Computer Networks

The Network Layer TCP/IP

Adrian Sergiu DARABANT

Lecture 8

IP Datagram

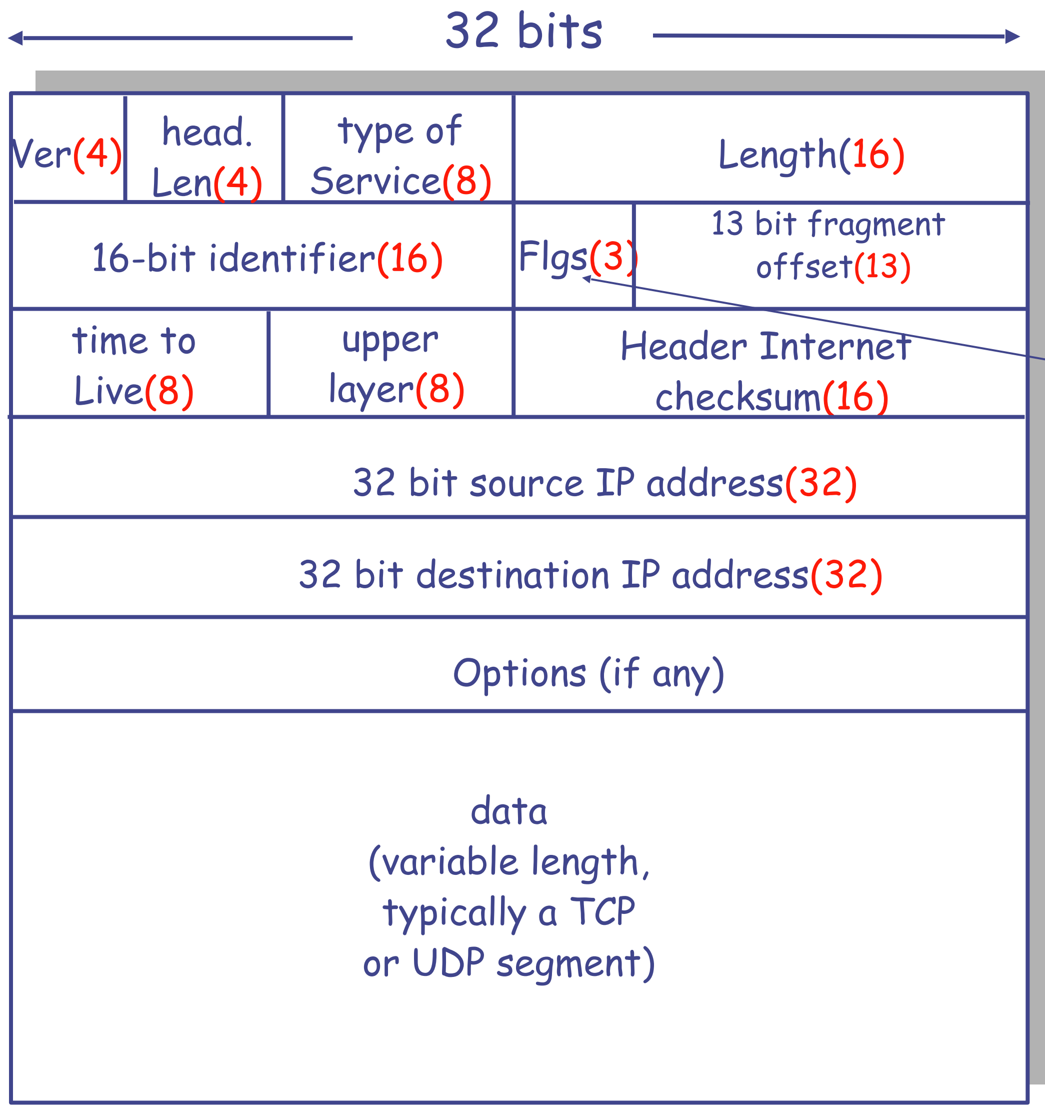
IP protocol version number

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

IP Checksum

The checksum field is the 16-bit one's complement of the one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

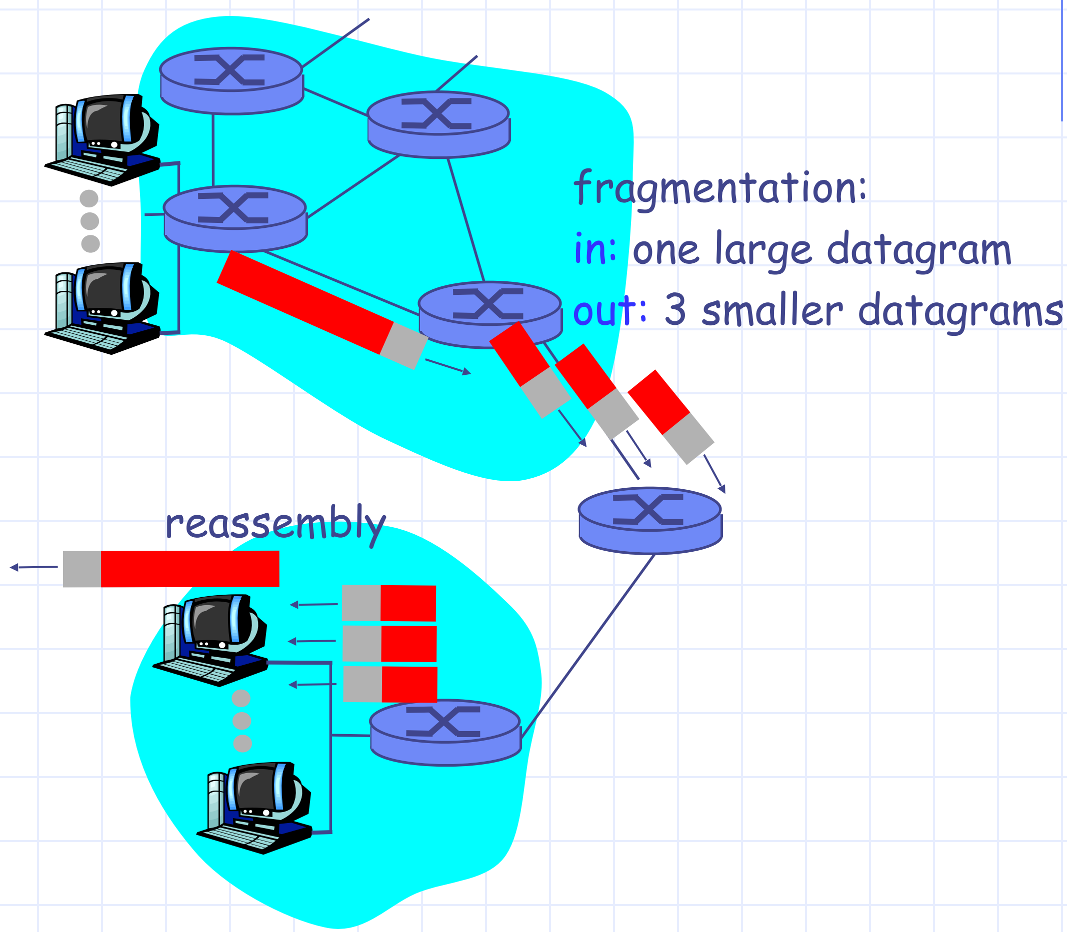
Hex 4500003044224000800600008c7c19acae241e2b (20 bytes IP header):

1.4500 + 0030 + 4422 + 4000 + 8006 + 0000 + 8c7c + 19ac + ae24 + 1e2b = 0002BBCF (32-bit sum)
2.0002 + BBCF = BBD1 = 1011101111010001 (1's complement 16-bit sum)
3.~BBD1 = 0100010000101110 = **442E** (1's complement of 1's complement 16-bit sum)

Verify the Checksum at destination:
1.2BBCF + 442E = 2FFFD
2.2 + FFFD = FFFF (the 1's complement of FFFF) = 0.
3.Result is 0 means datagram header is OK !

Fragmentation/Reassembly

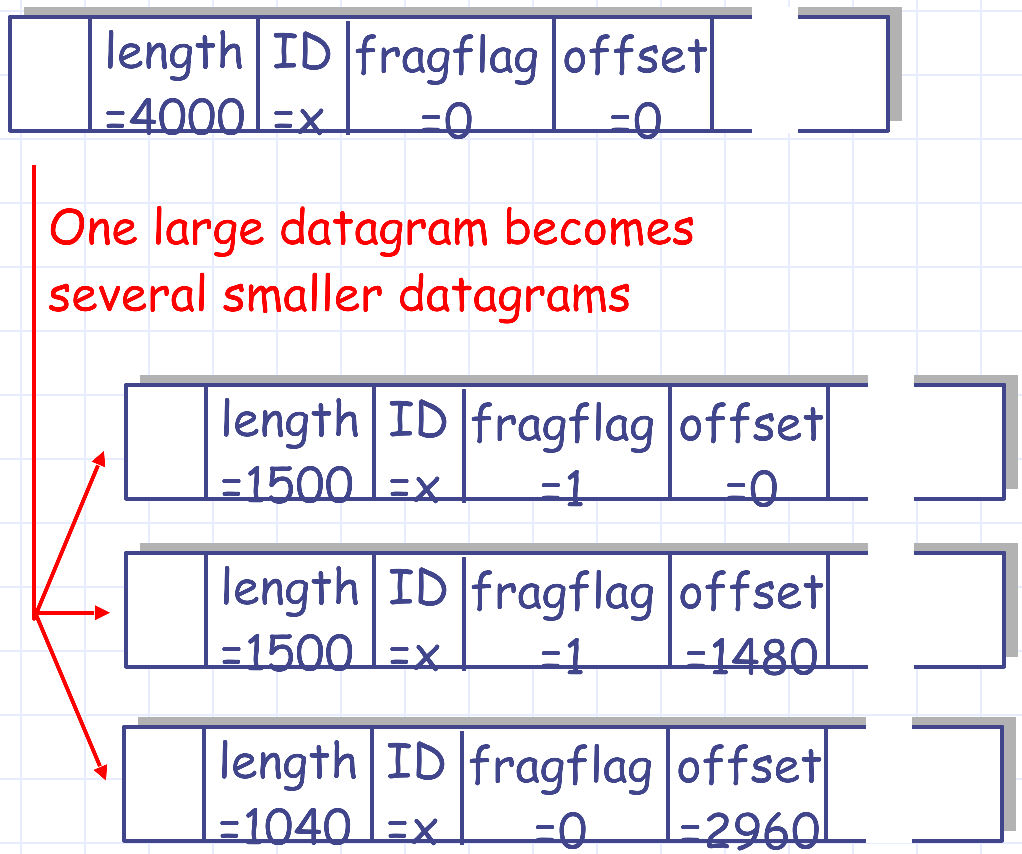
- network links have MTU (max.transfer size) - largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided (fragmented) within net
  - one datagram becomes several datagrams
  - reassembled only at final destination
  - IP header bits used to identify, order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



