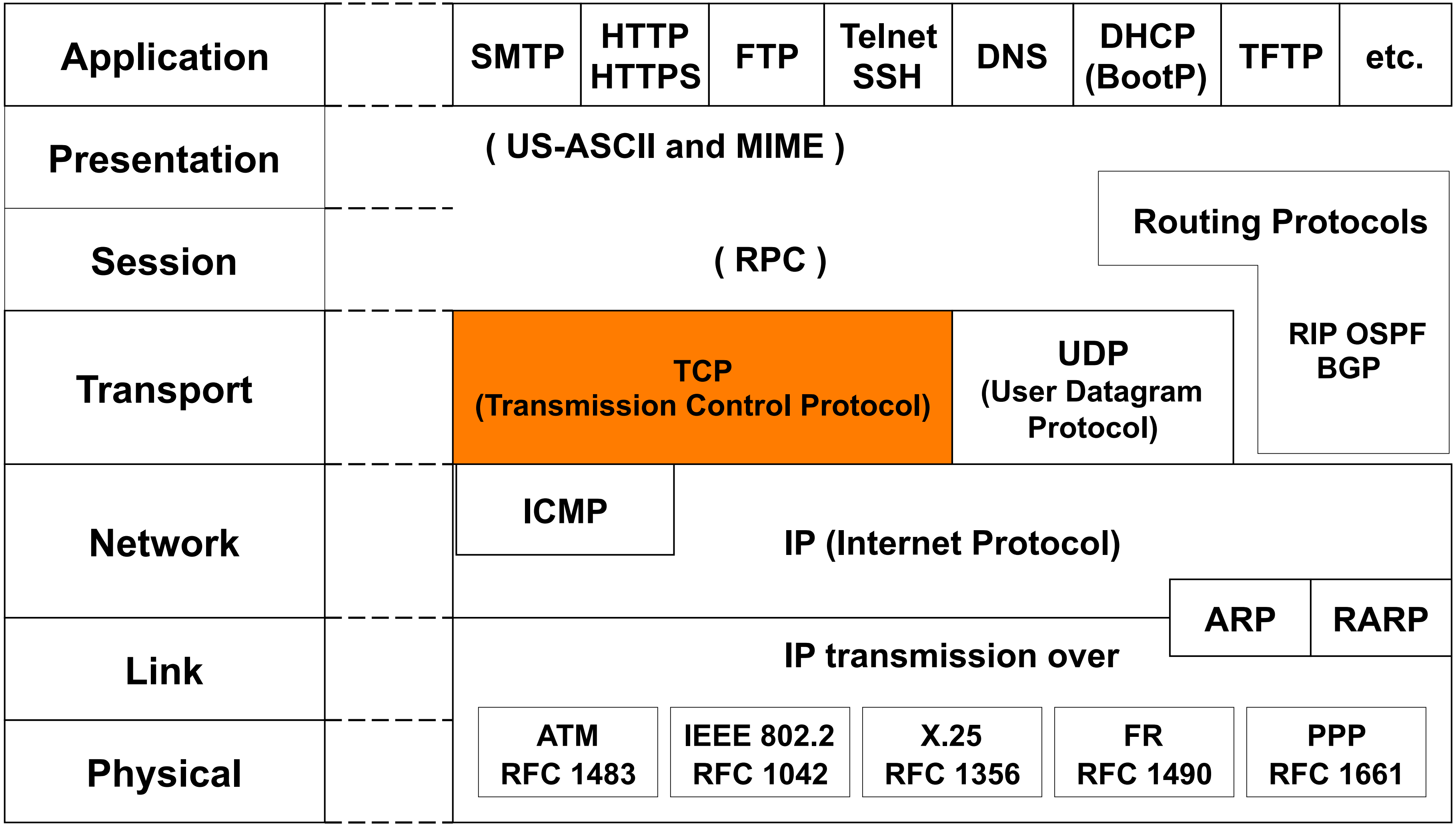
Internet Transport Layer

TCP Fundamentals, TCP Performance Aspects,
UDP (User Datagram Protocol),
NAT (Network Address Translation)

Agenda

- TCP Fundamentals
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- TCP Performance
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- UDP
- RFC Collection
- NAT

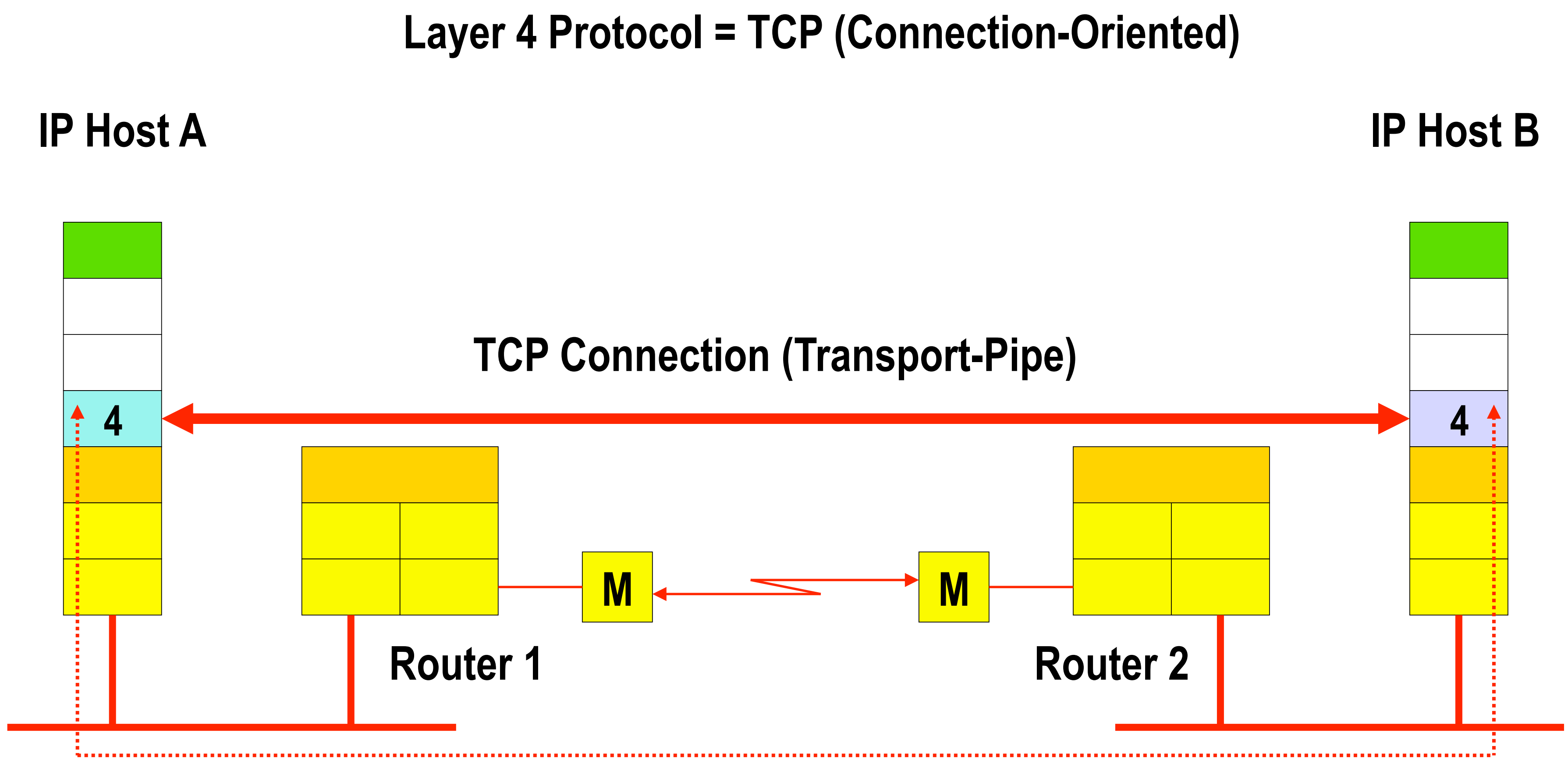
TCP/IP Protocol Suite



TCP (Transmission Control Protocol)

- **TCP is a connection oriented**
  - Call setup with "three way handshake"
- **Provides a reliable end-to-end transport of data between computer processes of different end systems**
  - Error detection and recovery
  - Maintaining the order of the data (sequencing) without duplication or loss
  - Flow control
- **Application's data is regarded as continuous byte stream**
  - TCP ensures a reliable transmission of segments of this byte stream
  - Handover to Layer 7 at so called "Ports"
    - OSI-Speak: Service Access Point
- **RFC 793**

TCP and OSI Transport Layer 4



TCP Protocol Functions

- **TCP transmission block**
  - Called segment transmitted inside IP datagram's payload field
- **ARQ Continuous Repeat Request**
  - With piggy-backed acknowledgments
- **Error recovery**
  - Positive & multiple acknowledgements using timeouts for each segment
    - Sequence numbers based on byte position within in the TCP stream
- **Flow control**
  - Sliding window and dynamically adjusted window size

TCP Ports

- **TCP provides its service to higher layers**
  - Through ports
- **Port numbers identify**
  - Communicating processes in an IP host
- **Using port numbers**
  - TCP can multiplex different layer-7 byte streams
- **Server processes are identified by**
  - Well known port numbers : 0..1023
  - Controlled by IANA
- **Client processes use**
  - Arbitrary port numbers > 1023
  - Better > 8000 because of registered ports

Well Known Ports

Some Well Known Ports

7	Echo
20	FTP (Data), File Transfer Protocol
21	FTP (Control)
23	TELNET, Terminal Emulation
25	SMTP, Simple Mail Transfer Protocol
53	DNS, Domain Name Server
69	TFTP, Trivial File Transfer Protocol
80	HTTP Hypertext Transfer Protocol
111	Sun Remote Procedure Call (RPC)
137	NetBIOS Name Service
138	NetBIOS Datagram Service
139	NetBIOS Session Service
161	SNMP, Simple Network Management Protocol
162	SNMPTRAP
322	RTSP (Real Time Streaming Protocol) Server

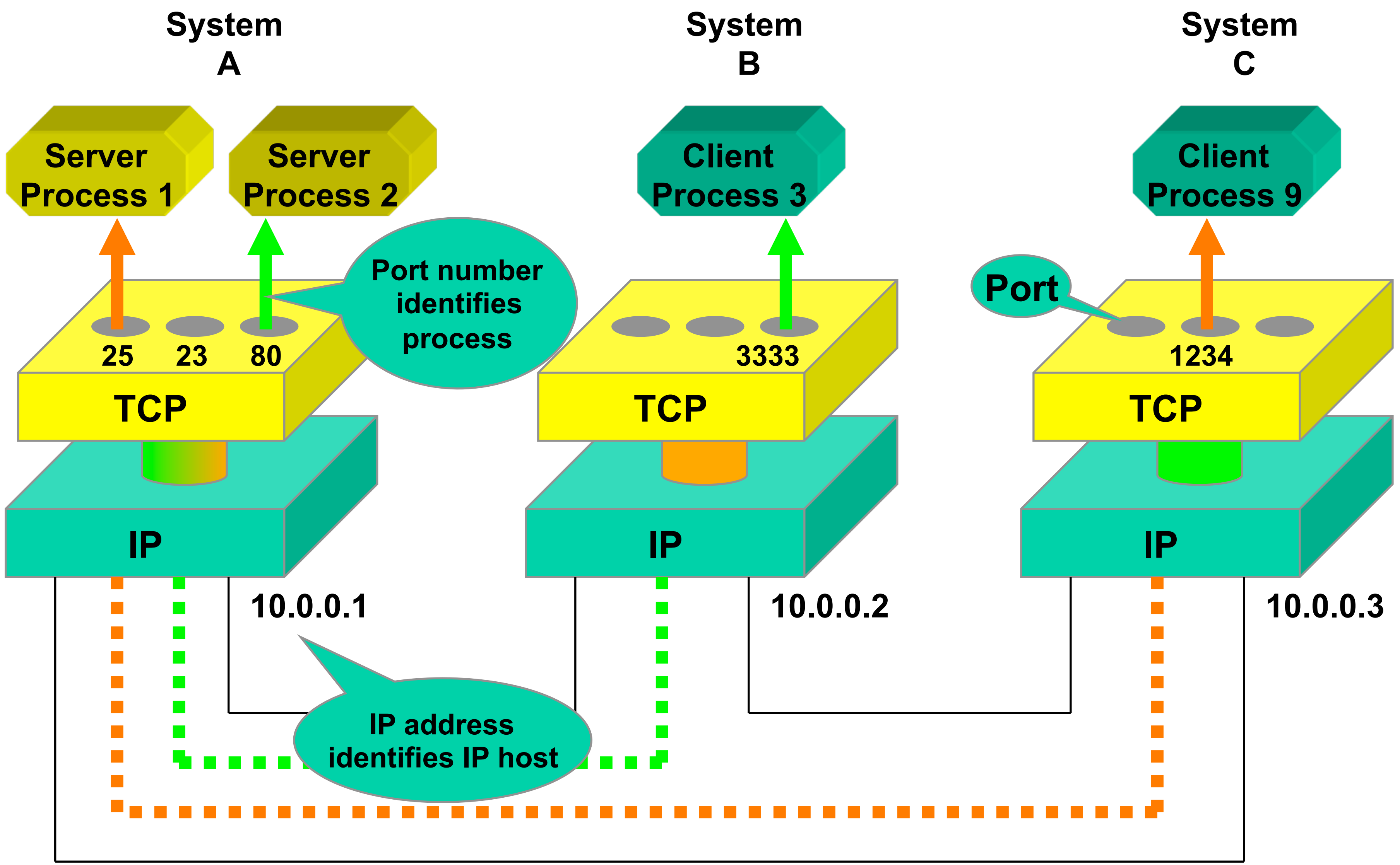
Some Registered Ports

1416	Novell LU6.2
1433	Microsoft-SQL-Server
1439	Eicon X25/SNA Gateway
1527	Oracle
1986	Cisco License Manager
1998	Cisco X.25 service (XOT)
5060	SIP (VoIP Signaling)
6000	\
.....	> X Window System
6063	/

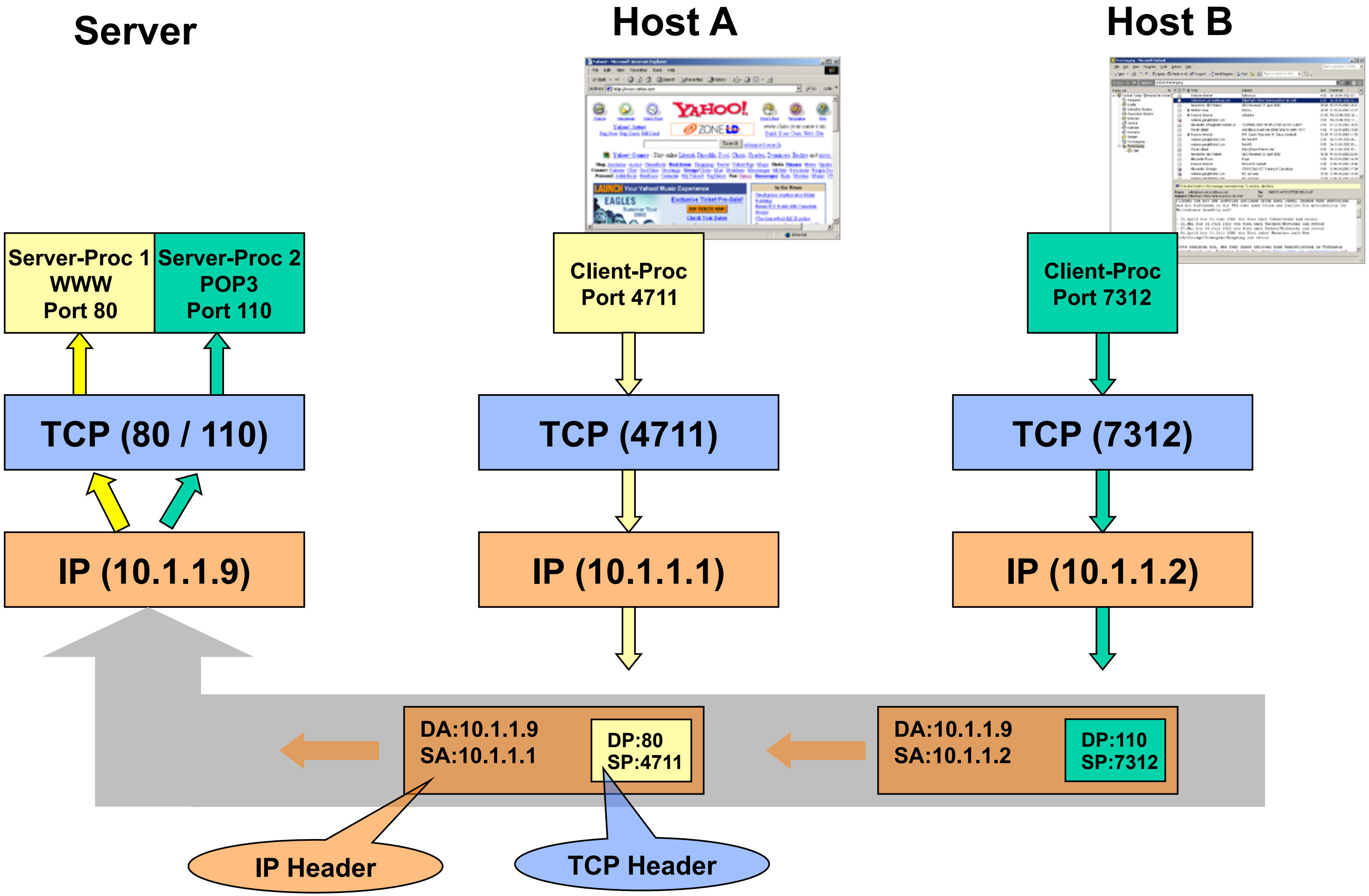
... etc.

(see RFC1700)

TCP Ports and TCP Connections



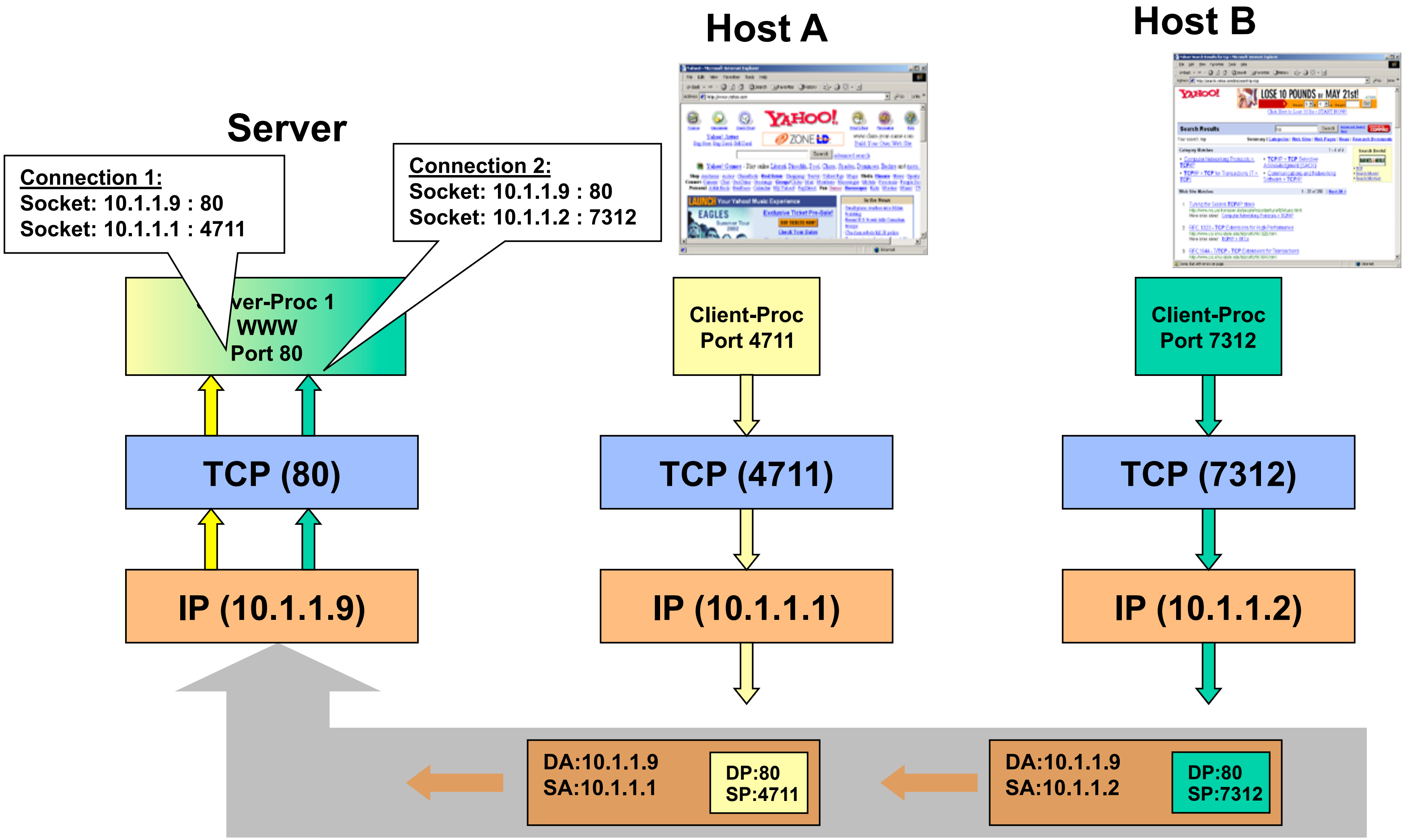
Example 1: TCP Port



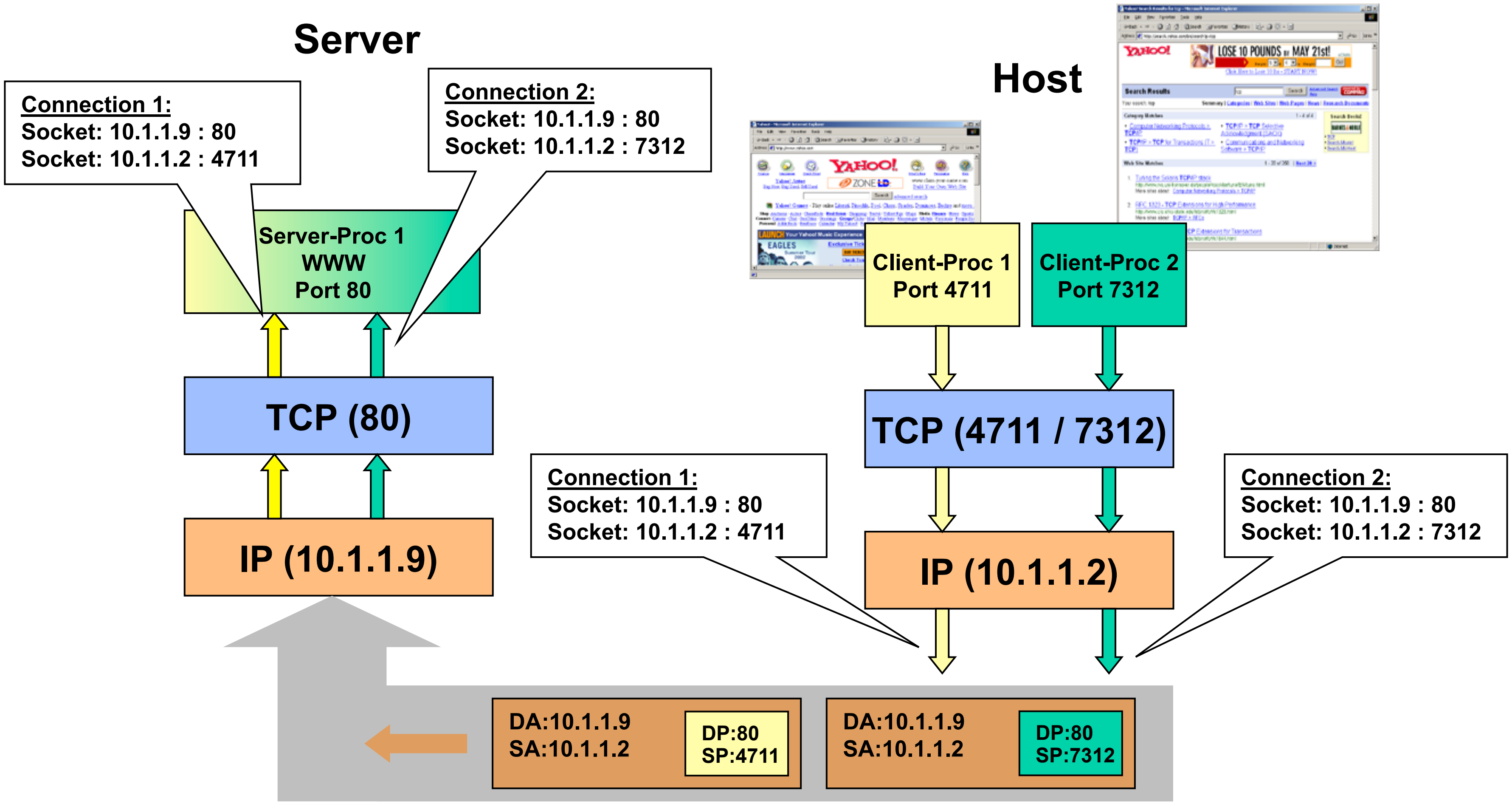
TCP Sockets and TCP Connection

- **Client-server environment**
  - Server-process has to maintain several TCP connections = TCP streams ("flow") to different targets at the same time
  - Hence a single port at the server side has to multiplex several virtual connections
- **How to distinguish these connections?**
  - Usage of so called <u>sockets</u>
- **Socket**
  - Combination IP address and port number
    - Note: similar to the OSI "CEP" Connection Endpoint Identifier
    - E.g.: 10.1.1.2:80 [IP-Address : Port-Number]
- **Each TCP connection is uniquely identified by**
  - A <u>pair of sockets</u>
    - Source-IP, Source-Port, Destination-IP, Destination-Port

Example 2: TCP Socket



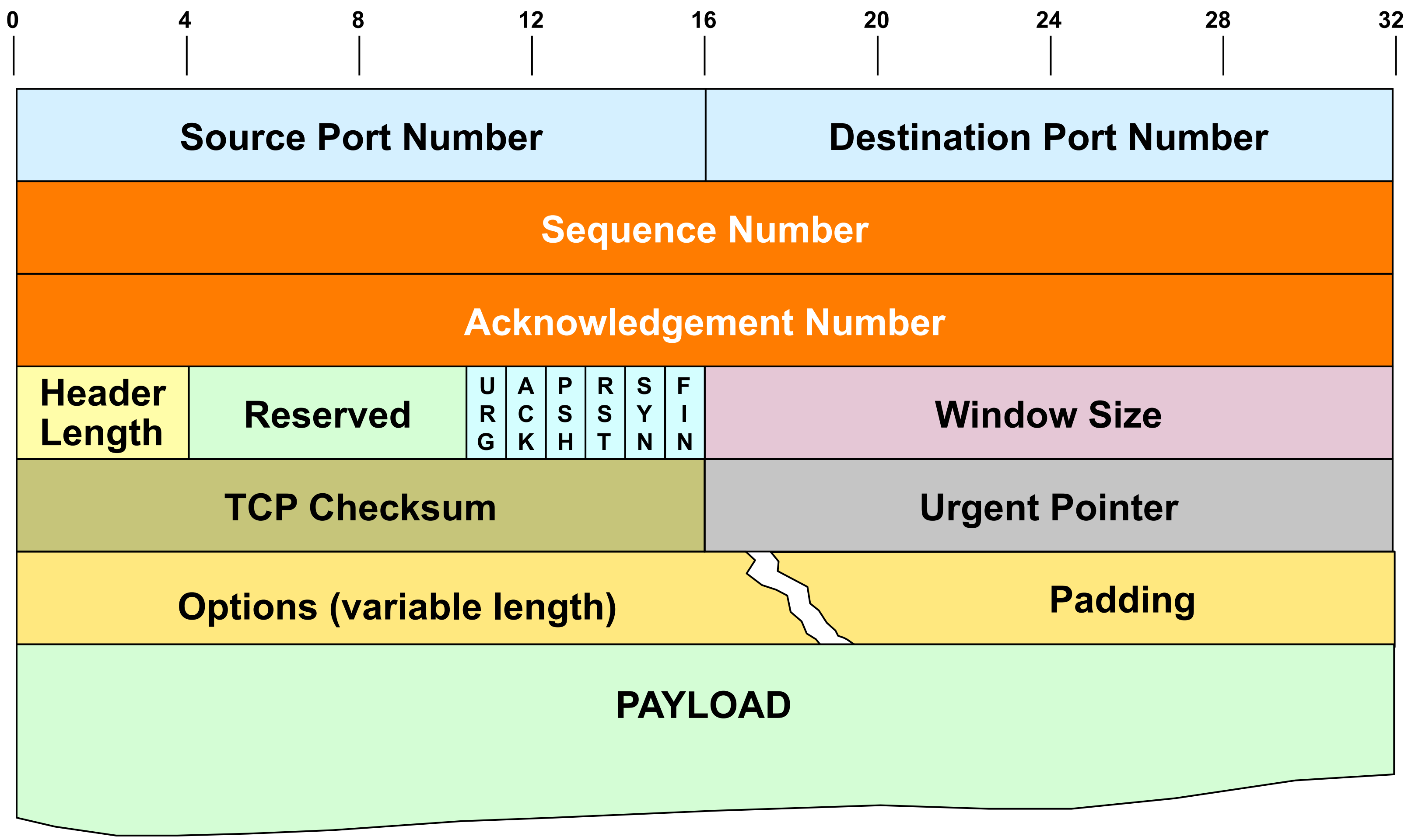
Example 3: TCP Socket



Agenda

- **<u>TCP Fundamentals</u>**
  - Principles, Port and Sockets
  - <u>Header Fields</u>
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

TCP Header


0
4
8
12
16
20
24
28
32
Source Port Number
Destination Port Number
Sequence Number
Acknowledgement Number
Header Length
Reserved
URG
ACK
PSH
RST
SYN
FIN
Window Size
TCP Checksum
Urgent Pointer
Options (variable length)
Padding
PAYLOAD

