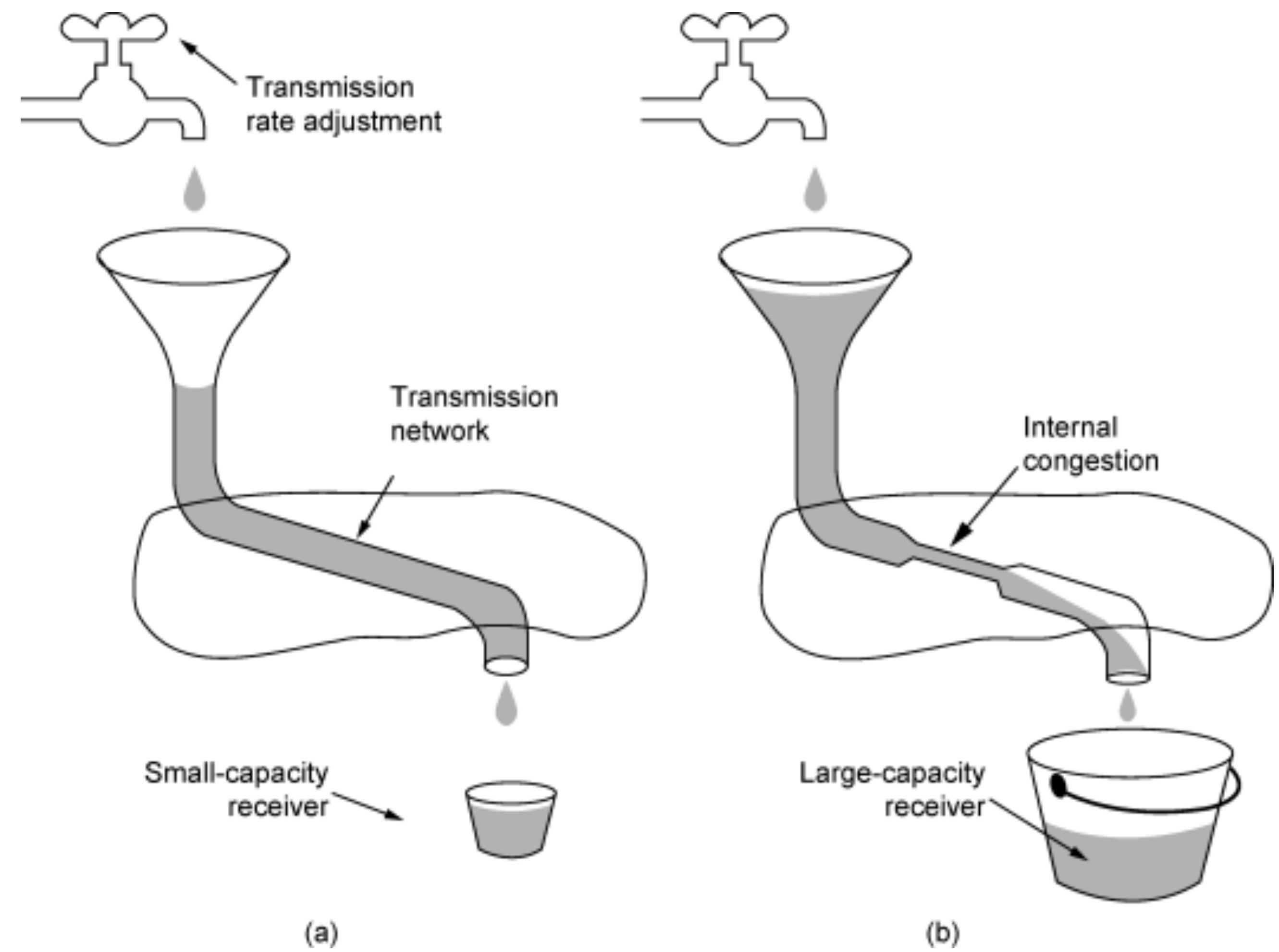
- Convergence
  - Good algorithms reach quickly a fair and efficient allocation of bandwidth



Changing bandwidth allocation over time.

TCP - Congestion Control

- Regulating the sending rate:
  - Sending rate is limited
    - By flow control if the receiver has insufficient buffering
    - By congestion, if there is insufficient bandwidth



(a) A fast network feeding a low-capacity receiver.
(b) A slow network feeding a high-capacity receiver.

TCP - Congestion Control

- eXplicit Congestion Protocol (Katabi, 2002)
  - Routers tell sources the rate at which they might send
- Explicit Congestion Notification with TCP
  - Routers set bits on packets that experience congestion to warn senders to slow down
- Fast TCP (Wei, 2006)
  - Measures round-trip delay as a signal
- Compound TCP (Windows)
  - Uses packet loss and end-to-end delay

