Computer Networks

The Network Layer TCP/IP

Adrian Sergiu DARABANT

Lecture 8

IP Datagram

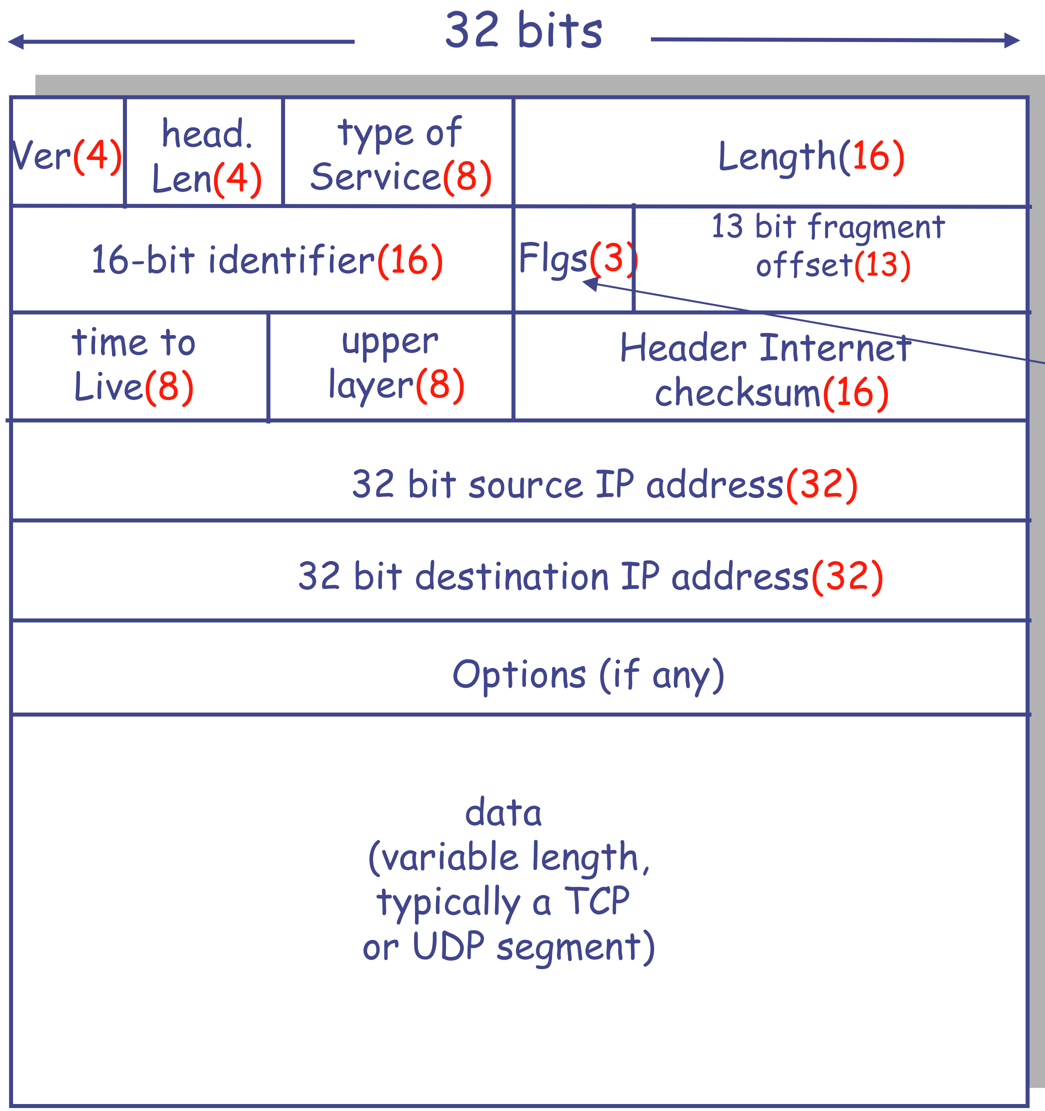
IP protocol version number

header length (bytes)

"type" of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to



total datagram length (bytes)

for fragmentation/ reassembly

DF+MF

E.g. timestamp, record route taken, specify list of routers to visit.

<u>how much overhead with IP?</u>

- 20 bytes of IP
- \+ transp layer overhead

Datagram: from source to destination

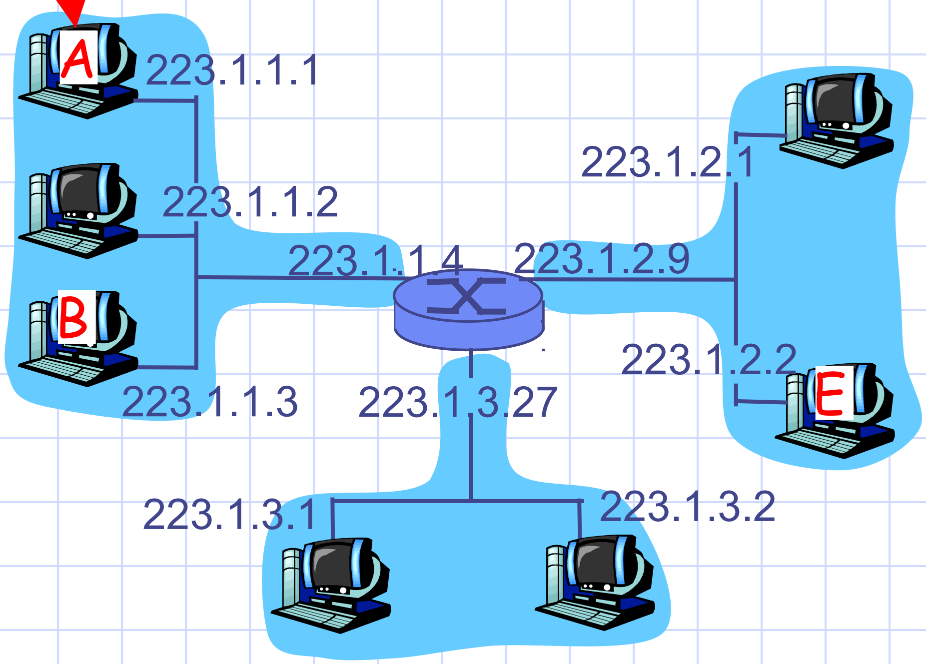
forwarding table in A

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2

IP datagram:

misc fields	source IP addr	dest IP addr	data

- datagram remains unchanged, as it travels source to destination
- addr fields of interest here

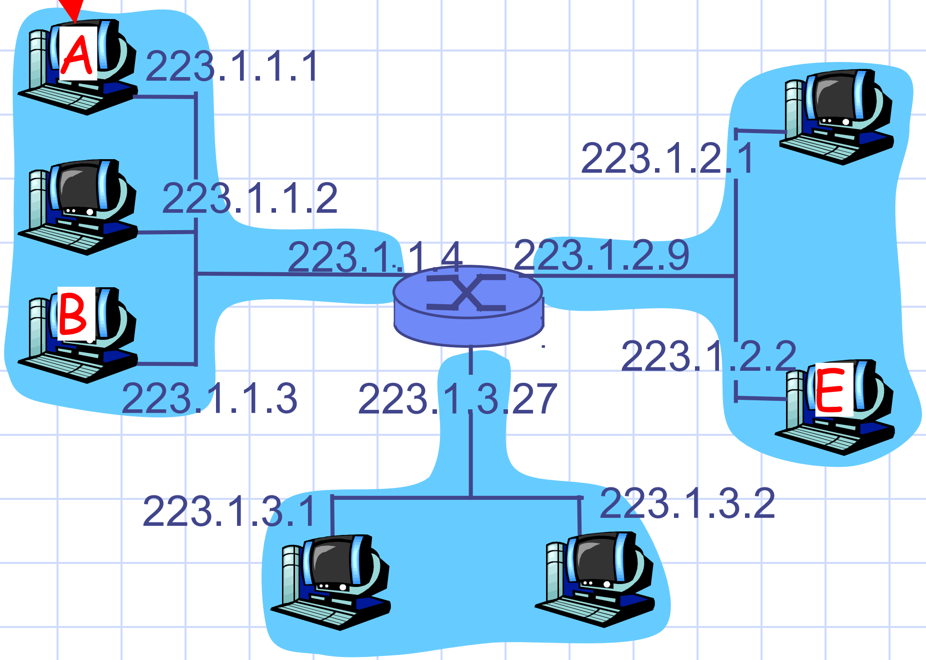


Datagram: from source to destination

misc fields	223.1.1.1	223.1.1.3	data

forwarding table in A

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



Starting at A, send IP datagram addressed to B:

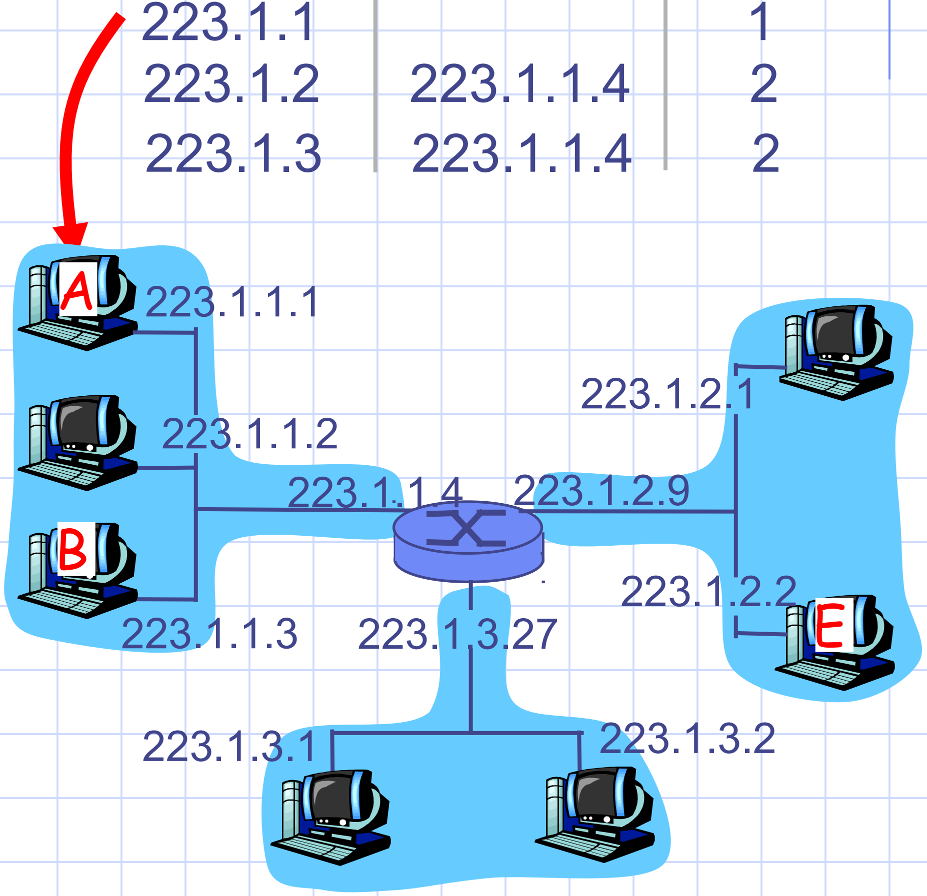
- look up net. address of B in forwarding table
- find B is on same net. as A
- link layer will send datagram directly to B inside link-layer frame
  - B and A are directly connected