Fragmentation/Reassembly

Example

- 4000 byte datagram
- MTU = 1500 bytes



ARP: IP Addres vs Network Adapter Address

IP datagram: comm IP -> IP

misc fields	source IP addr	dest IP addr	data

Ethernet: MAC -> MAC

How does one translates IP to MAC addresses ?

Where can I see my MAC addresses ?

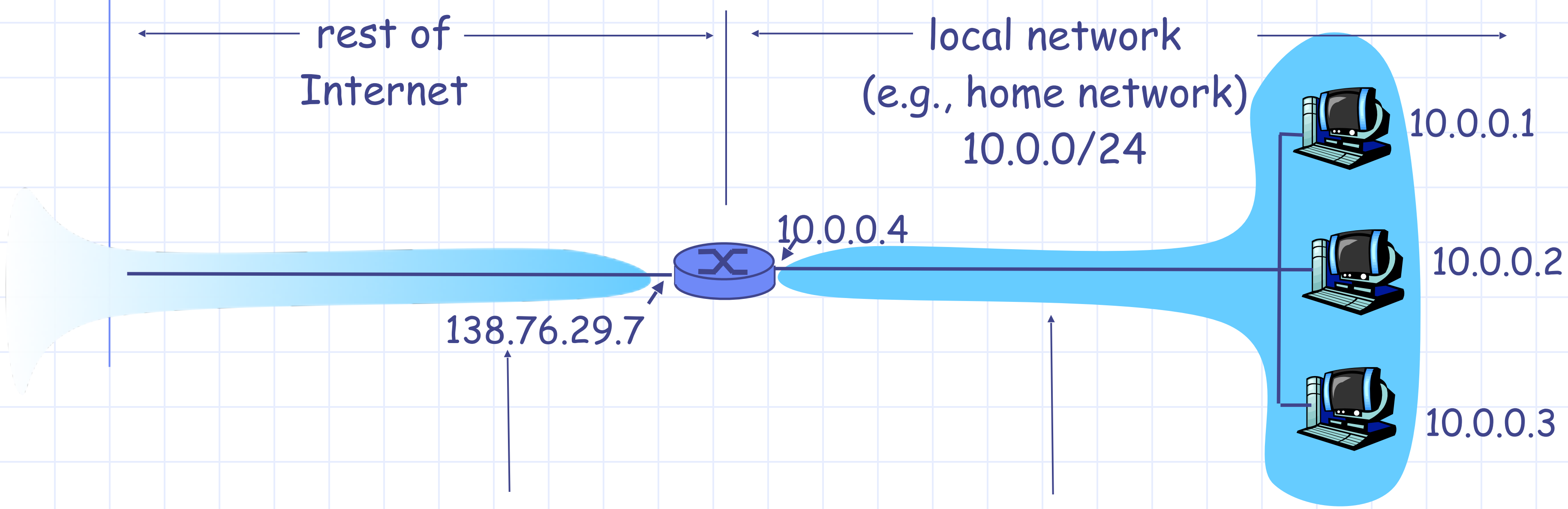
Who manages these mapping ?

ARP – Address Resolution Protocol

Ifconfig or ipconfig – shows also the MAC address

TCP/IP stack – using ARP tables – see **arp** command

NAT – Network Address Translation



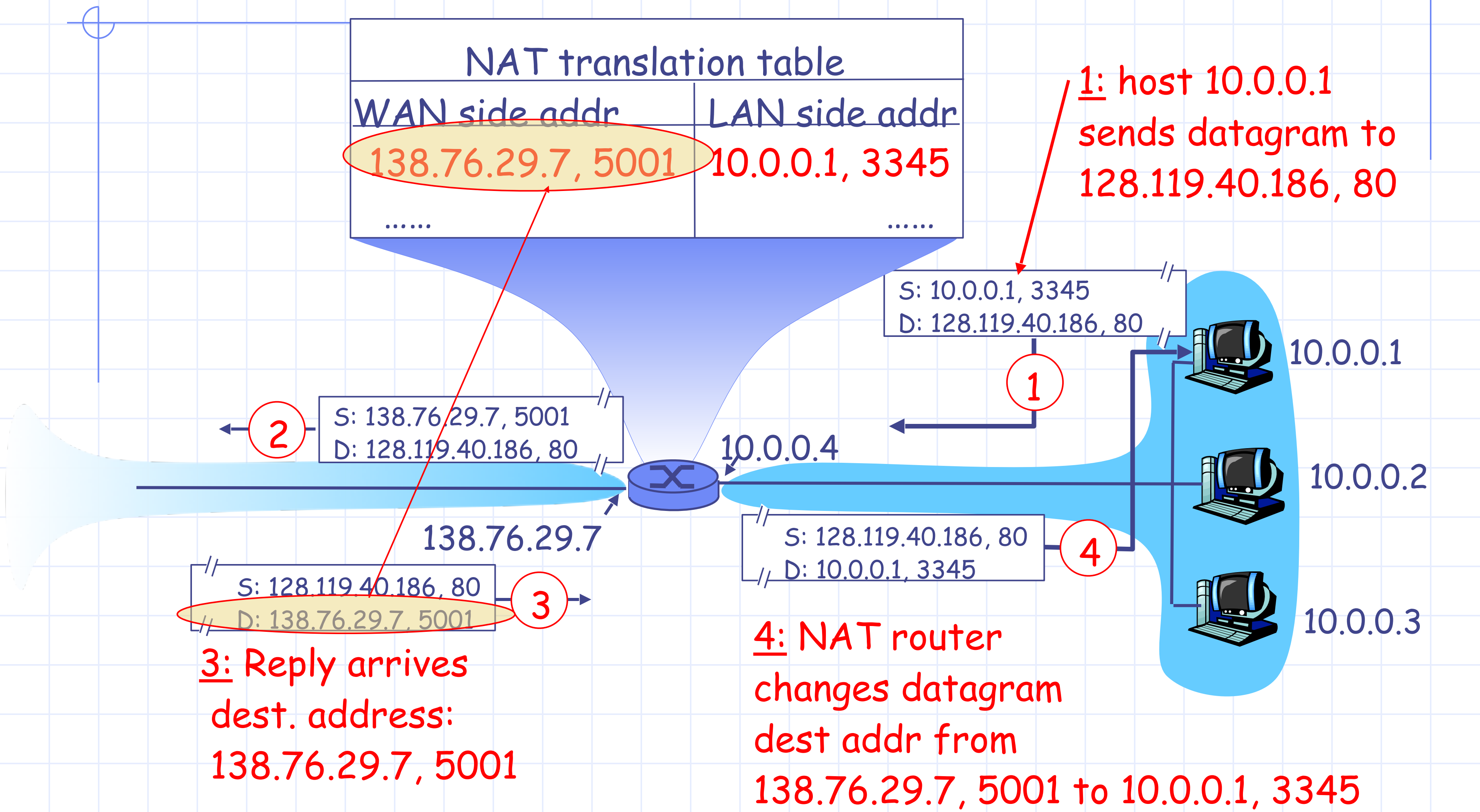
All datagrams leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