TCP Header Entries (1)

- **Source and Destination Port**
  - 16 bit port number for source and destination process
- **Header Length**
  - Indicates the length of the header given as a multiple 4 bytes
  - Necessary, because of the variable header length in case of options
- **Sequence Number (32 Bit)**
  - Position number of the first byte of this segment
    - In relation to the byte stream flowing through a TCP connection
  - Wraps around to 0 after reaching 2^{32} -1
- **Acknowledge Number (32 Bit)**
  - Number of next byte expected by receiver
  - Acknowledges the correct reception of all bytes up to ACK-number minus 1

TCP Header Entries (2)

- **SYN-Flag**
  - Indicates a connection request
  - Sequence number synchronization
- **ACK-Flag**
  - Acknowledge number is valid
  - Always set, except in very first segment
- **FIN-Flag**
  - Indicates that this segment is the last
  - Other side must also finish the conversation
- **RST-Flag**
  - Immediately kill the conversation
  - Used to refuse a connection-attempt

TCP Header Entries (3)

- **PSH-Flag**
  - TCP should push the segment immediately to the application without buffering
  - To provide low-latency connections
  - Often ignored

TCP Header Entries (4)

- **URG-Flag**
  - Indicates urgent data
  - If set, the 16-bit "Urgent Pointer" field is valid and points to the last byte of urgent data
  - There is no way to indicate the beginning of urgent data (!)
  - Applications switch into the "urgent mode"
  - Used for quasi outband signaling
- **Urgent Pointer**
  - Points to the last octet of urgent data

TCP Header Entries (5)

- **Window (16 Bit)**
  - Adjusts the send-window size of the other side
  - Flow control STOP and GO
  - Receiver-based flow control
  - Used with every segment
  - Sequence number of last byte allowed to send = ACK number + window value seen in this segment

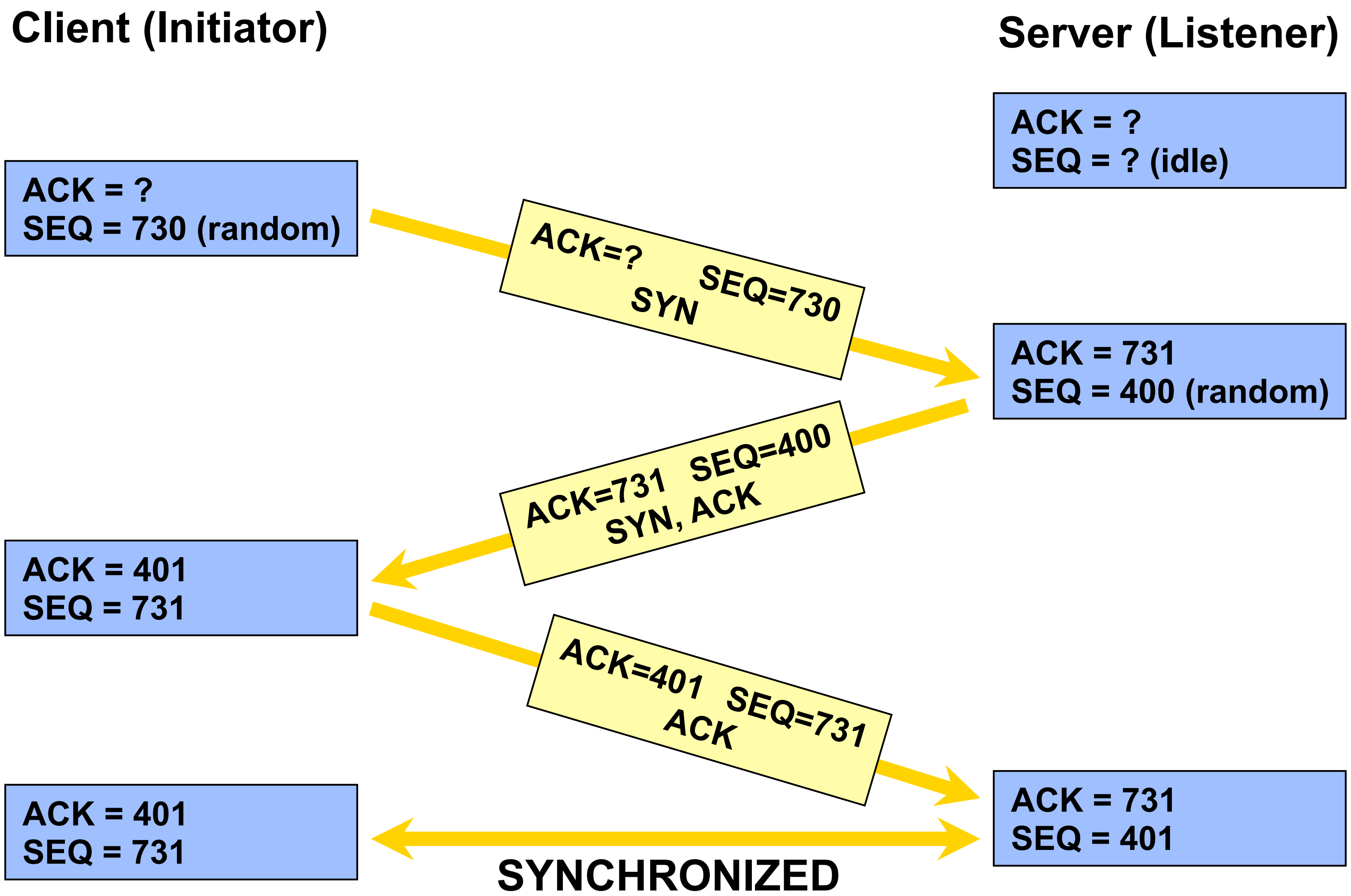
TCP Header Entries (6)

- **Checksum**
  - Calculated over TCP header, payload and 12 byte pseudo IP header
  - Pseudo IP header consists of source and destination IP address, IP protocol type, and IP total length
  - Complete socket information is protected
  - Thus TCP can also detect IP errors
- **Options**
  - Only MSS (Maximum Message Size) is used
  - Other options are defined in RFC1146, RFC1323 and RFC1693
- **Pad**
  - Ensures 32 bit alignment

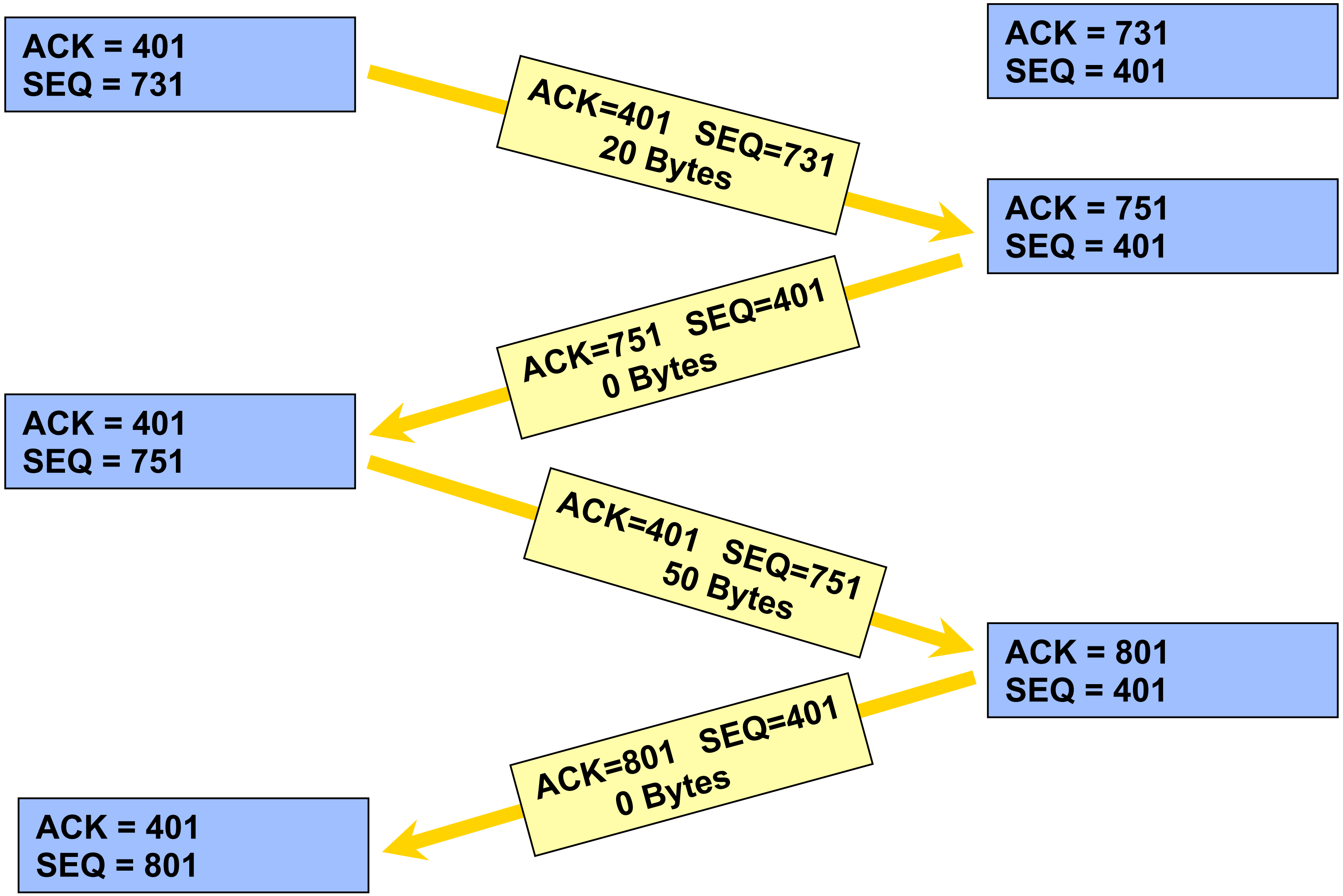
Agenda

- TCP Fundamentals
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- TCP Performance
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- UDP
- RFC Collection
- NAT

TCP 3-Way-Handshake



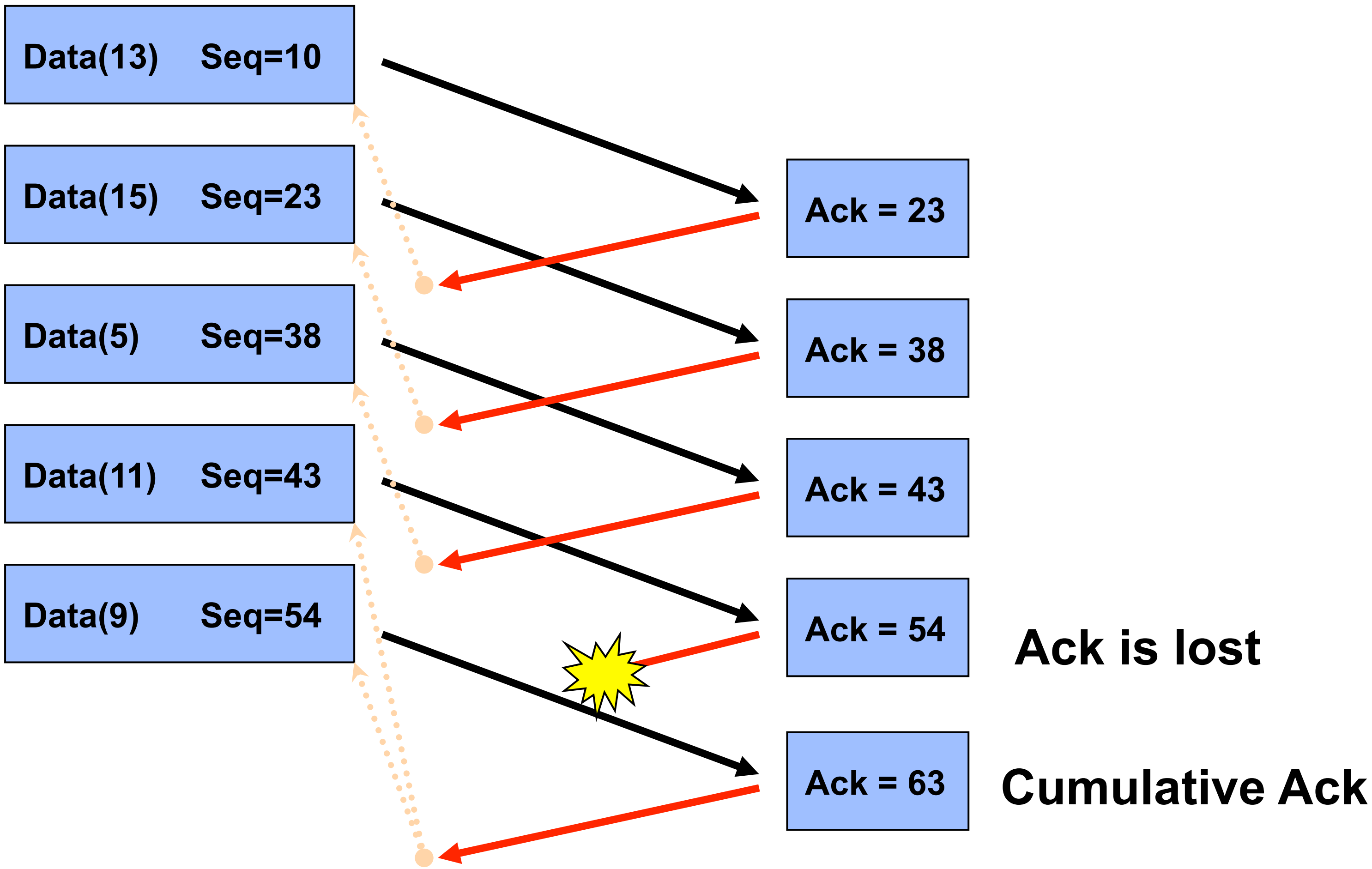
TCP Data Transfer



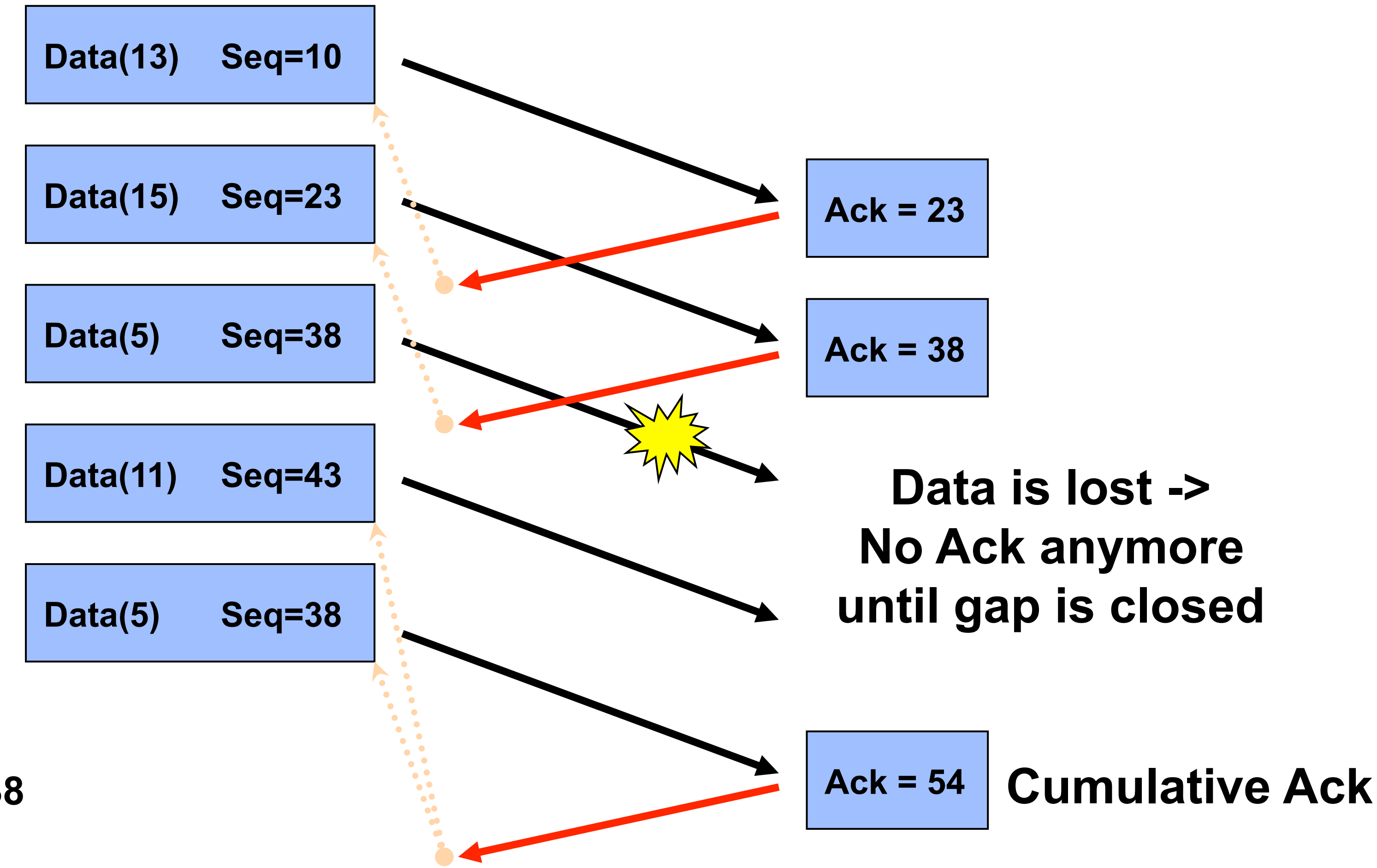
TCP Data Transfer

- **Acknowledgements are generated for all bytes which arrived <u>in sequence without errors</u>**
  - Positive acknowledgement
- **If a segment arrives out of sequence, no acknowledges are sent until this "gap" is closed (old TCP)**
  - Timeout will initiate a retransmission of unacknowledged data
- **Duplicates are also acknowledged (!)**
  - Receiver cannot know why duplicate has been sent; maybe because of a lost acknowledgement
- **The acknowledge number indicates the sequence number of the next byte to be received**
- **Acknowledgements are cumulative**
  - Ack(N) confirms all bytes with sequence numbers up to N-1
  - Therefore lost acknowledgements are no problem

Cumulative Acknowledgement

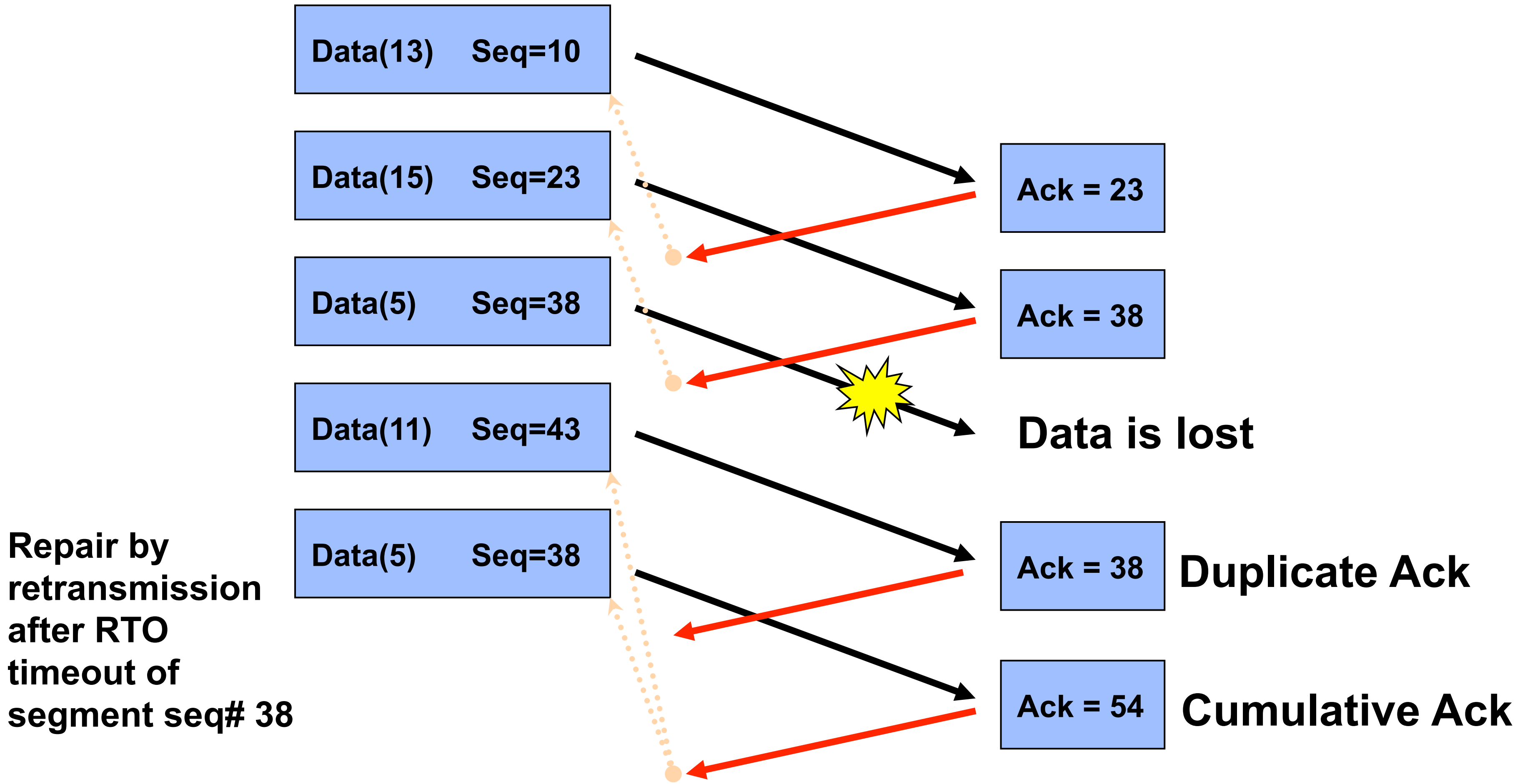


TCP Duplicates, Lost Original (old TCP)

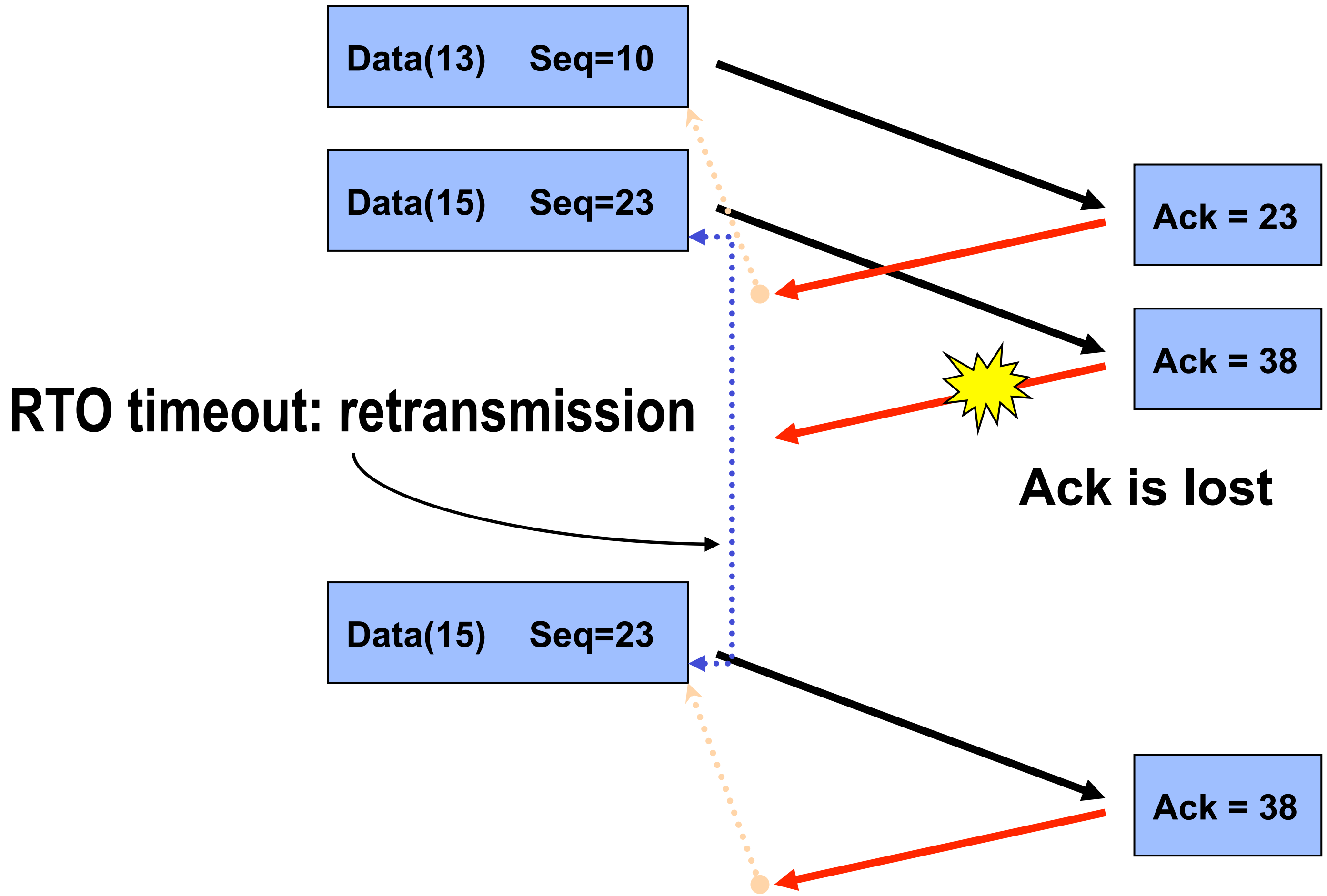


Repair by retransmission after RTO timeout of segment seq# 38

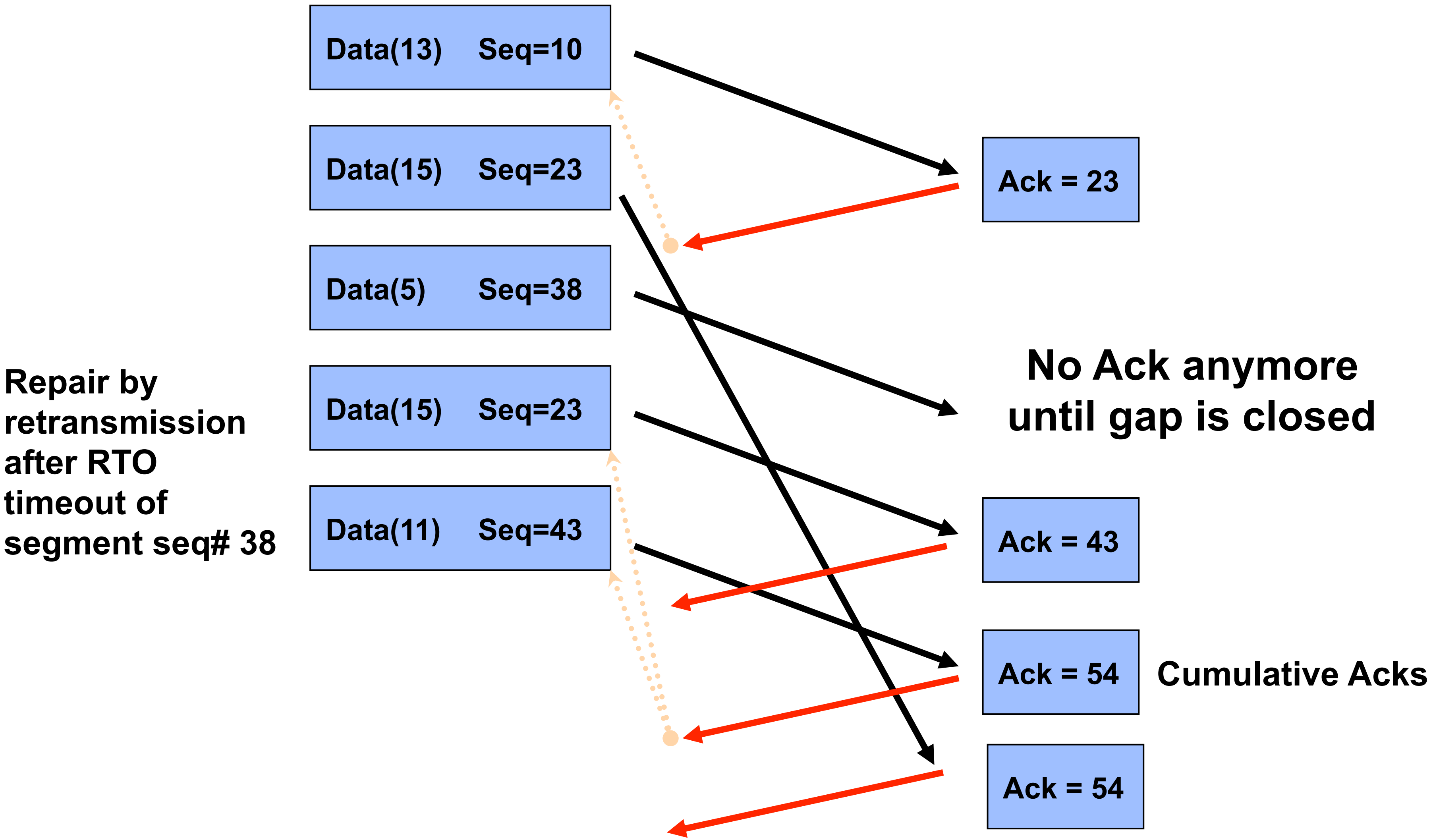
Duplicate Acknowledgement (new TCP)



TCP Duplicates, Lost Acknowledgement



TCP Duplicates, Delayed Original



TCP Retransmission Timeout

- **Retransmission timeout (RTO) will initiate a retransmission of unacknowledged segments**
  – High timeout results in long idle times if an error occurs
  – Low timeout results in unnecessary retransmissions
- **Constant timeout will never fit**
  – Remember: RTT is a statistic value in the packet switching world
- **Adaptive timeout is necessary**
- **For TCP's performance a precise estimation of the current RTT is crucial**
  – TCP continuously measures RTT to adapt RTO