Datagram: from source to destination

misc fields	223.1.1.1	223.1.2.3	data

forwarding table in A

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



Starting at A, dest. E:

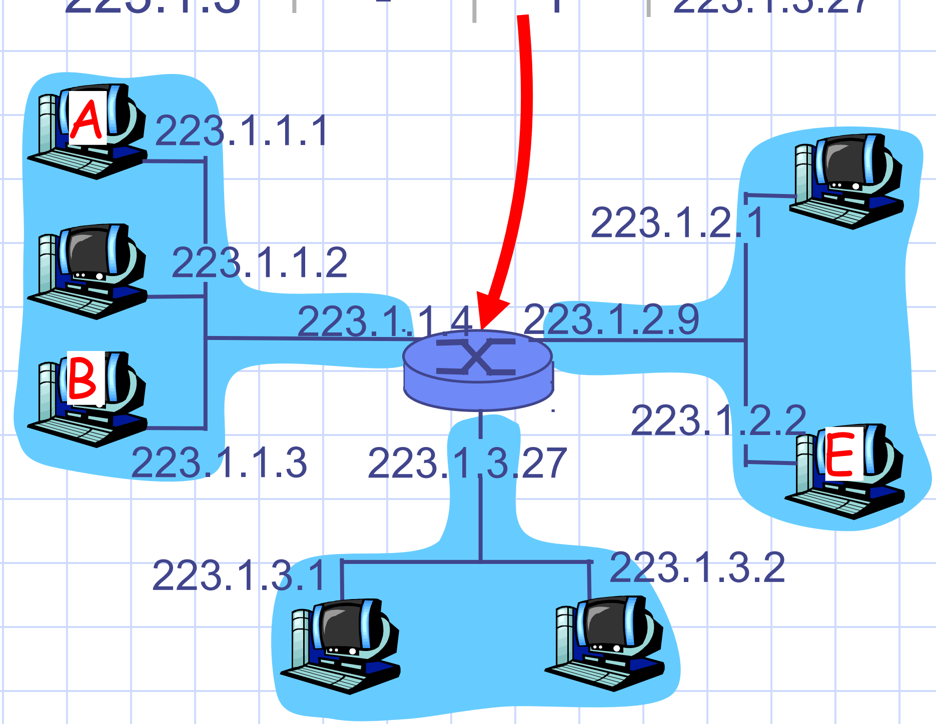
- look up network address of E in forwarding table
- E on *different* network
  - A, E not directly attached
- routing table: next hop router to E is 223.1.1.4
- link layer sends datagram to router 223.1.1.4 inside link-layer frame
- datagram arrives at 223.1.1.4

Datagram: from source to destination

misc fields	223.1.1.1	223.1.2.3	data

forwarding table in router

Dest. Net.	router	Nhops	interface
223.1.1	-	1	223.1.1.4
223.1.2	-	1	223.1.2.9
223.1.3	-	1	223.1.3.27

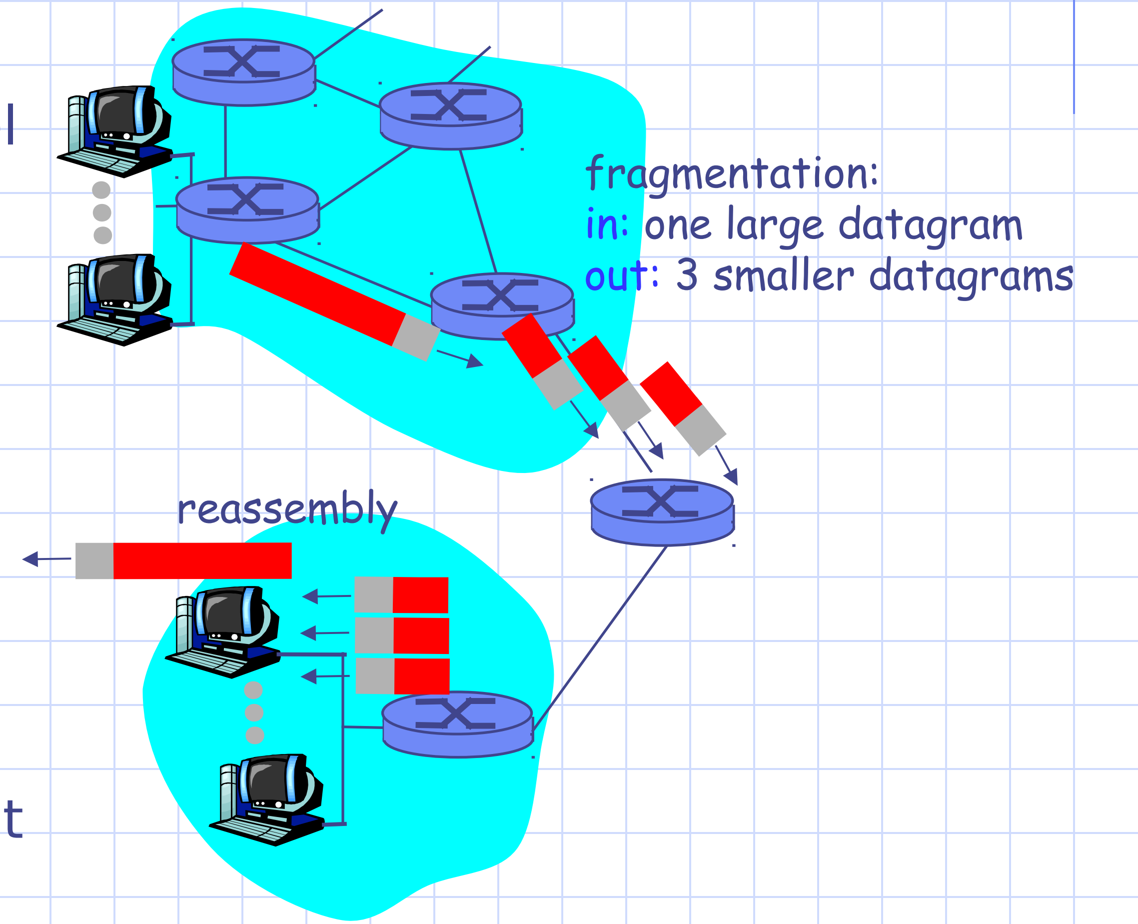


Arriving at 223.1.4, destined for 223.1.2.2

- look up network address of E in router's forwarding table
- E on *same* network as router's interface 223.1.2.9
  - router, E directly attached
- link layer sends datagram to 223.1.2.2 inside link-layer frame via interface 223.1.2.9

Fragmentation/Reassembly

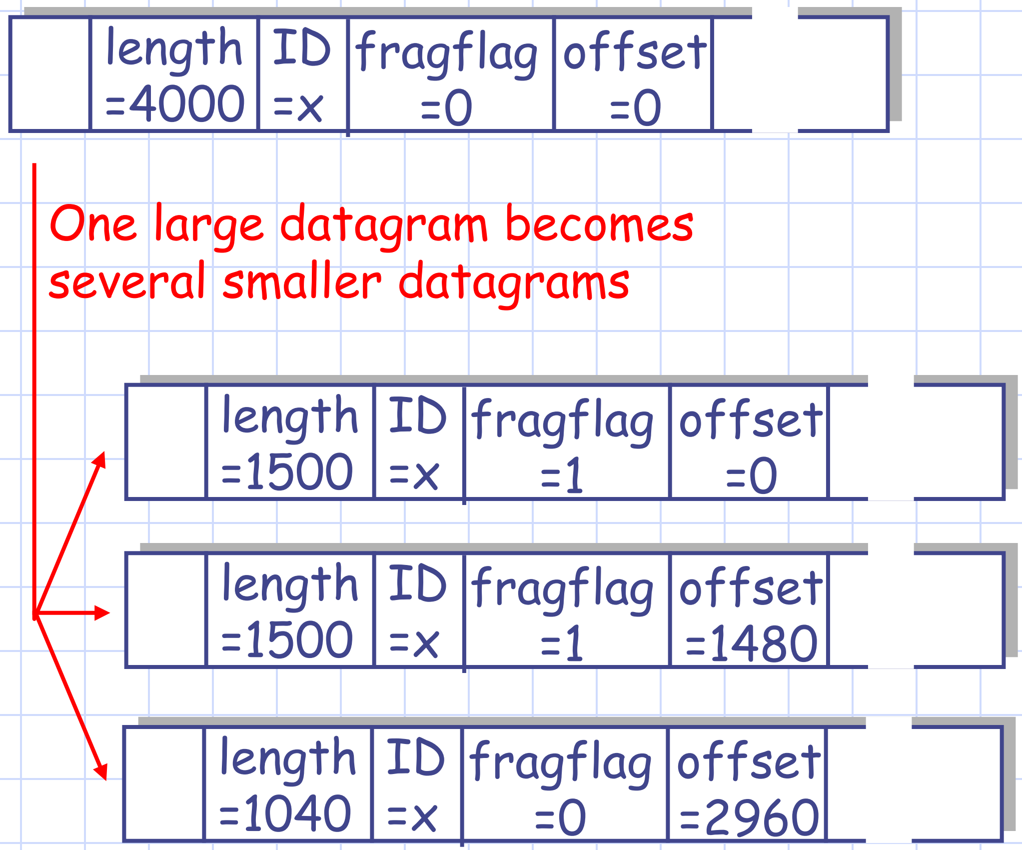
- network links have MTU (max.transfer size) - largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related