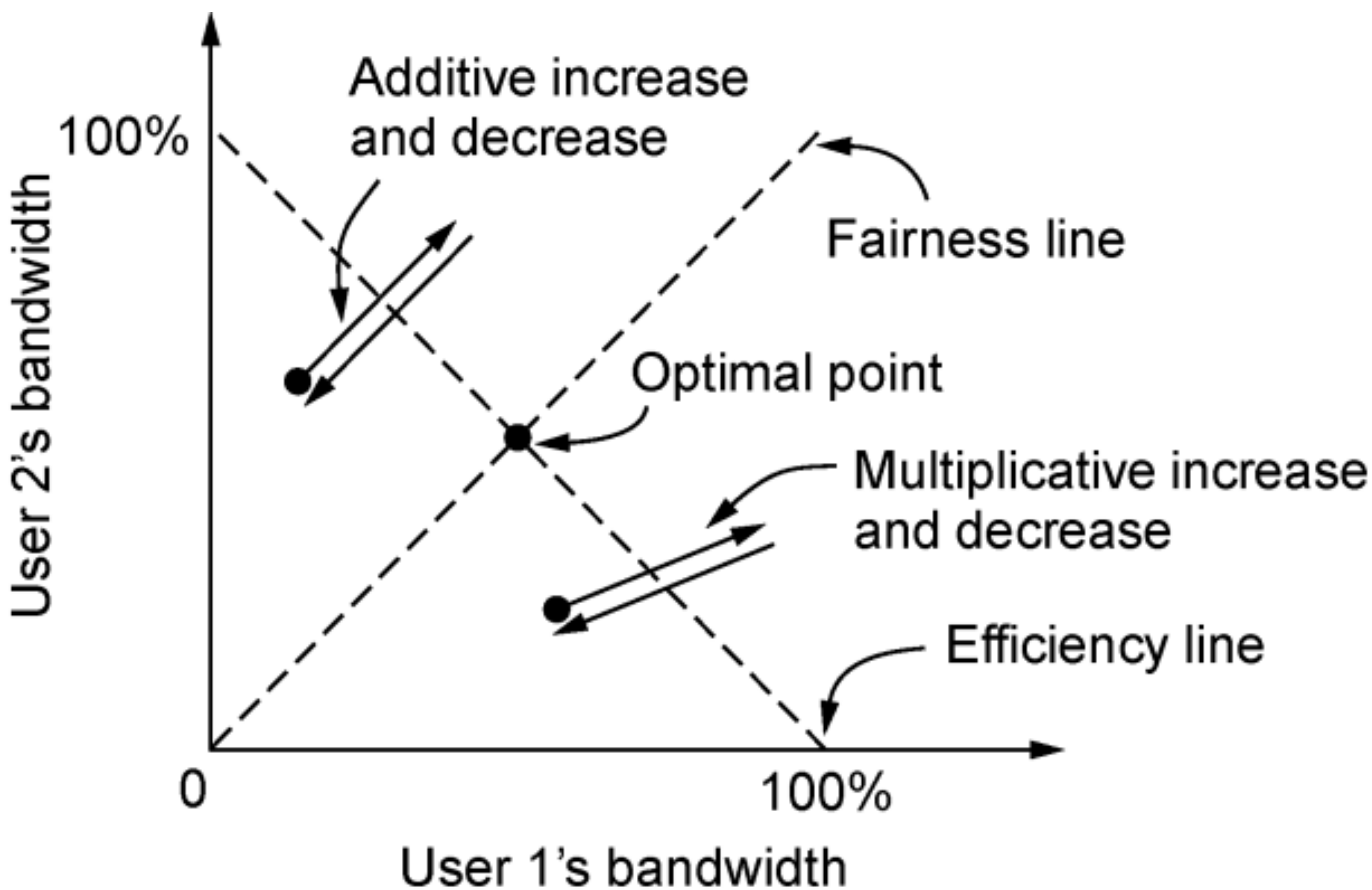
TCP - Congestion Control

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
Compound TCP	Packet loss & end-to-end	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

TCP - Congestion Control

- Control Laws
  - Congestion signal tells when senders need to change their rate
  - Control laws specify **how** they adjust their rates