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



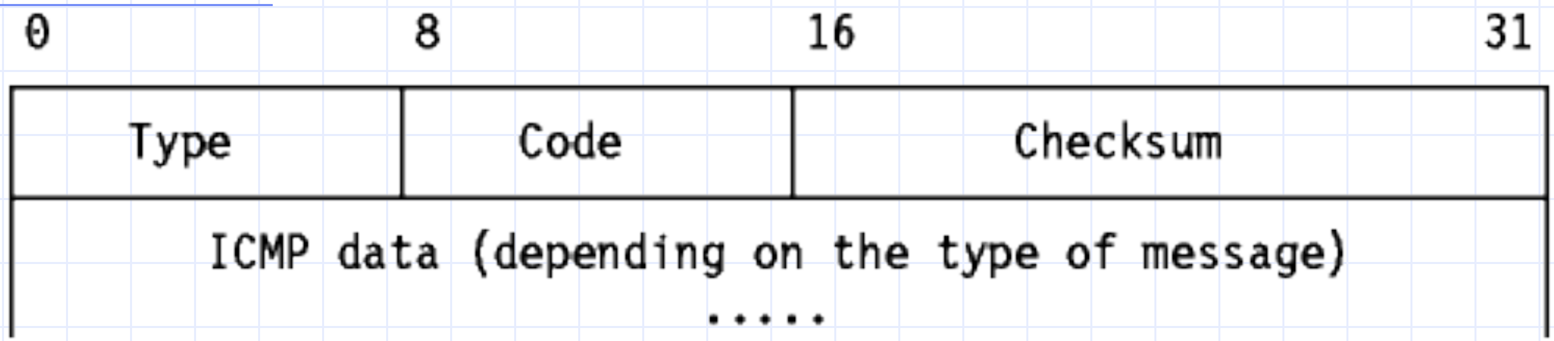
NAT – Network Address Translation

- 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 solved by IPv6

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

ICMP



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 - Debuging

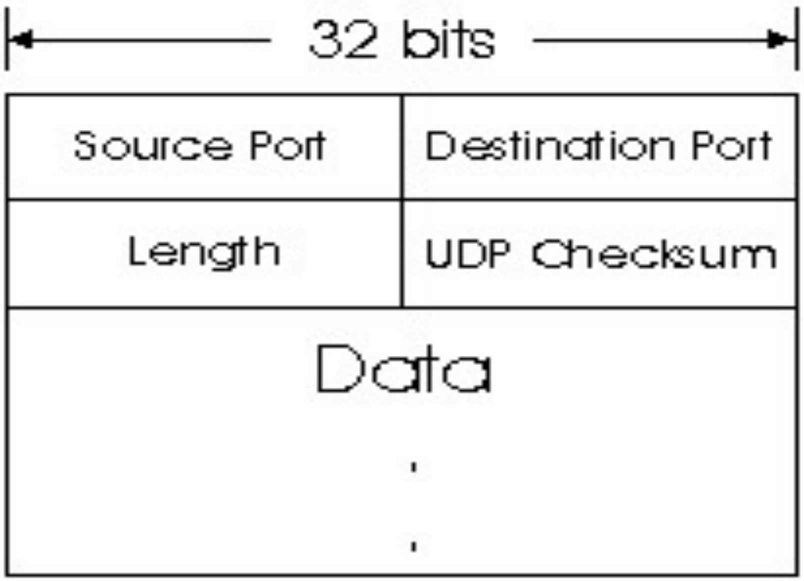
- **Tcpdump – Wireshark –** your network debuger

- **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).

Traceroute

- Uses IP TTL to trace packet paths
- S1: Set TTL=1 send a UDP datagram to destination => discarded on first Router and ICMP sent back with the Router IP as source (display it)
- S2: TTL=2 send a UDP datagram to destination => discarded on 2nd Router and ICMP sent back with the Router IP as source (display it)
- S3: TTL=3 repeat ….
- S1…Sn – UDP port used is a rarely used port. When destination is reached => send back ICMP port unreachable – Clever !

UDP



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

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

Checksum – for the entire datagram (header + data)

Length >=8 – entire datagram

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 congestion control measures at the application level.

TCP - Data Transfer

- **Ordered data transfer** — the destination host rearranges data according to the Sequence number
- **Retransmission of lost packets** — any cumulative stream not acknowledged is retransmitted
- **Error-free data transfer –**consequence of the above
- **Flow control (Window based)**— 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 vs UDP –fun way ☺

TCP:

Hi, I'd like to hear a TCP joke.
Hello, would you like to hear a TCP joke?
Yes, I'd like to hear a TCP joke.
OK, I'll tell you a TCP joke.
Ok, I will hear a TCP joke.
Are you ready to hear a TCP joke?
Yes, I am ready to hear a TCP joke.
Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline.
Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline.

I'm sorry, your connection has timed out.

Hello, would you like to hear a TCP joke?

UDP:

Want to hear an UDP joke?
Yes.

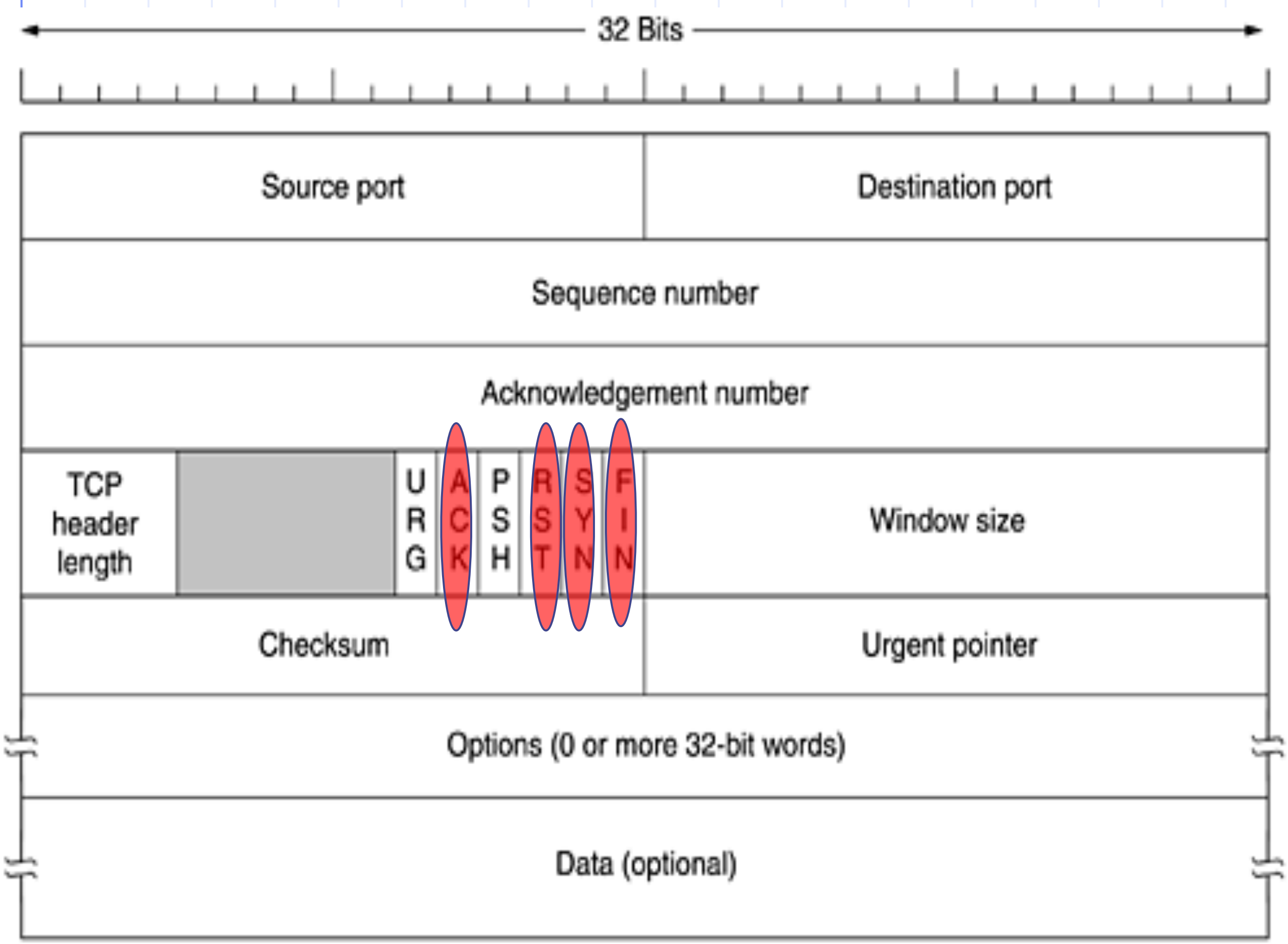
Was it good?

What?

TCP implementation- what we need ?

- **Data Loss ?** => each sent packet needs confirmation
- **Out of order ?** => data buffering on reception
- **Data Flow ?**=> infer network and peer status
- **Network Congestion ?**=> infer network status
- **All of the above ?** =>keep status on each side and initialize it at begining