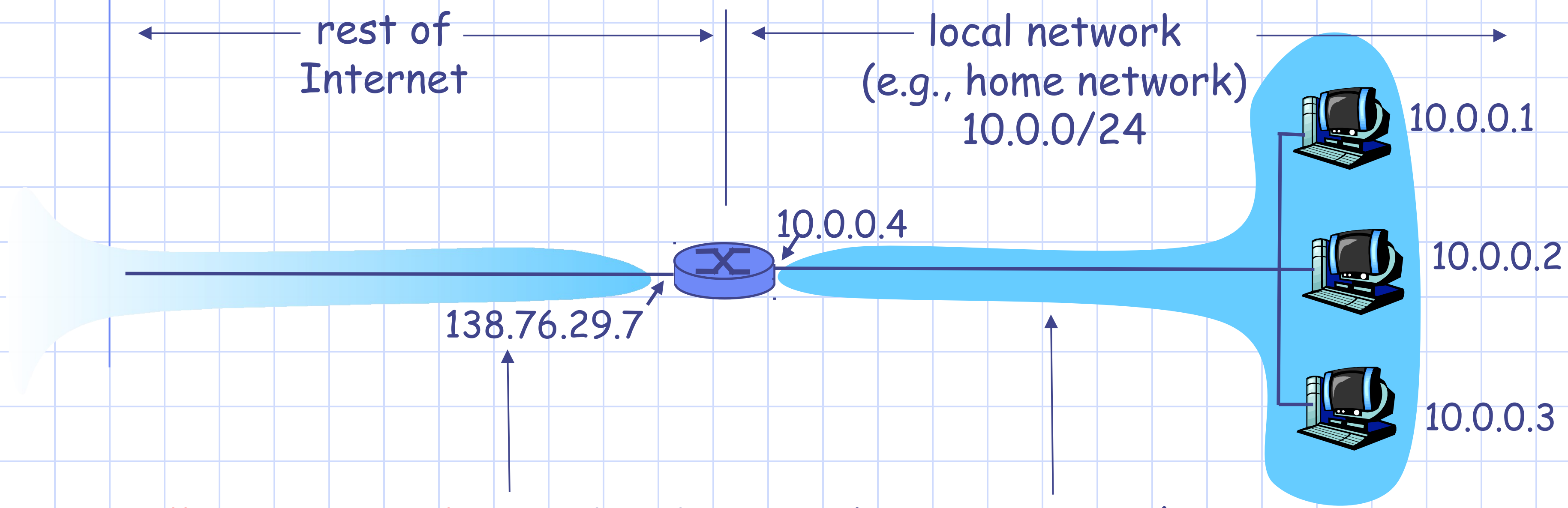
Fragmentation/Reassembly

Example

- 4000 byte datagram
- MTU = 1500 bytes



NAT – Network Address Translation



All datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

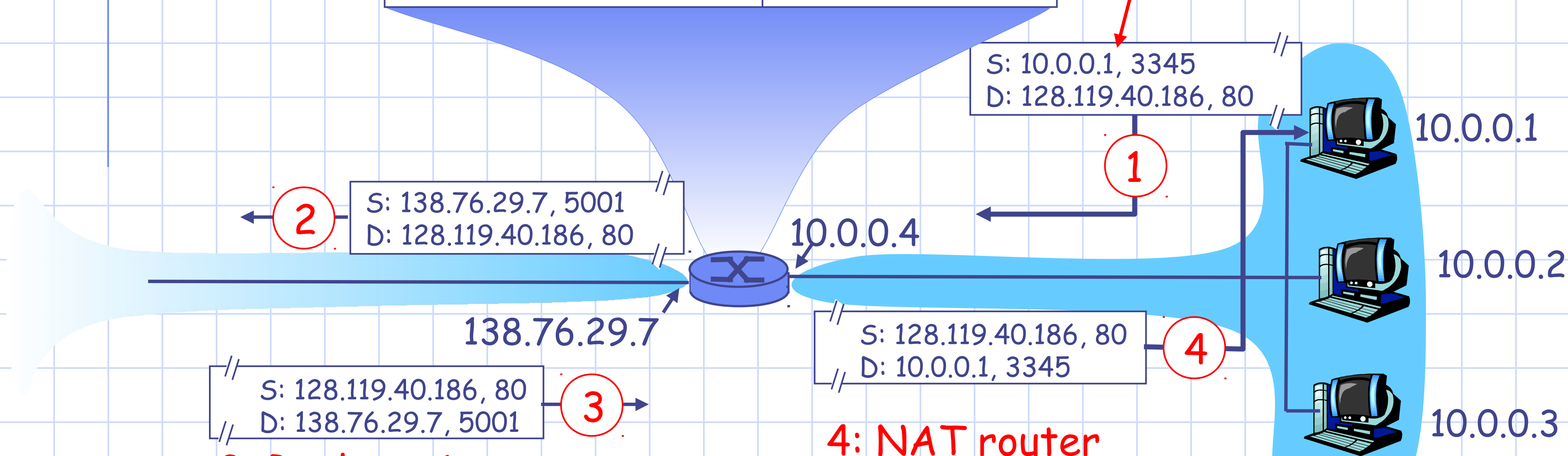
NAT – Network Address Translation

- Motivation: local network uses just one IP address as far as outside word is concerned:
  - no need to be allocated range of addresses from ISP: - just one IP address is used for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).

NAT – Network Address Translation

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40, 80



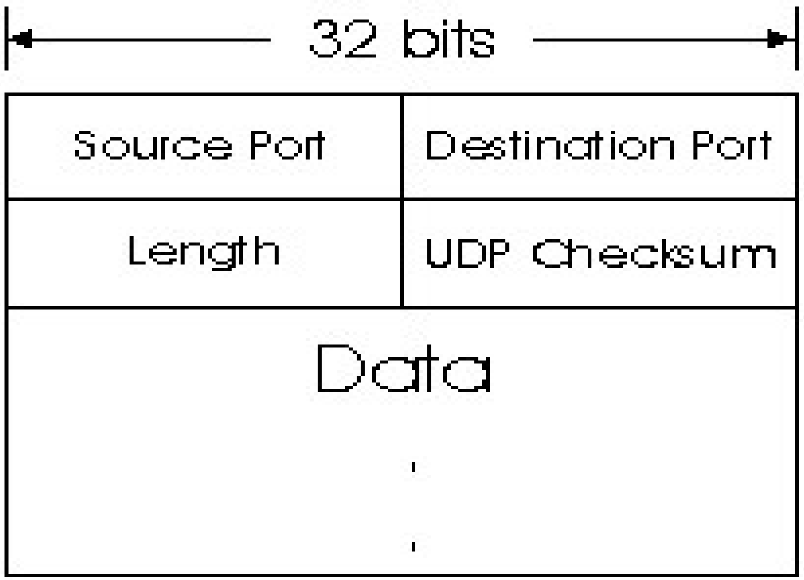
3: Reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

NAT – Network Address Translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - address shortage should instead be

UDP



how much overhead with UDP?

- 20 bytes of IP
- 8 bytes of UDP
- = 28 bytes + app layer overhead

Checksum – for the entire datagram (header + data)

Length >=8 – entire datagram

ICMP

- Used by hosts, routers, gateways to communication network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- Network-layer “above” IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

UDP Rules

Unreliable – When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission, or timeout.

Not ordered – If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.

Lightweight – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

Datagrams – Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.

No congestion control – UDP itself does not avoid congestion, and it's possible for high bandwidth applications to trigger congestion collapse, unless they implement

ICMP

0	8	16	31
Type	Code	Checksum	
ICMP data (depending on the type of message)			

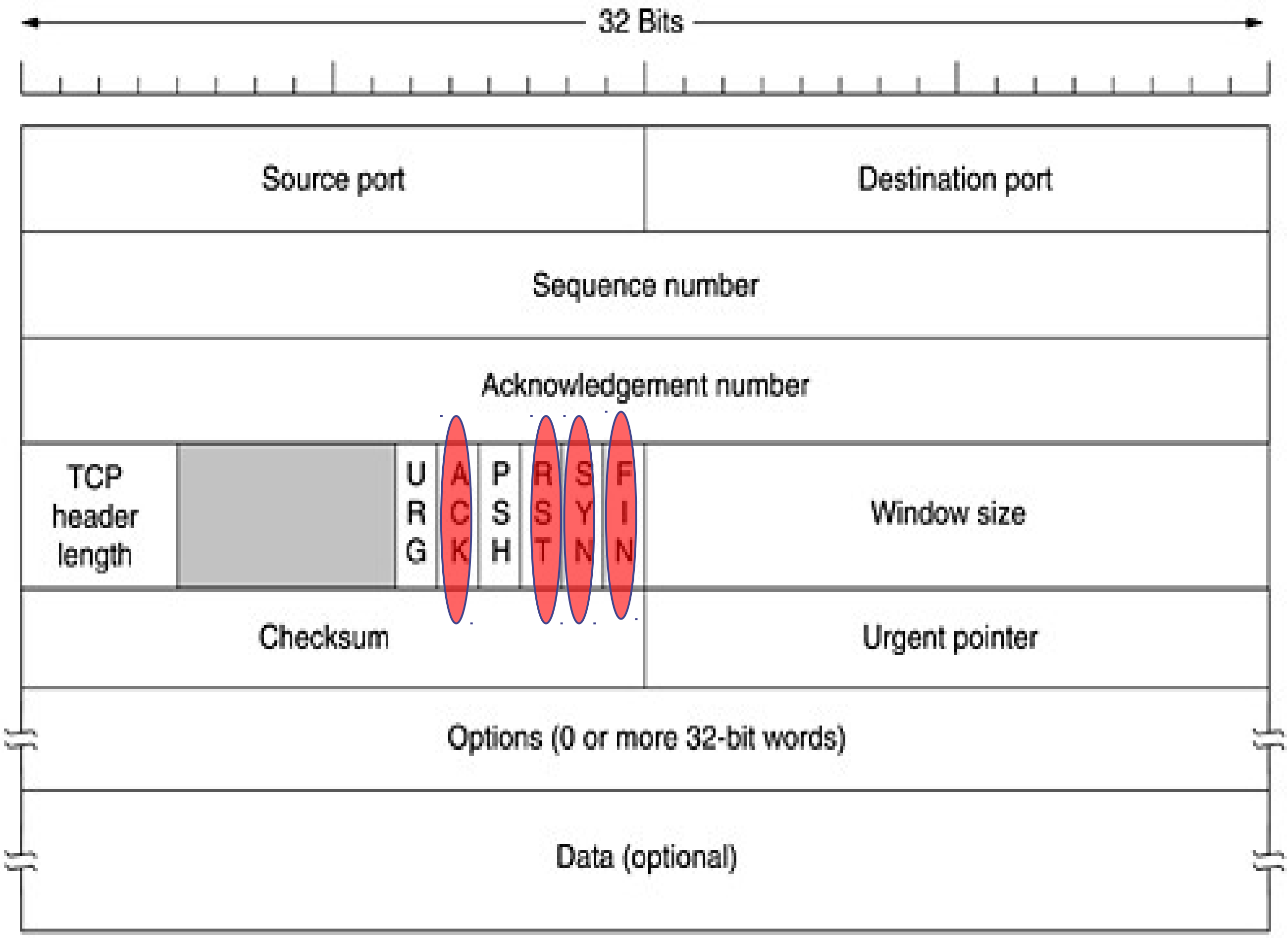
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown