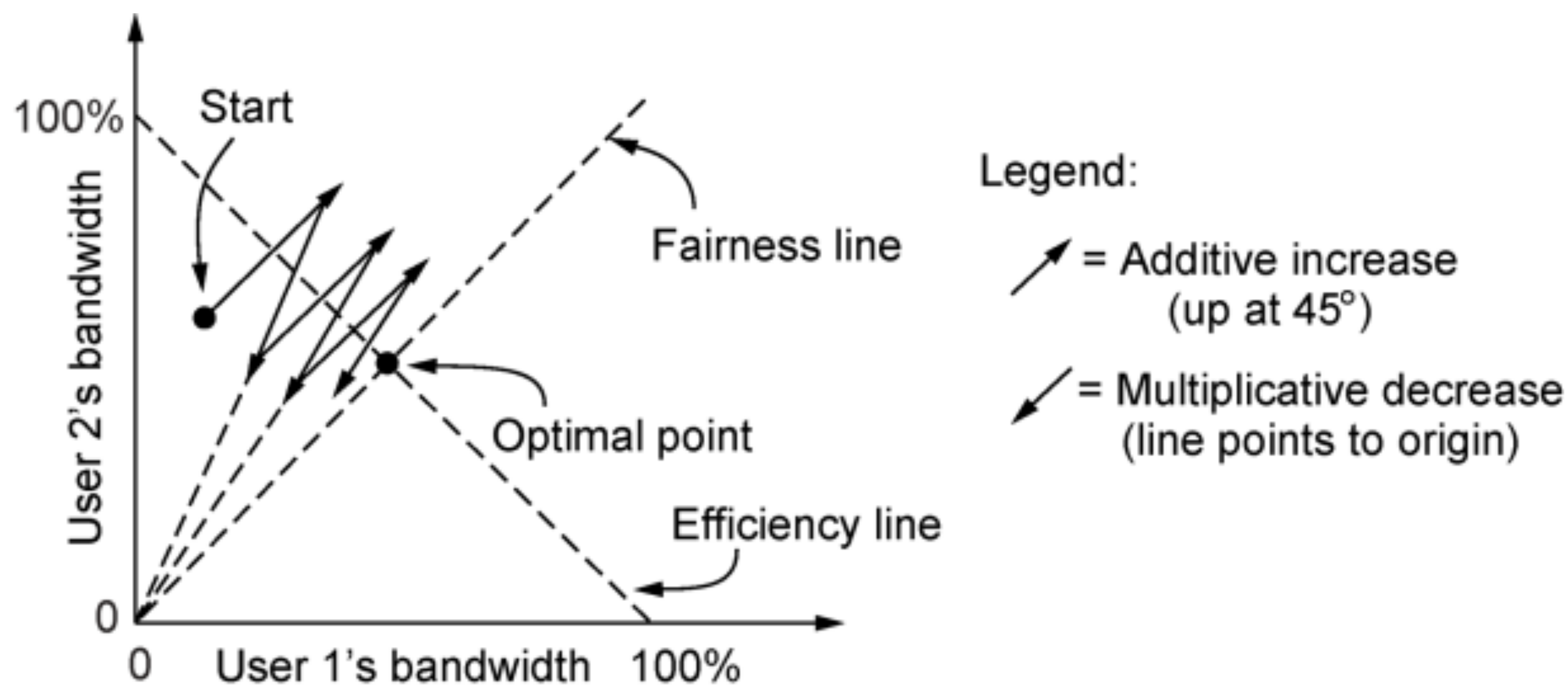
TCP - Congestion Control



Additive and multiplicative bandwidth adjustments.

TCP - Congestion Control

- Additive Increase — Multiplicative Decrease (AIMD) law



Additive Increase Multiplicative Decrease (AIMD) control law.

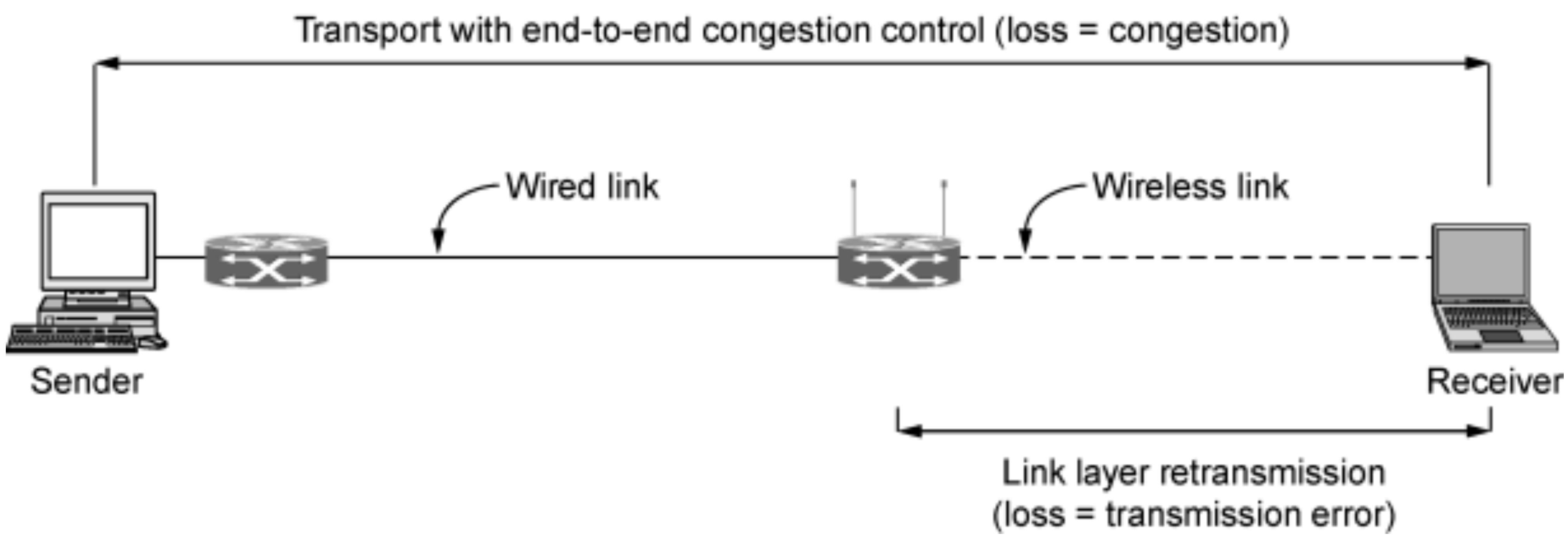
TCP - Congestion Control

- Competition with other protocols
  - TCP is the dominant flow protocol with congestion control
  - Other streaming protocols are **TCP-friendly** if and only if they are fair to TCP

TCP - Congestion Control

- TCP over wireless links
  - Loss rates of over 10% are common for wireless frames
  - Congestion control schemes that use packet loss as indicator
    - Will throttle TCP over wireless unnecessary
  - Can:
    - Use masking: retransmission of wireless frames
    - Use different timescales (tiny for layer 2, large for layer 4)

TCP - Congestion Control



Congestion control over a path with a wireless link.