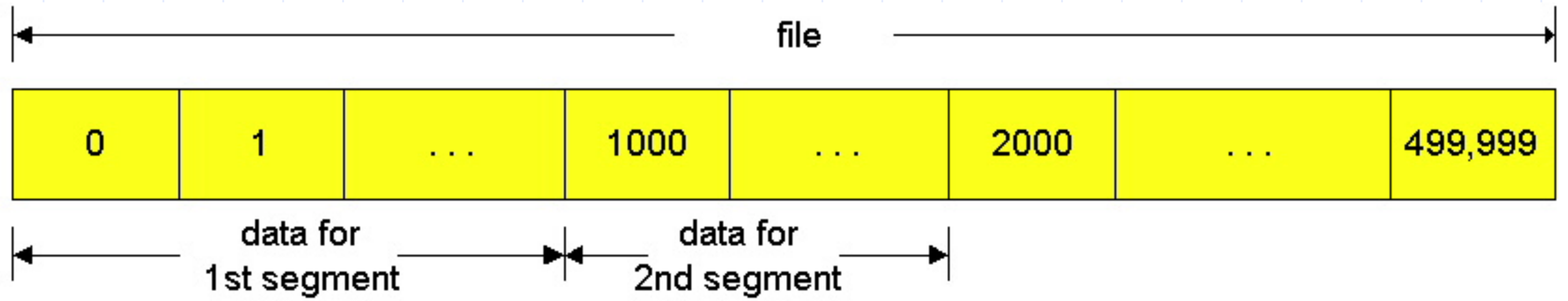
TCP Segment



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

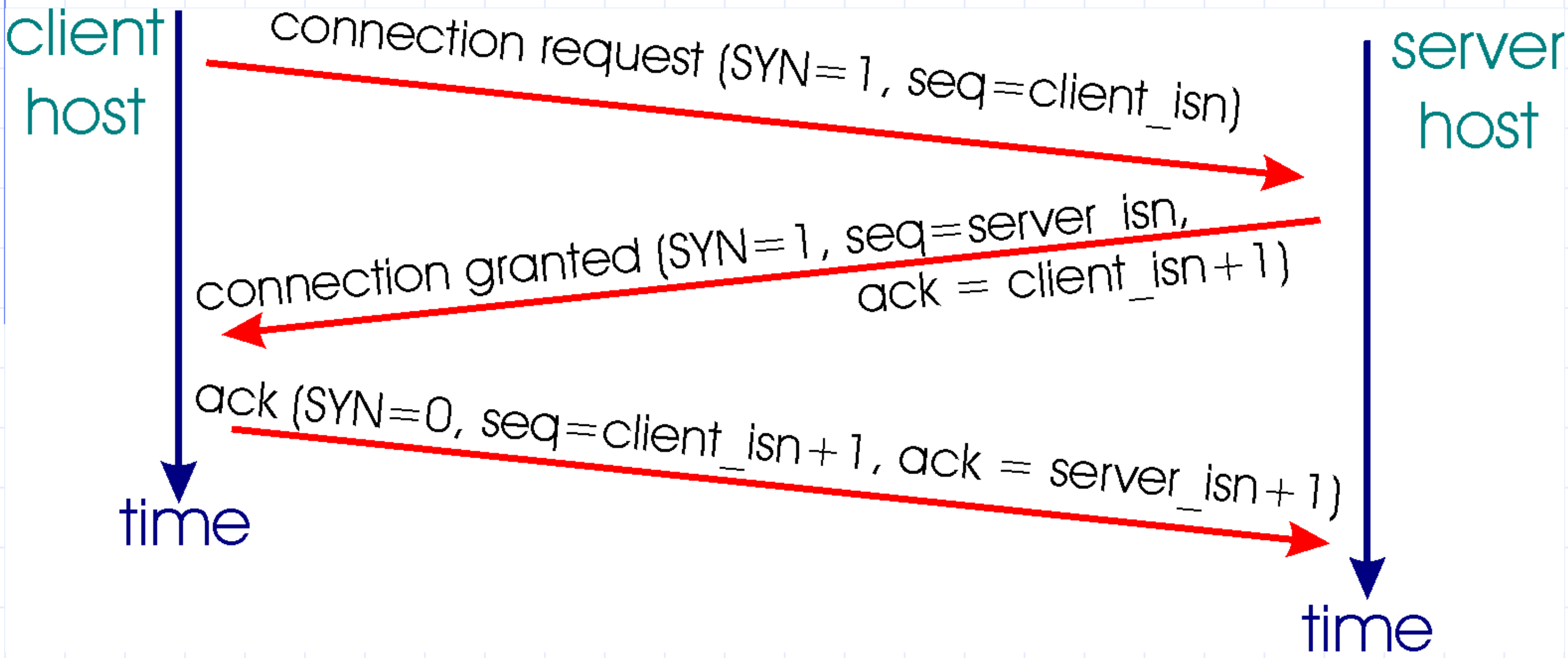
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

TCP Segments



- Data is a contiguous stream split into **segments**
- Each segment is carried into one (or multiple) IP datagrams
- Each segment needs an acknowledgement from receiver
- Send/Receive operations act **on the stream** they are not paired to each-other (i.e. one read for multiple sends)

TCP Open – 3-way handshake



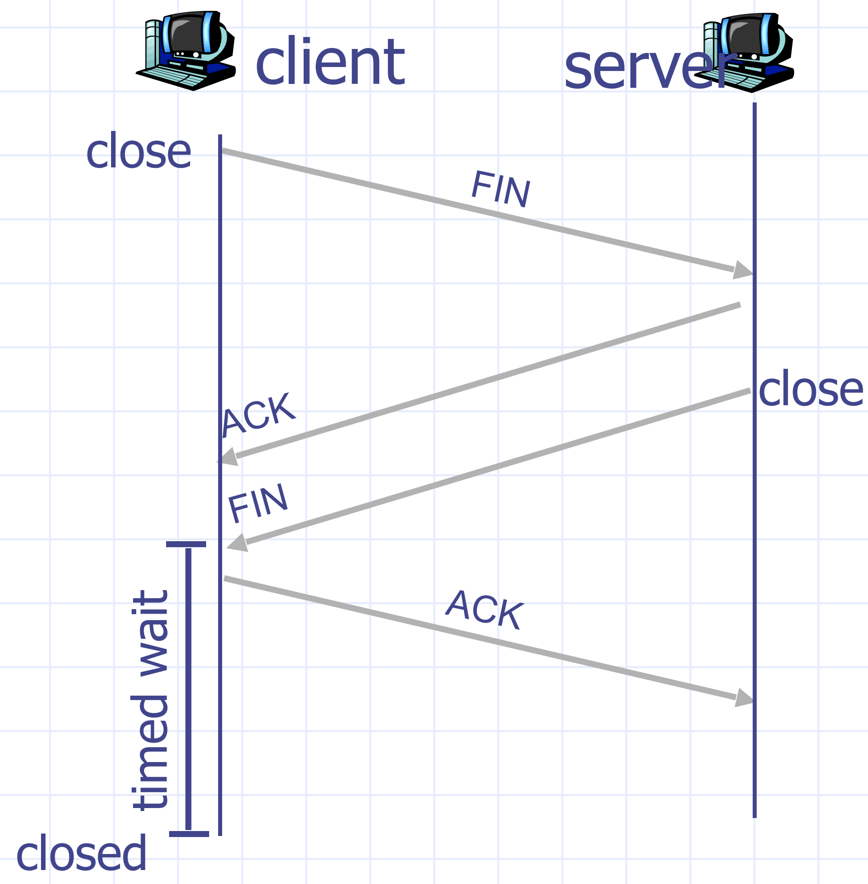
TCP Connection Teardown

Closing a connection:

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

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

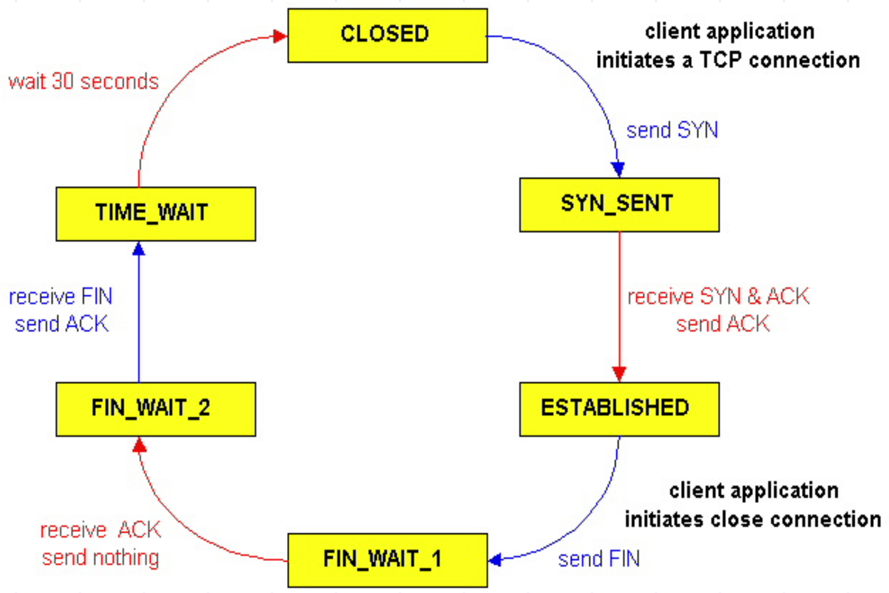
Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



Seq numbers and Acks

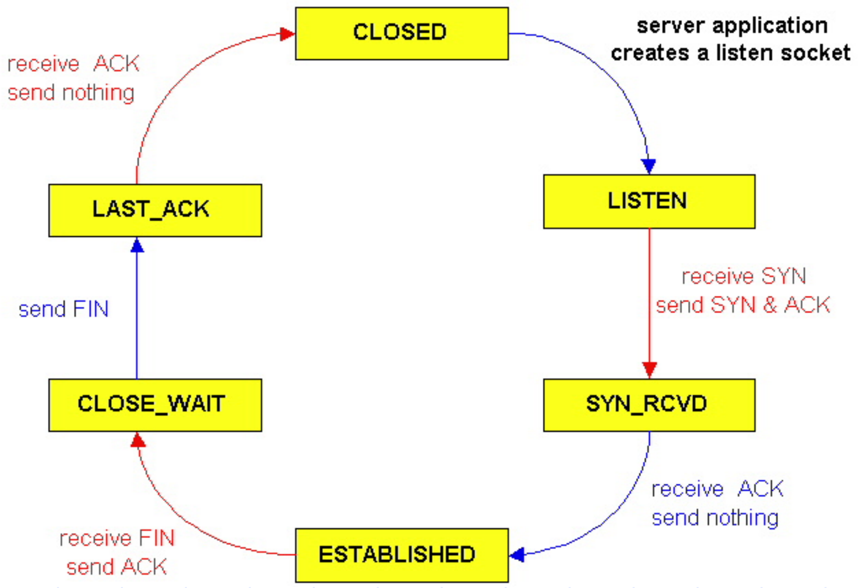
- Sequence numbers are used to reassemble data in the order in which it was sent.
- Sequence numbers increment based on the number of bytes in the TCP data field.
  - –Known as a Byte Sequencing Protocol
- Each segment transmitted must be acknowledged.
  - –Multiple segments can be acknowledged
- The 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