Type	Code	description
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Network diagnostic

- **Ping** – uses ICMP **Echo** and **Reply** to determine if a host is “up”
- **Traceroute** – determine the path (as routers) from a source host to a destination host using UDP(usually).

TCP Datagrams



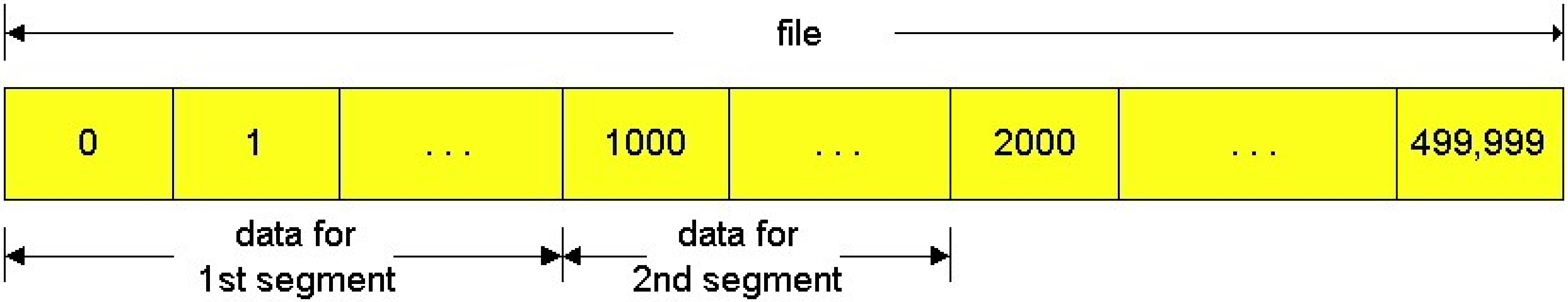
how much overhead with TCP?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

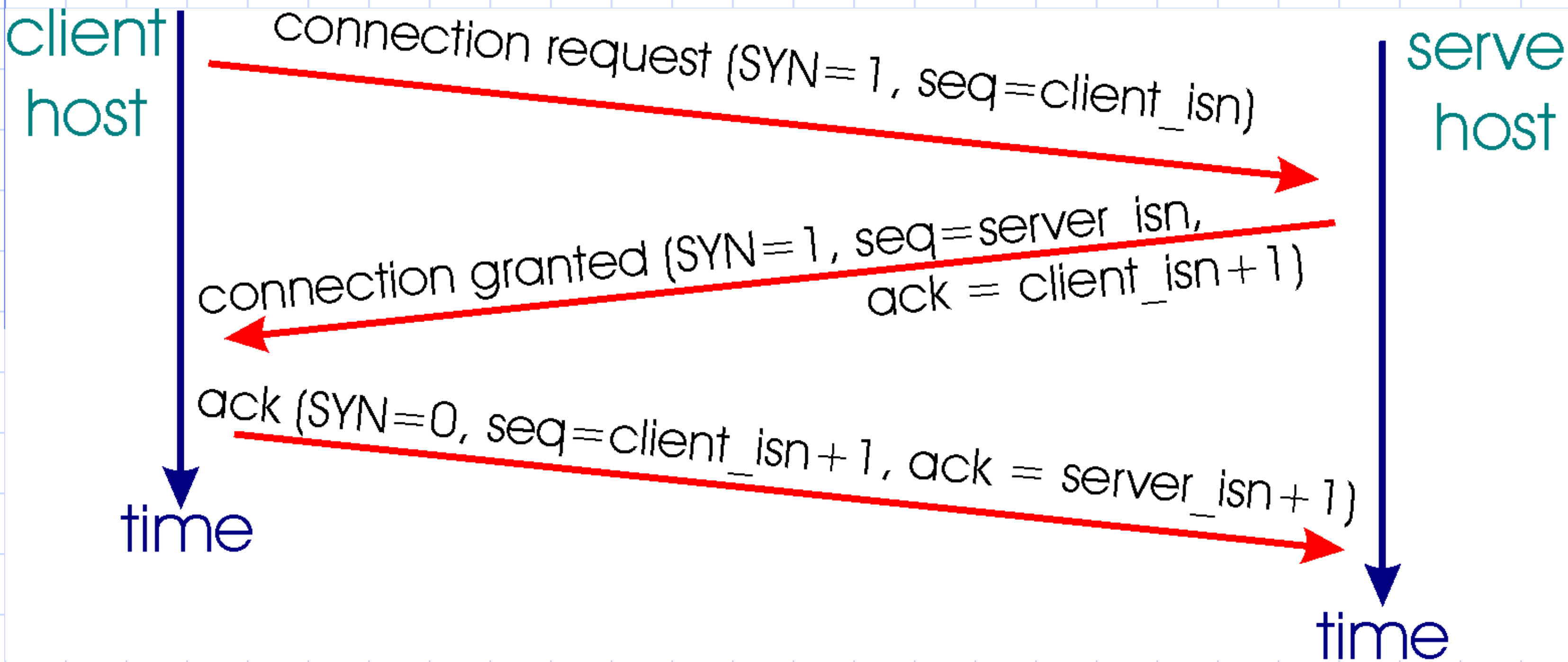
TCP - Data Transfer

- **Ordered data transfer** — the destination host rearranges according to sequence number
- **Retransmission of lost packets** — any cumulative stream not acknowledged is retransmitted
- **Error-free data transfer**
- **Flow control** — limits the rate a sender transfers data to guarantee reliable delivery. The receiver continually hints the sender on how much data can be received (controlled by the sliding window). When the receiving host's buffer fills, the next acknowledgment contains a 0 in the window size, to stop transfer and allow the data in the buffer to be processed.
- **Congestion control**

TCP Segments



TCP Open – 3-way handshake



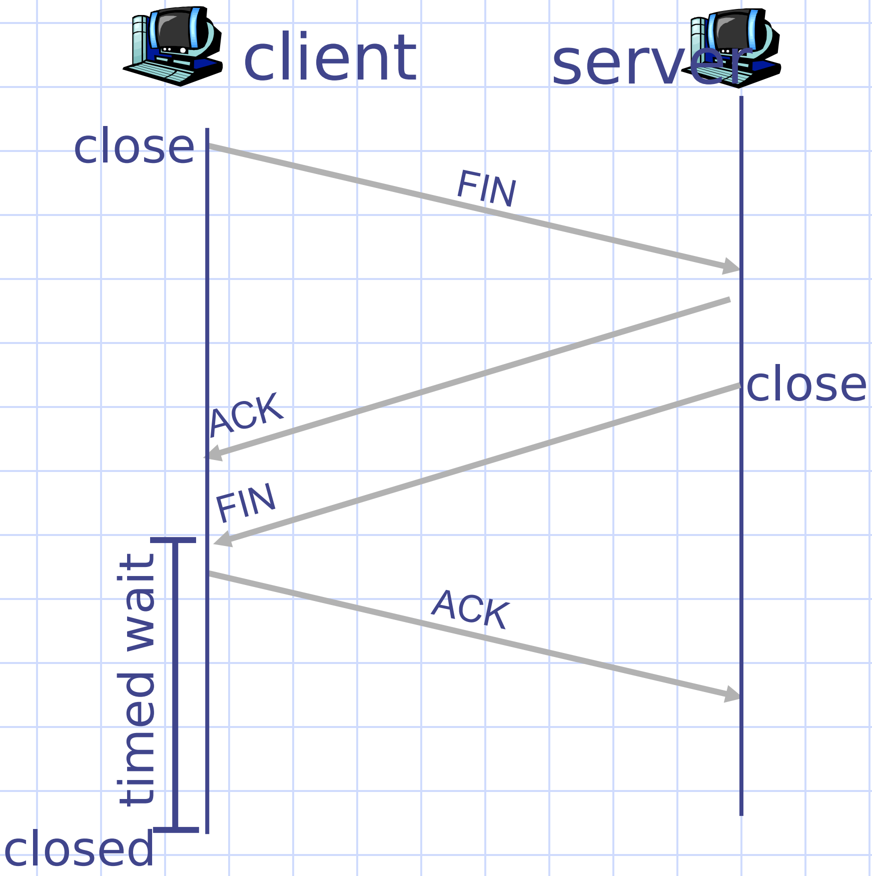
TCP Connection Teardown

Closing a connection:

client closes socket: `clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

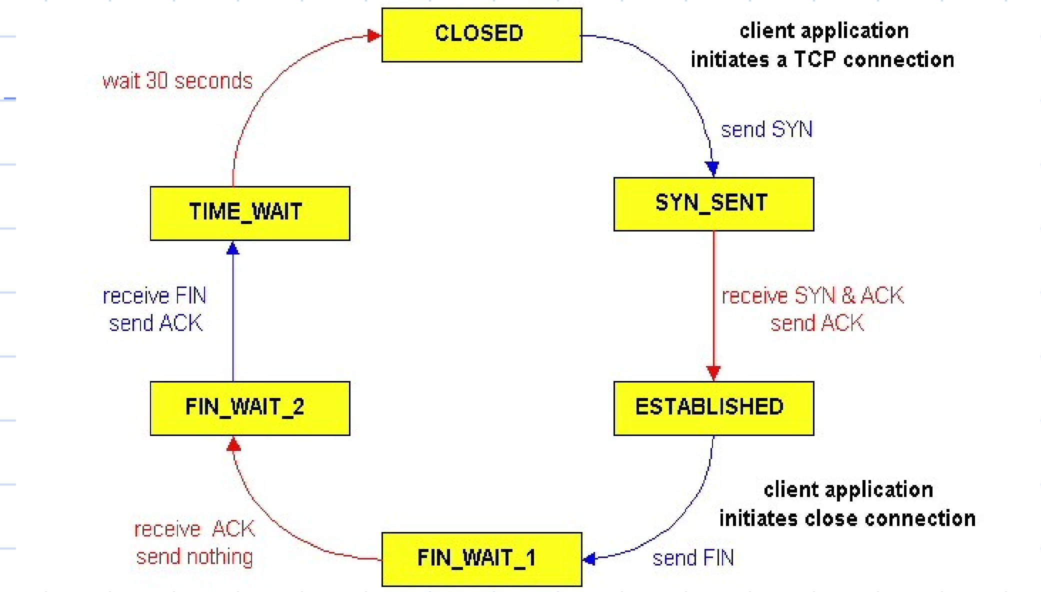
Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