Retransmission Ambiguity Problem

- **If a segment has been retransmitted and an ACK follows: Does this ACK belong to the retransmission or to the original packet?**
  - Could distort RTT measurement dramatically
- **Solution: Phil Karn's algorithm**
  - Ignore ACKs of a retransmission for the RTT measurement
  - And use an exponential backoff method

RTT Estimation

FYI

- **Originally a smooth RTT estimator was used (a low pass filter)**
  - M denotes the observed RTT (which is typically imprecise because there is no one-to-one mapping between data and ACKs)
  - $R = \alpha R + (1 - \alpha) M$ with smoothing factor $\alpha = 0.9$
  - Finally $RTO = \beta \cdot R$ with variance factor $\beta = 2$

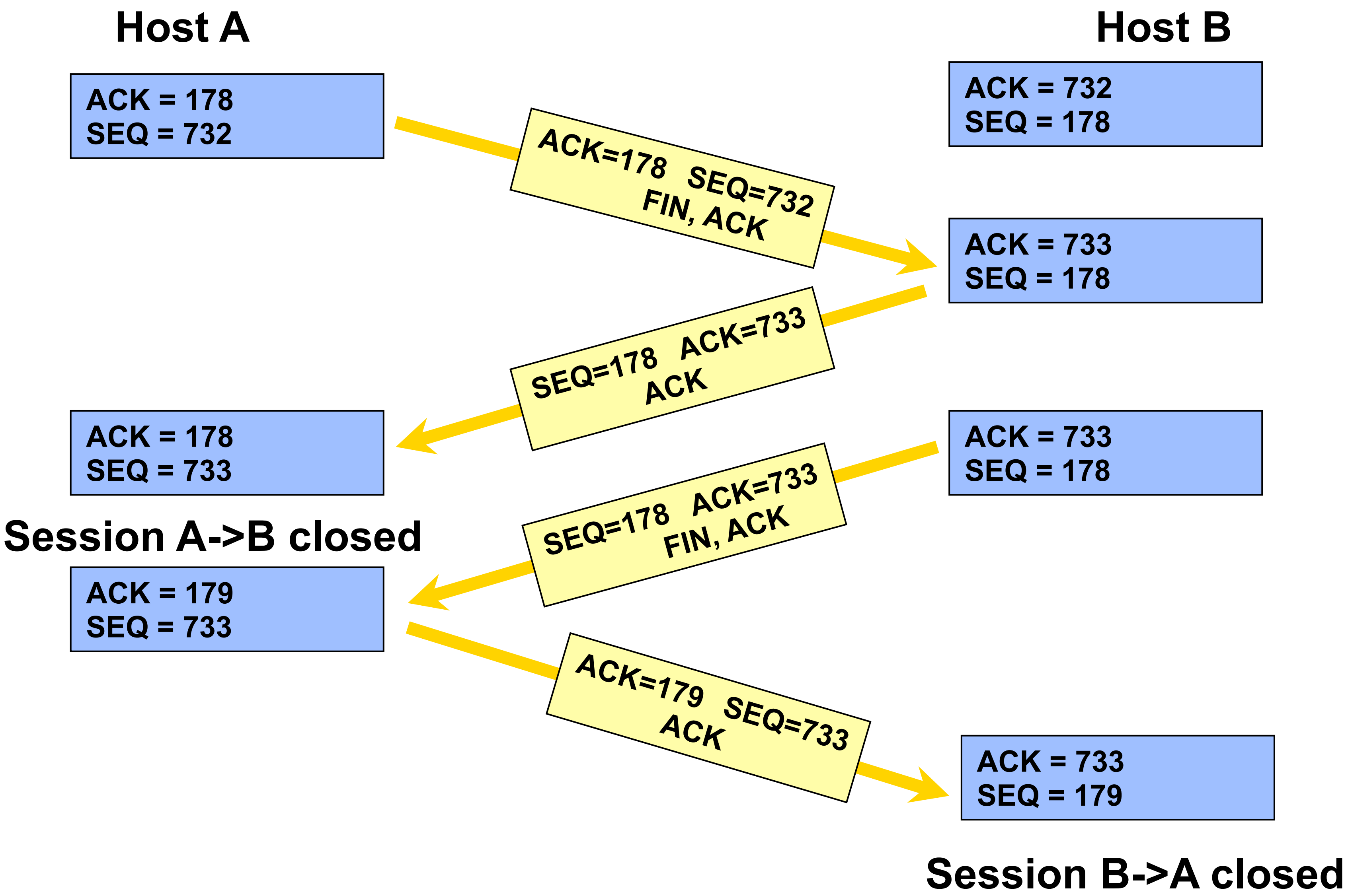
- **Initial smooth RTT estimator could not keep up with wide fluctuations of the RTT**
  - Led to too many retransmissions

- **Jacobson's suggested to take the RTT variance also into account**
  - $Err = M - A$
    - The deviation from the measured RTT (M) and the RTT estimation (A)
  - $A = A + g \cdot Err$
    - with gain $g = 0.125$
  - $D = D + h\,(|Err| - D)$
    - with $h = 0.25$
  - $RTO = A + 4D$

TCP Keepalive Timer

- **Note that absolutely no data flows during an idle TCP connection!**
  - Even for hours, days, weeks!
- **Usually needed by a server that wants to know which clients are still alive**
  - To close stale TCP sessions
- **Many implementations provide an optional TCP keepalive mechanism**
  - Not part of the TCP standard!
  - Not recommended by RFC 1122 (TCP/IP hosts requirements)
  - Minimum interval must be 2 hours

TCP Disconnect



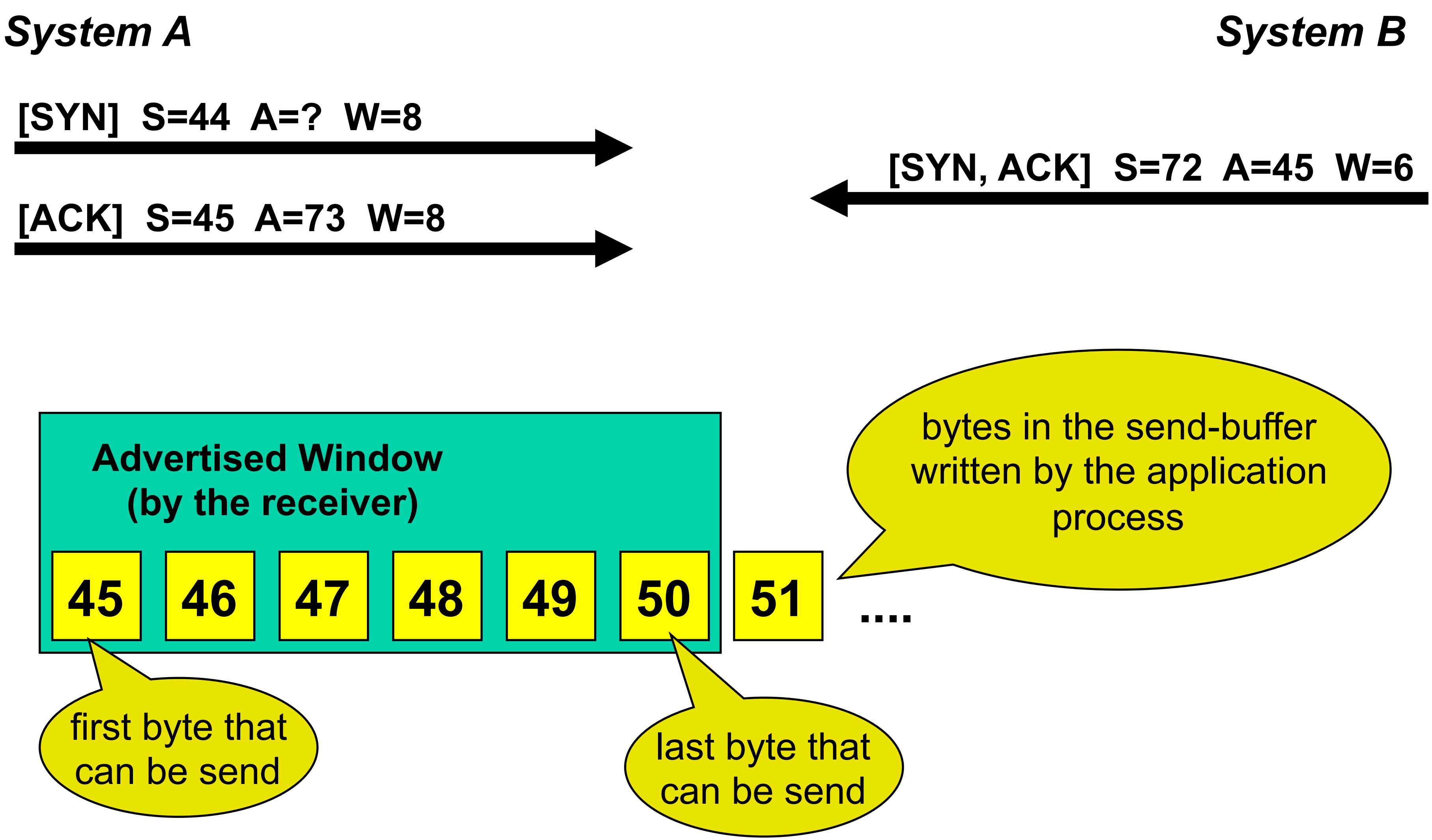
Agenda

- **<u>TCP Fundamentals</u>**
  – Principles, Port and Sockets
  – Header Fields
  – Three Way Handshake
  – <u>Windowing</u>
  – Enhancements
- **TCP Performance**
  – Slow Start and Congestion Avoidance
  – Fast Retransmit and Fast Recovery
  – TCP Window Scale Option and SACK Options
  – Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

Flow control: "Sliding Window"

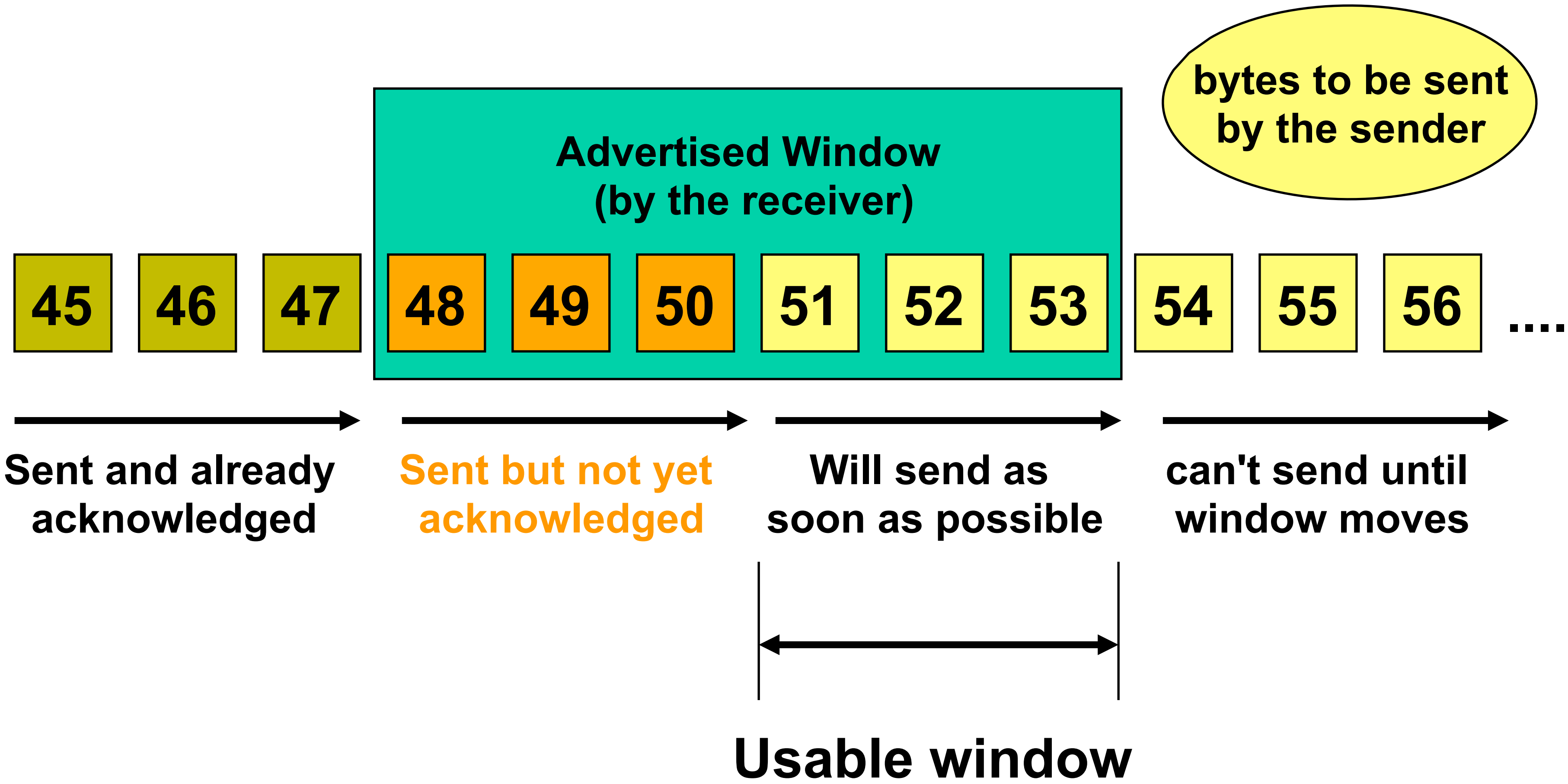
- **TCP flow control is done with dynamic windowing using the sliding window protocol**
- **The receiver advertises the current amount of octets it is able to receive**
  - Using the window field of the TCP header
  - Values 0 through 65535
- **Sequence number of the last octet a sender may send = received ack-number -1 + window size**
  - The starting size of the window is negotiated during the connect phase
  - The receiving process can influence the advertised window, hereby affecting the TCP performance

Sliding Window: Initialization

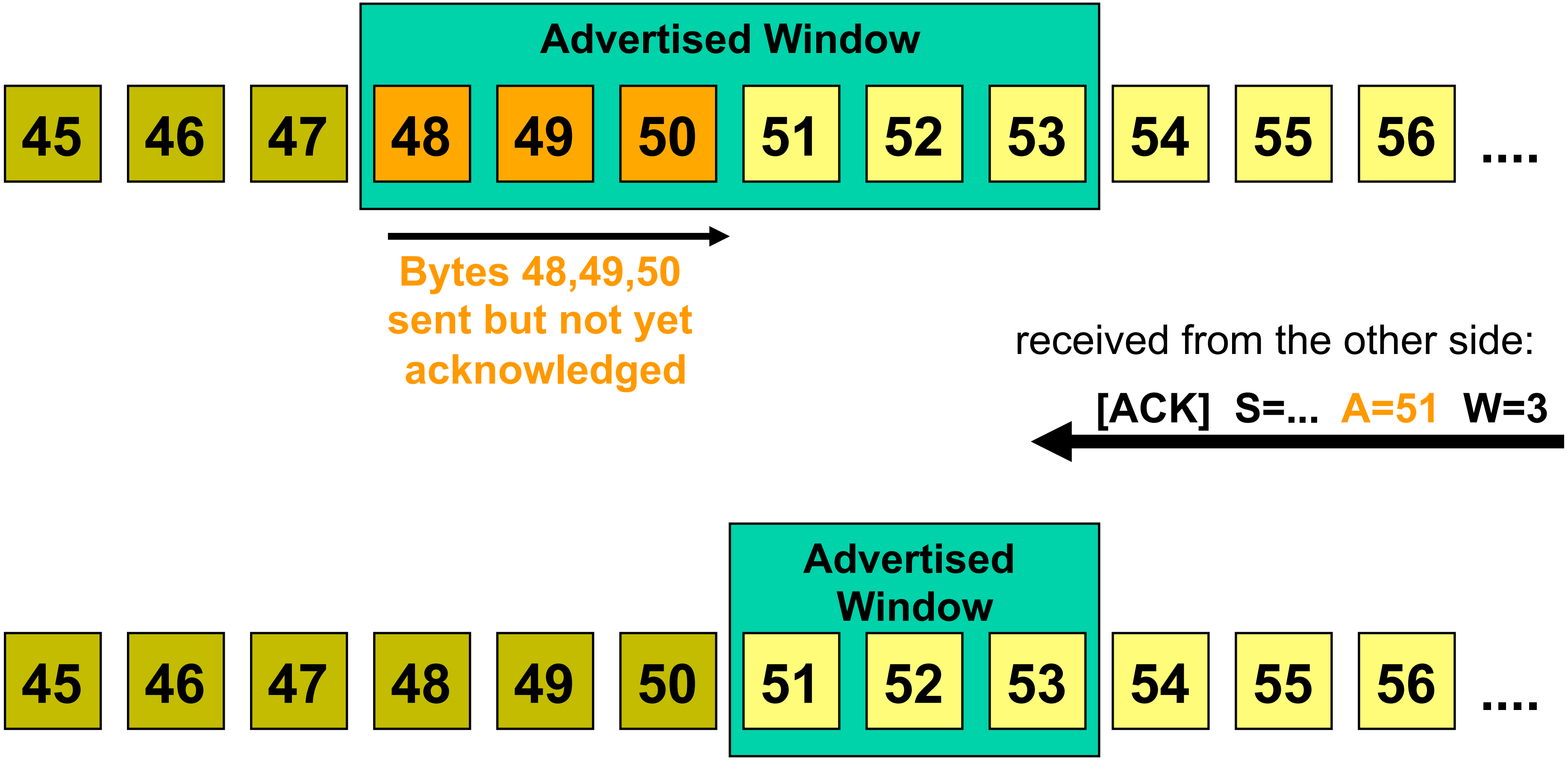


Sliding Window: Principle

Sender's (System A) point of view after sender got {ACK=48, WIN=6} from the receiver (System B)

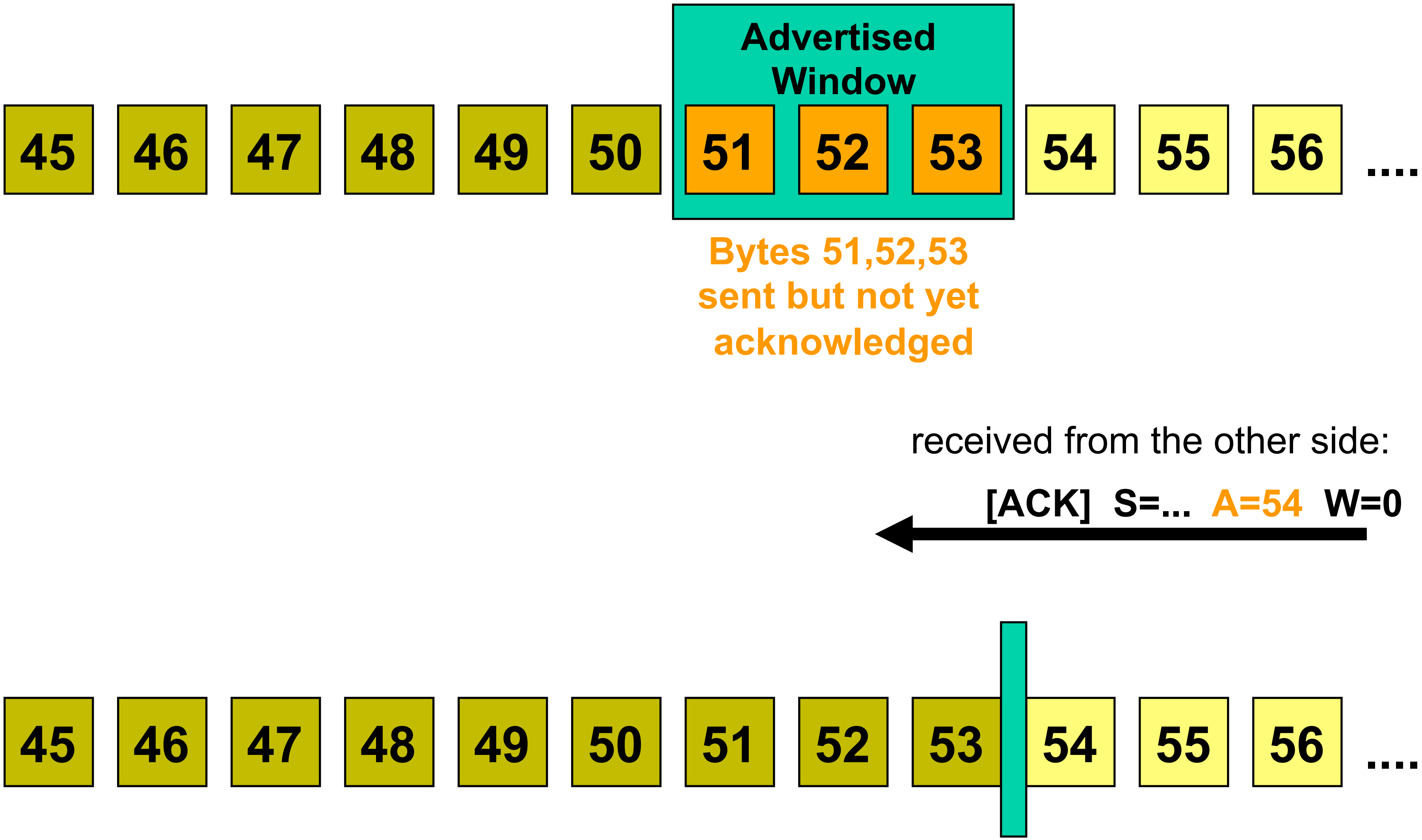


Closing the Sliding Window

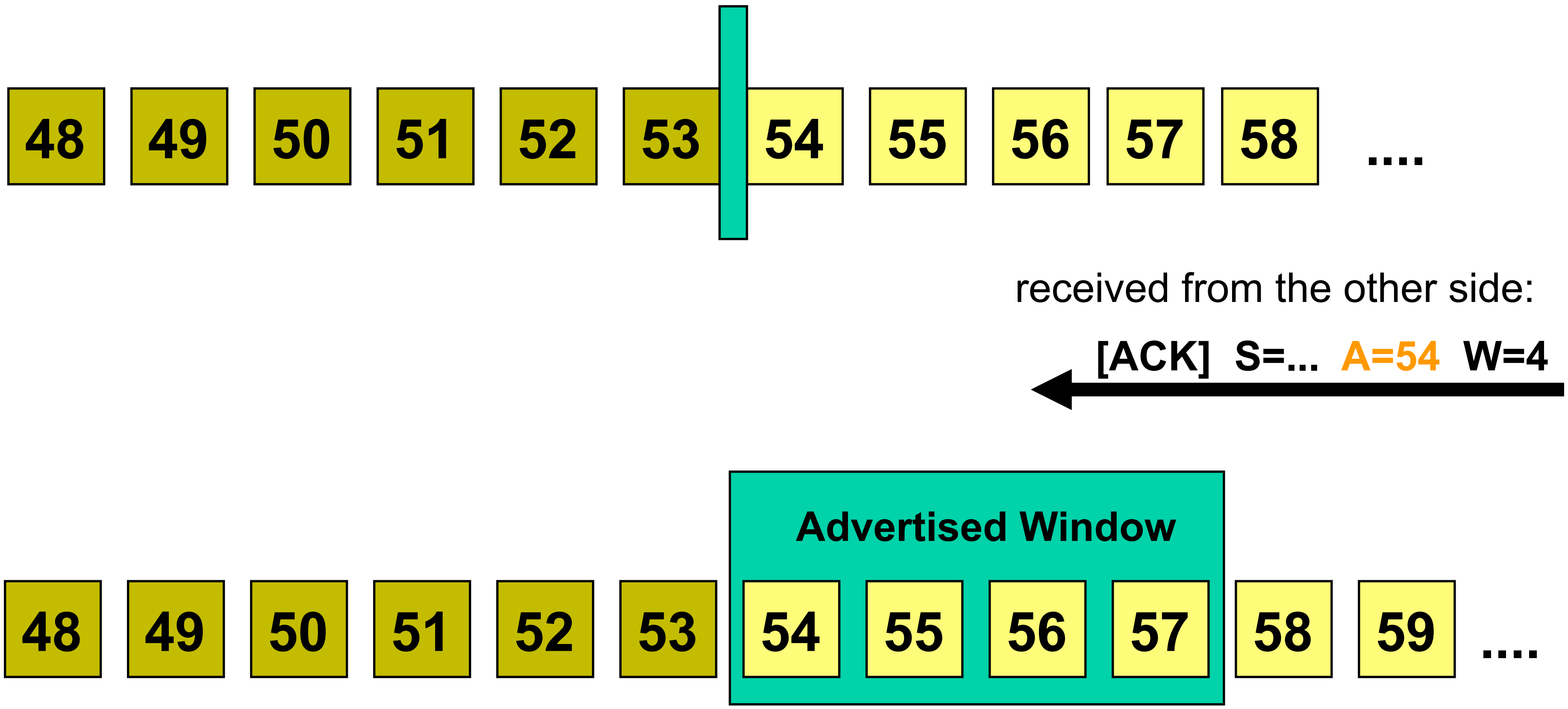


Now the sender may send bytes 51, 52, 53. The receiver didn't open the window (W=3, right edge remains constant) because of congestion. However, the remaining three bytes inside the window are already granted, so the receiver cannot move the right edge leftwards.

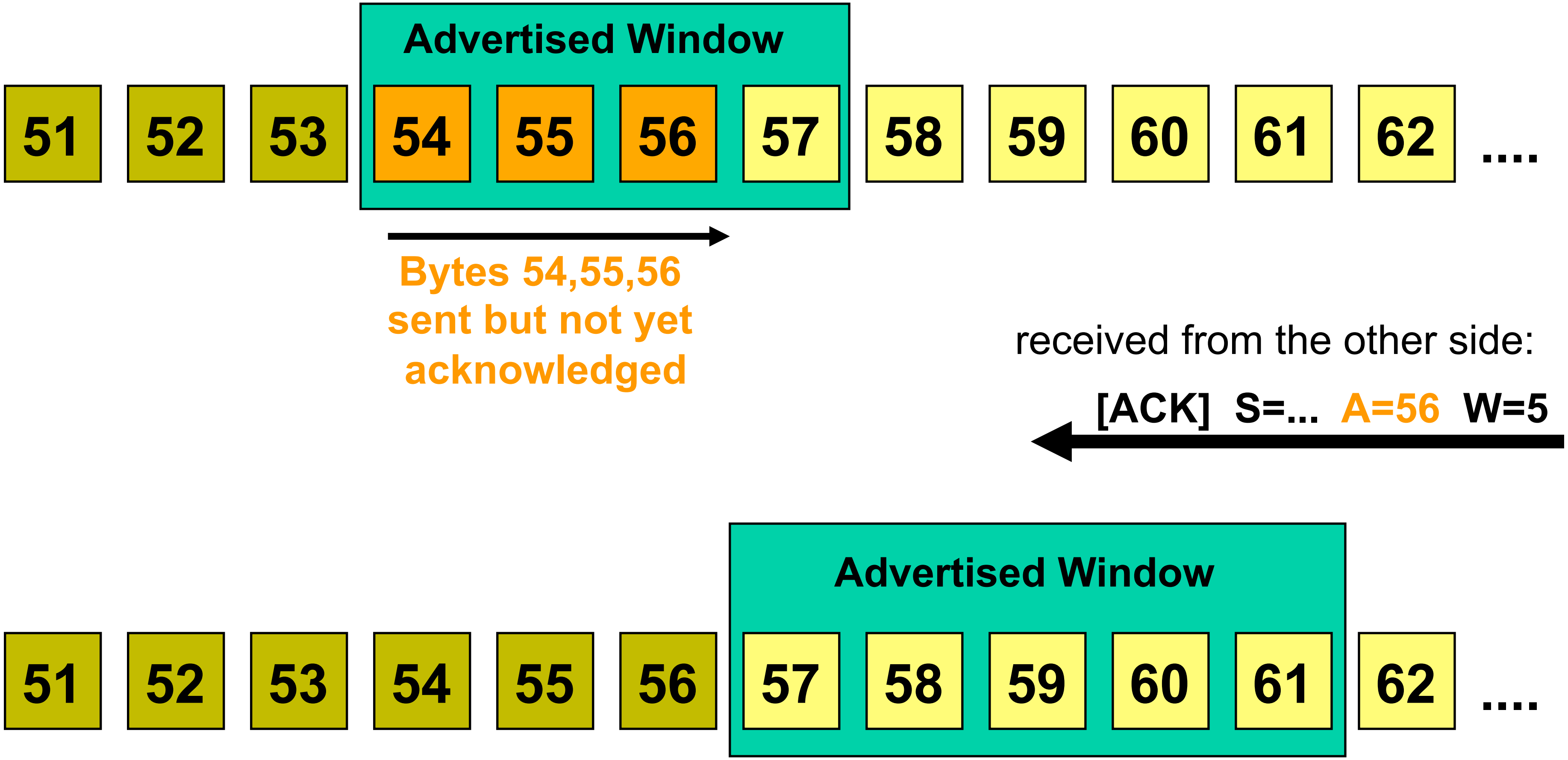
Flow Control -> STOP, Window Closed



Opening the Window -> Flow Control GO

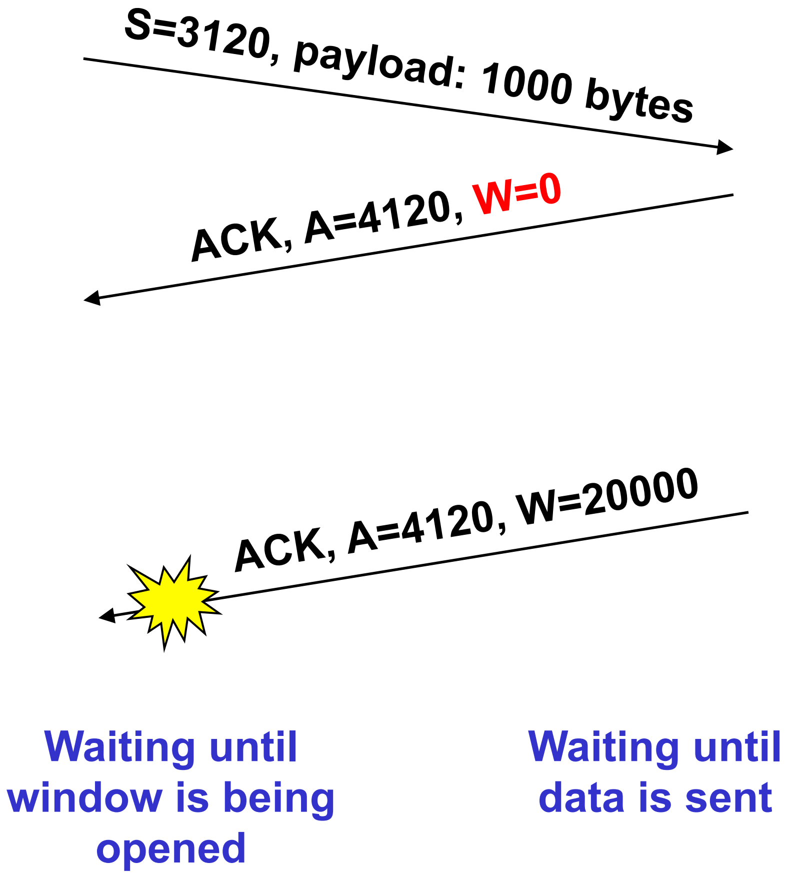


Increasing the Sliding Window



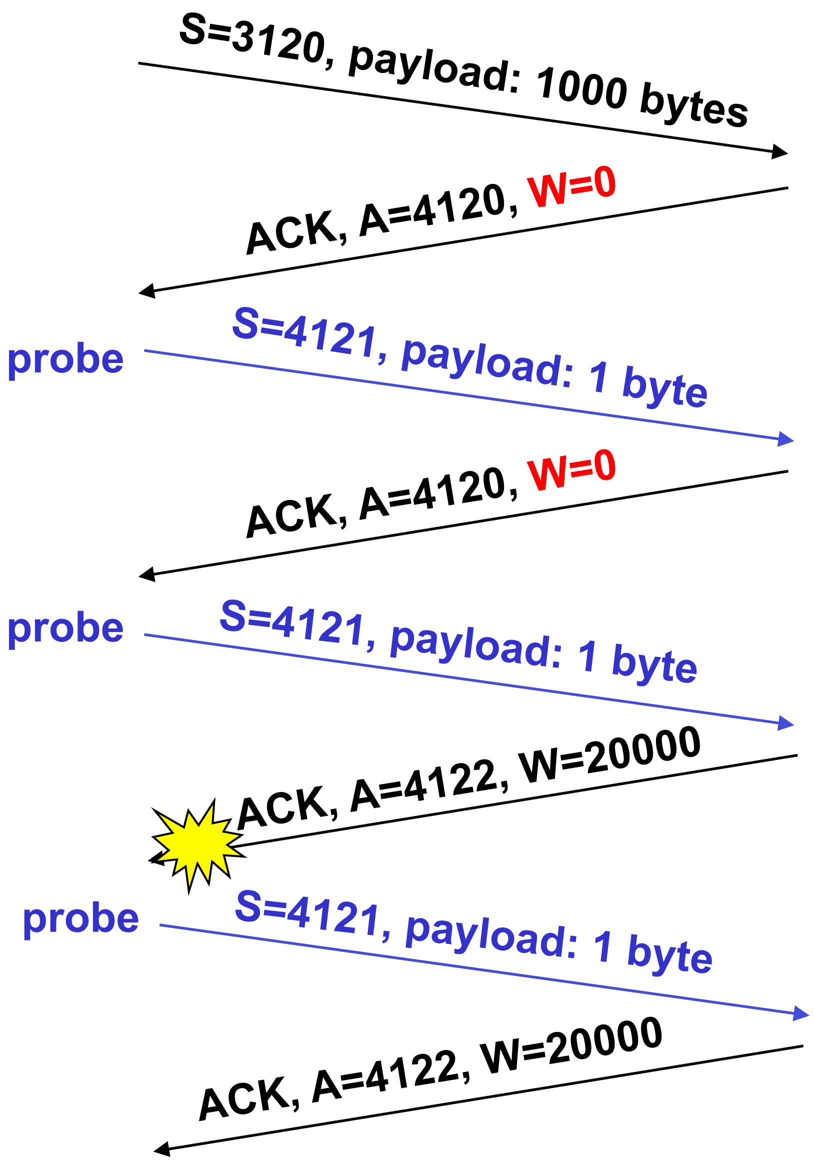
TCP Persist Timer (1/2)

- **Deadlock possible: Window is zero and window-opening ACK is lost!**
  - ACKs are sent unreliable!
  - Now both sides wait for each other!