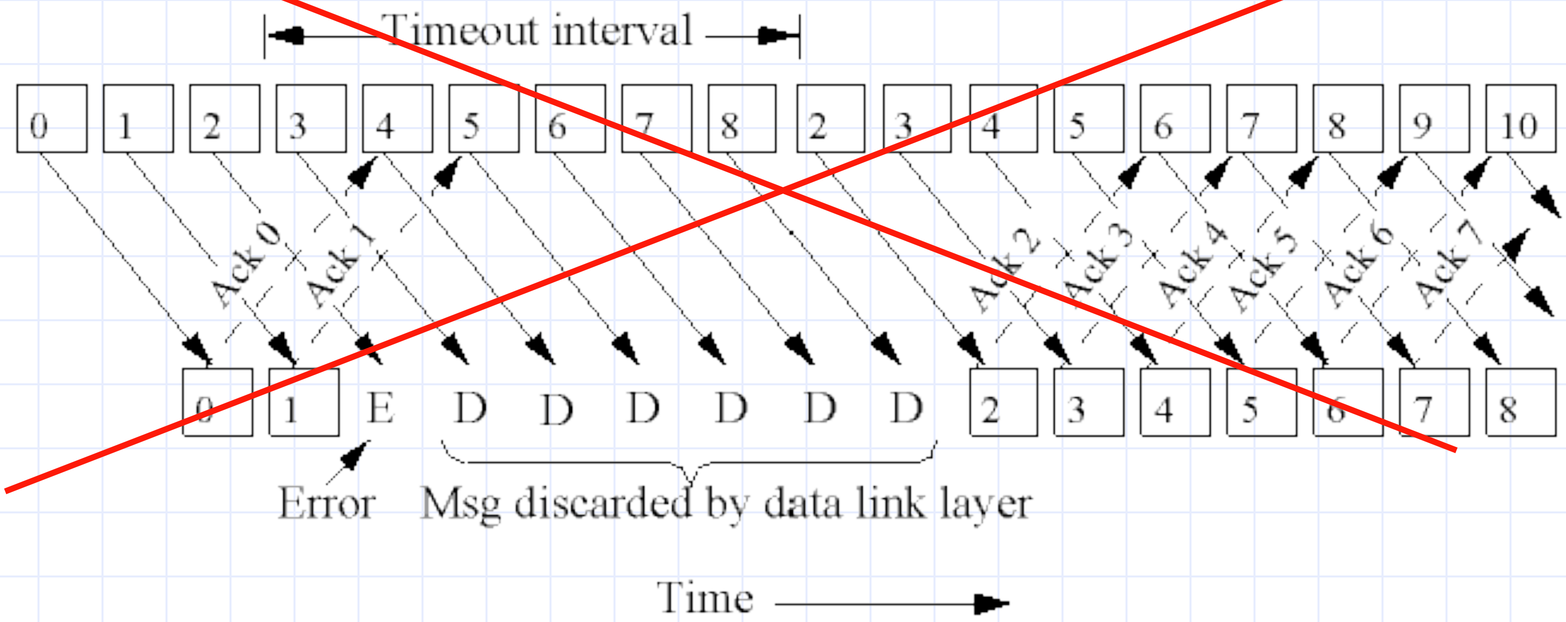


TCP Flow & Window Control

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

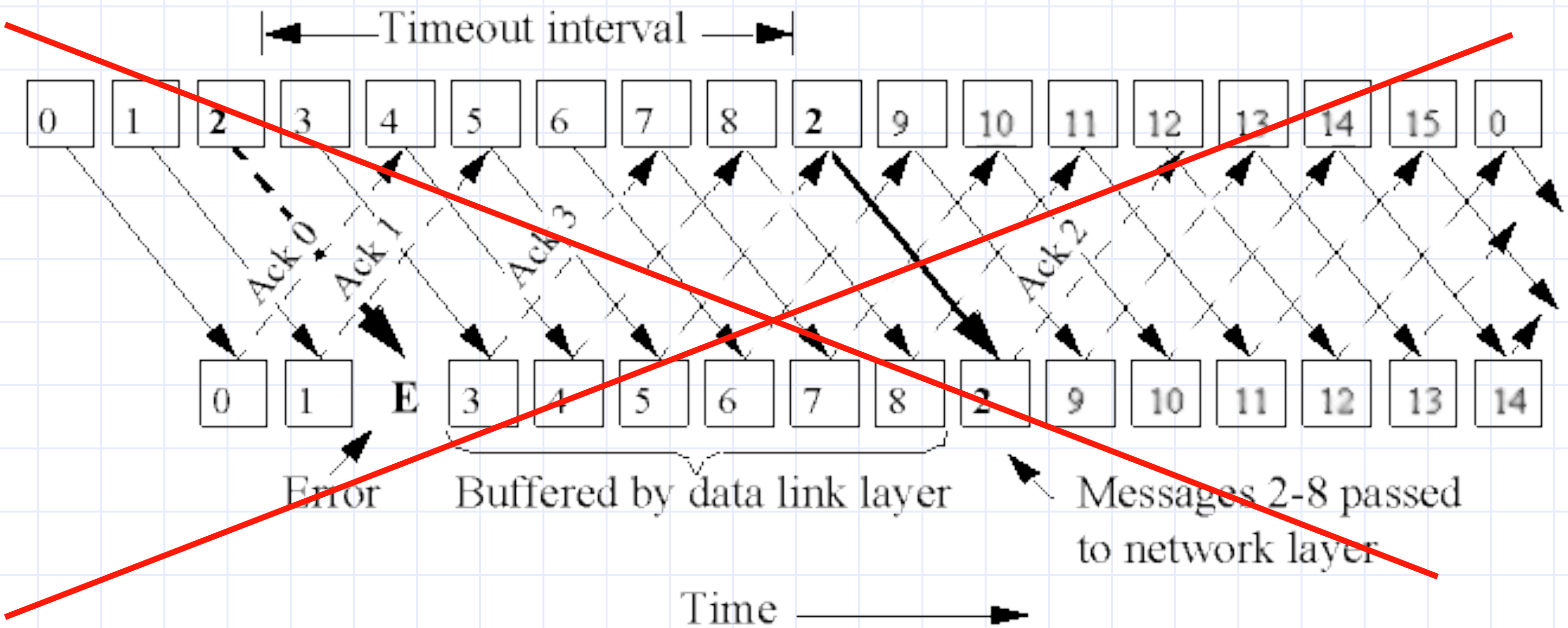
Go Back N

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



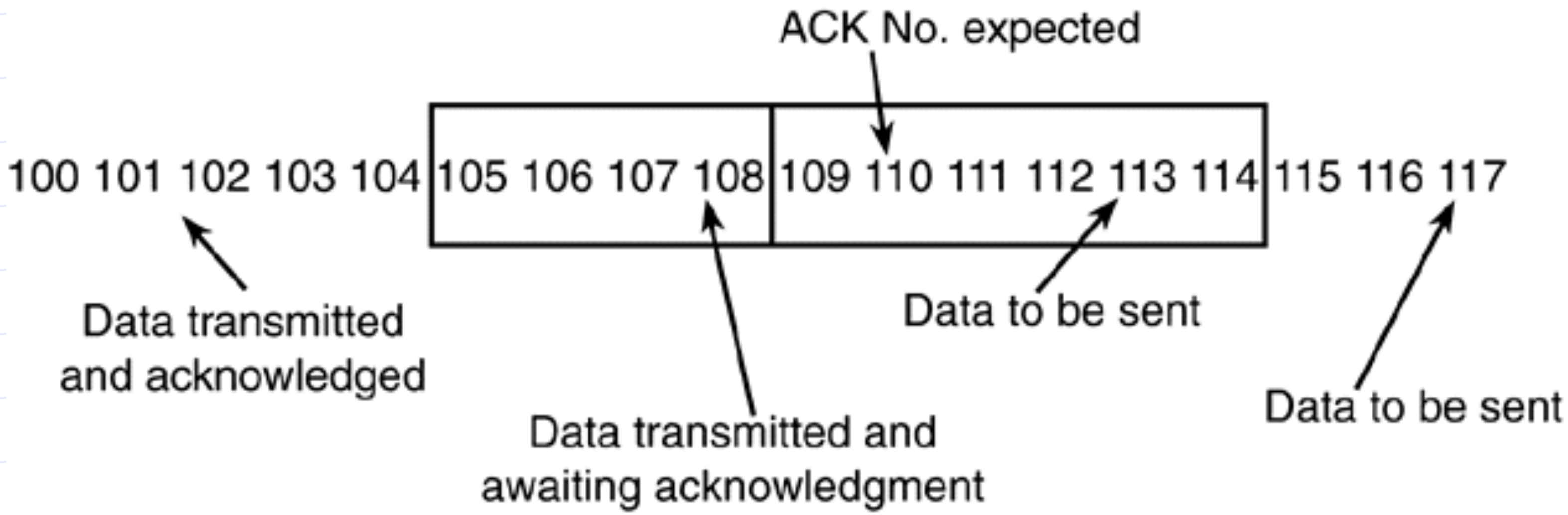
This implicitly sets the Window Size =1

Selective Repeat



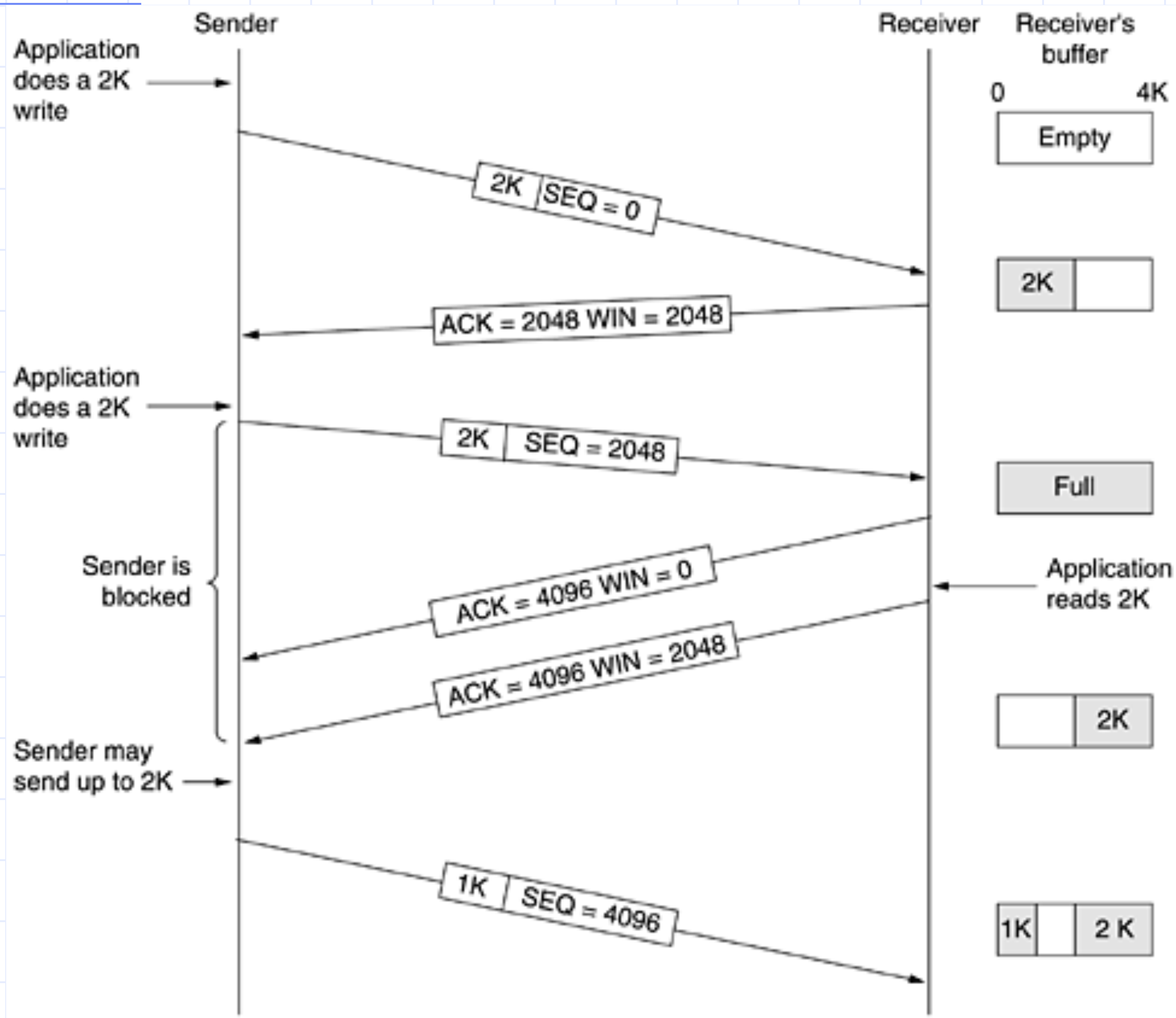