Seq numbers and Acks

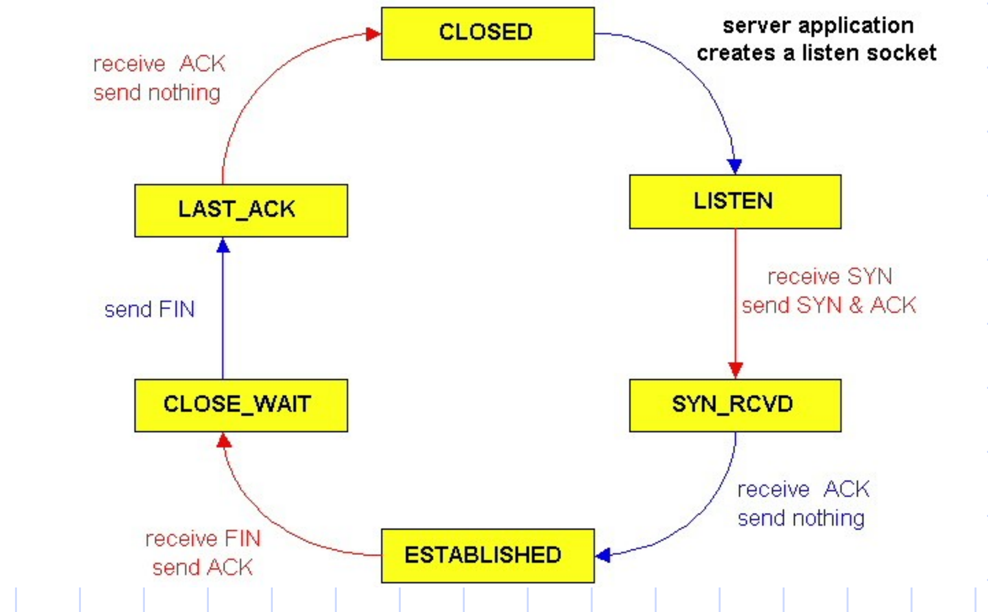
- Sequence numbers are used to reassemble data in the order in which it was sent.
- Sequence numbers increment based on the number of bytes in the TCP data field.
  - –Known as a Byte Sequencing Protocol
- Each segment transmitted must be acknowledged.
  - –Multiple segments can be acknowledged
- The ACK (Acknowledgement) field indicates the next byte (sequence) number the receiver expects to receive.
- The sender, no matter how many transmitted segments, expects to receive an ACK that is one more than the number of the last transmitted byte.

TCP States



TCP client

TCP Server

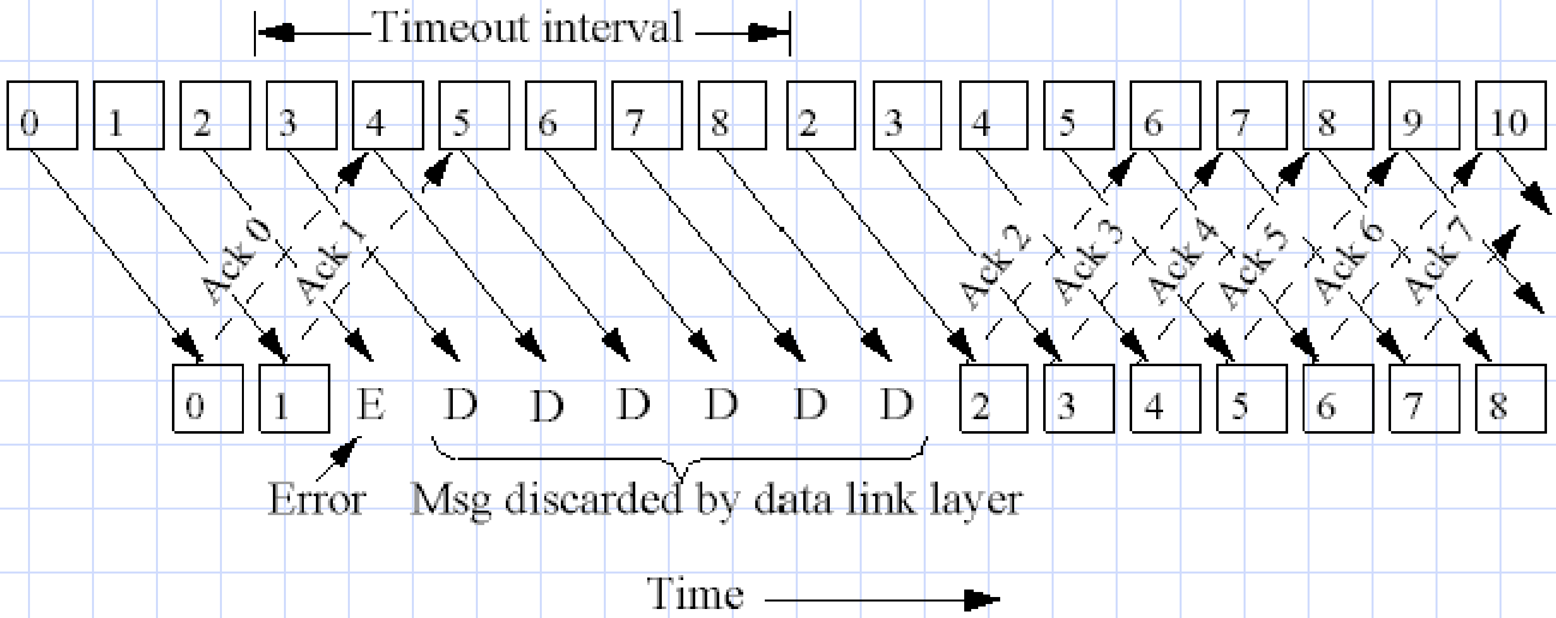


TCP Flow & Window Control

- **Sliding Window mechanism** -> the number of allowed unacknowledged bytes
  - Stop and Wait
  - Go-back N (TCP)
  - Selective Repeat
- Receiver Window
- Sender Window