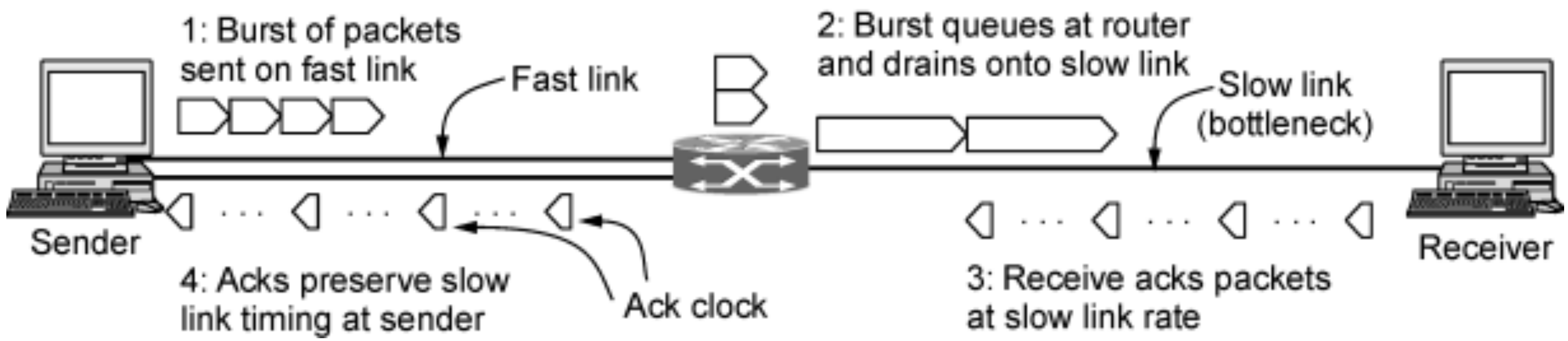
TCP - Congestion Control

- TCP Congestion Control
  - Congestion Window — Number of bytes that a sender may have in the network at any time
  - Different from the flow control window
  - Uses AIMD
  - Developed by van Jacobson
    - Based on congestion collapse in the early internet (1986)

TCP - Congestion Control

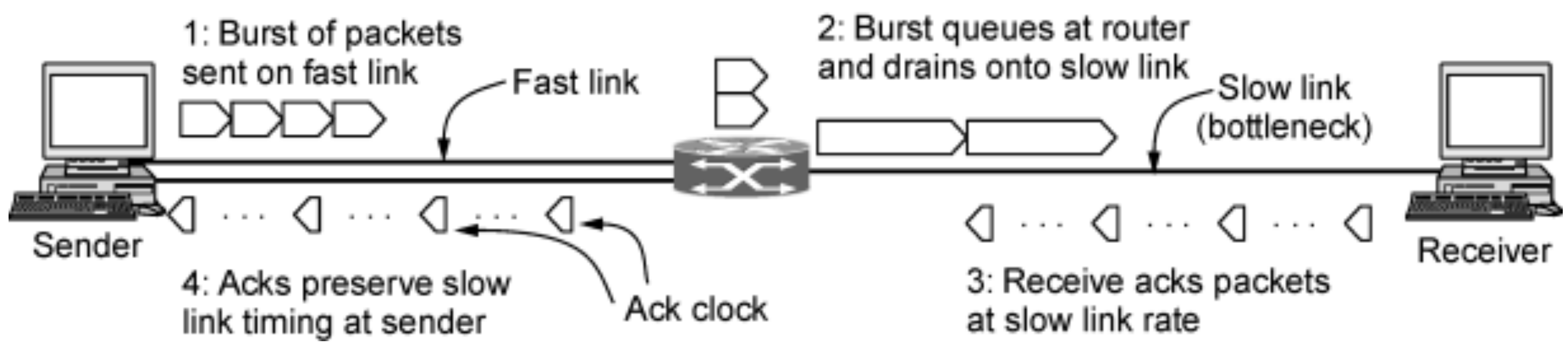
- All TCP algorithms assume that lost packets are caused by congestion and monitor time-outs
  - Good timers are essential

TCP - Congestion Control



A burst of packets from a sender and the returning ack clock.