TCP Send Window

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.

Receiver Window Management



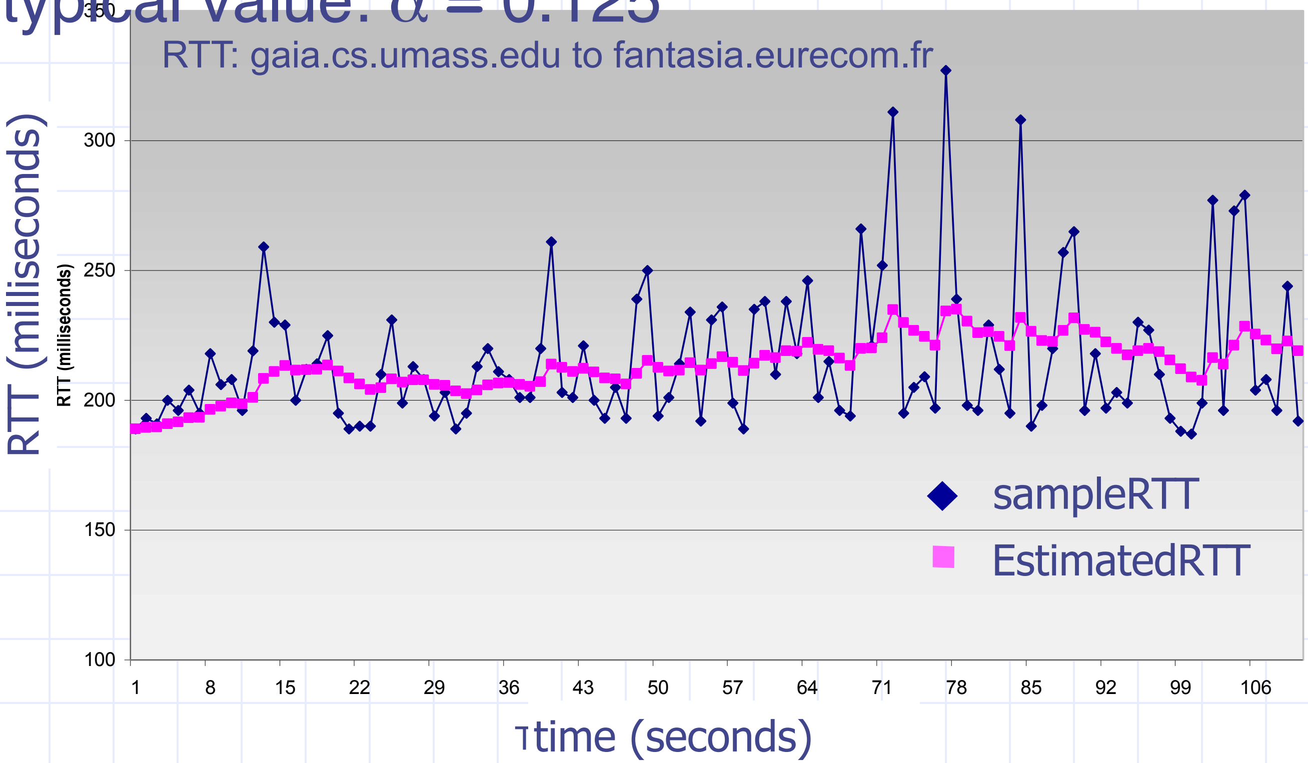
TCP Retransmission

- 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.

TCP round trip time, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



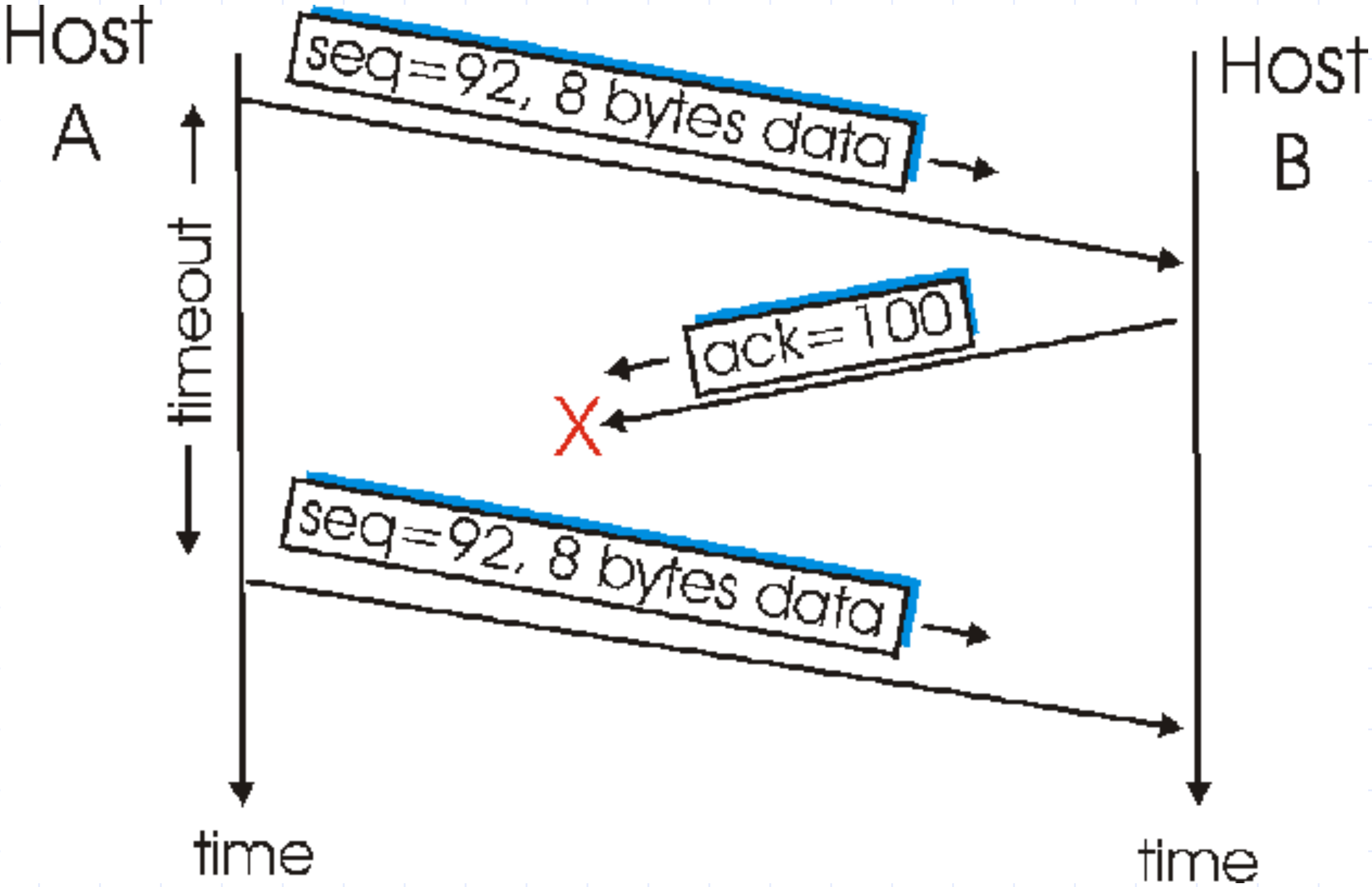
Timeout value ?