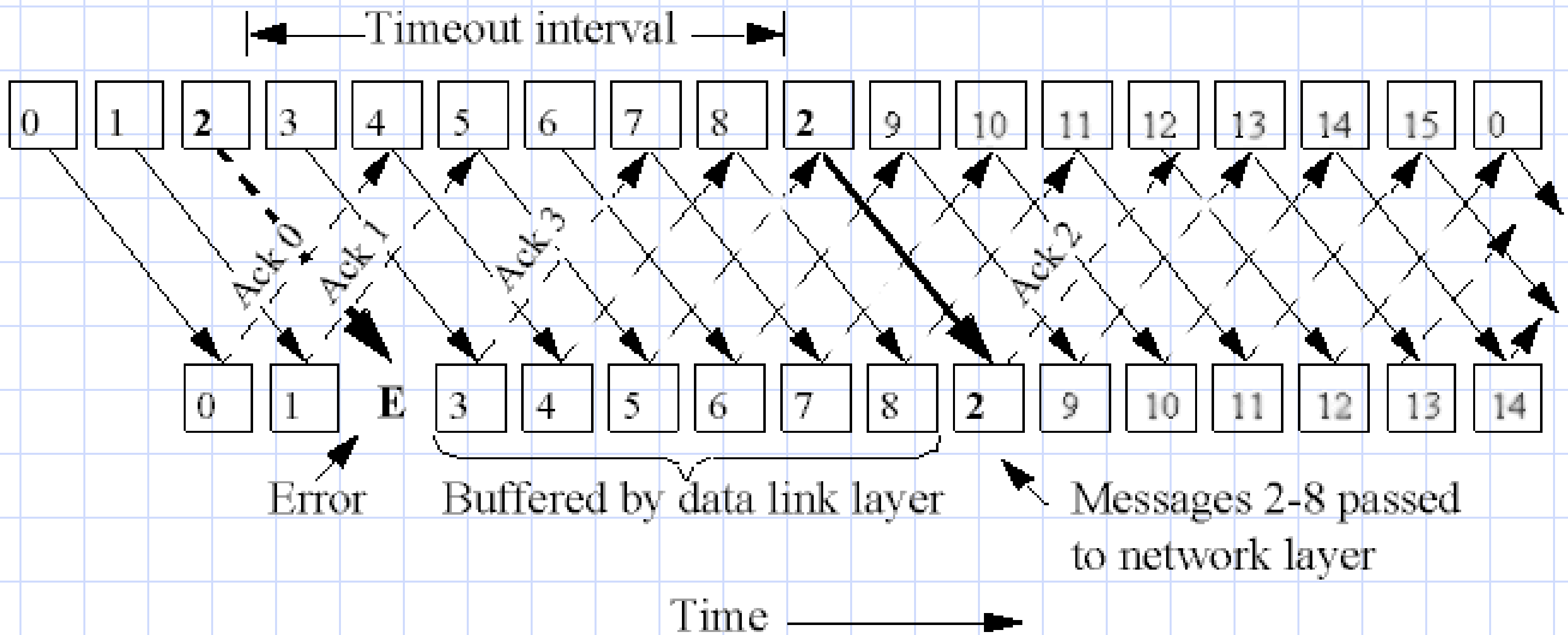
Go Back N

If seg k not received => discard k+1, k+2, etc



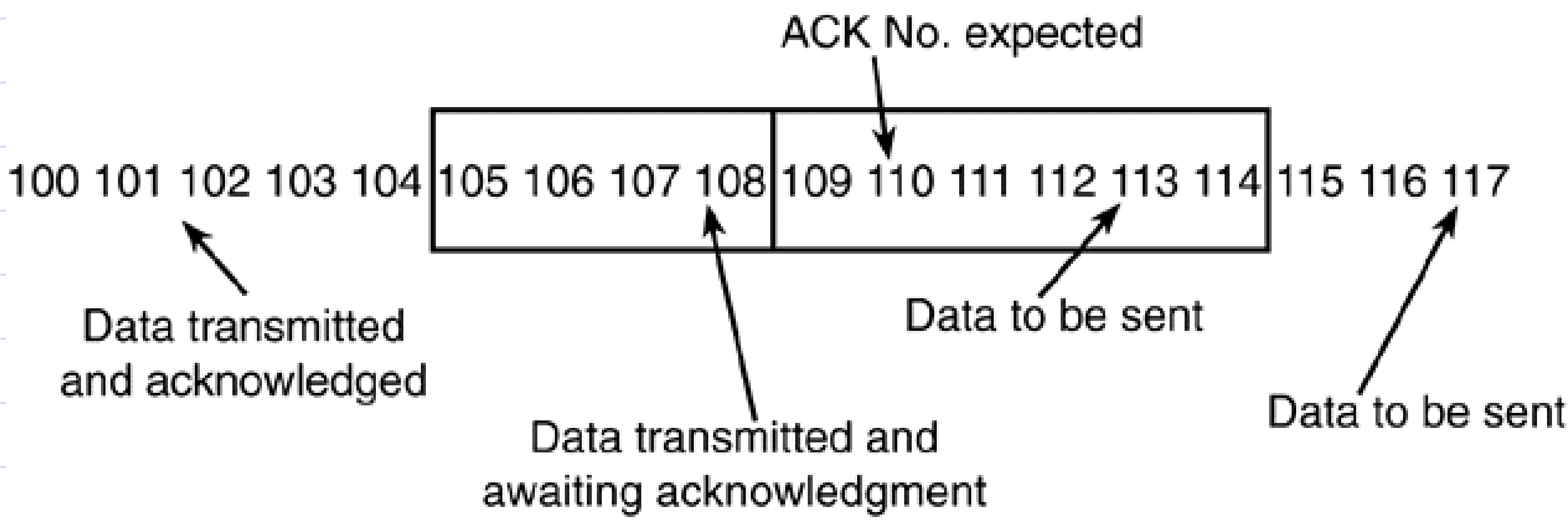
This implicitly sets the Window Size =1

Selective Repeat



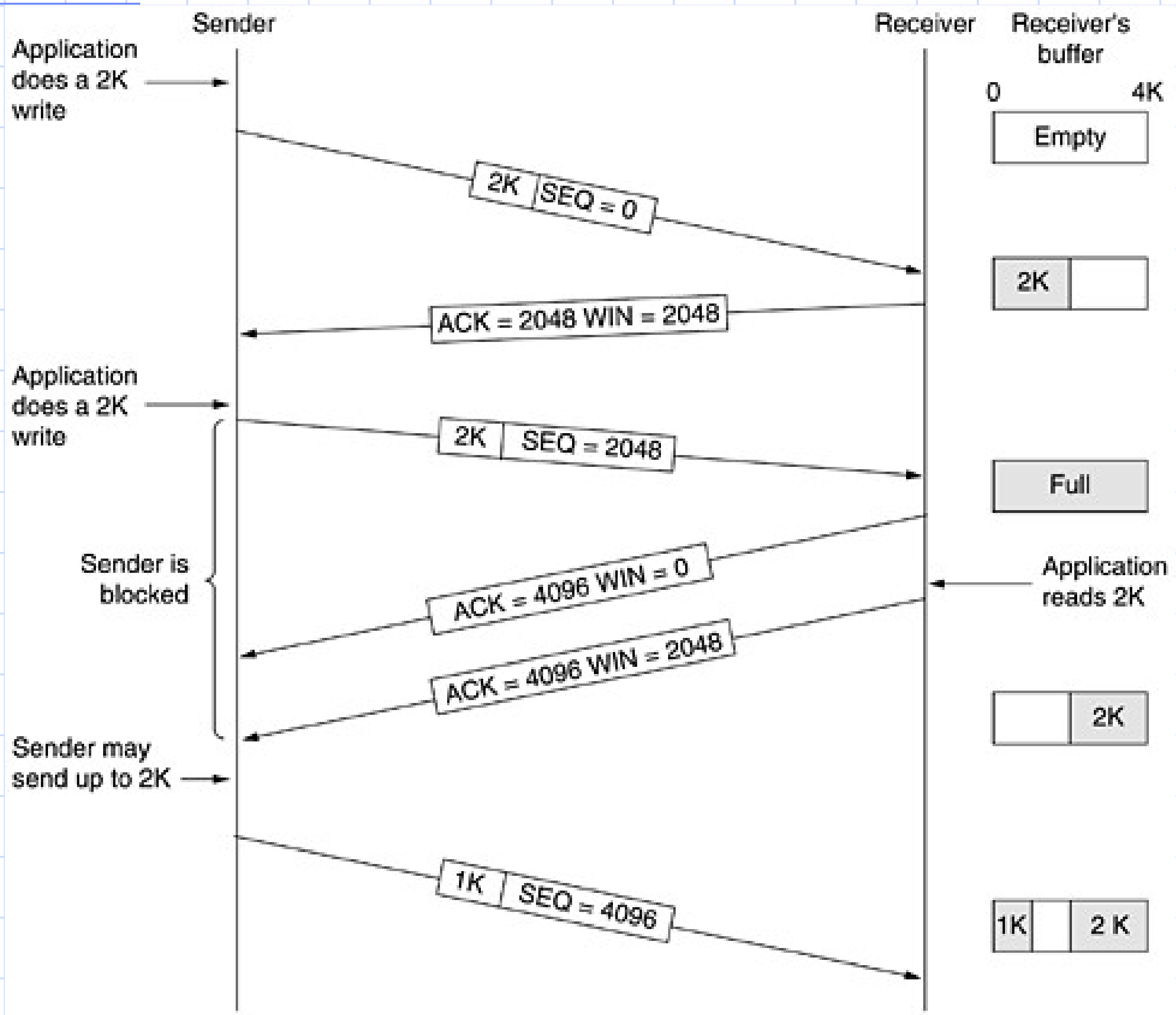
TCP Send Window

Windows based on advertised window in the received packet from the partner



Note: The actual segment size is usually 512 or 536 bytes each, but for clarity, I have shown a much smaller size.

Window Management



TCP Retransmission

- TCP will retransmit a segment upon expiration of an adaptive transmission timer.
- The timer is variable.
- When TCP transmits a segment, it records the time of transmission and the sequence number of the segment.
- When TCP receives an acknowledgment, it records the time.
- This allows TCP to build a sample round-trip delay time. **(RTT)**
- TCP will build an average delay time for a packet to be sent and received.
- The timer is slowly changed to allow for the varying differences in the Internet.

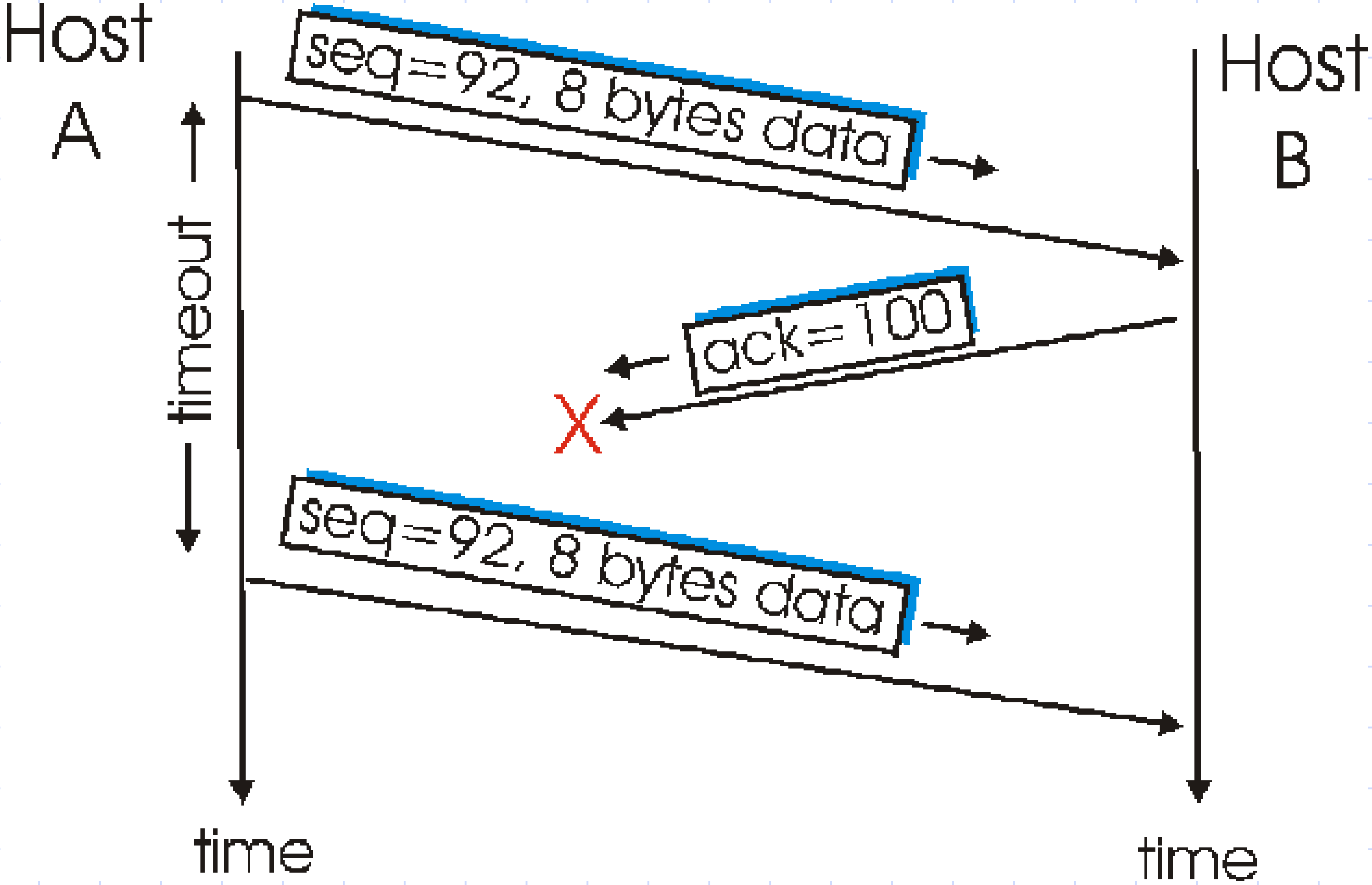
Timeout value ?