TCP Persist Timer (2/2)

- **Solution: Sender may send *window probes:***
  - Send one data byte *beyond* window
  - If window remains closed then this byte is not acknowledged—so this byte keeps being retransmitted
- **TCP sender remains in persist state and continues retransmission forever (until window size opens)**
  - Probe intervals are increased exponentially between 5 and 60 seconds
  - Max interval is 60 seconds (forever)



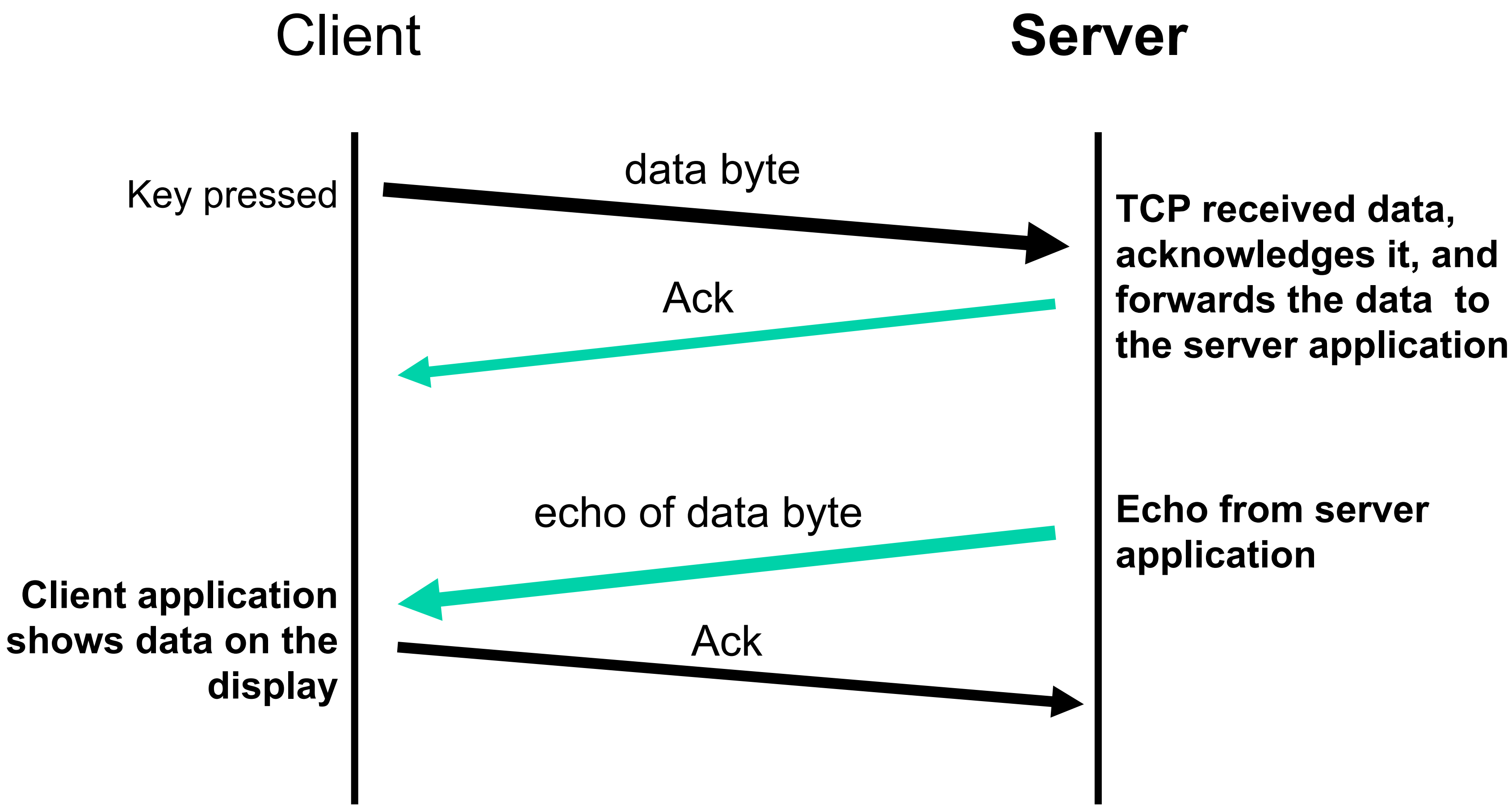
Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

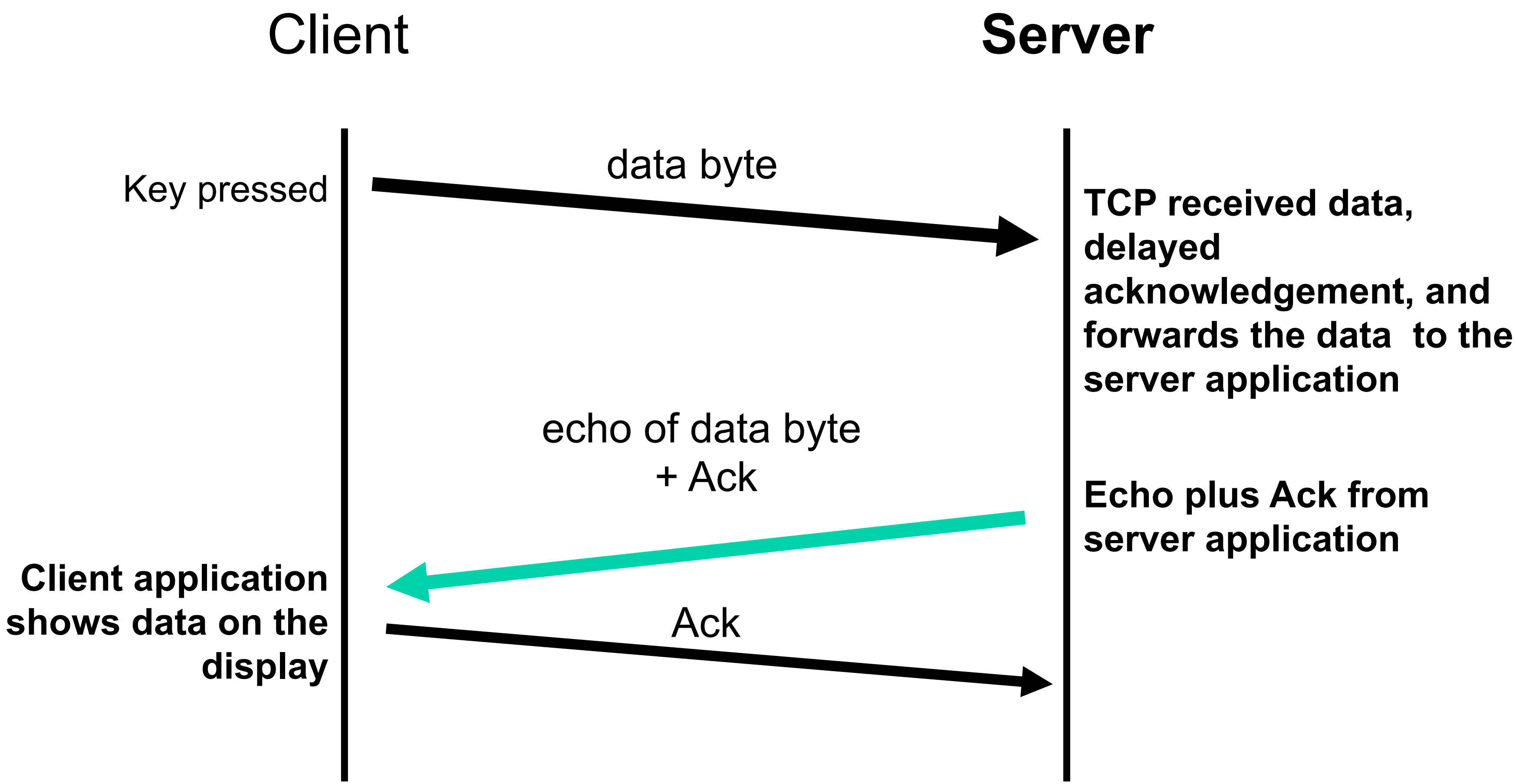
TCP Enhancements

- **So far, only the very basic TCP procedures have been mentioned**
- **But TCP has much more magic built-in algorithms which are essential for operation in today's IP networks:**
  - "Slow Start" and “Congestion Avoidance”
  - "Fast Retransmit" and "Fast Recovery"
  - "Delayed Acknowledgements"
  - "The Nagle Algorithm“
  - Selective ACK (SACK), Window Scaling
  - Silly windowing avoidance
  - ....
- **Additionally, there are different implementations (Reno, Vegas, …)**
  - …

Interactive Traffic



Interactive Traffic with Delayed ACK

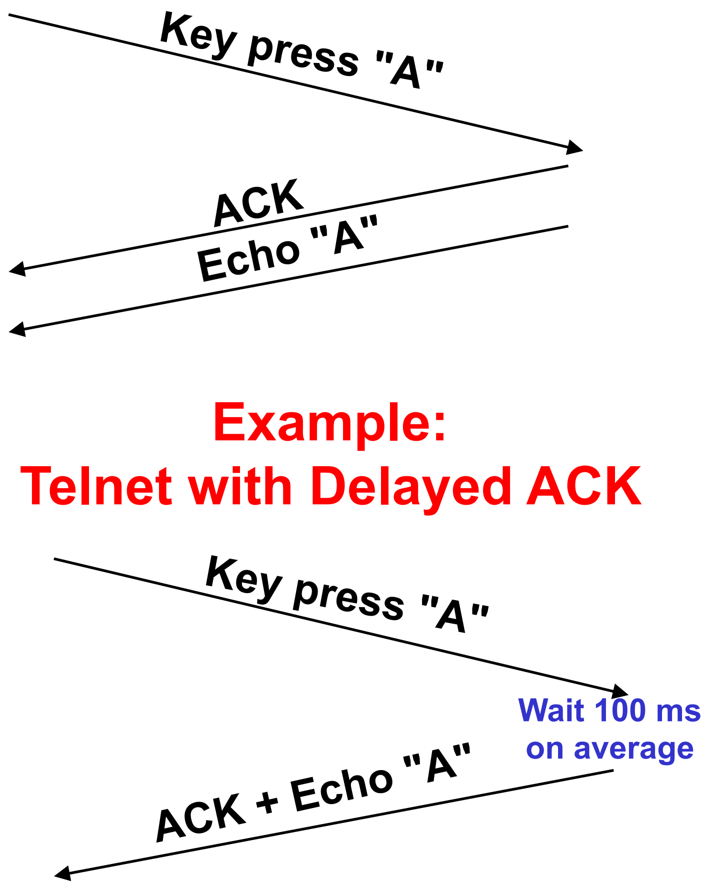


Delayed ACKs

- Goal: Reduce traffic, support piggy-backed ACKs
- Normally TCP, after receiving data, does not immediately send an ACK
- Typically TCP waits (typically) 200 ms and hopes that layer-7 provides data that can be sent along with the ACK

Example:
Telnet and no Delayed ACK

Example:
Telnet with Delayed ACK



Nagle Algorithm

- **Goal: Avoid tinygrams on expensive (and usually slow) WAN links**
- **In RFC 896 John Nagle introduced an efficient algorithm to improve TCP**
- **Idea: In case of outstanding (=unacknowledged) data, small segments should not be sent until the outstanding data is acknowledged**
- **In the meanwhile small amount of data (arriving from Layer 7) is collected and sent as a single segment when the acknowledgement arrives**
- **This simple algorithm is self-clocking**
  - The faster the ACKs come back, the faster data is sent
- **Note: The Nagle algorithm can be disabled!**
  - Important for real-time services

Agenda

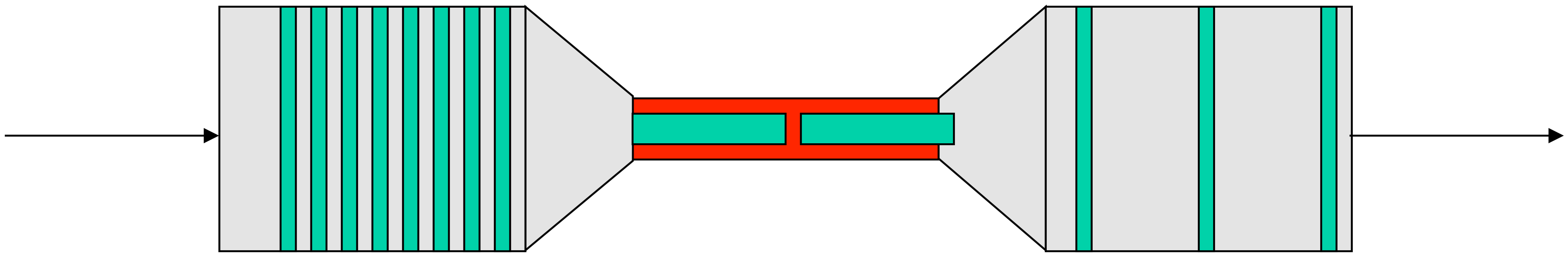
- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

Once again: The Window Size

- **The windows size (announced by the peer) indicates how many bytes I may send at once**
  - Without having to wait for acknowledgements
- **Before 1988, TCP peers tend to exploit the whole window size at once after startup**
  - Sending several segments in a sequence
  - Usually no problem for hosts
  - But led to frequent network congestions
- **Another problem:**
  - In case of segment loss sender can use the window given by the receiver but when window becomes closed the sender must wait until retransmission timer times out
  - That means during that time sender may not fully use the offered bandwidth of the network even if its available
- **TCP performance degradation**

Congestion

- **Problem (buffer overflows) appears at bottleneck links**
  - Some intermediate router must queue packets
  - Queue overflow -> retransmission -> even more overflow!
  - Can't be solved by traditional receiver-imposed flow control (using the window field)



Pipe model of a network path: Big fat pipes (high data rates) outside, a bottleneck link in the middle. The green packets are sent at the maximum achievable rate so that the interpacket delay is almost zero at the bottleneck link; however there is a significant interpacket gap in the fat pipes.

How to Improve TCP Performance?

- **TCP should be "ACK-clocking"**
  – New packets should be injected at the rate at which ACKs are received
  – Duplicate ACKs are necessary to feel the ACK clocking in case of some segments get lost.

- **Ideal case:**
  – Rate at which new segments are injected into the network = acknowledgment-rate of the other end
  – Requires a sensitive algorithm to catch the equilibrium point between high data throughput and packet dropping due to queue overflow:

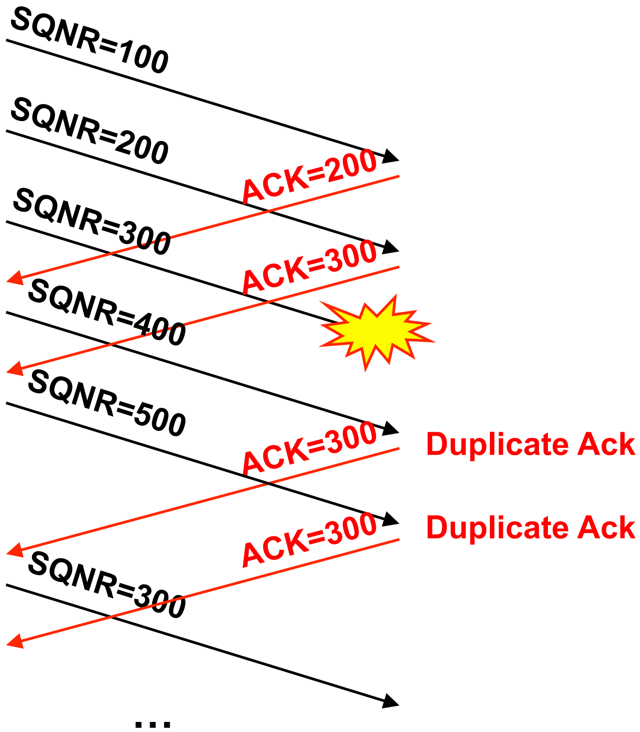
    Van Jacobson's Slow Start and Congestion Avoidance

    (sender-imposed flow control)

- **Assumption:**
  – Packet loss in today's networks are mainly caused by congestion but not by bit errors on physical lines (optical, digital transmission)
    - Note: but not valid for WLAN

Once again: Duplicate ACKs

- **TCP receivers send duplicate ACKs if segments are missing**
  - ACKs are cumulative (each ACK acknowledges all data until specified ACK-number)
  - Duplicate ACKs should not be delayed
- **ACK=300 means: *"I am <u>still</u> waiting for packet with SQNR=300"***

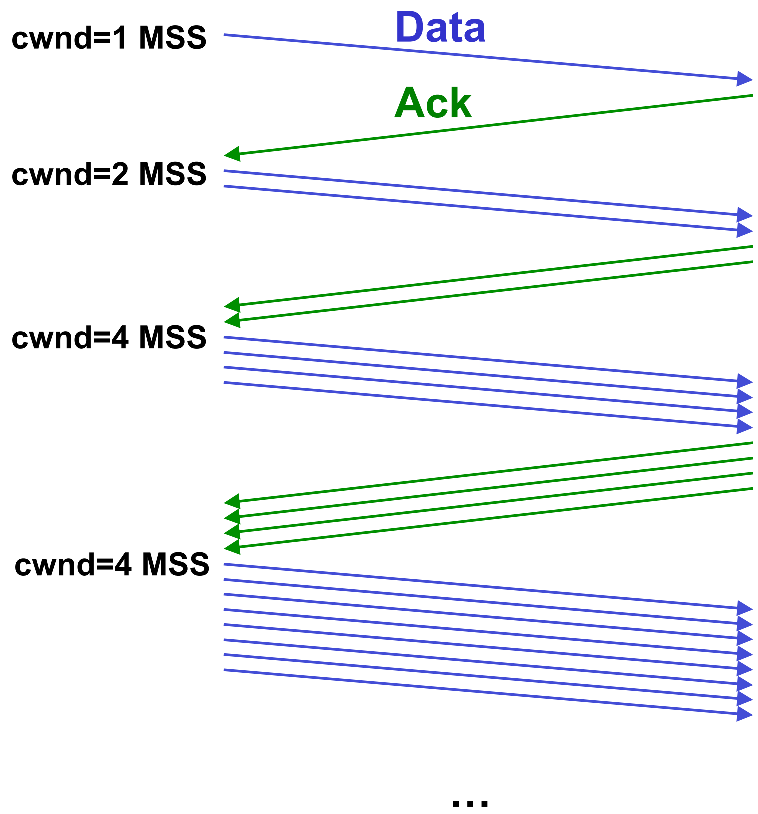


Slow Start Parameters

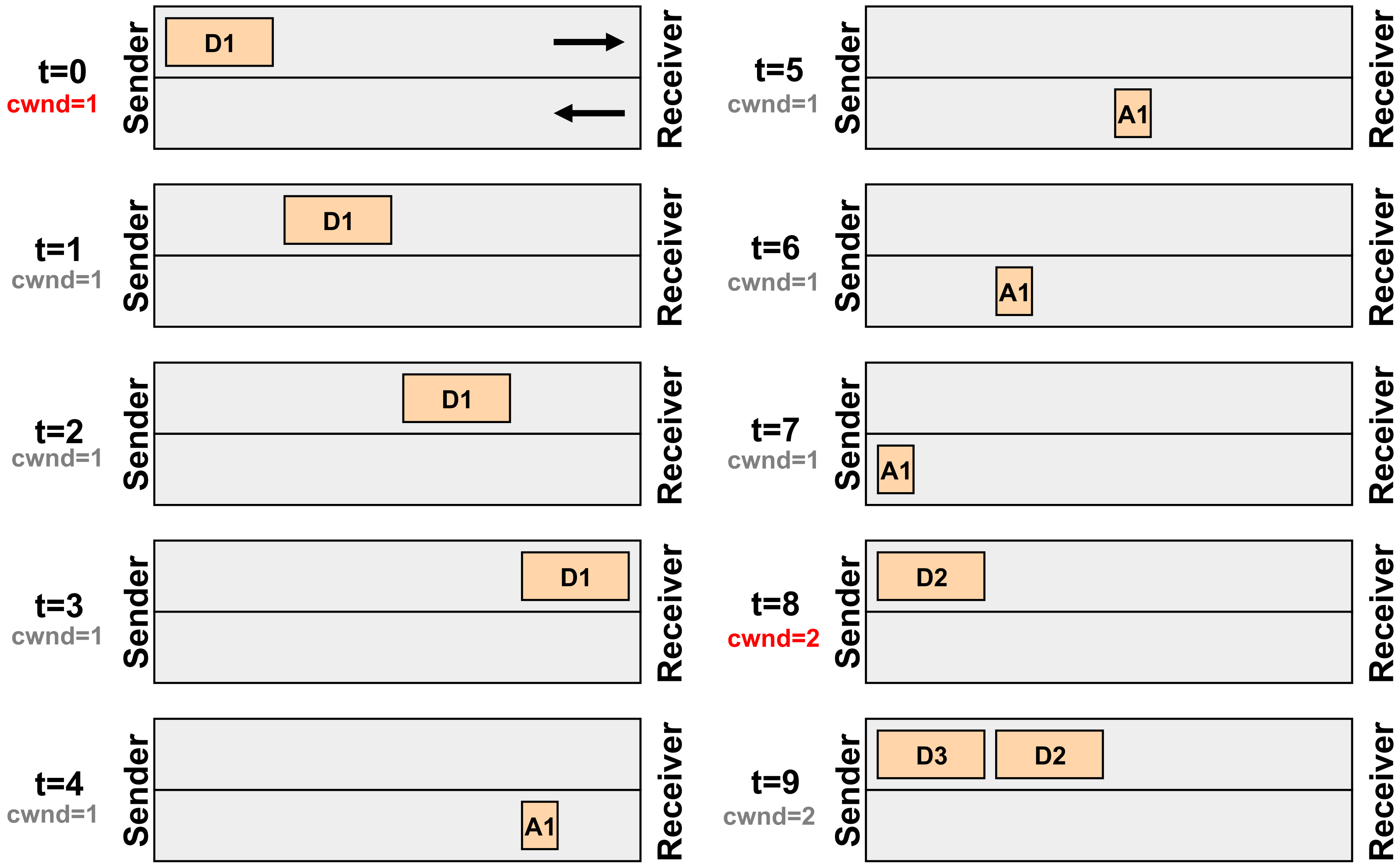
- **Two important parameters are communicated during the TCP three-way handshake**
  - The maximum segment size (MSS)
  - The advertized window size W
- **Now Slow Start introduces the *congestion window (cwnd)***
  - Only locally valid and locally maintained
  - Like window field stores a byte count
- **Rule:**
  - The sender may transmit up to the minimum of the congestion window and the advertised window

Idea of Slow Start

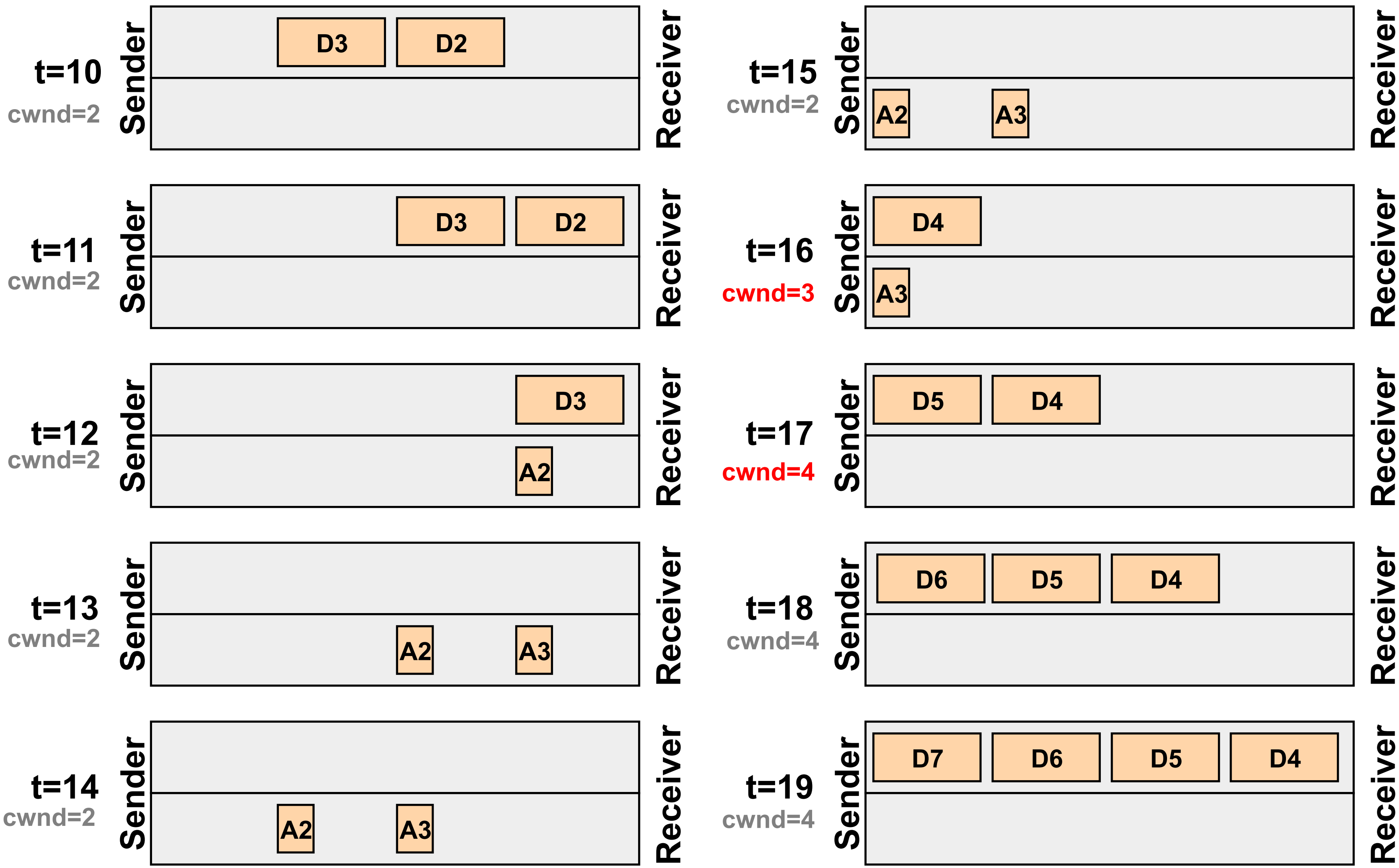
- **Upon new session, cwnd is initialized with MSS (= 1 segment)**
- **Allowed bytes to be sent:**
  - Current window size = Minimum (W, cwnd)
- **Each time an ACK is received, cwnd is incremented by 1 segment**
  - That is, cwnd doubles every RTT (!)
  - Exponential increase!