$EstimatedRTT = (1-\alpha)EstimatedRTT + \alpha\ SampleRTT$

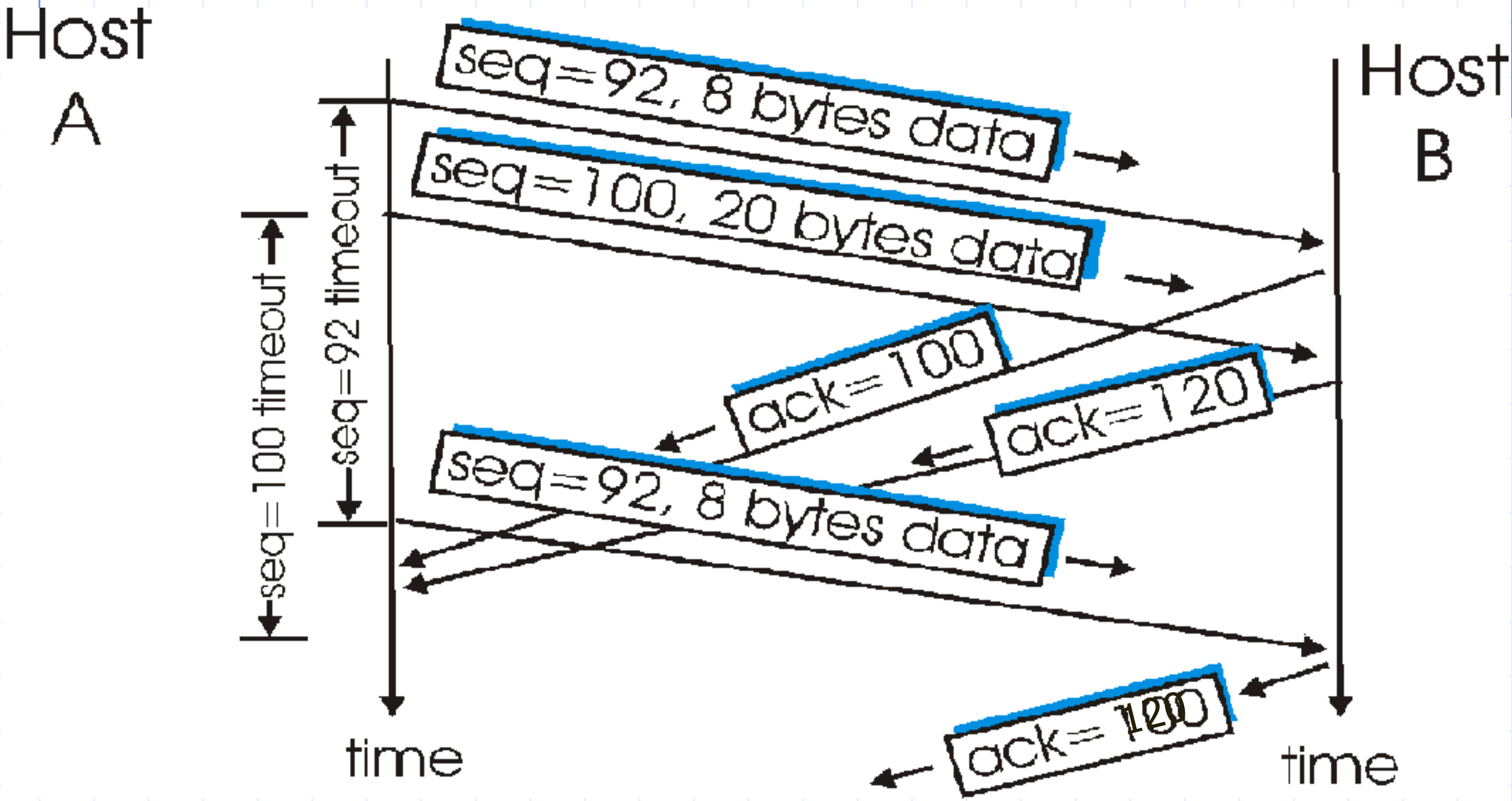
$DevRTT = (1-\beta)DevRTT + \beta\ |\ SampleRTT - EstimatedRTT\ |$

$\mathbf{TimeoutInterval = EstimatedRTT + 4\ DevRTT}$

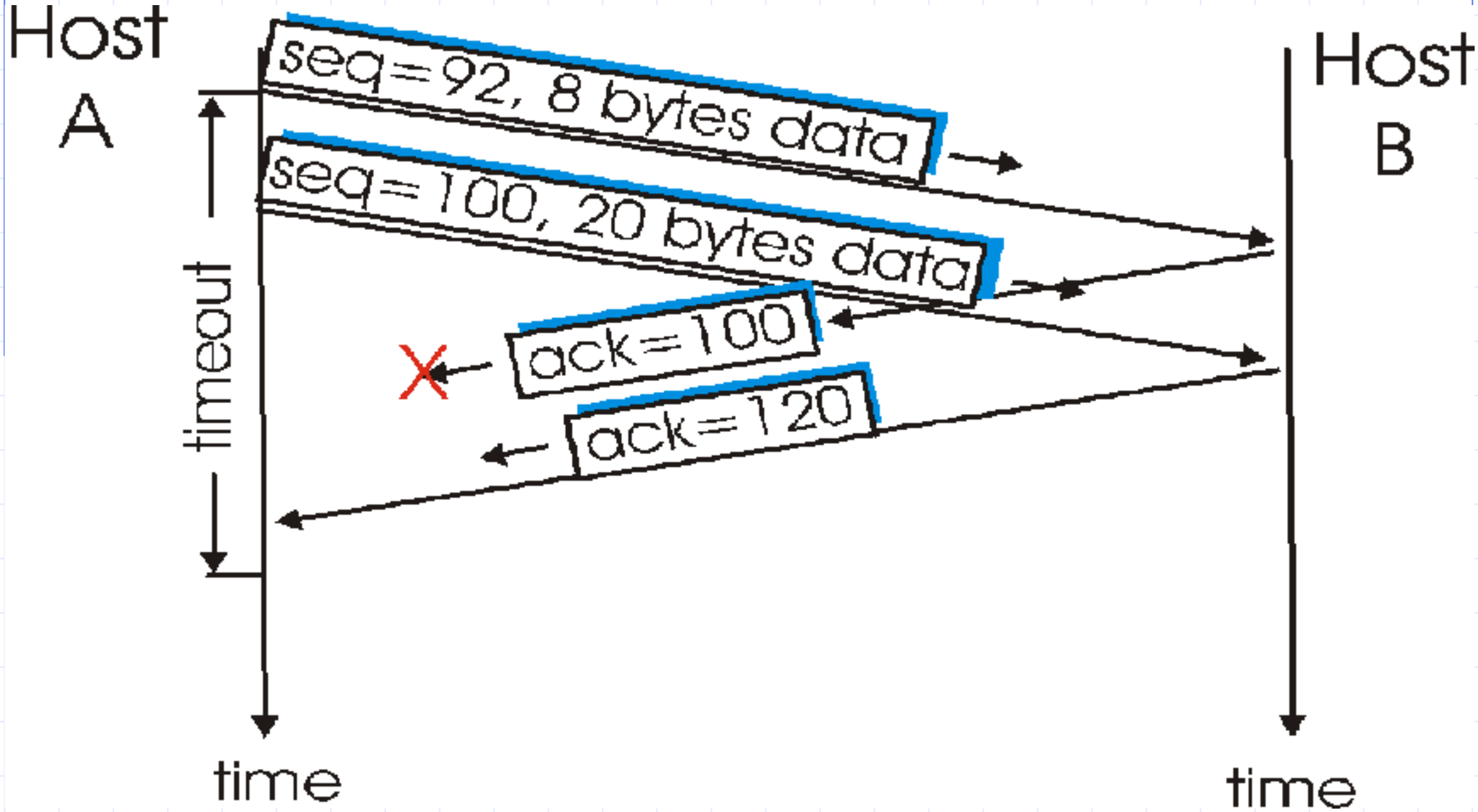
Retransmission-1



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 **congestion window**. –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,