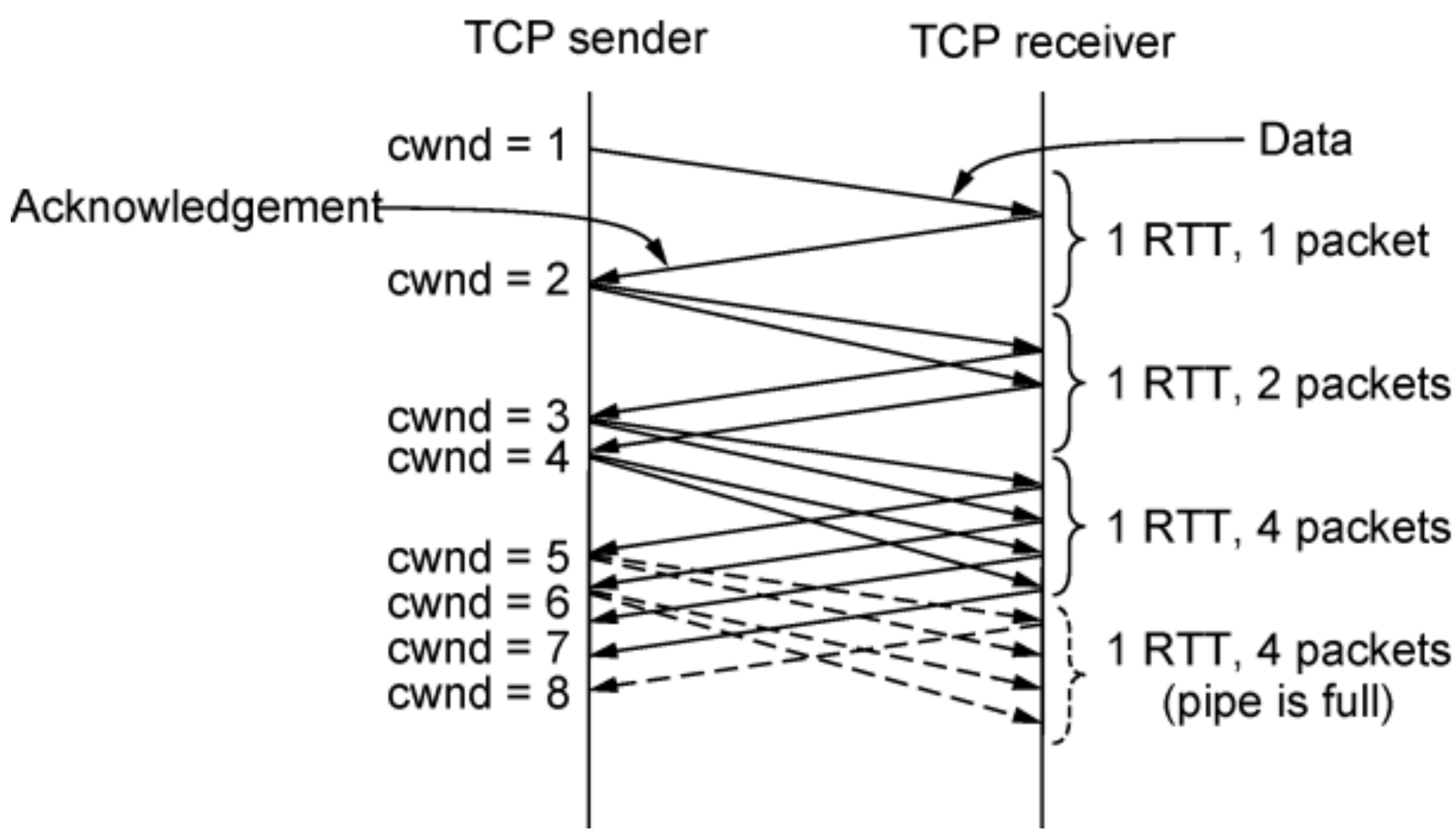
TCP - Congestion Control



- Acks timing gives the rate at which the slow link can digest packages

TCP - Congestion Control

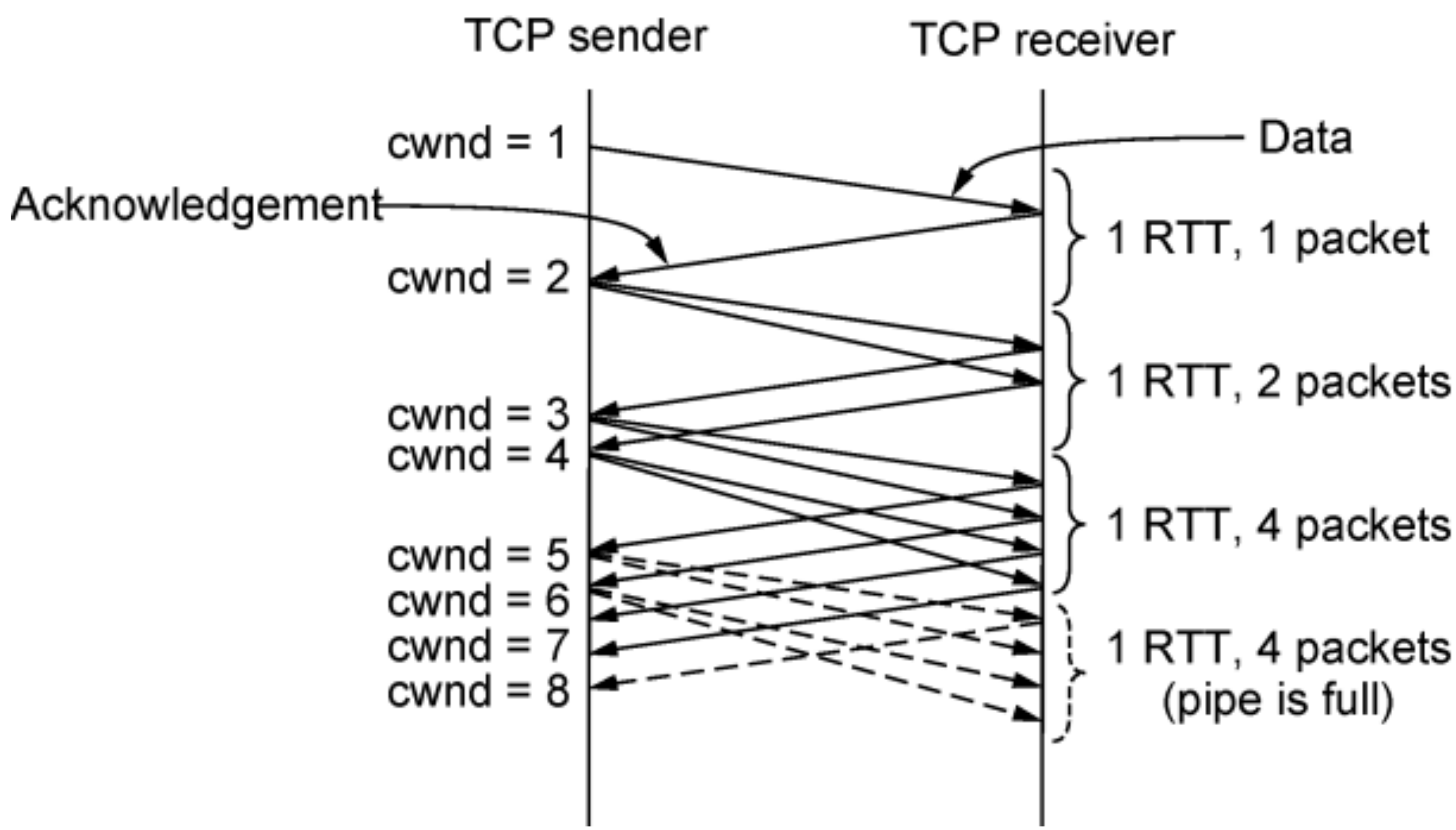
- Slow Start algorithm
  - Exponential growth of segments sent per round-trip time.