EstimatedRTT=(1-α)EstimatedRTT+ α SampleRTT

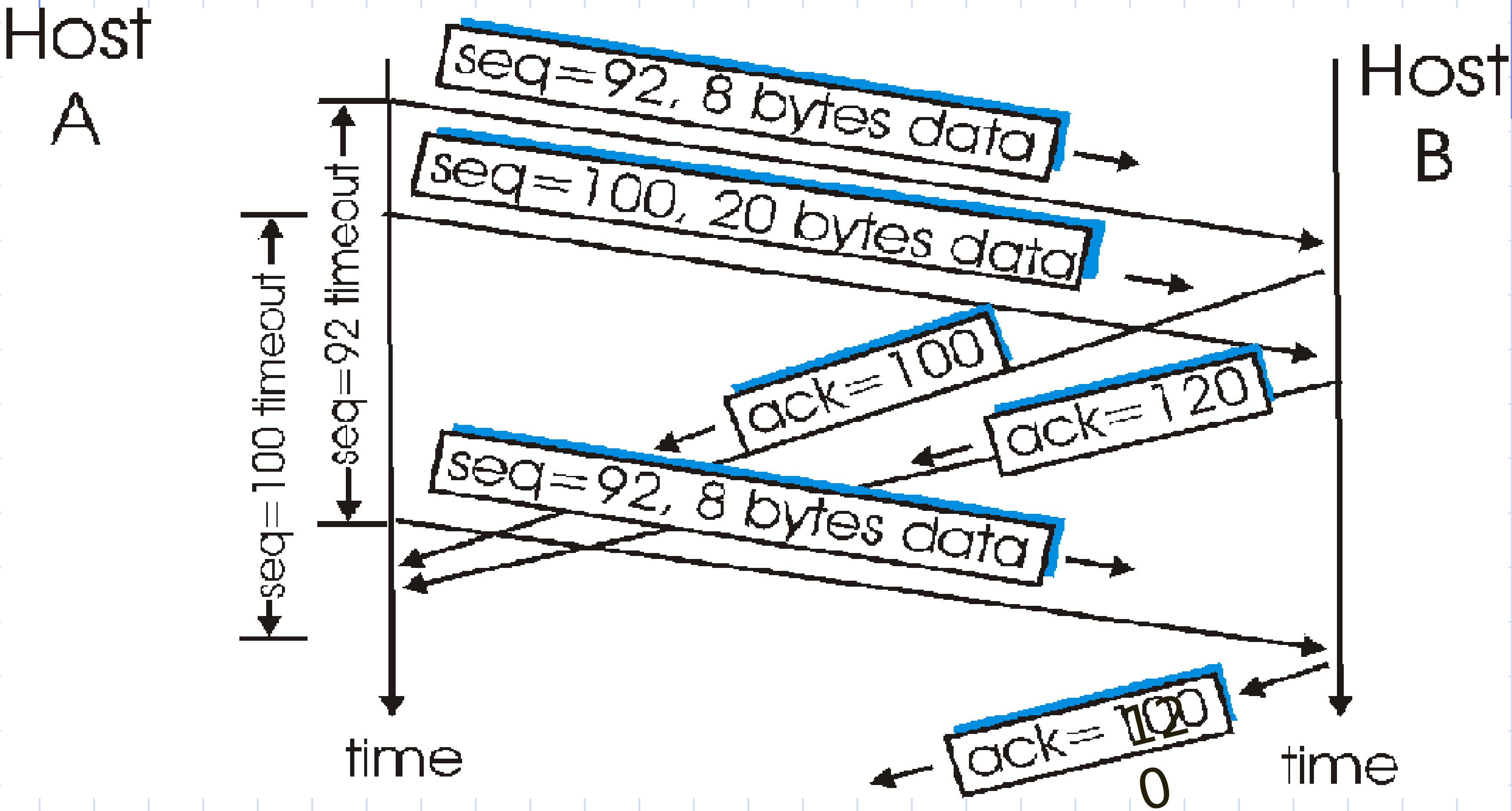
DevRTT = (1-β)DevRTT + β | SampleRTT-EstimatedRTT |

TimeoutInterval = EstimatedRTT + 4 DevRTT

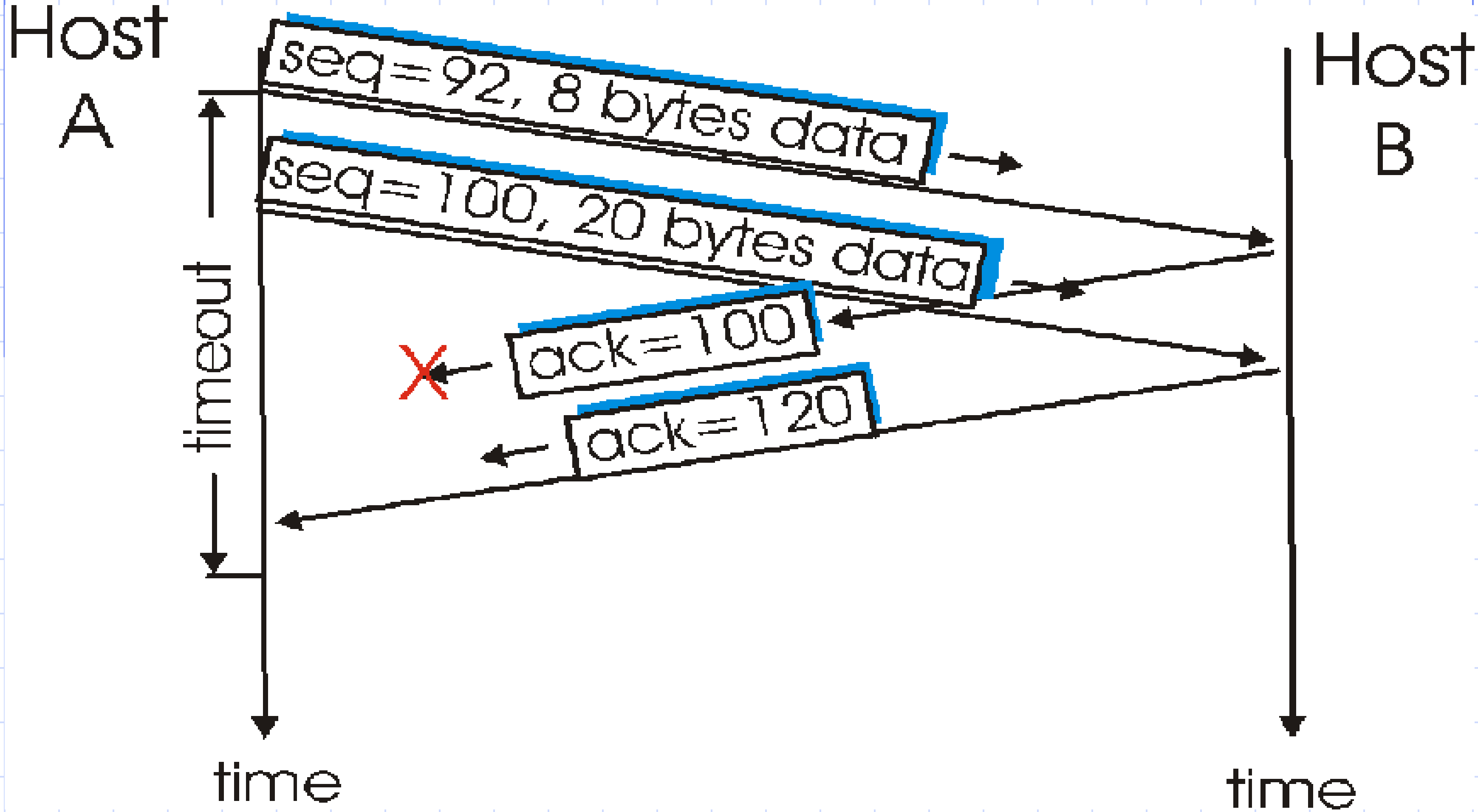
Retransmission-1



Retransmission-2



Retransmission-3



Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Approaches towards congestion control

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at

Congestion Control

- Previously, TCP would start to transmit as much data as was allowed in the advertised window.
- What about congestion ? What is it ?
- A new window was added called the <u>**congestion window**</u>. –It is not negotiated, it is assumed. It starts out with *one segment* !

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:

`LastByteSent-LastByteAcked` $\leq$ **`CongWin`**

- Roughly,

$$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$$

- `CongWin` is **dynamic, function of perceived network congestion**

How does sender perceive congestion?

- loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (`CongWin`) after loss event

three mechanisms:

- AIMD ***(additive increase, multiplicative decrease)***
- slow start
- conservative after timeout events

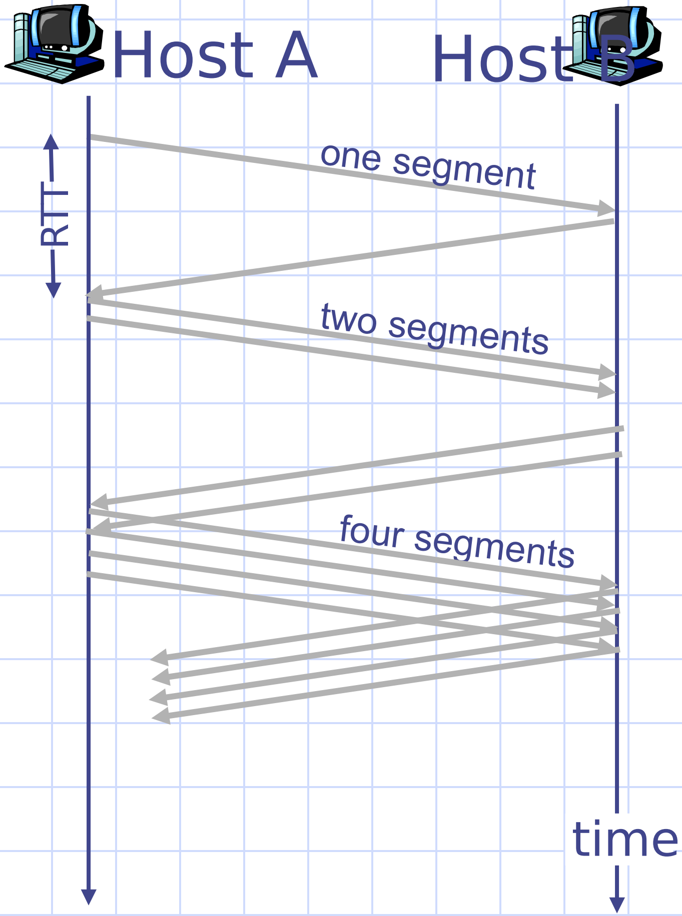
TCP Slow Start

- **When connection begins**, `CongWin` = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- **available bandwidth may be >> MSS/RTT**
  - desirable to quickly ramp up to respectable rate

When connection begins, increase rate exponentially fast until first loss event

TCP Slow Start -2

- When connection begins, increase rate exponentially until first loss event:
  - double **`CongWin`** every RTT
  - done by incrementing **`CongWin`** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement

- After 3 dup ACKs:
  - **`CongWin`** is cut in half
  - window then grows linearly
- But after timeout event:
  - **`CongWin`** instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

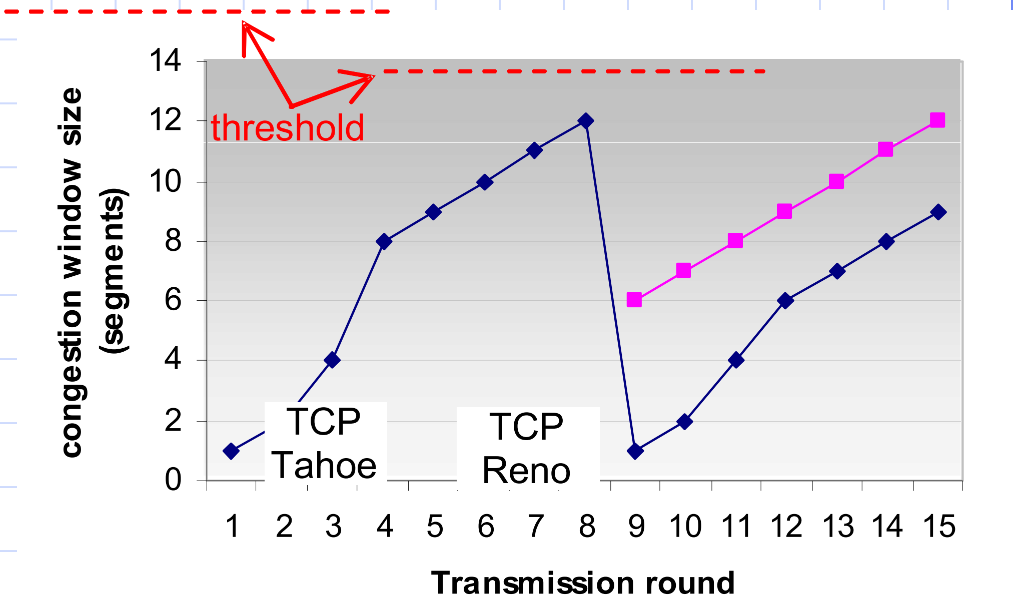
Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout before 3 dup ACKs is "more alarming"

Refinement -2

Q: When should the exponential increase switch to linear?

A: When **CongWin** gets to 1/2 of its value before timeout.



Implementation:

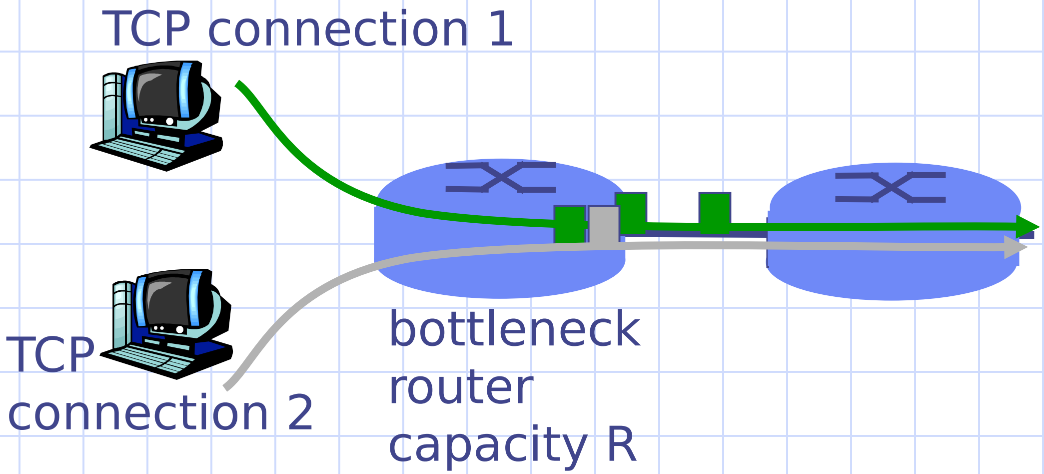
- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

Summary: TCP Congestion Control

- When **`CongWin`** is below **`Threshold`**, sender in slow-start phase, window grows exponentially.
- When **`CongWin`** is above **`Threshold`**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs,
  - **`Threshold =CongWin/2`**
  - **`CongWin = Threshold`**.
- When timeout occurs,
  - **`Threshold = CongWin/2`**
  - **`CongWin`** =1 MSS.

TCP Fairness

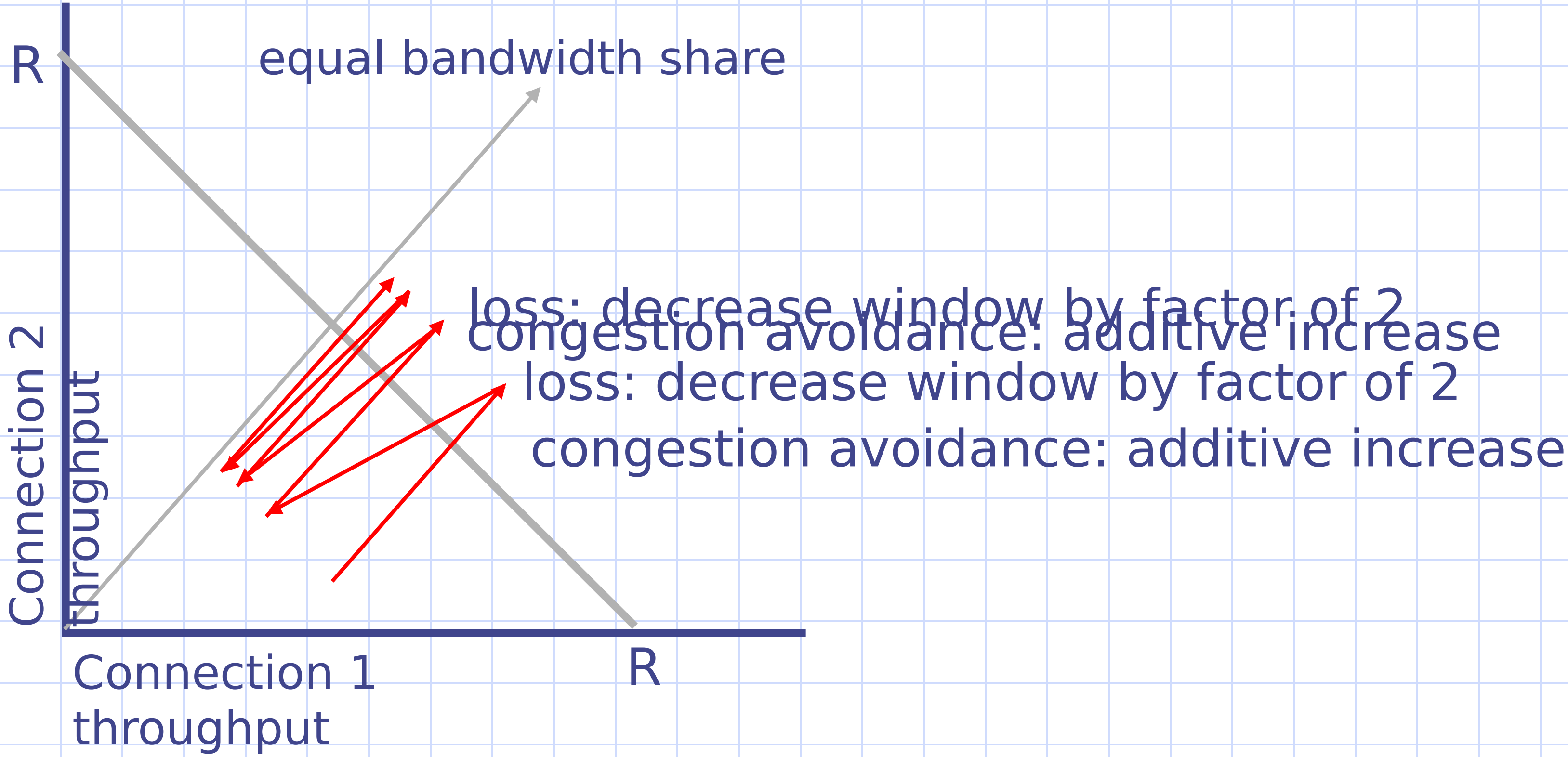
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair ?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Fairness !!!!

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel cnctions between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 cnctions;
  - new app asks for 1 TCP, gets rate R/10
  - **new app asks for 11 TCPs**, **gets R/2** !