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

- **`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

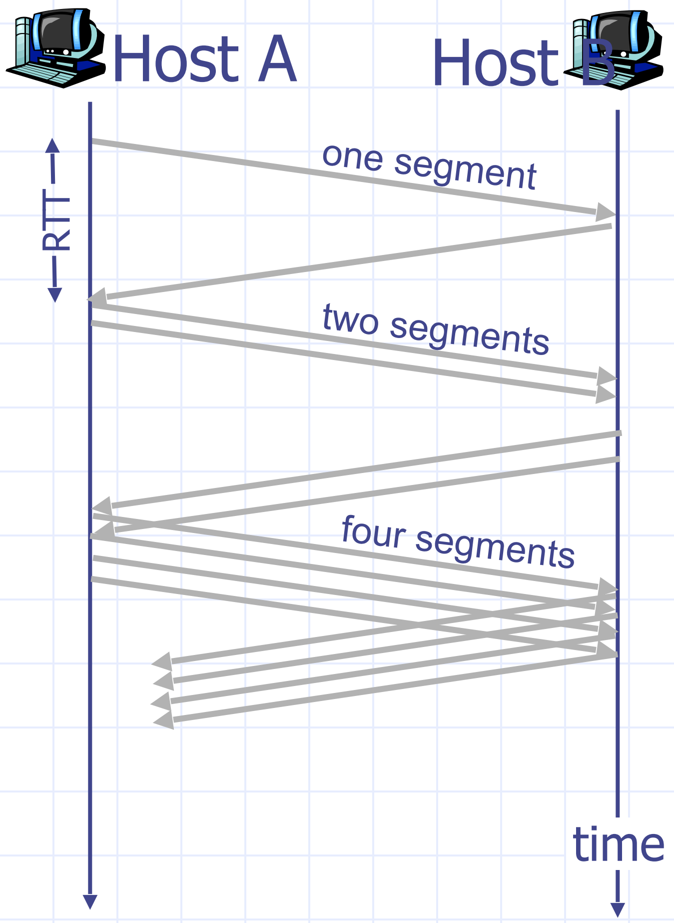
TCP Slow Start

- **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
- <u>Summary:</u> 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

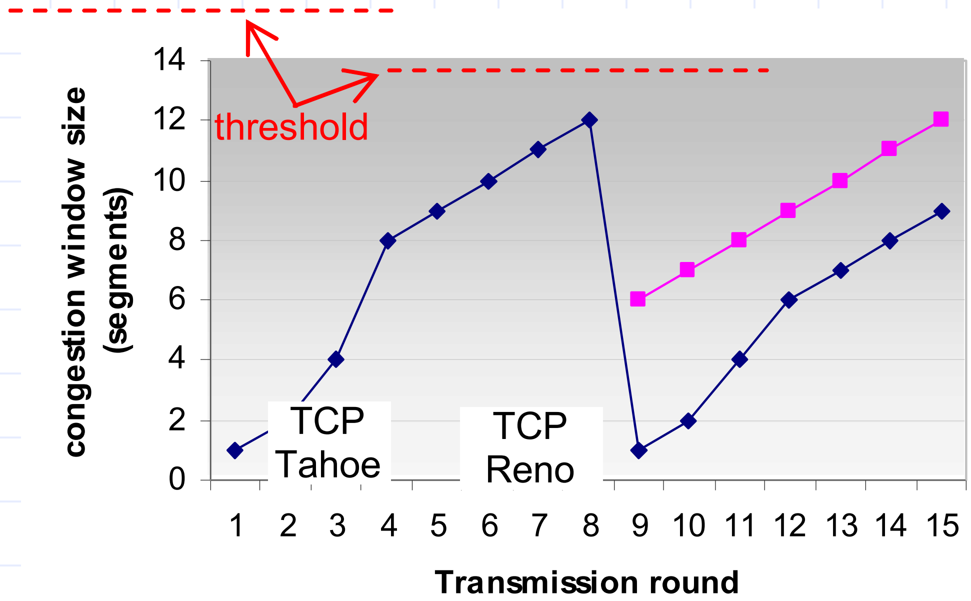
Philosophy:

- 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:

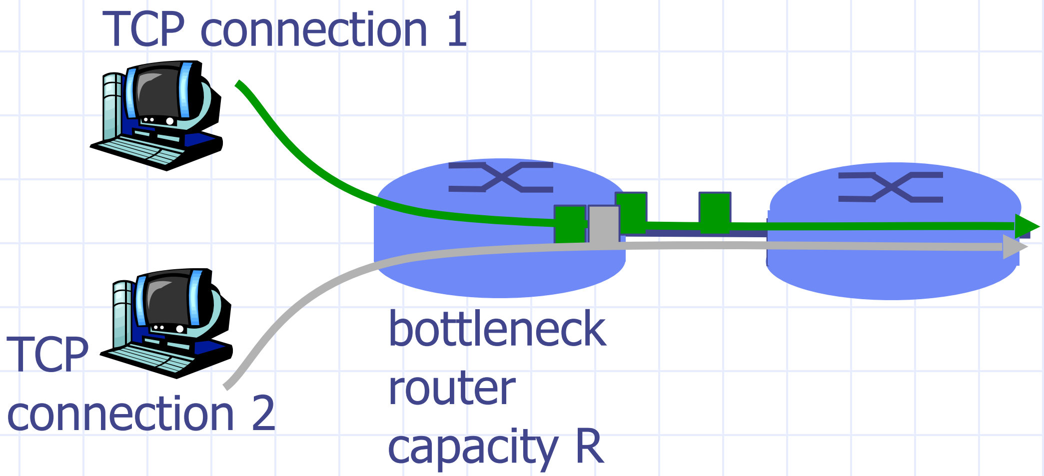
- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

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

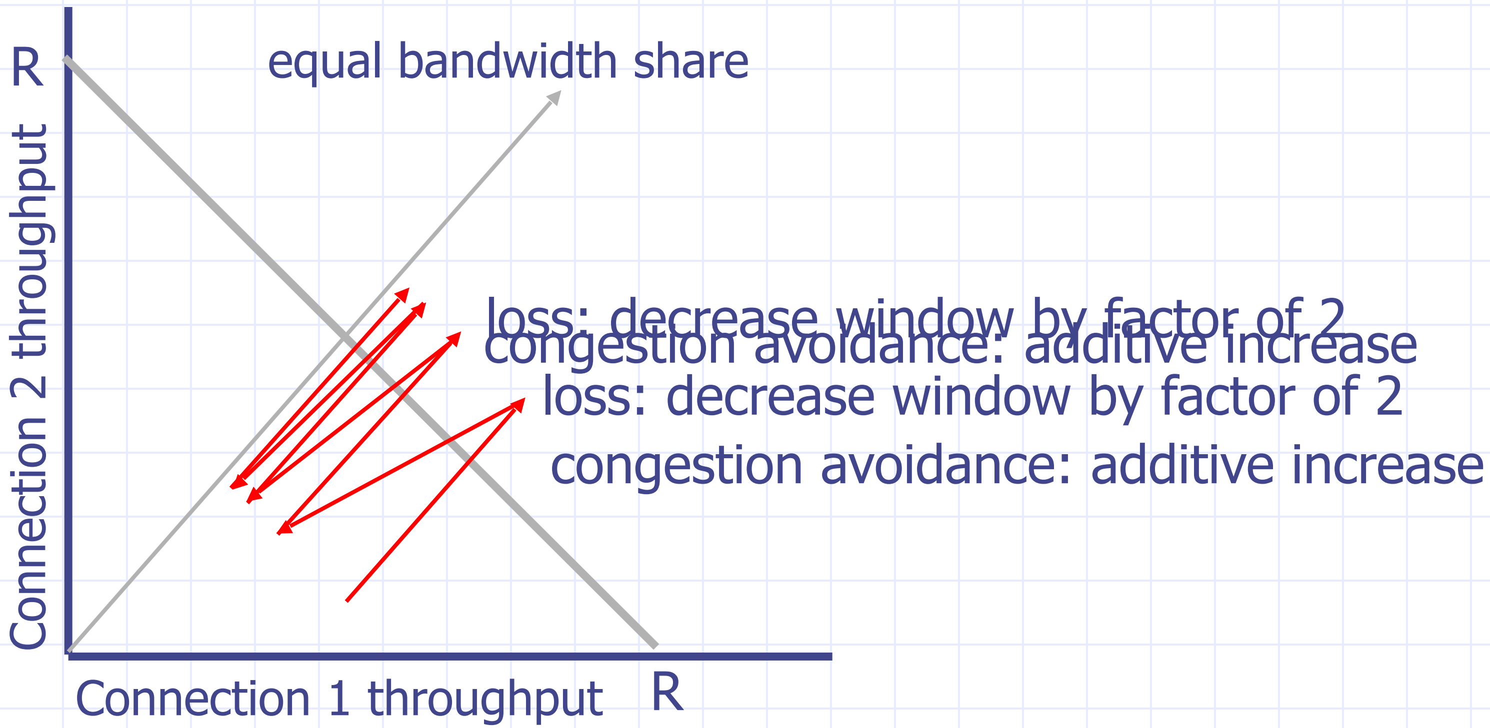
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



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 connections 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** !