TCP - Congestion Control

- Slow Start Algorithm
  - Pretty soon, this will fill up a network connection
  - Algorithm defines a **slow start threshold**
    - Initially very high
    - Get’s reduced whenever there is congestion
  - Algorithm switches from exponential to additive increase once the slow start threshold is crossed

TCP - Congestion Control

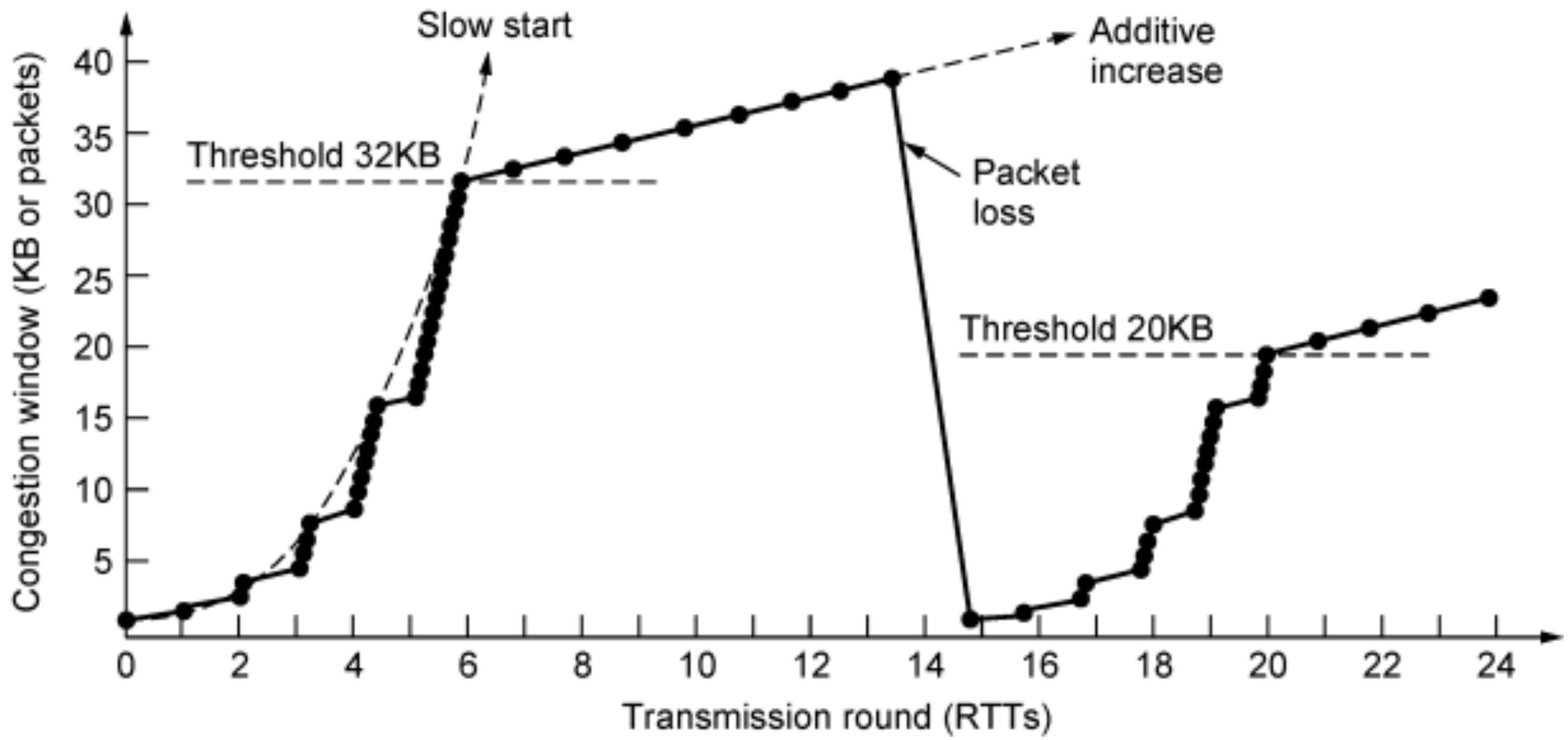


Additive increase from an init
segment

TCP - Congestion Control