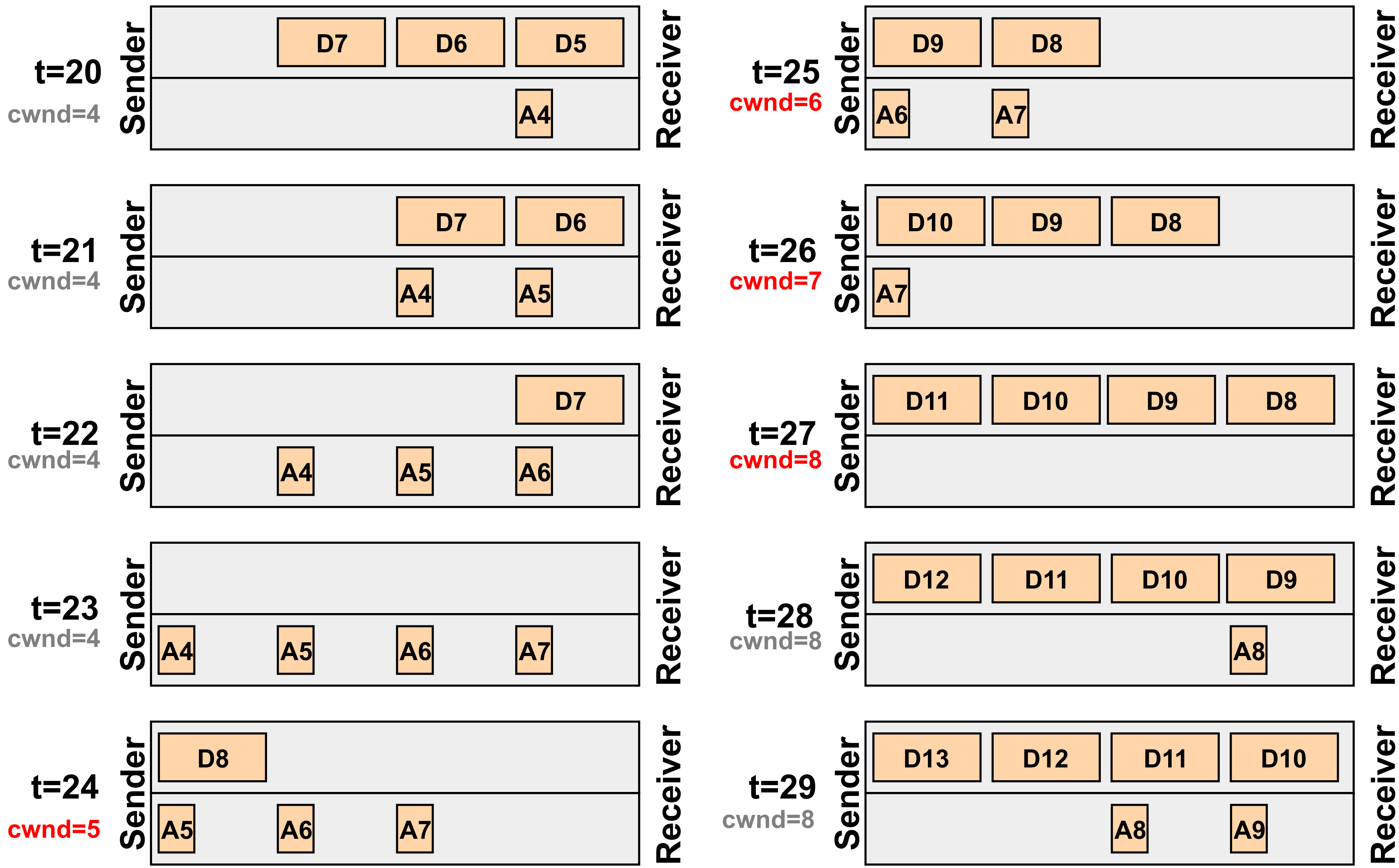
Graphical Illustration (1/4)



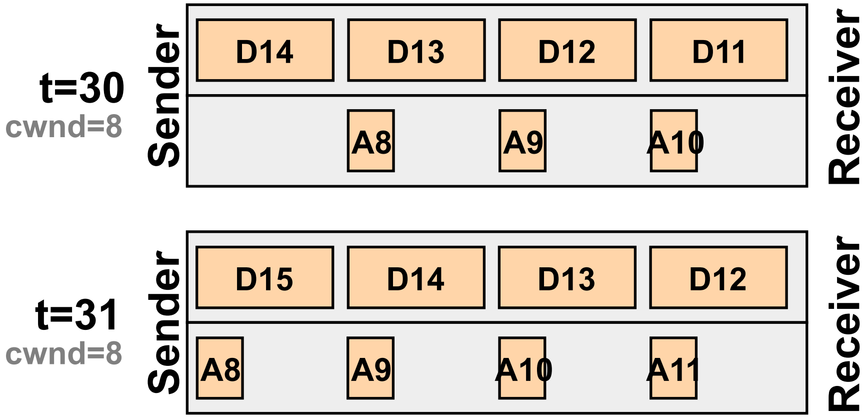
Graphical Illustration (2/4)



Graphical Illustration (3/4)



Graphical Illustration (4/4)

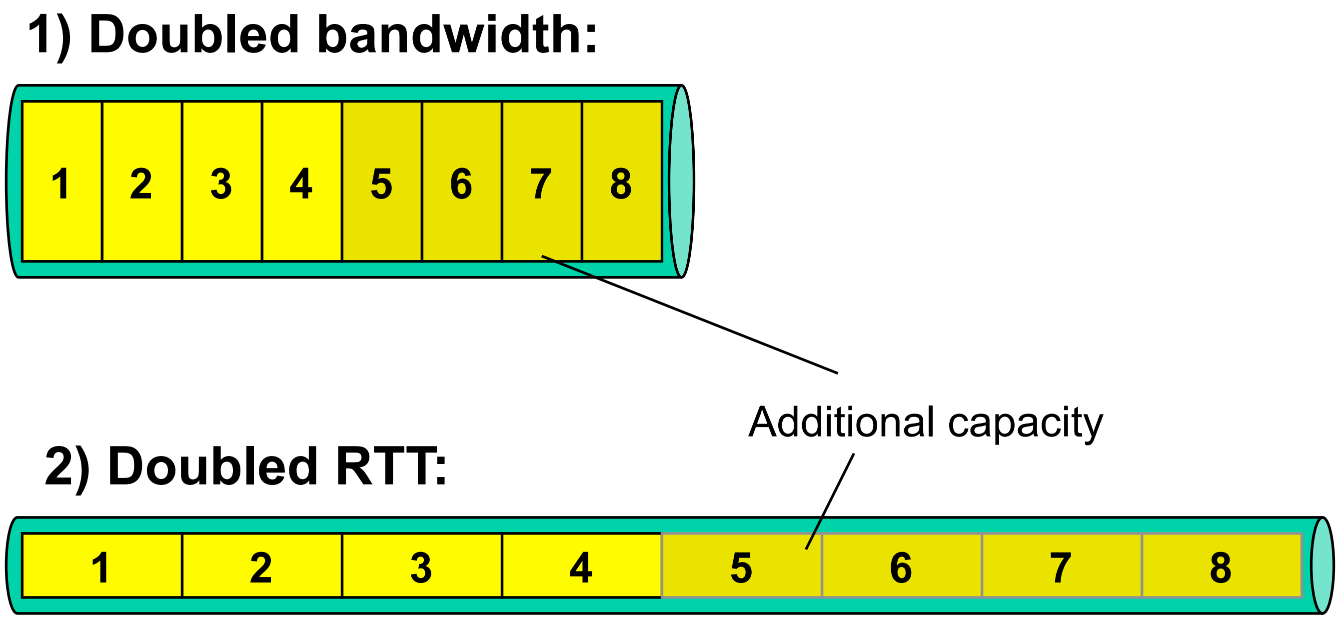


cwnd=8 => Pipe is full (ideal situation) – cwnd should not be increased anymore!

- **TCP is *"self-clocking"***
  - The spacing between the ACKs is the same as between the data segments
  - The number of ACKs is the same as the number of data segments
- **In our example, cwnd=8 is the optimum**
  - This is the bandwidth-delay product (8 = RTT x BW)
  - In other words: the pipe can accept 8 segments per round-trip-time

Performance Limitation of all ARQ Protocols

- **By "Bandwidth-Delay Product" = "Channel Volume"**
- **Continuous RQ with sliding window**
  - The sender's window must be large enough to avoid stopping of sending
- **Channel volume maybe increased**
  - By delays caused by buffers
  - Limited signal speed
  - Bandwidth



End of Slow Start -> Congestion

- **Slow start leads to an exponential increase of the data rate until some network bottleneck is congested and some segments get dropped!**

- **Congestion can be detected by the sender through <u>timeouts</u> or <u>duplicate acknowledgements</u>**

- **Slow start reduces its sending rate with the help of a companion algorithm, called <u>"Congestion Avoidance"</u>**

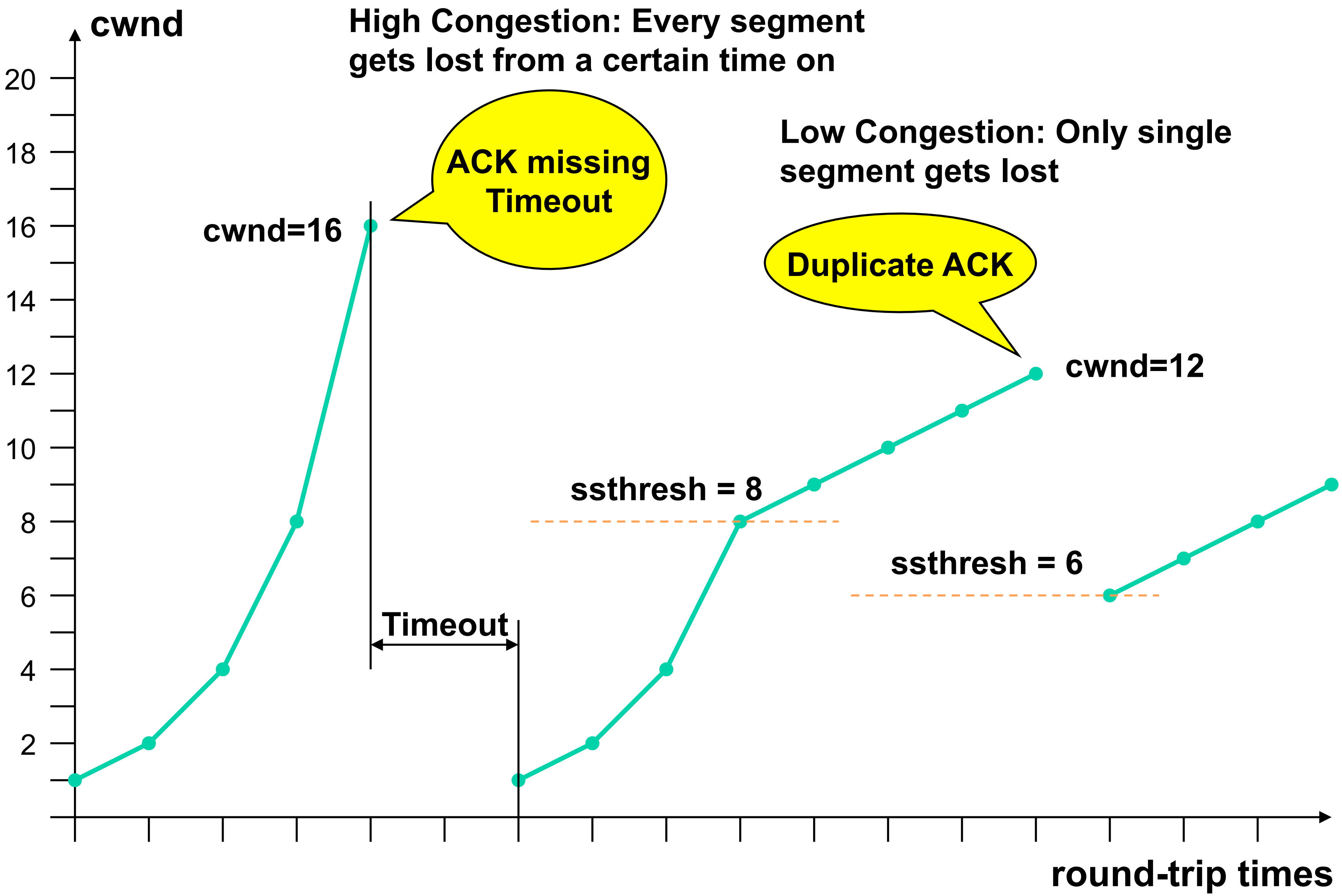
Congestion Avoidance (1)

- **Upon congestion (=duplicate ACKs)**
  – Reduce the sending rate by half and now increase the rate *linearly* until duplicate ACKs are seen again (and repeat this continuously)
- **Congestion Avoidance requires TCP to maintain another variable**
  – Slow Start Threshold" (ssthresh)
  – ssthresh is set to half the current window size in case a duplicate ACK is received
    - Initially, ssthresh is set to TCP's maximum possible MSS (i.e. 65,535 bytes)
    - Note: ssthresh marks a safe window size because congestion occurred at a window size of 2 x ssthresh

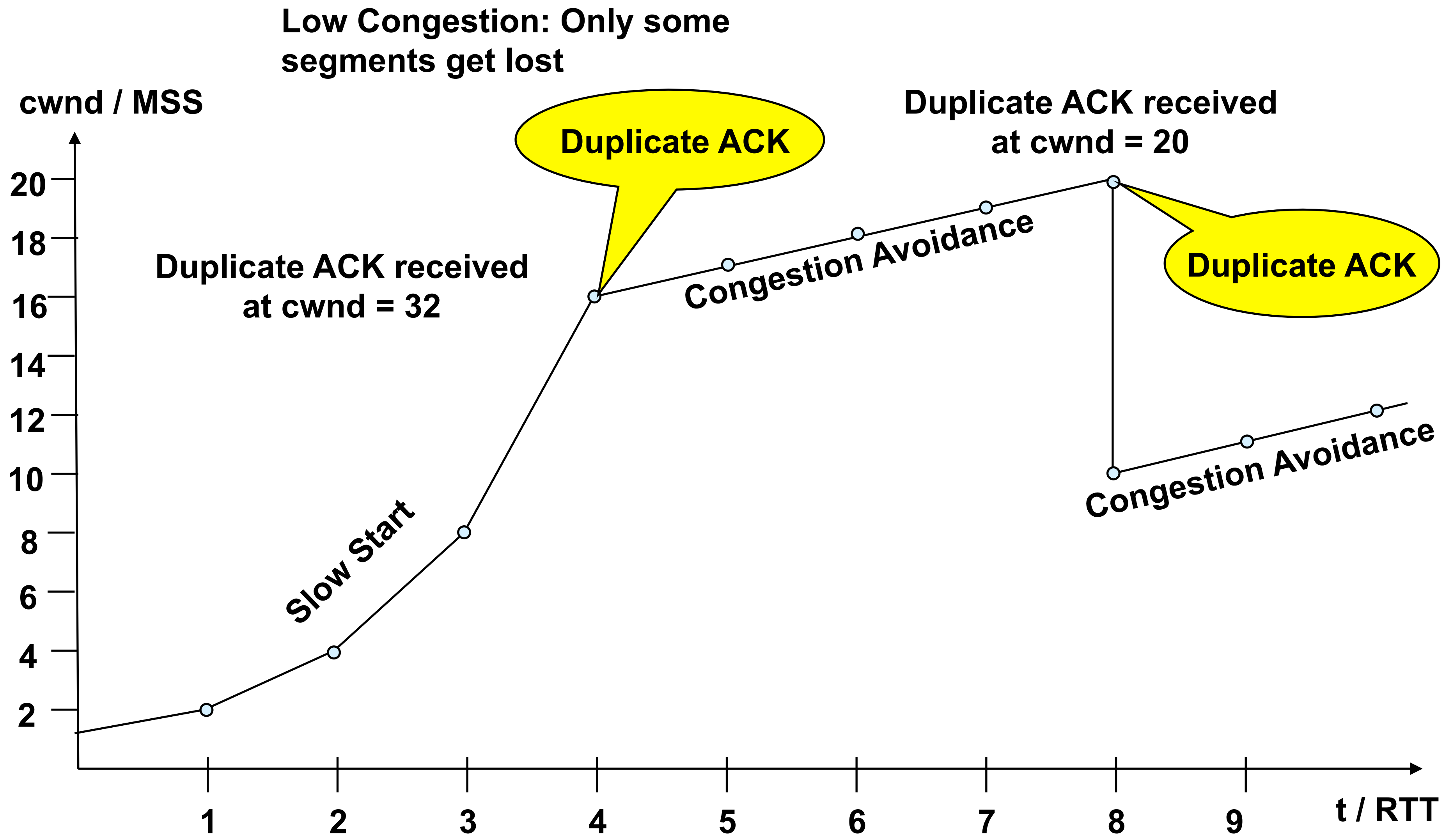
Congestion Avoidance (2)

- **If the congestion is indicated by**
  - A timeout:
    - cwnd is set to 1 -> forcing slow start again
  - A duplicate ACK:
    - cwnd is set to ssthresh (= 1/2 current window size)
- **cwnd ≤ ssthresh:**
  - Slow start, doubling cwnd every round-trip time
  - Exponential growth of cwnd
- **cwnd > ssthresh:**
  - Congestion avoidance, cwnd is incremented by MSS × MSS / cwnd every time an ACK is received
  - linear growth of cwnd

Slow Start and Congestion Avoidance

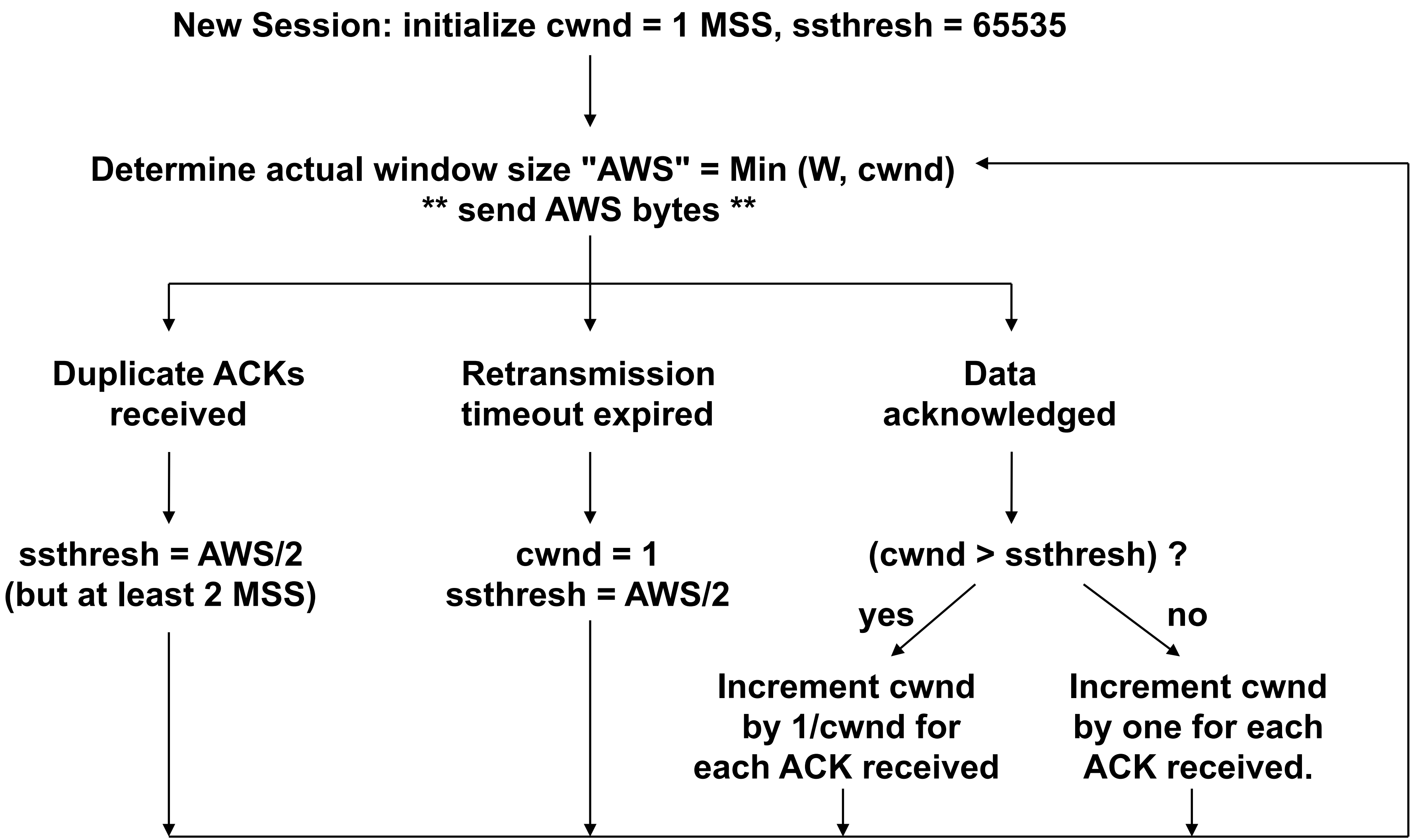


Slow Start and Congestion Avoidance

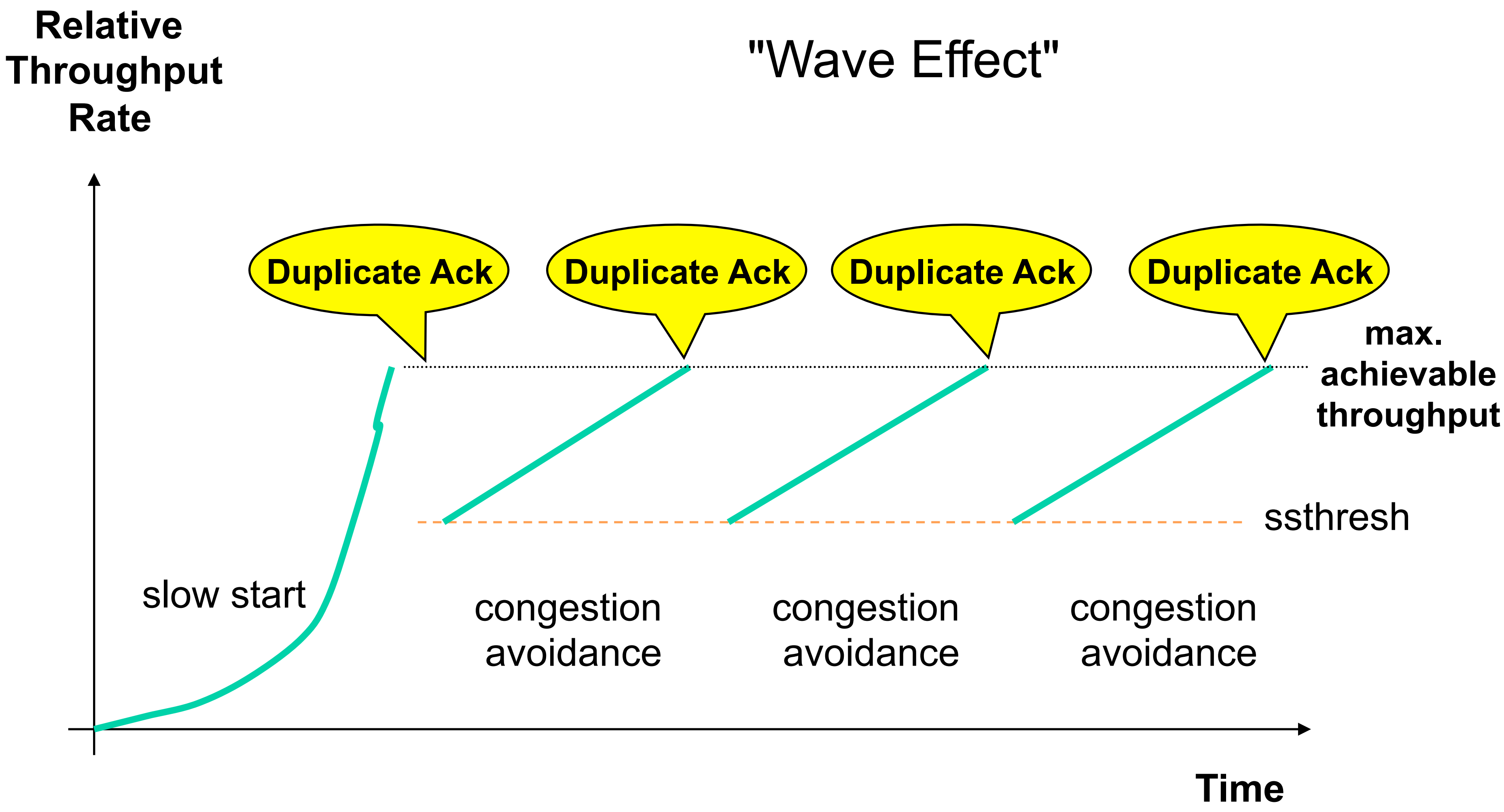


The Combined Algorithm

FYI



Long Term View of TCP Throughput



Real TCP Performance

- **TCP always tries to minimize the data delivery time**
- **Good and proven self-regulating mechanism to avoid congestion**
- **TCP is "hungry but fair"**
  - Essentially fair to other TCP applications
  - Unreliable traffic (e. g. UDP) is not fair to TCP…

Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

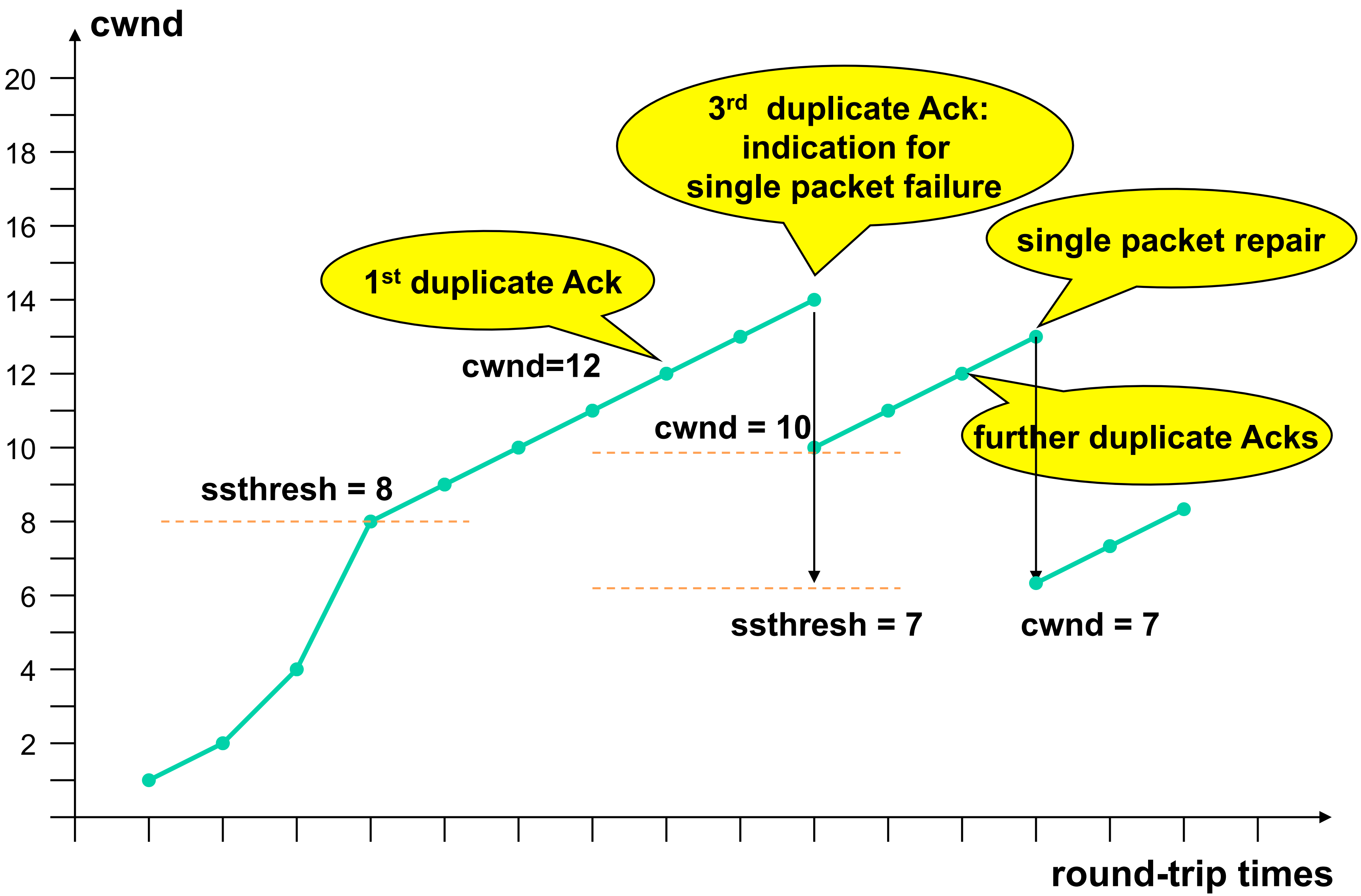
"Fast Retransmit"

- **Note that duplicate ACKs are also sent upon packet reordering**
- **Therefore TCP waits for 3 duplicate ACKs before it really assumes congestion**
  - Immediate retransmission (don't wait for timer expiration)
- **This is called the *Fast Retransmit* algorithm**

"Fast Recovery"

- **After Fast Retransmit TCP continues with Congestion Avoidance**
  - ssthresh is set to half the current window size
  - cwnd is set to ssthresh plus 3 times the maximum segment size.
  - Does NOT fall back to Slow Start
- **Every another duplicate ACK tells us that a "good" segment has been received by the peer**
  - cwnd = cwnd + MSS
  - => Send one additional segment
- **As soon a normal ACK is received**
  - cwnd = ssthresh = Minimum (W, cwnd)/2
- **This is called Fast Recovery**

Fast Retransmit and Fast Recovery



All Together!

Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery

FYI

New Session: initialize cwnd = 1 MSS, ssthresh = 65535

Determine actual window size "AWS" = Min (W, cwnd)
**** send AWS bytes ****

3 duplicate ACKs received

ssthresh = AWS/2 (but at least 2 MSS),
retransmit the segment,
cwnd = ssthresh+3 MSS,
for each 3+nth duplicate ACK increase cwnd by 1 MSS;
then set cwnd=ssthresh upon first "normal" ACK

Retransmission timeout expired

cwnd = 1
ssthresh = AWS/2

Data acknowledged

(cwnd > ssthresh) ?

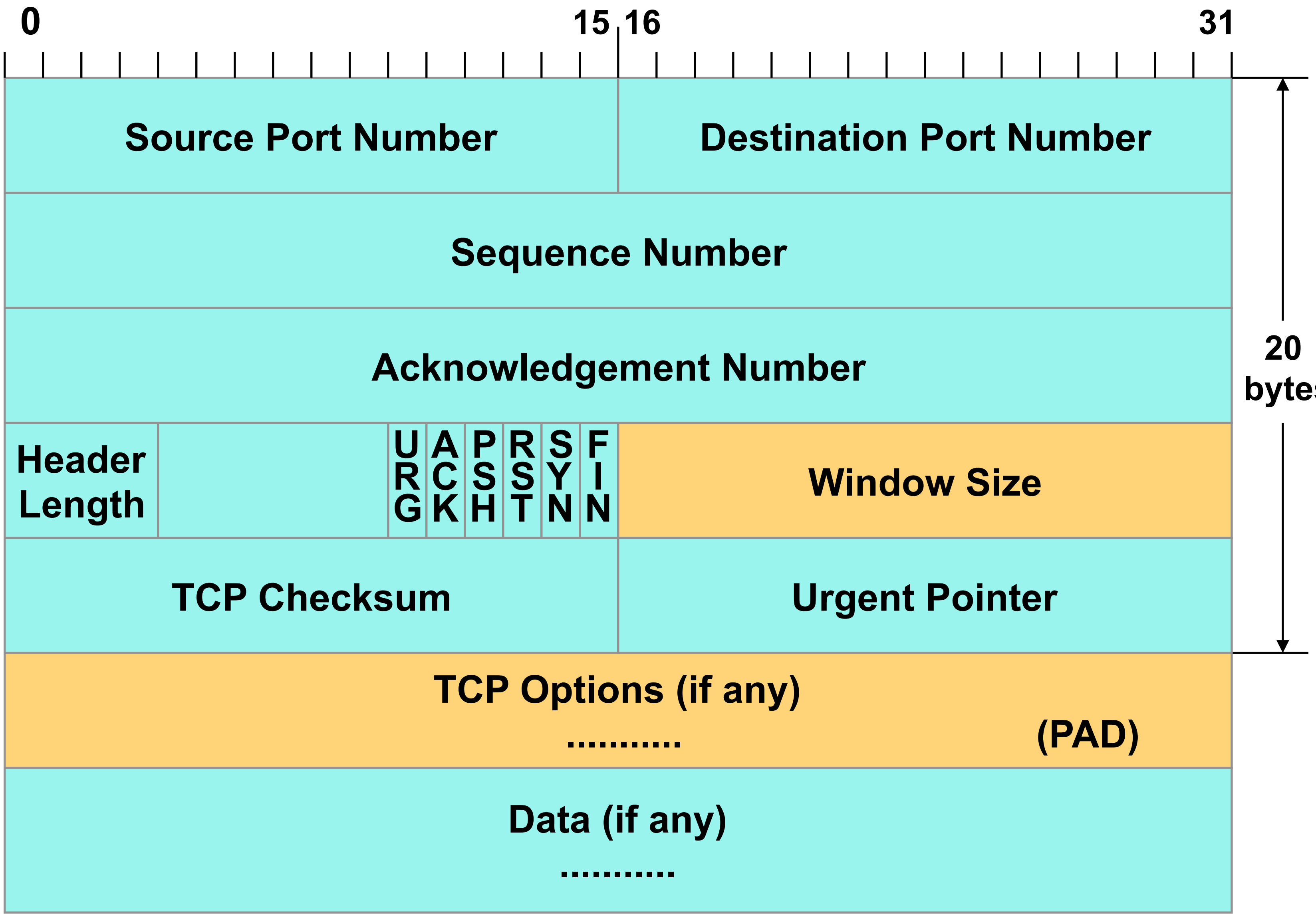
yes: **Increment cwnd by 1/cwnd for each ACK received**

no: **Increment cwnd by one for each ACK received.**

Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

TCP Header Window Field



TCP Options

- **Window-scale option**
  - a maximum segment size of 65,535 octets is inefficient for high delay-bandwidth paths
  - the window-scale option allows the advertised window size to be left-shifted (i.e. multiplication by 2)
  - <u>enables a maximum window size of 2^30 octets</u> !
  - negotiated during connection establishment

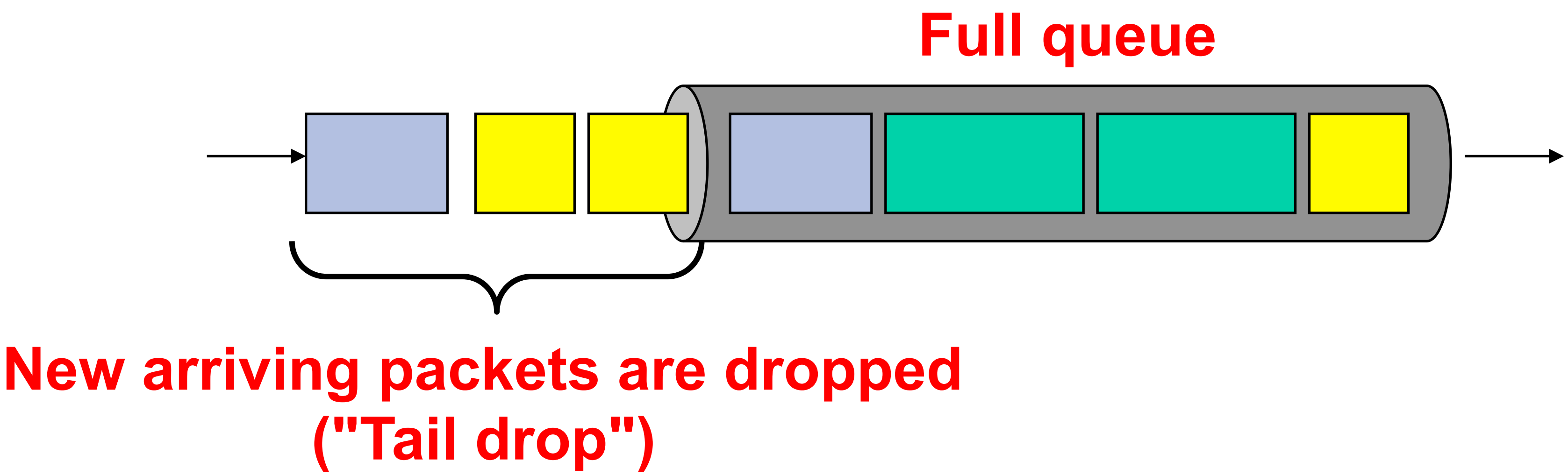
- **SACK (Selective Acknowledgement)**
  - if the SACK-permitted option is set during connection establishment, the receiver may selectively acknowledge already received data even if there is a gap in the TCP stream (Ack-based synchronization maintained)

Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

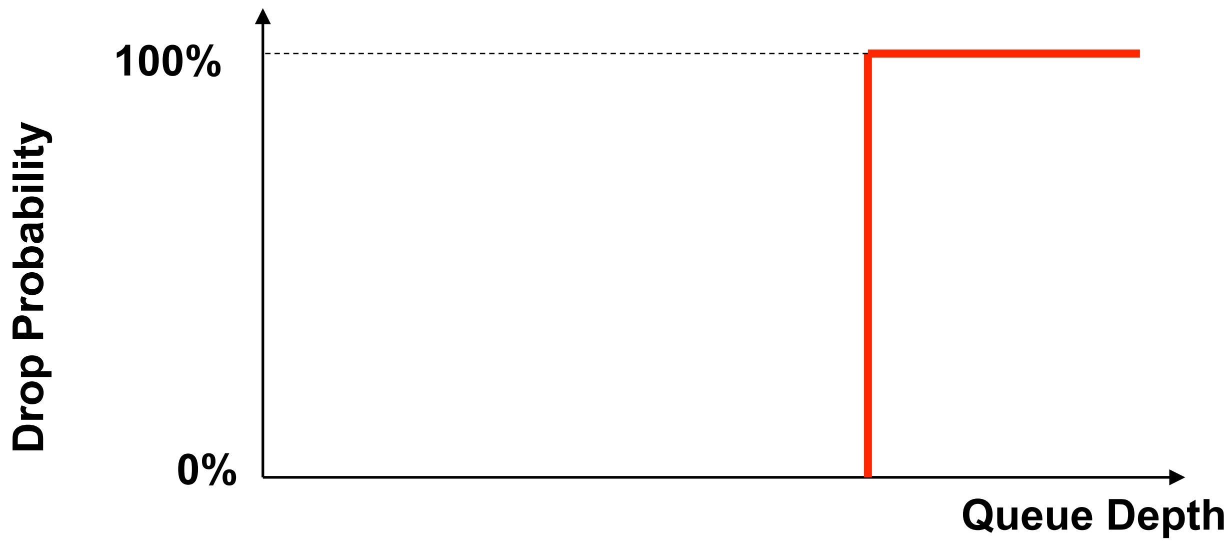
What's Happening in the Network?

- ***Tail-drop queuing*** **is the standard dropping behavior in FIFO queues**
  – If queue is full all subsequent packets are dropped



Tail-drop Queuing (cont.)

- **Another representation:**
  Drop probability versus queue depth

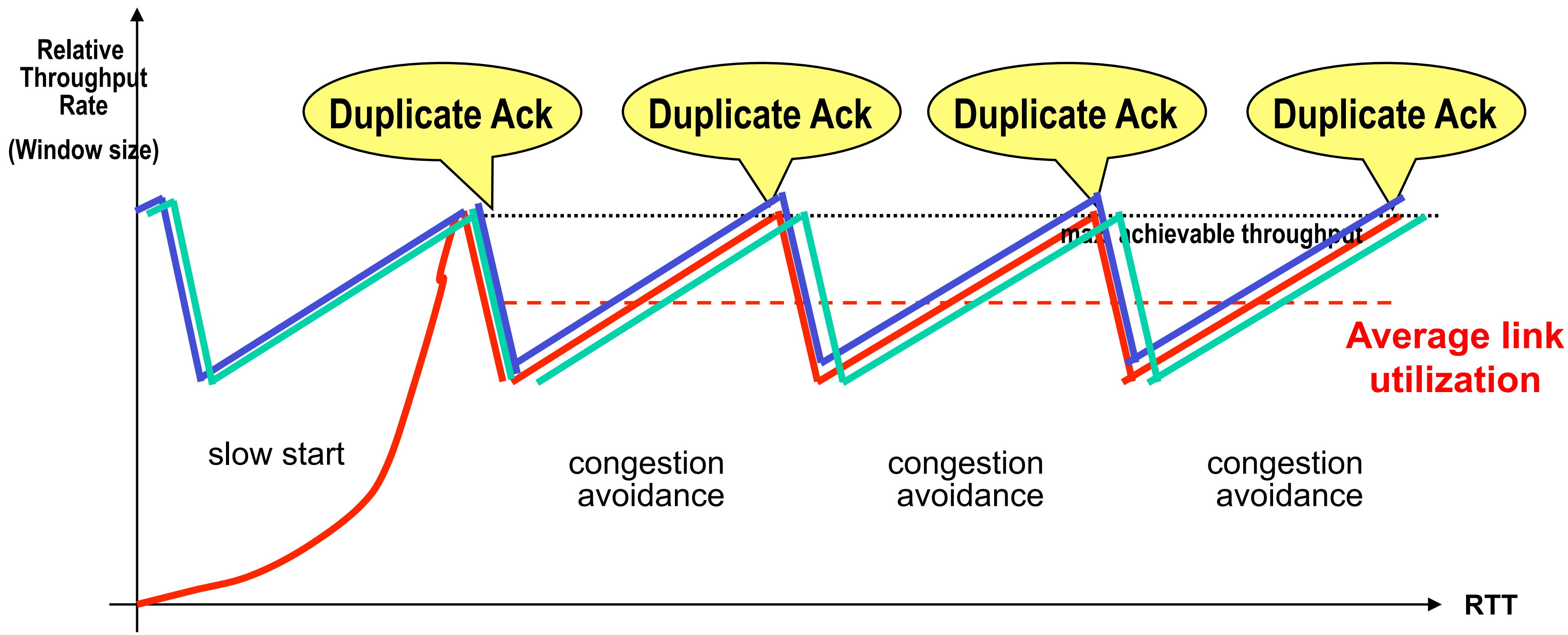


Tail-drop Problems

- **No flow differentiation**
- **TCP starvation upon multiple packet drop**
  - TCP receivers may keep quiet (not even send duplicate ACKs) and sender falls back to slow start
    – worst case!
  - TCP fast retransmit and/or selective acknowledgement may help
- **TCP synchronization**

TCP Synchronization

- **Tail-drop drops many segments of different sessions at the same time**
- **All these sessions experience duplicate ACKs and perform synchronized congestion avoidance**



Random Early Detection (RED)

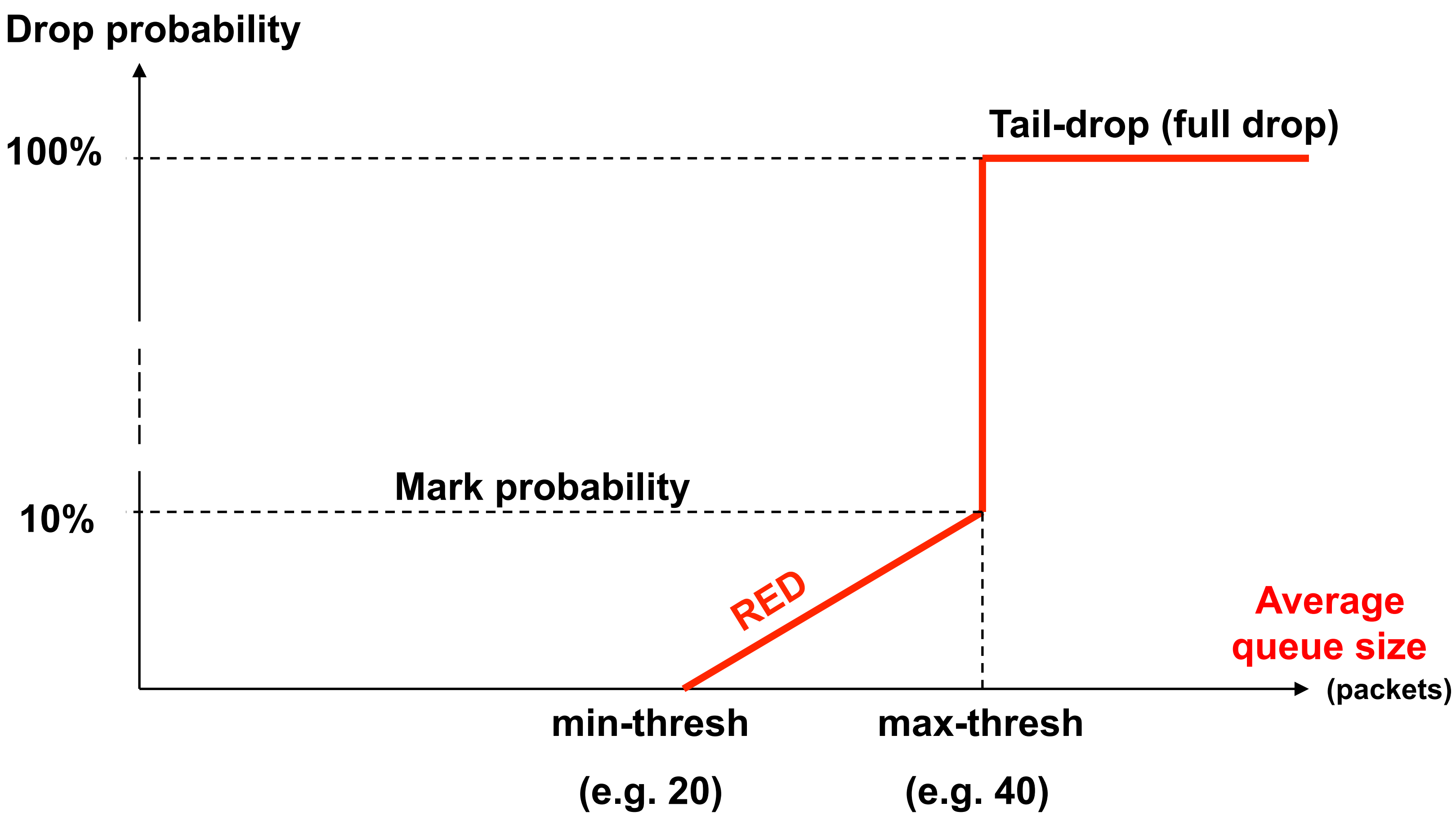
- **Utilizes TCP specific behavior**
  - TCP dynamically adjusts traffic throughput by reducing window size
    - in order to accommodate to the minimal available bandwidth (bottleneck)
- **"Missing" (dropped) TCP segments cause window size reduction!**
  - Idea: Start dropping TCP segments before queuing "tail-drops" occur
  - Make sure that "important" traffic is not dropped
- **RED randomly drops segments before queue is full**
  - Drop probability increases linearly with queue depth

RED

FYI

- **Important RED parameters**
  - Minimum threshold
  - Maximum threshold
  - Average queue size (running average)
- **RED works in three different modes**
  - No drop
    - If average queue size is between 0 and minimum threshold
  - Random drop
    - If average queue size is between minimum and maximum threshold
  - Full drop
    - If average queue size is equal or above maximum threshold = "tail-drop"

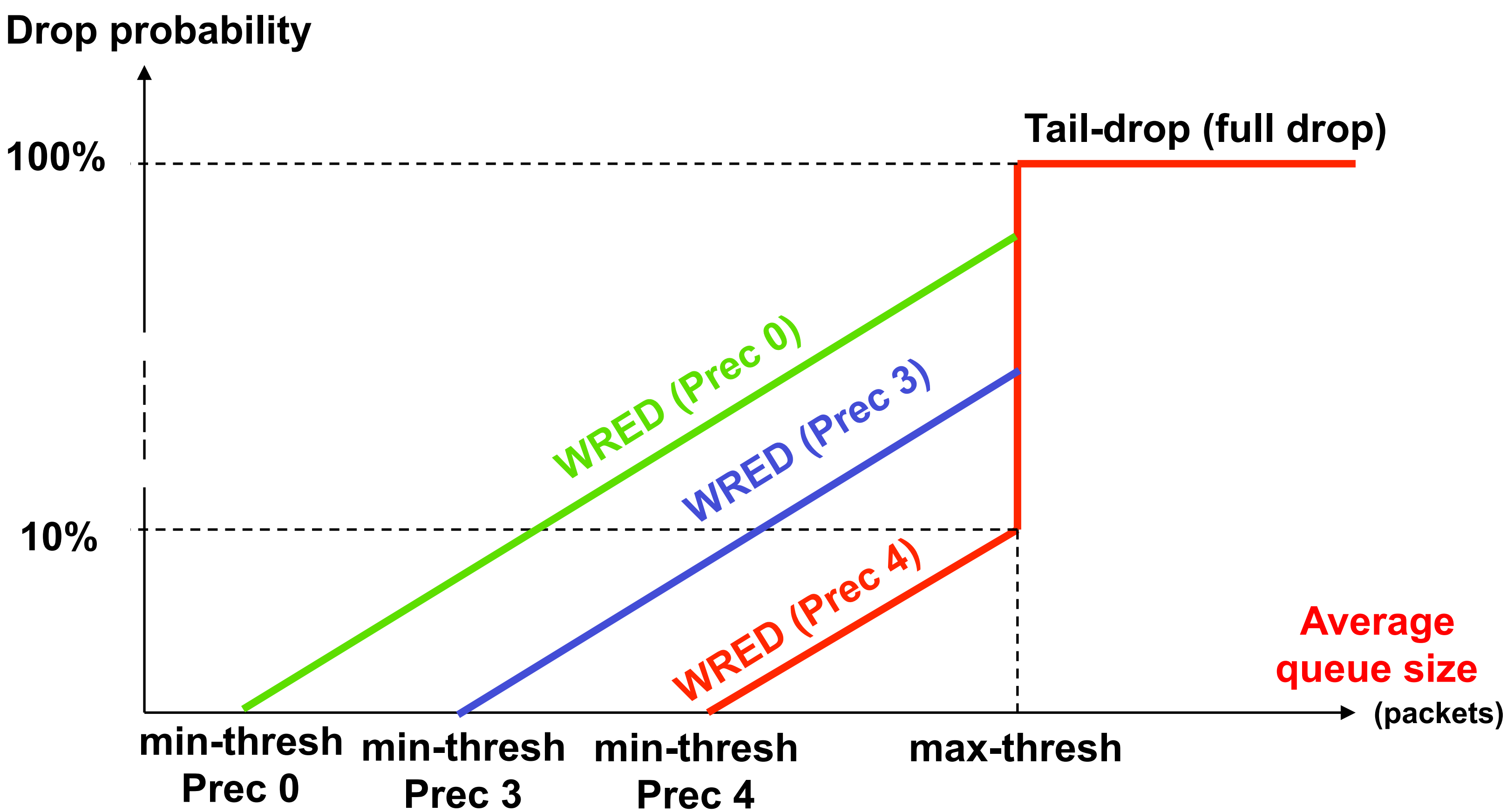
RED Parameters



Weighted RED (WRED)

- **Drops less important packets more aggressively than more important packets**
- **Importance based on:**
  - IP precedence 0-7 (ToS byte)
  - DSCP value 0-63 (ToS byte)
- **Classified traffic can be dropped based on the following parameters**
  - Minimum threshold
  - Maximum threshold
  - Mark probability denominator
    (Drop probability at maximum threshold)

WRED Parameters



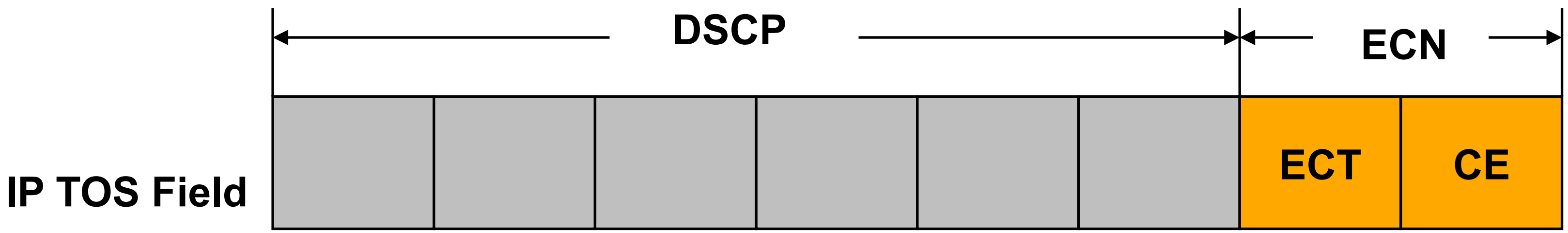
RED Problems

FYI

- **RED performs "Active Queue Management" (AQM) and drops packets before congestion occurs**
  – But an uncertainty remains whether congestion will occur at all
- **RED is known as "difficult to tune"**
  – Goal: Self-tuning RED
  – Running estimate weighted moving average (EWMA) of the average queue size

Explicit Congestion Notification (ECN)

- **Traditional TCP stacks only use segment loss as indicator to reduce window size**
  - But some applications are sensitive to packet loss and delays
- **Routers with ECN enabled mark packets when the average queue depth exceeds a threshold**
  - Instead of randomly dropping them
  - Hosts may reduce window size upon receiving ECN-marked packets
- **Least significant two bits of IP TOS used for ECN**



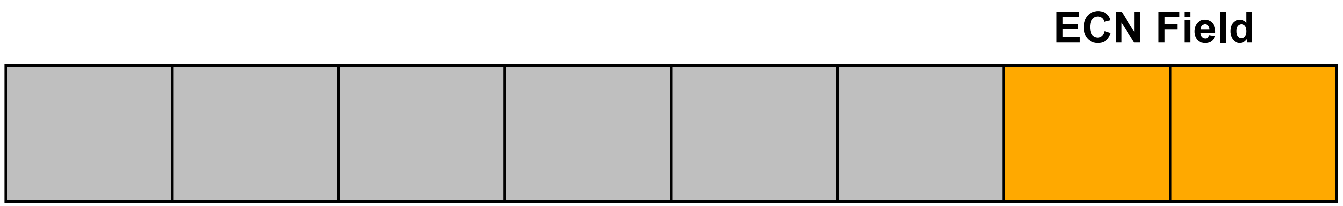
Obsolete (but widely used) RFC 2481 notation of these two bits:

ECT	ECN-Capable Transport
CE	Congestion Experienced

Usage of CE and ECT

FYI

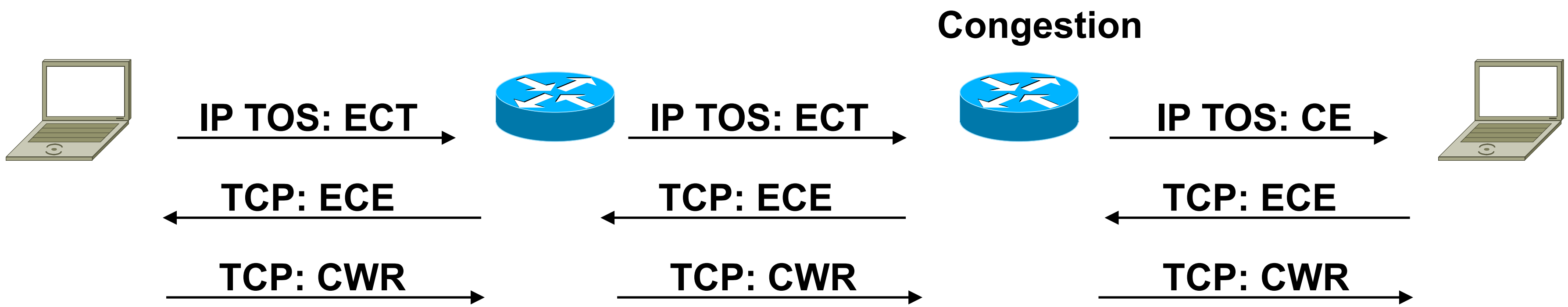
- **RFC 3168 redefines the use of the two bits: ECN-supporting hosts should set one of the two ECT code points**
  - ECT(0) or ECT(1)
  - ECT(0) SHOULD be preferred
- **Routers that experience congestion set the CE code point in packets with ECT code point set (otherwise: RED)**
- **If average queue depth is exceeding max-threshold: Tail-drop**
- **If CE already set: forward packet normally (abuse!)**



Non ECN-capable transport	**0**	**0**	
Codepoints for ECN-capable transport	**0**	**1**	**ECT(1)**
	1	**0**	**ECT(0)**
CE codepoint	**1**	**1**	

CWR and ECE

- **RFC 3168 also introduced two new TCP flags**
  – ECN Echo (ECE)
  – Congestion Window Reduced (CWR)
- **Purpose:**
  – ECE used by data receiver to inform the data sender when a CE packet has been received
  – CWR flag used by data sender to inform the data receiver that the congestion window has been reduced



Part of TCP header:

Header Length	Reserved	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size

Note

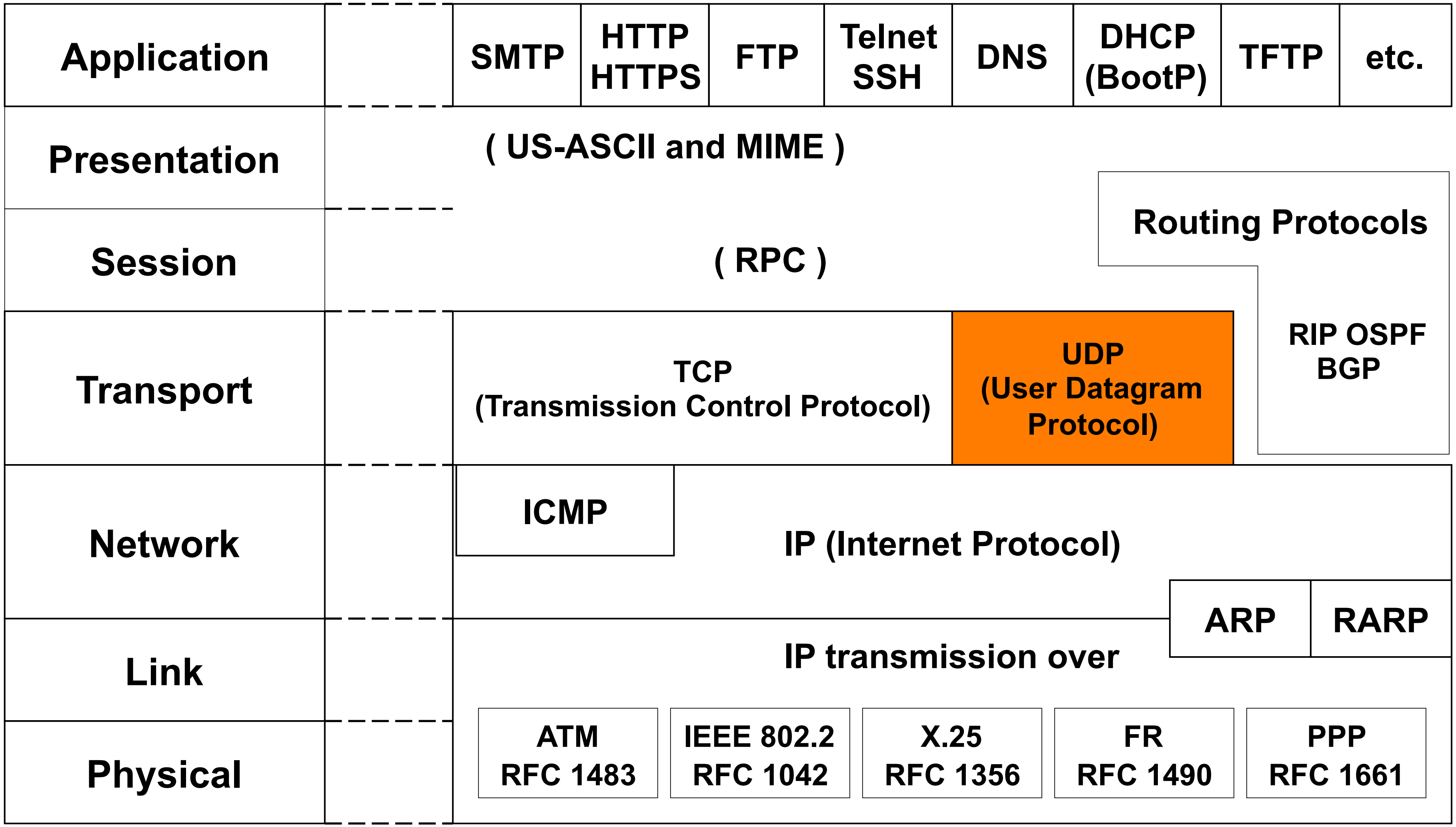
FYI

- **CE is only set when average queue depth exceeds a threshold**
  - End-host would react immediately
  - Therefore ECN is not appropriate for short term bursts (similar as RED)
- **Therefore ECN is different as the related features in Frame Relay or ATM which acts also on short term (transient) congestion**

Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Delay Bandwidth Product
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

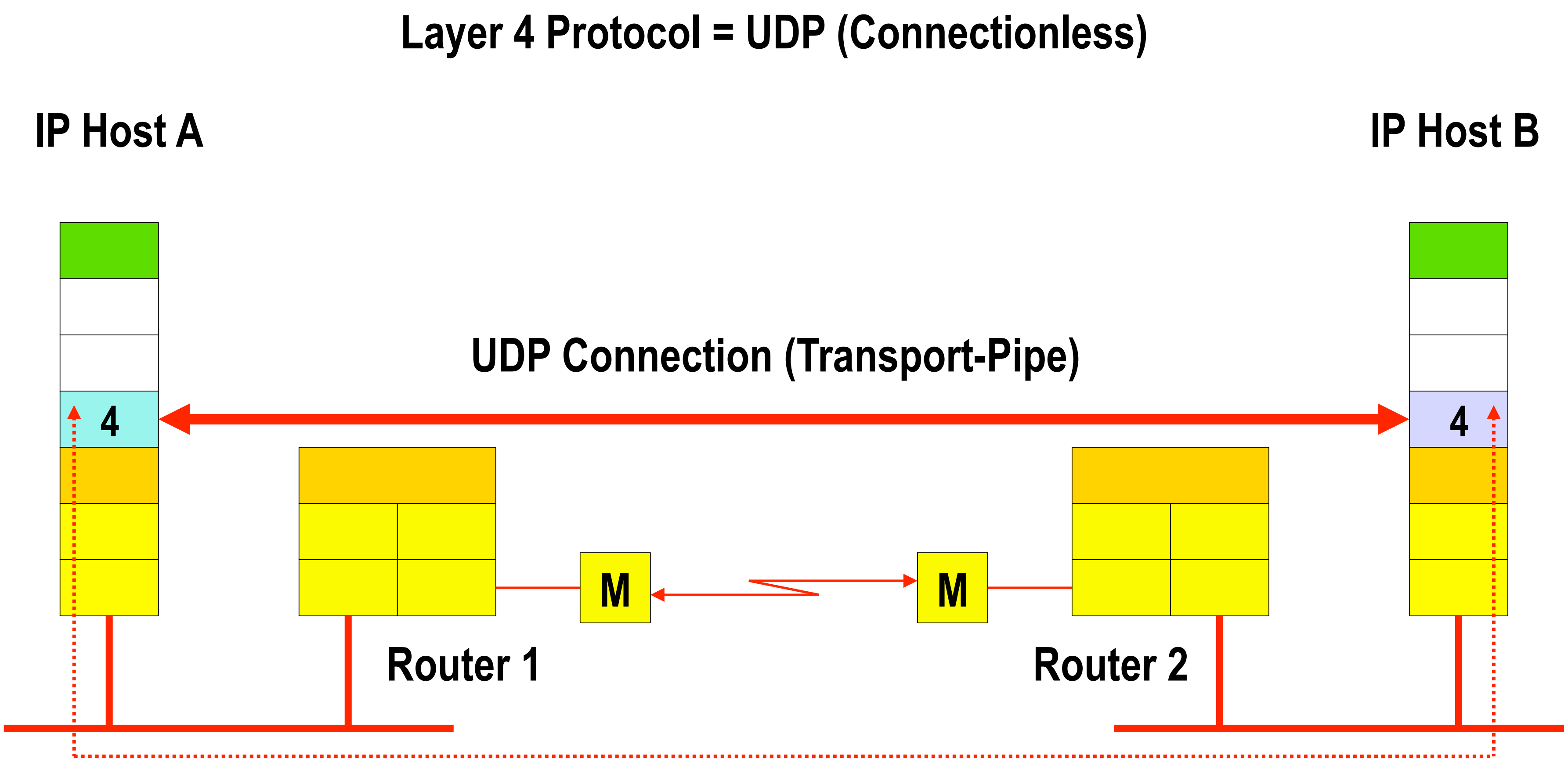
TCP/IP Protocol Suite



UDP (User Datagram Protocol, RFC 768)

- UDP is a connectionless layer 4 service (datagram service)
- Layer 3 Functions are extended by port addressing and a checksum to ensure integrity
- UDP uses the same port numbers as TCP (if applicable)
- Less complex than TCP, easier to implement

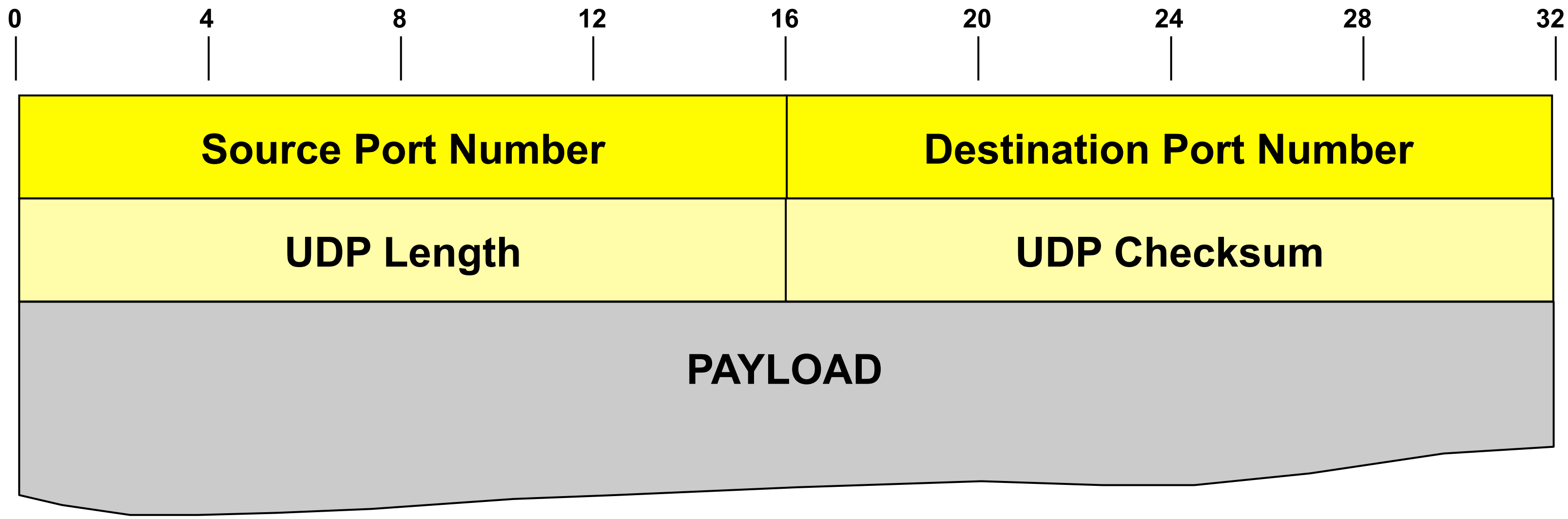
UDP and OSI Transport Layer 4



UDP Usage

- **UDP is used**
  - When the overhead of a connection oriented service is undesirable
    - E.g. for short DNS request/reply
  - When the implementation has to be small
    - e.g. BootP, TFTP, DHCP, SNMP
  - Where retransmission of lost segments makes no sense
    - Voice over IP
    - Multimedia streams

UDP Header


0
4
8
12
16
20
24
28
32
Source Port Number
Destination Port Number
UDP Length
UDP Checksum
PAYLOAD

Important UDP Port Numbers

– 7	Echo
– 53	DOMAIN, Domain Name Server
– 67	BOOTPS, Bootstrap Protocol Server
– 68	BOOTPC, Bootstrap Protocol Client
– 69	TFTP, Trivial File Transfer Protocol
– 79	Finger
– 111	SUN RPC, Sun Remote Procedure Call
– 137	NetBIOS Name Service
– 138	NetBIOS Datagram Service
– 161	SNMP, Simple Network Management Protocol
– 162	SNMP Trap
– 322	RTSP (Real Time Streaming Protocol) Server
– 520	RIP
– 5060	SIP (VoIP Signaling)
– xxxx	RTP (Real-time Transport Protocol)
– xxxx+1	RTCP (RTP Control Protocol)

Agenda

- **TCP Fundamentals**
  - Principles, Port and Sockets
  - Header Fields
  - Three Way Handshake
  - Windowing
  - Enhancements
- **TCP Performance**
  - Slow Start and Congestion Avoidance
  - Delay Bandwidth Product
  - Fast Retransmit and Fast Recovery
  - TCP Window Scale Option and SACK Options
  - Explicit Congestion Notification (ECN)
- **UDP**
- **RFC Collection**
- **NAT**

RFCs

- **0761 - TCP**
- **0813 - Window and Acknowledgement Strategy in TCP**
- **0879 - The TCP Maximum Segment Size**
- **0896 - Congestion Control in TCP/IP Internetworks**
- **1072 - TCP Extension for Long-Delay Paths**
- **1106 - TCP Big Window and Nak Options**
- **1110 - Problems with Big Window**
- **1122 - Requirements for Internet Hosts -- Com. Layer**
- **1185 - TCP Extension for High-Speed Paths**
- **1323 - High Performance Extensions (Window Scale)**

RFCs

- 2001 - Slow Start and Congestion Avoidance (Obsolete)
- 2018 - TCP Selective Acknowledgement (SACK)
- 2147 - TCP and UDP over IPv6 Jumbograms
- 2414 - Increasing TCP's Initial Window
- 2581 - TCP Slow Start and Congestion Avoidance (Current)
- 2873 - TCP Processing of the IPv4 Precedence Field
- 3168 - TCP Explicit Congestion Notification (ECN)

Agenda

- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

Private Address Range - RFC 1918

- **Three blocks of address ranges are reserved for addressing of private networks**
  - 10.0.0.0 - 10.255.255.255 (10/8 prefix)
  - 172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
  - 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

- **NAT (Network Address Translation)**
  - Performs translation between private addresses and globally unique addresses
  - Was originally developed as an interim solution to combat IPv4 address depletion by allowing IP addresses to be reused by several hosts

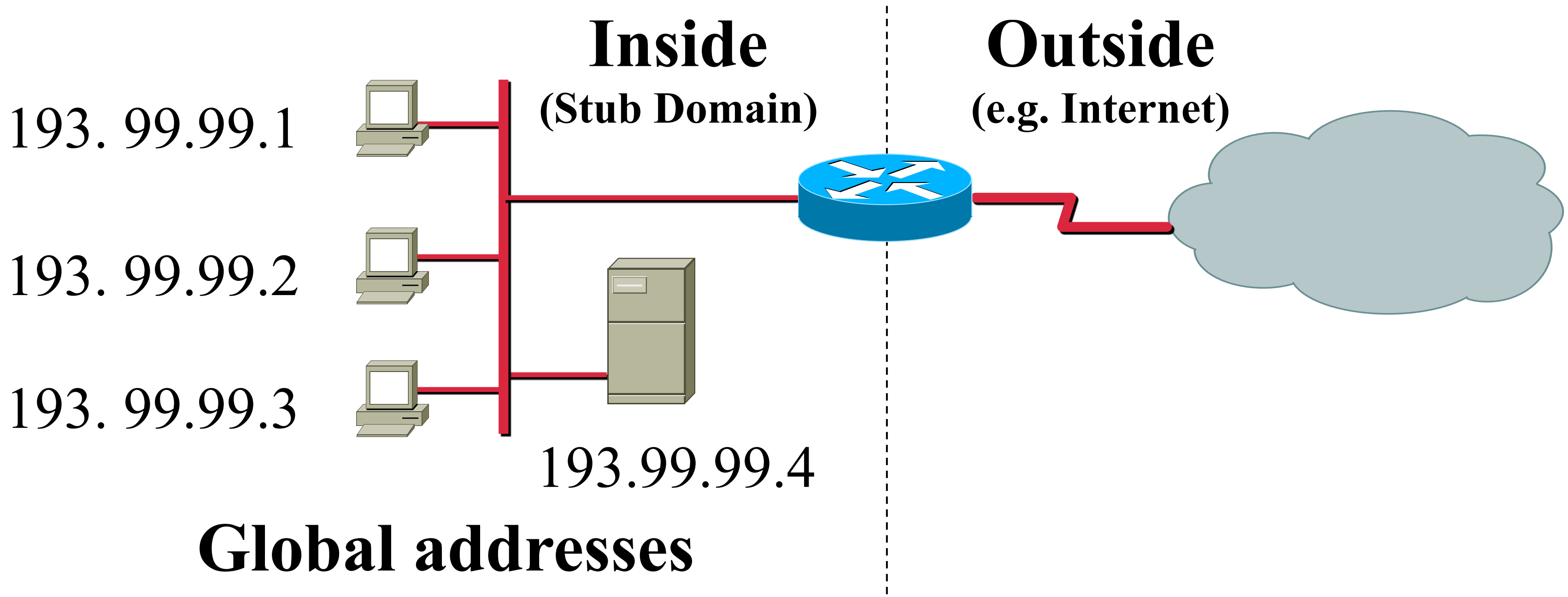
Network Address Translation (NAT)

- **NAT**
  - First explained in RFC 1631
    - The address reuse solution is to place Network Address Translators (NAT) at the borders of stub domains
    - Each NAT box has a table consisting of pairs of local IP addresses and globally unique addresses performing address translation when passing IP Datagram's between a stub domain and the Internet and vice versa
    - The IP addresses inside the stub domain are not globally unique, they are reused in other domains, thus solving the address depletion problem
    - In most cases private addresses (RFC 1918) are used inside the stub domain (10.0.0.0/8, 172.16.0.0/16, 192.168.0.0/16)

Reasons for NAT

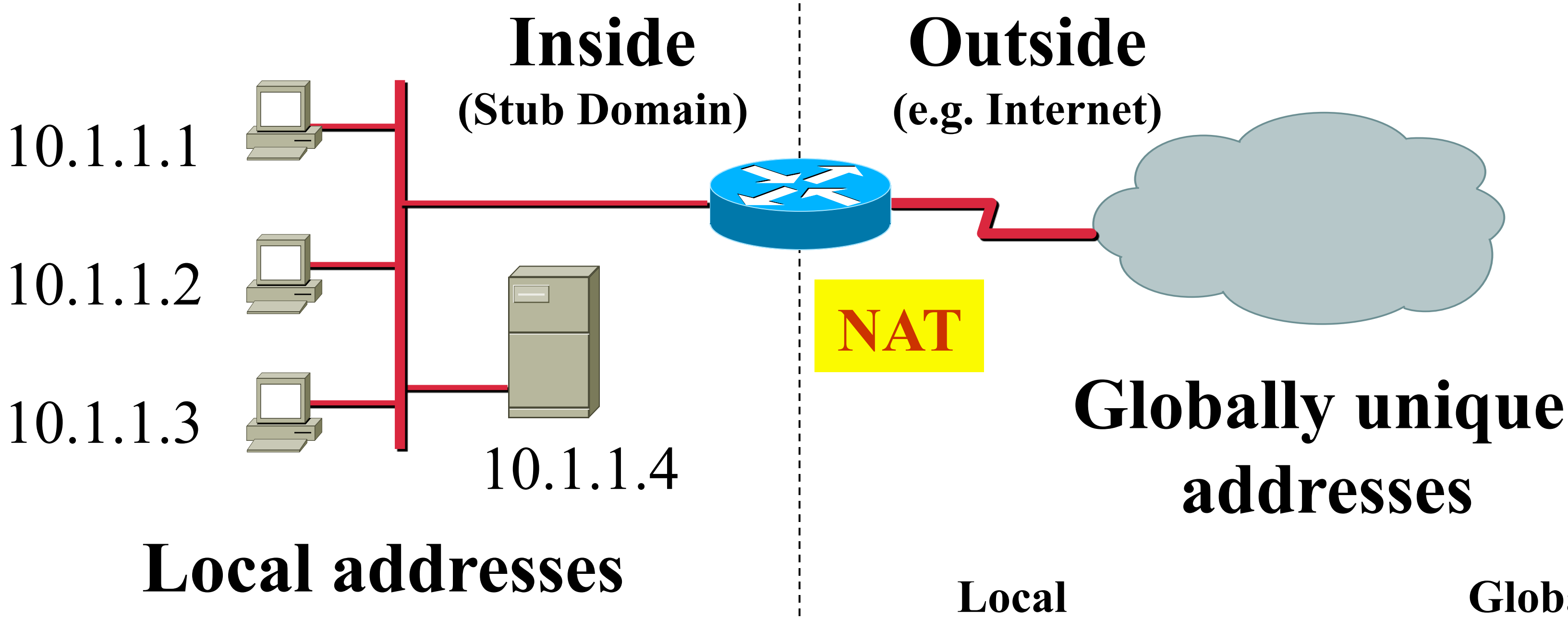
- **Mitigate Internet address depletion**
  - As temporary solution before IPv6 is there
- **Save global addresses (and money)**
  - NAT is most often to map the nonroutable private address spaces defined by RFC 1918 to an official address
    - 10.0.0.0/8, 172.16.0.0/16, 192.168.0.0/16
- **Conserve internal address plan**
- **TCP load sharing**
  - Several physical servers are hided behind one IP address and traffic to them is balanced
- **Hide internal topology**
  - Security aspect

Terms (1)



(NAT not necessary in this case)

Terms (2)



Static one-to-one mapping (NAT-Binding) is maintained by router-internal static NAT–Table

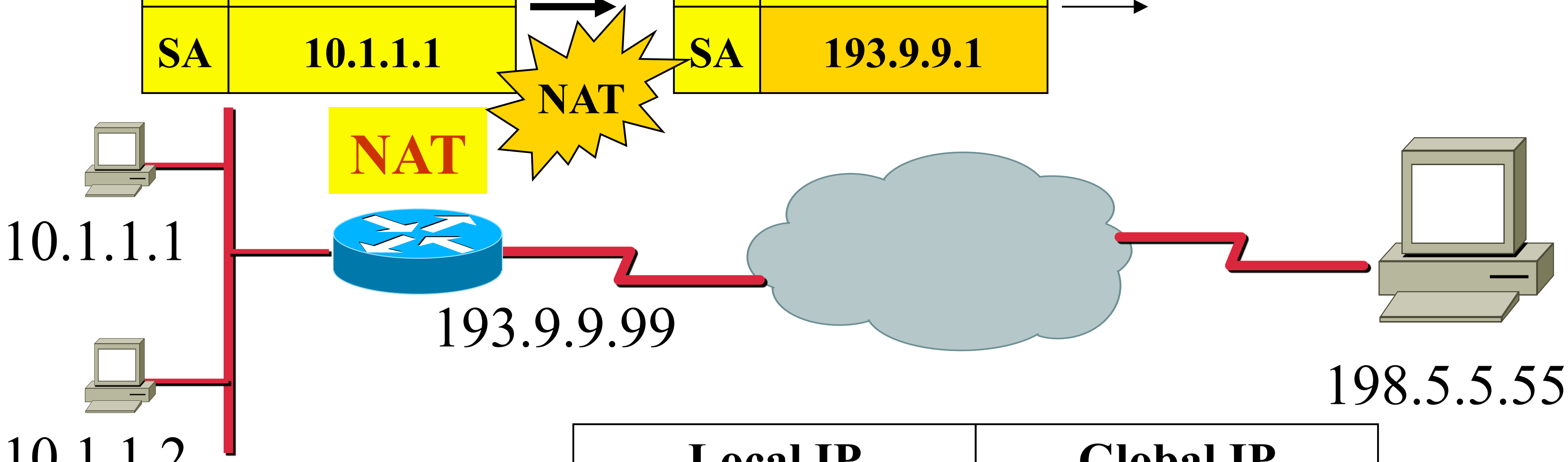
Local IP address		Global IP address
10.1.1.1	⟷	193.99.99.1
10.1.1.2	⟷	193.99.99.2
10.1.1.3	⟷	193.99.99.3
10.1.1.4	⟷	193.99.99.4

Basic Principle (1)

Binding is maintained by static NAT–Table

DA	198.5.5.55
SA	10.1.1.1

DA	198.5.5.55
SA	193.9.9.1

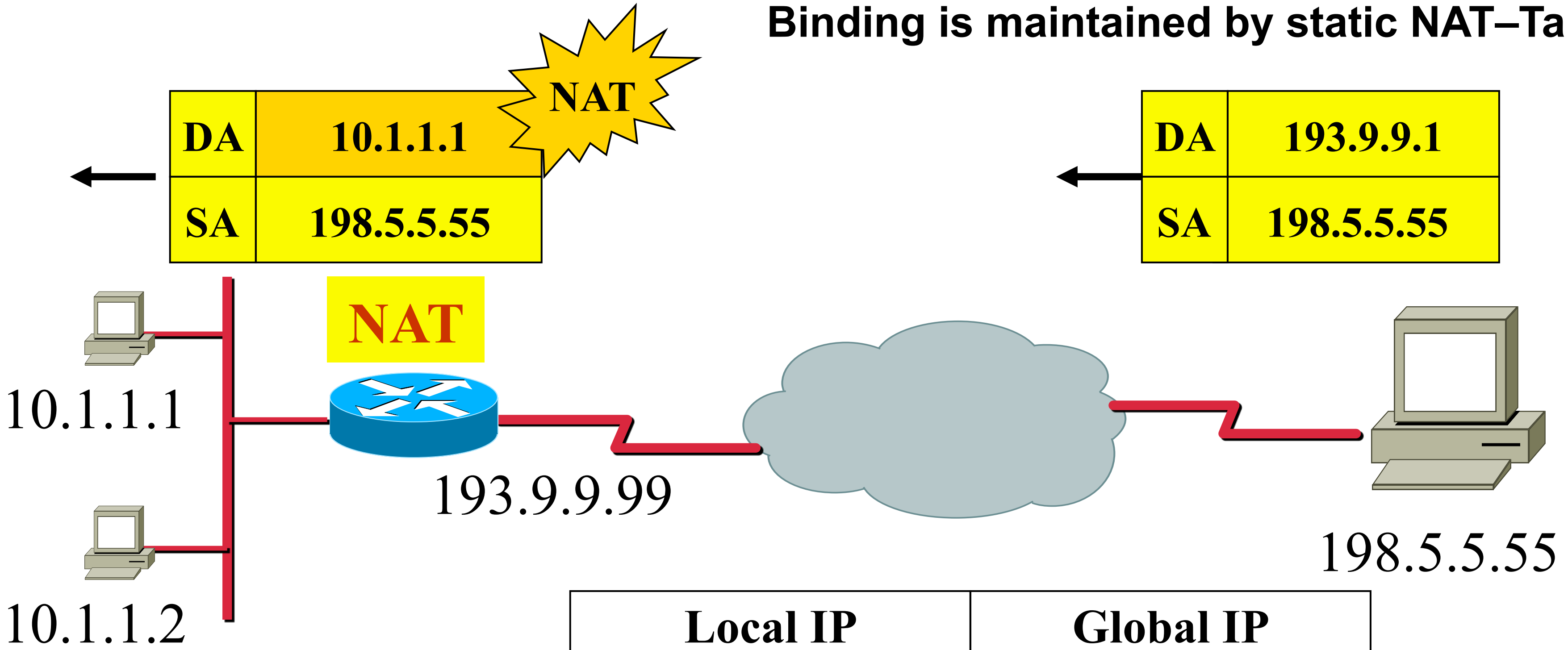


Simple Static NAT Table

Local IP	Global IP
10.1.1.1	193.9.9.1
10.1.1.2	193.9.9.2
....	....

Basic Principle (2)

Binding is maintained by static NAT–Table



Local IP	Global IP
10.1.1.1	193.9.9.1
10.1.1.2	193.9.9.2
....	....

Simple Static
NAT Table

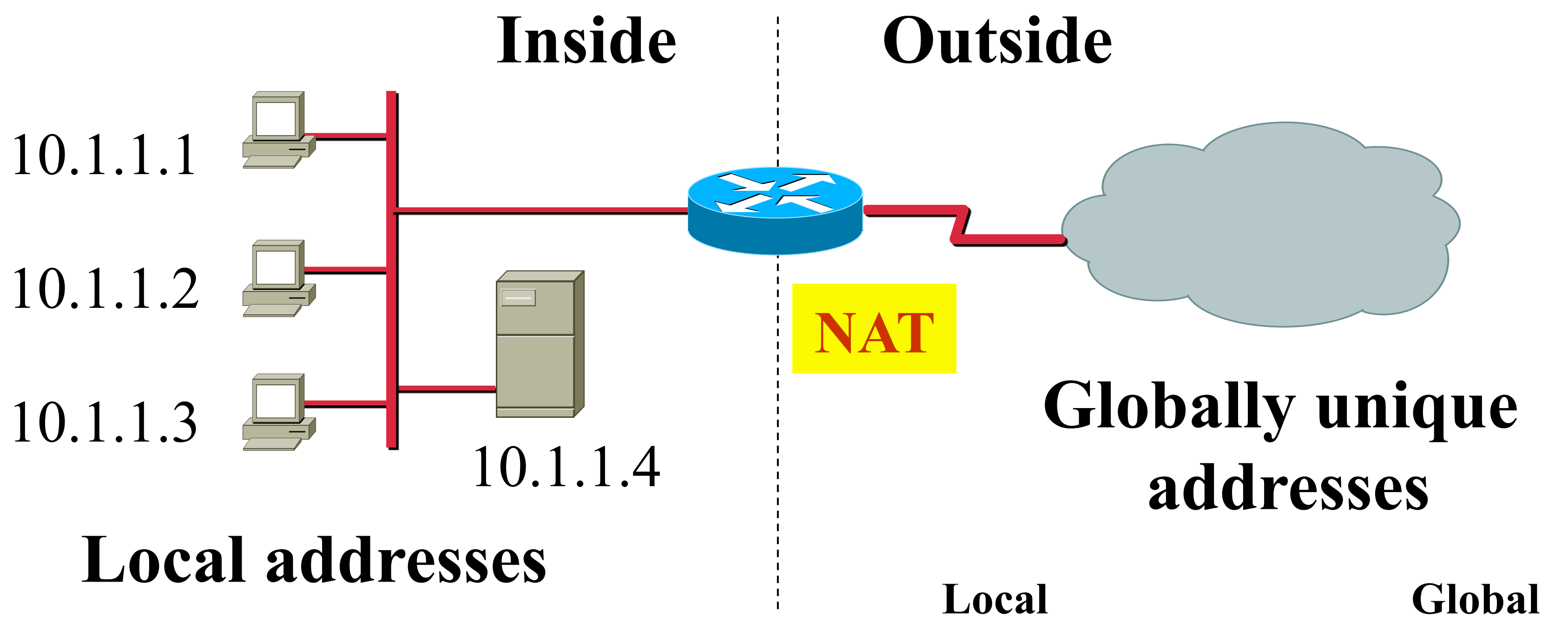
NAT Tasks and Behaviour

- Modify IP addresses according to NAT table
- But also must modify the IP checksum and the TCP checksum
- Must also look out for ICMP and modify the places where the IP address appears
- There may be other places, where modifications must be done
  - E.g. FTP, NetBIOS over TCP/IP, SNMP, DNS, Kerberos, X-Windows, SIP, H.323, IPsec, IKE…
- The sender and receiver (should) remain unaware that NAT is taking place

NAT Binding Possibilities

- **Static (“Fixed Binding”)**
  - In case of one-to-one mapping of local to global addresses
- **Dynamic (“Binding on the fly”)**
  - In case of sharing a pool of global addresses
  - Connections initiated by private hosts are assigned a global address from the pool
  - As long as the private host has an outgoing connection, it can be reached by incoming packets sent to this global address
  - After the connection is terminated (or a timeout is reached), the binding expires, and the address is returned to the pool for reuse
  - Is more complex because state must be maintained, and connections must be rejected when the pool is exhausted
  - Unlike static binding, dynamic binding enables address reuse, reducing the demand for globally unique addresses.