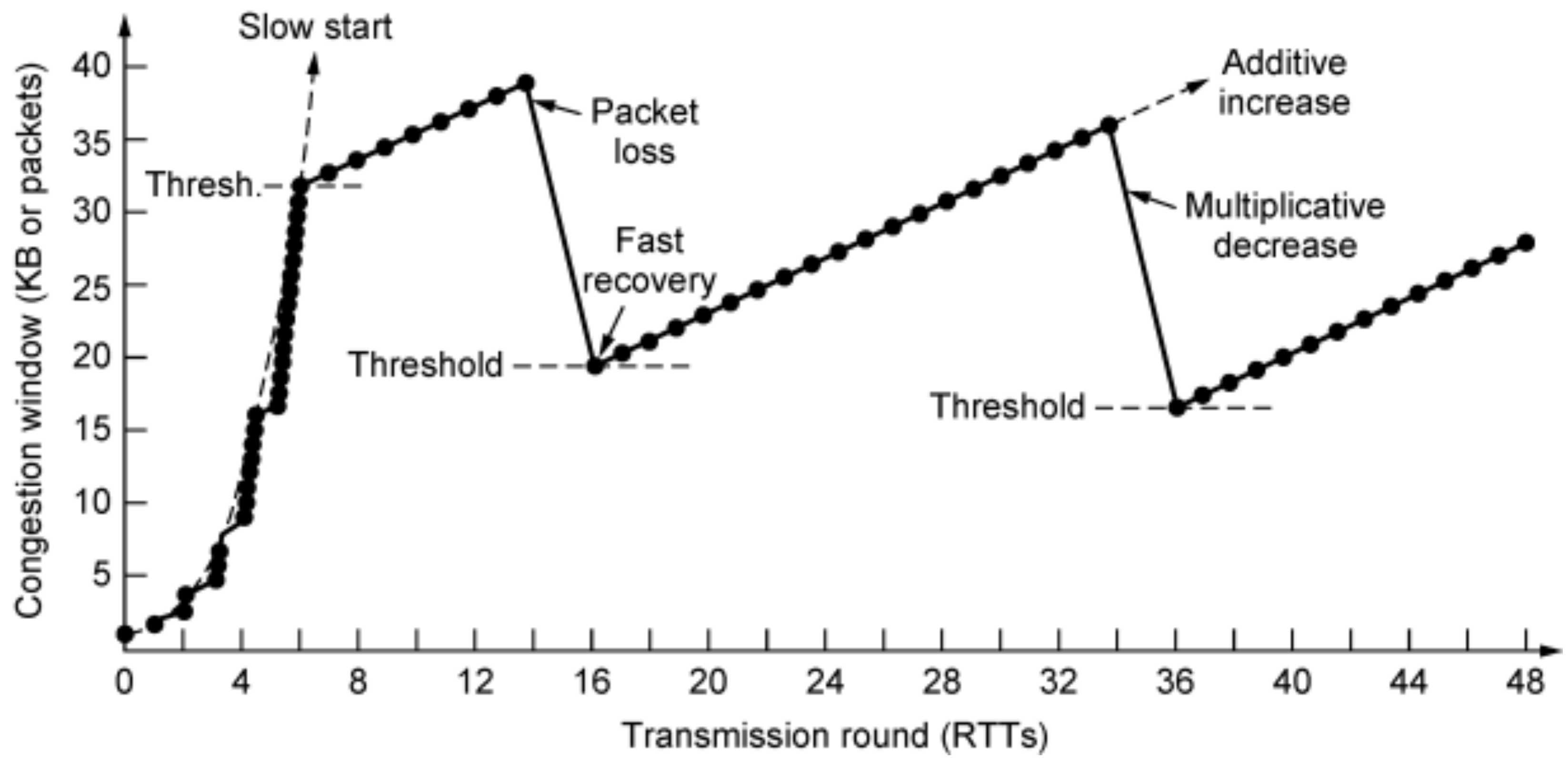
- Duplicate acknowledgments
  - Acks with the same byte acknowledged
    - Likely that another packet has arrived out of order
    - **Fast retransmission**:
      - Retransmit after receiving three duplicate acks

TCP - Congestion Control



Slow start followed by additive increase in TCP Tahoe.

TCP - Congestion Control



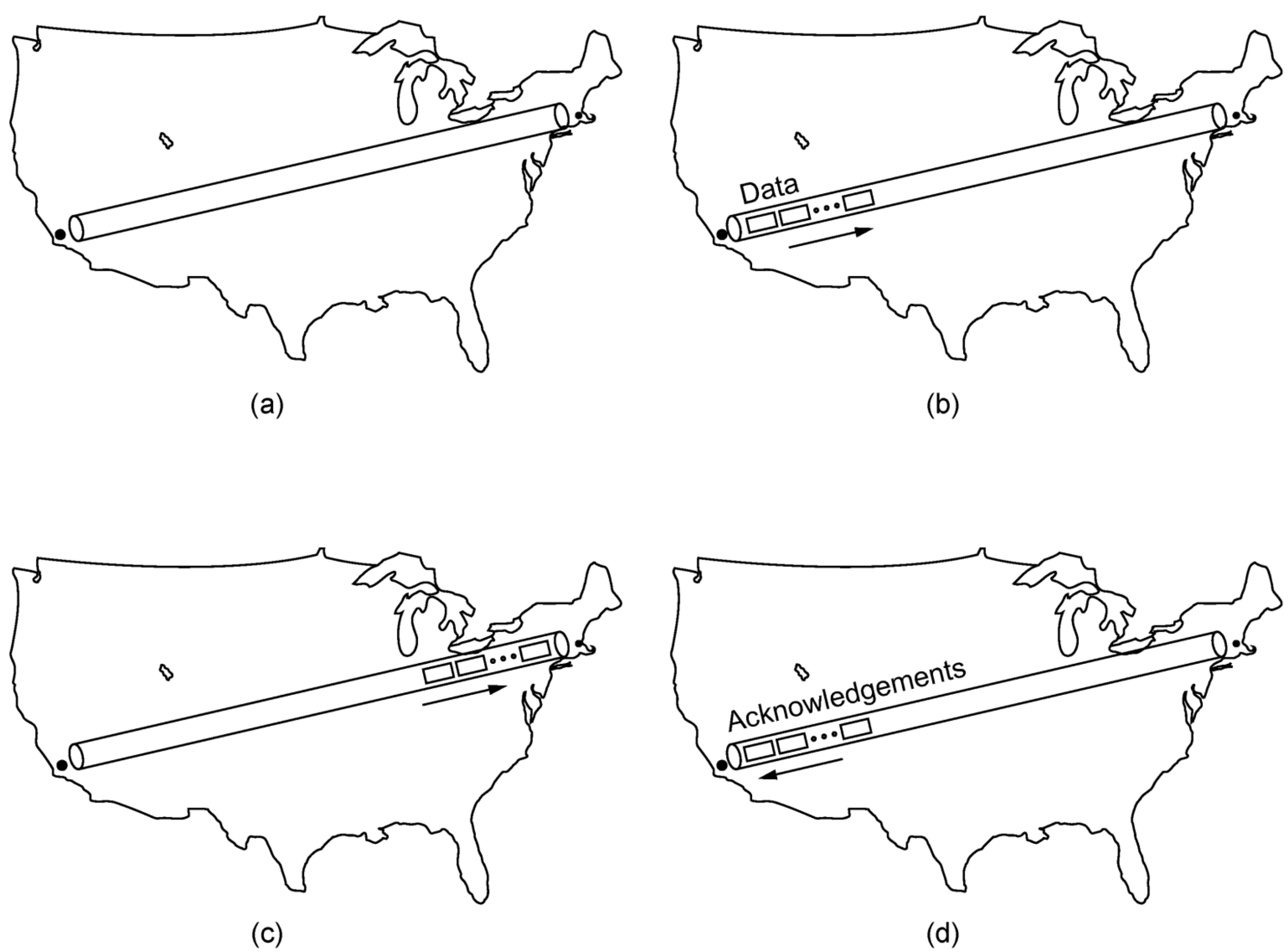
Fast recovery and the sawtooth pattern of TCP Reno.

Long Fat Networks

- Long distance high bandwidth does not lend itself to existing protocols
  - 32b sequence number
    - 56 kbps leased lines between routers (original internet)
      - takes 1 week to cycle through sequence numbers
    - 10 Mbps:
      - takes 57 minutes to wrap around
    - 1 Gbps:
      - takes 34 seconds
      - less than 120 second maximum packet lifetime

Long Fat Networks

- Flow control window is too small



The state of transmitting 1 Mbit from San Diego to Boston. (a) At t = 0. (b) After 500 μsec. (c) After 20 msec. (d) After 40 msec.

Long Fat Networks

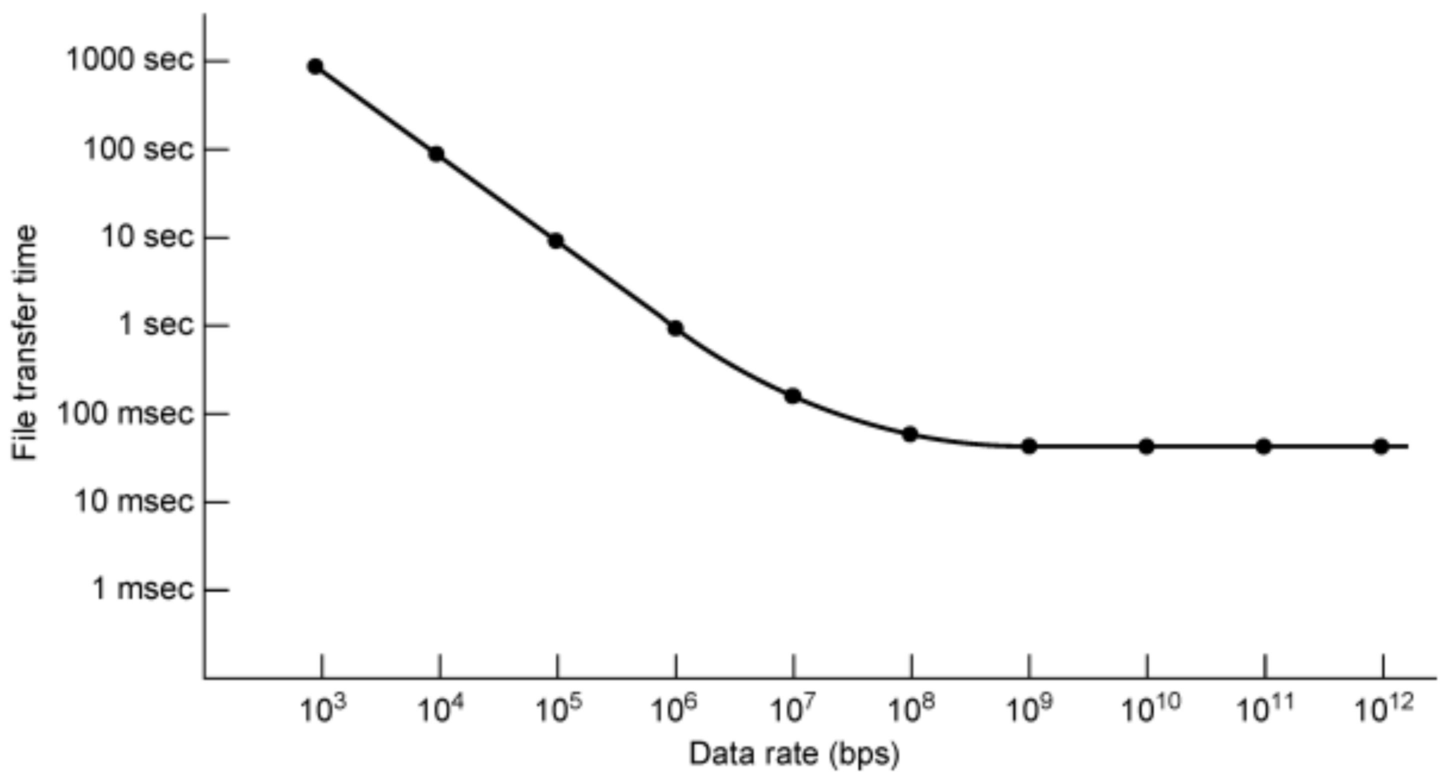
- Bandwidth Delay Product
  - Useful measure for analyzing network performance
  - Represents the capacity of the pipe
    - 1Gbps link between San Diego and Boston
    - Bandwidth delay product is 40 million bits
    - Burst of 0.5MB only fills 1.25% of capacity

Long Fat Networks

- Simple retransmission schemes
  - When sender discovers that a segment has been lost
  - Needs to resend that segment and all previous ones
  - Since packets are now big, bit loss

Long Fat Networks

- Long fat networks are bound by delay
  - Remote procedure call protocols e.g. will function poorly



Time to transfer and acknowledge a 1-Mbit file over a 4000-km line.

Long Fat Networks

- Communication speeds improve faster than computing speeds
  - Need protocols that are designed for speed
    - Not for bandwidth optimization