Scenario Dynamic Binding



Local IP address		Global IP address
10.1.1.1	⟷	193.99.99.1
10.1.1.2	⟷	193.99.99.2
10.1.1.3	Currently not possible	
10.1.1.4	Currently not possible	

Binding is maintained by dynamic NAT–Table

Note: a connection state or timer must be maintained per mapping

Agenda

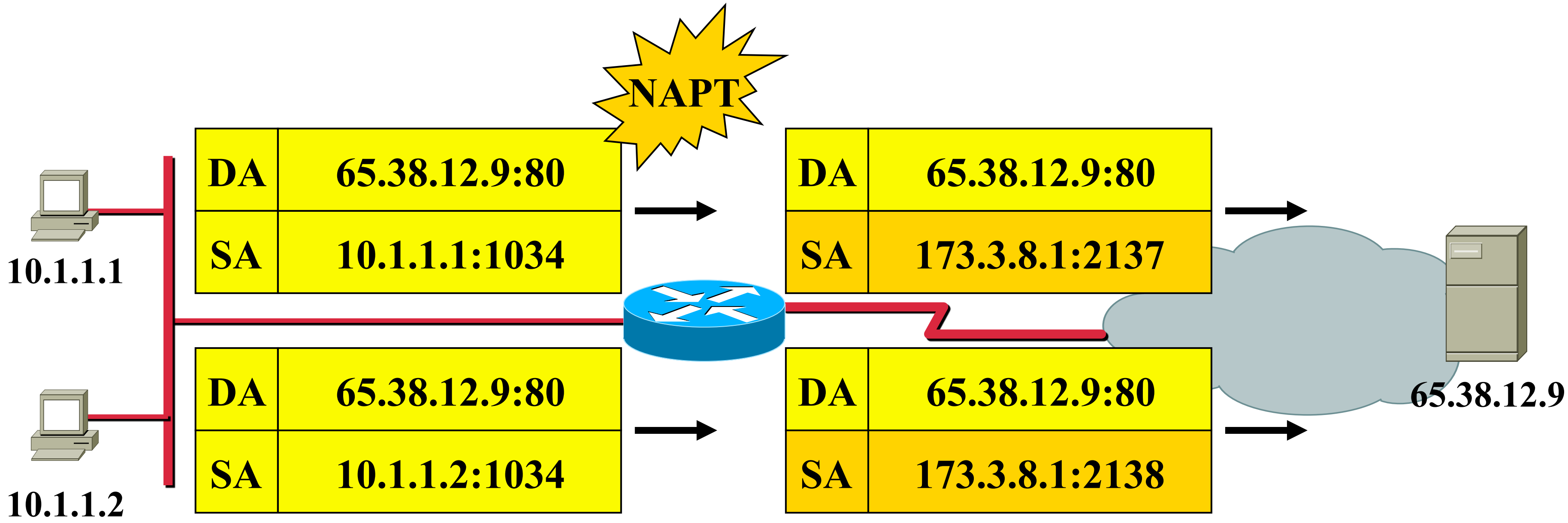
- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

Overloading (NAPT)

- Common problem:
  - **Many hosts inside initiating connections to the outside world**
  - **But only one or a few inside-global addresses available**

- Solution:
  - **Many-to-one Translation with NAPT (Network Address Port Translation)**
  - **Usable in context of TCP and UDP sessions**
  - **Aka *"Overloading Global Addresses*"**
  - **Aka "*PAT„ (Port Address Translation)***

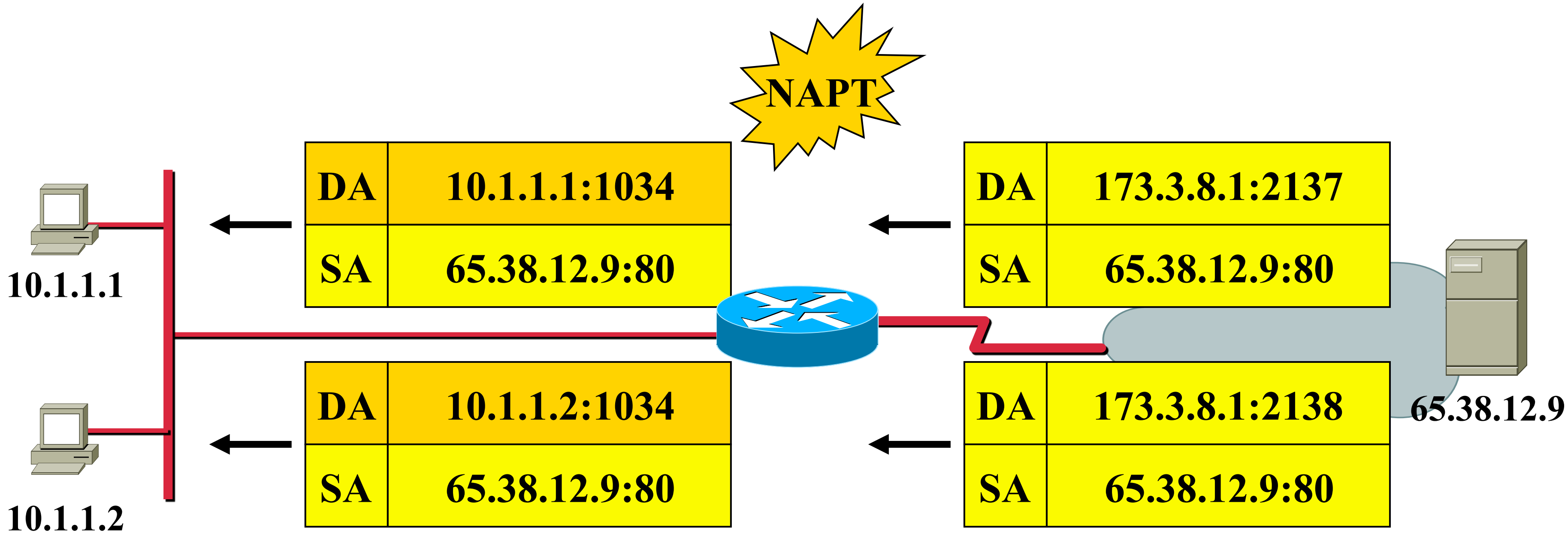
NAPT Example (1)



Prot.	Local	Global
TCP	10.1.1.1:1034	173.3.8.1:2137
TCP	10.1.1.2:1034	173.3.8.1:2138

Extended Translation Table

NAPT Example (2)



Prot.	Local	Global
TCP	10.1.1.1:1034	173.3.8.1:2137
TCP	10.1.1.2:1034	173.3.8.1:2138

Extended Translation Table

Agenda

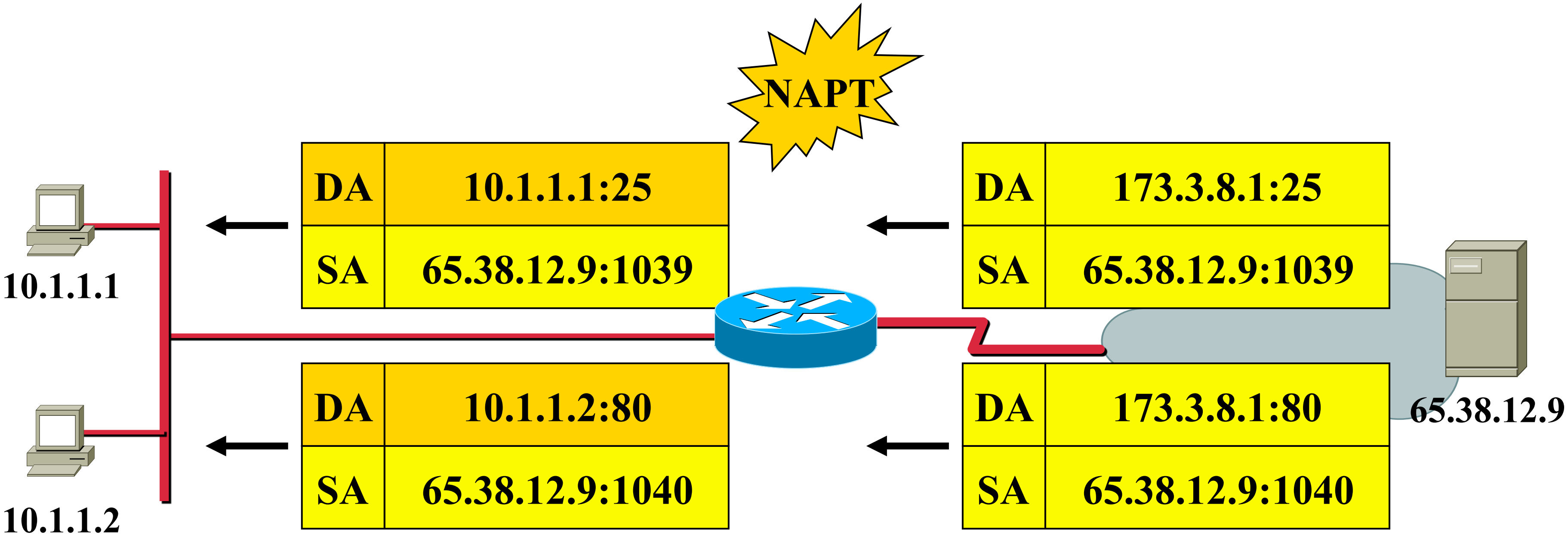
- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

Virtual Server Table

- Problem:
  - **How to reach an inside server from the outside**
  - **NAPT/NAT let IP datagram's (with UDP or TCP segments as payload) from to outside only in if a binding is found**
  - **But server waits for connections from the outside hence cannot install binding in the NAPT/NAT device**

- Solution:
  - **Virtual Server Table**
  - **Creating manually a static binding in the NAPT/NAT device to forward IP datagram's to the real inside server**

Virtual Server Table Example



Prot.	Local	Global
TCP	10.1.1.1:25	173.3.8.1:25
TCP	10.1.1.2:80	173.3.8.1:80

Extended Translation Table

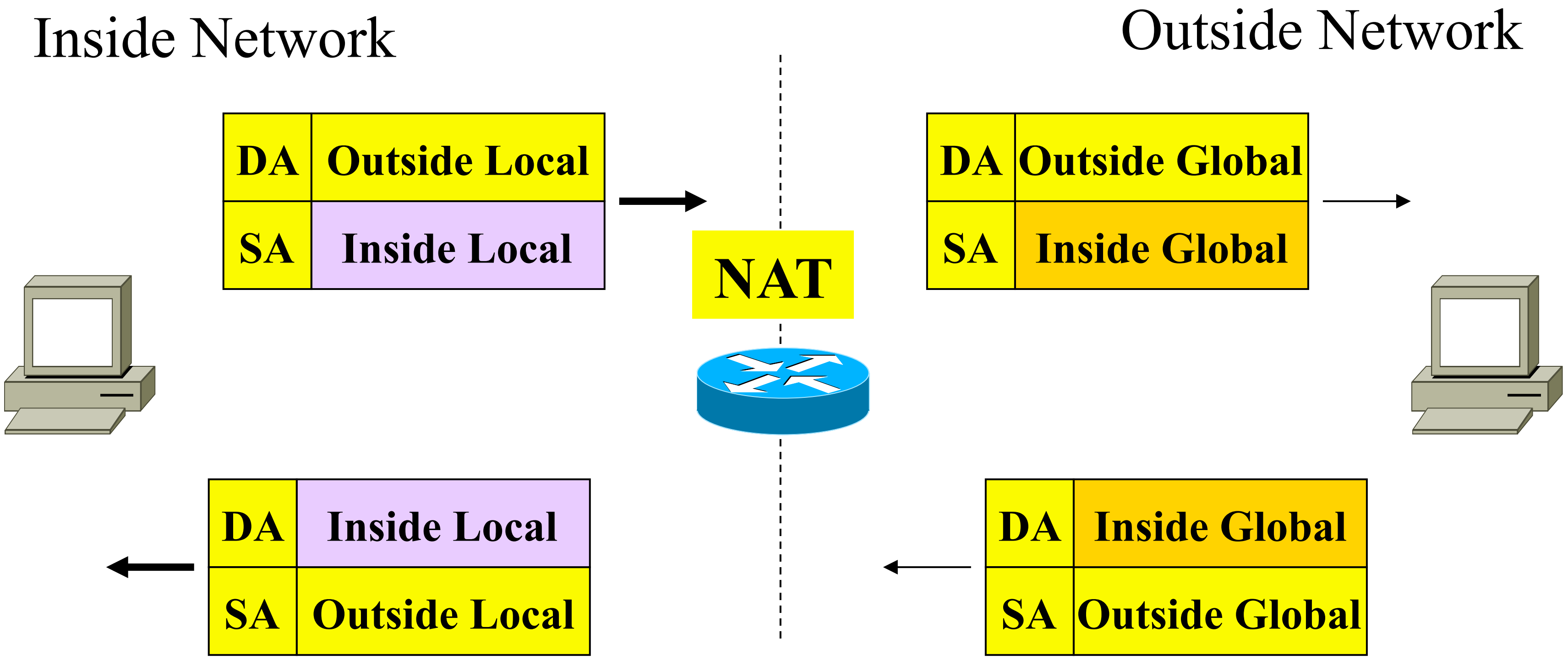
Agenda

- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

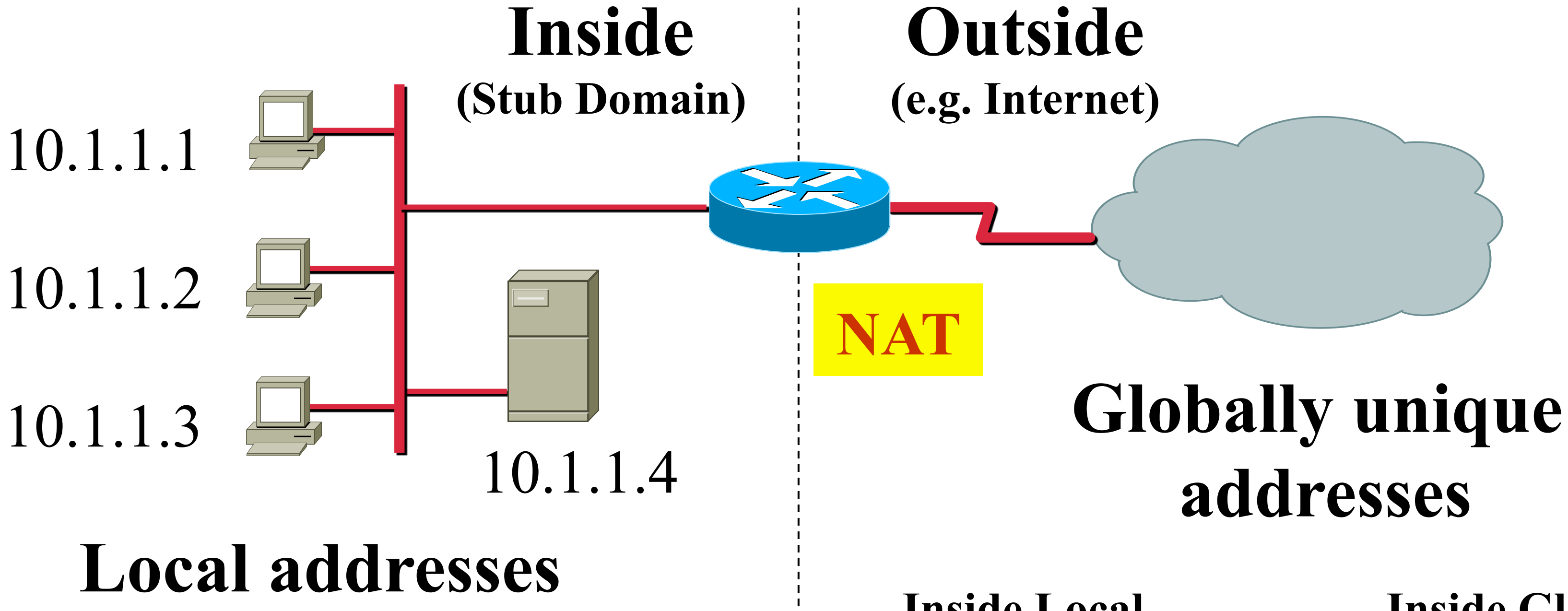
Terms Used in complex NAT Devices

FYI

- *Local* versus *global* address
  - Reflects area of usage (inside or outside)
- *Inside* versus *outside* world
  - Reflects the origin



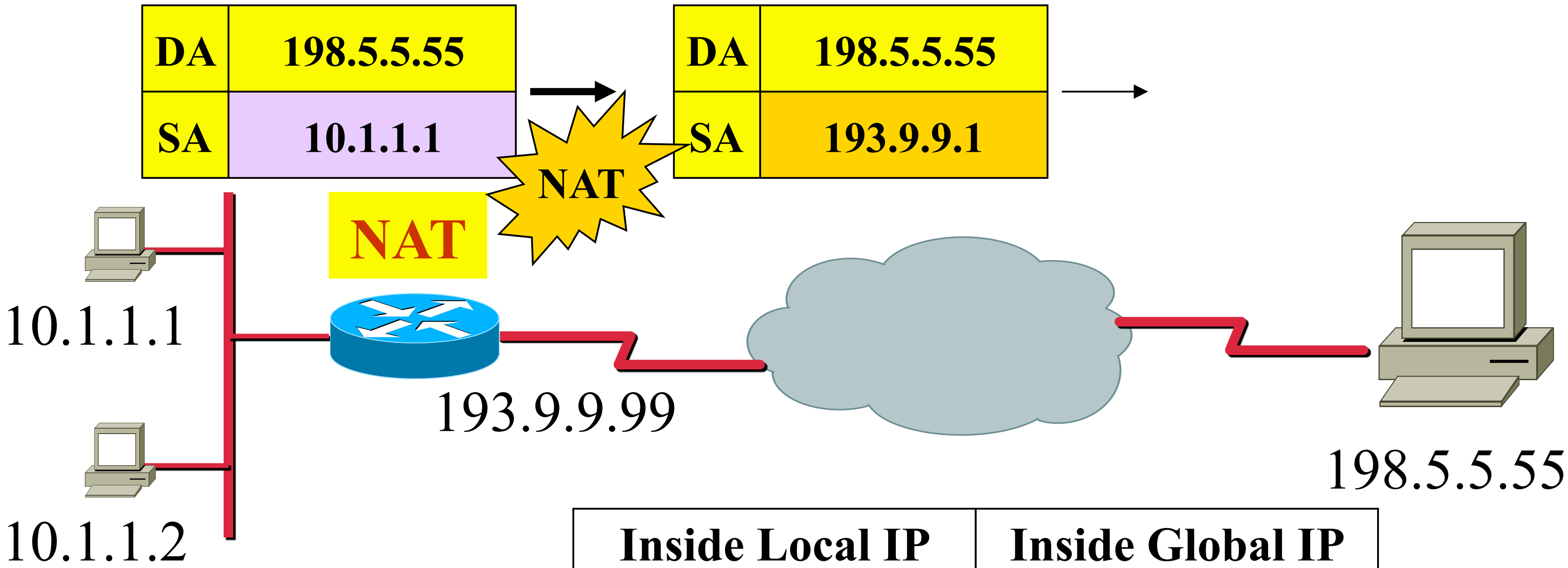
Static NAT Example with New Terms



Binding is maintained by static NAT–Table

Inside Local IP address		Inside Global IP address
10.1.1.1	⟷	193.99.99.1
10.1.1.2	⟷	193.99.99.2
10.1.1.3	⟷	193.99.99.3
10.1.1.4	⟷	193.99.99.4

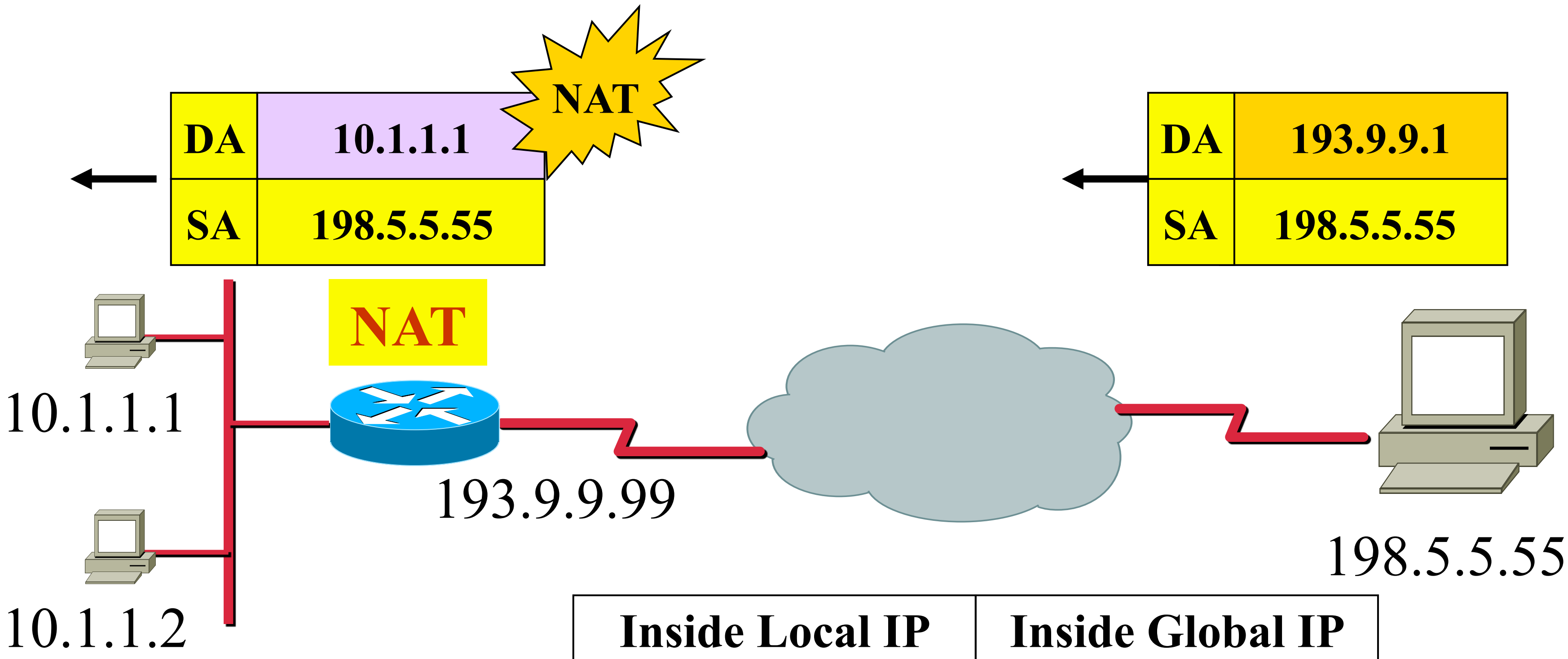
Basic Principle (1a) with New Terms Inside Address Translation



Simple NAT Table

Inside Local IP	Inside Global IP
10.1.1.1	193.9.9.1
10.1.1.2	193.9.9.2
....	....

Basic Principle (1b) with New Terms
Inside Address Translation



Simple NAT Table

Inside Local IP	Inside Global IP
10.1.1.1	193.9.9.1
10.1.1.2	193.9.9.2
....	....

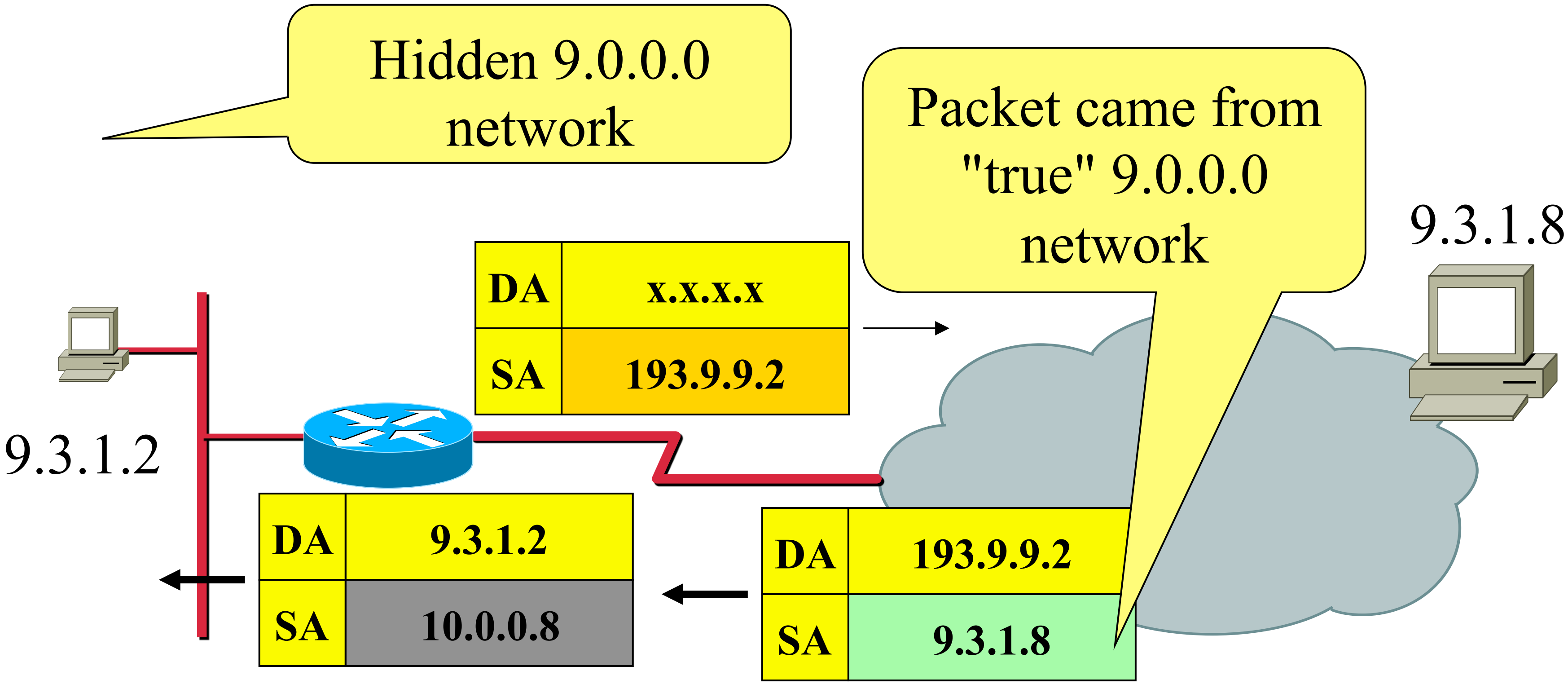
Overlapping Networks

= Same addresses **are used**

locally and ***globally***

What can happen?

Outside Address Translation

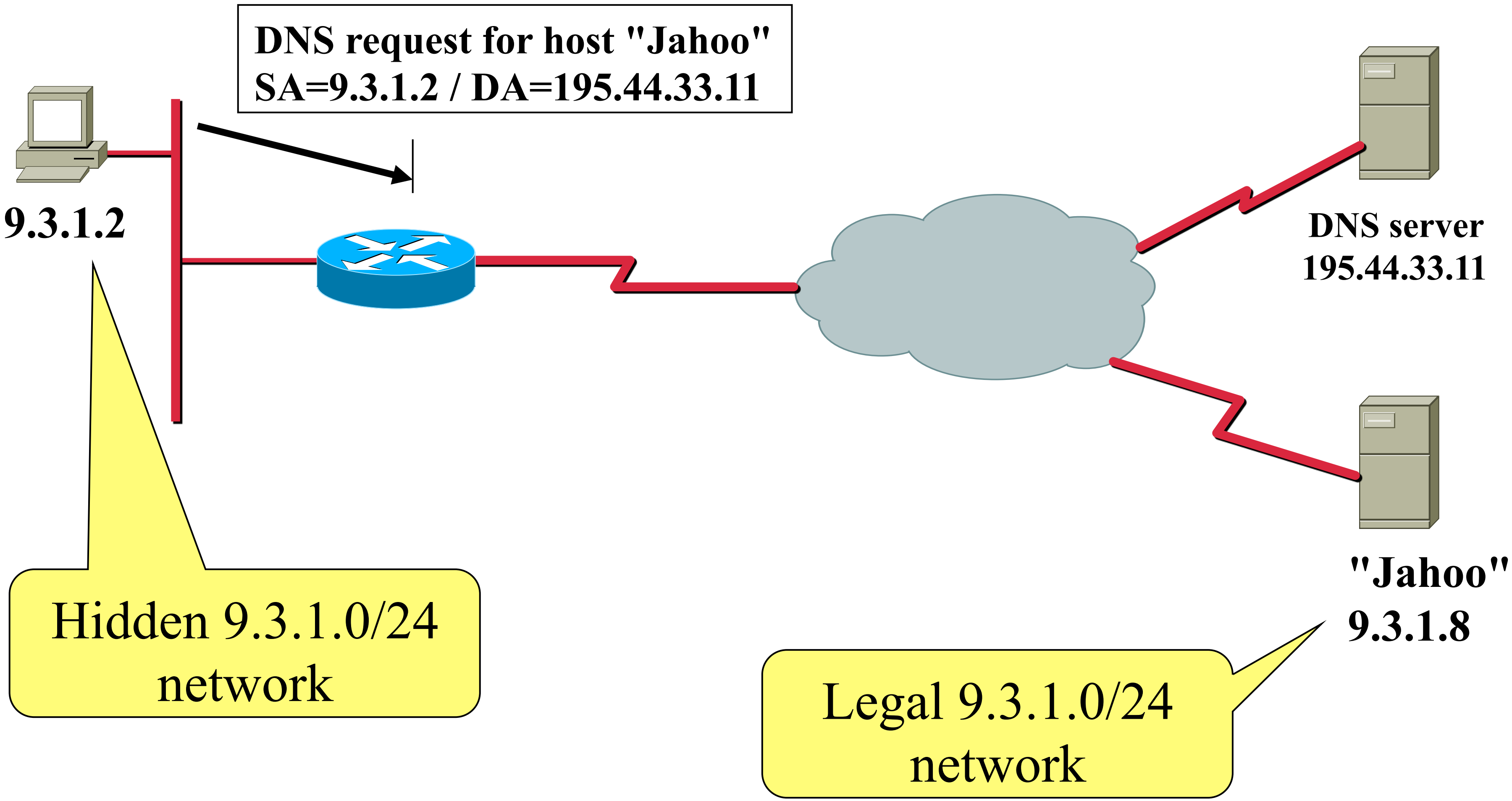


Inside Local	Inside Global	Outside Local	Outside Global
9.3.1.2	193.9.9.2	10.0.0.8	9.3.1.8

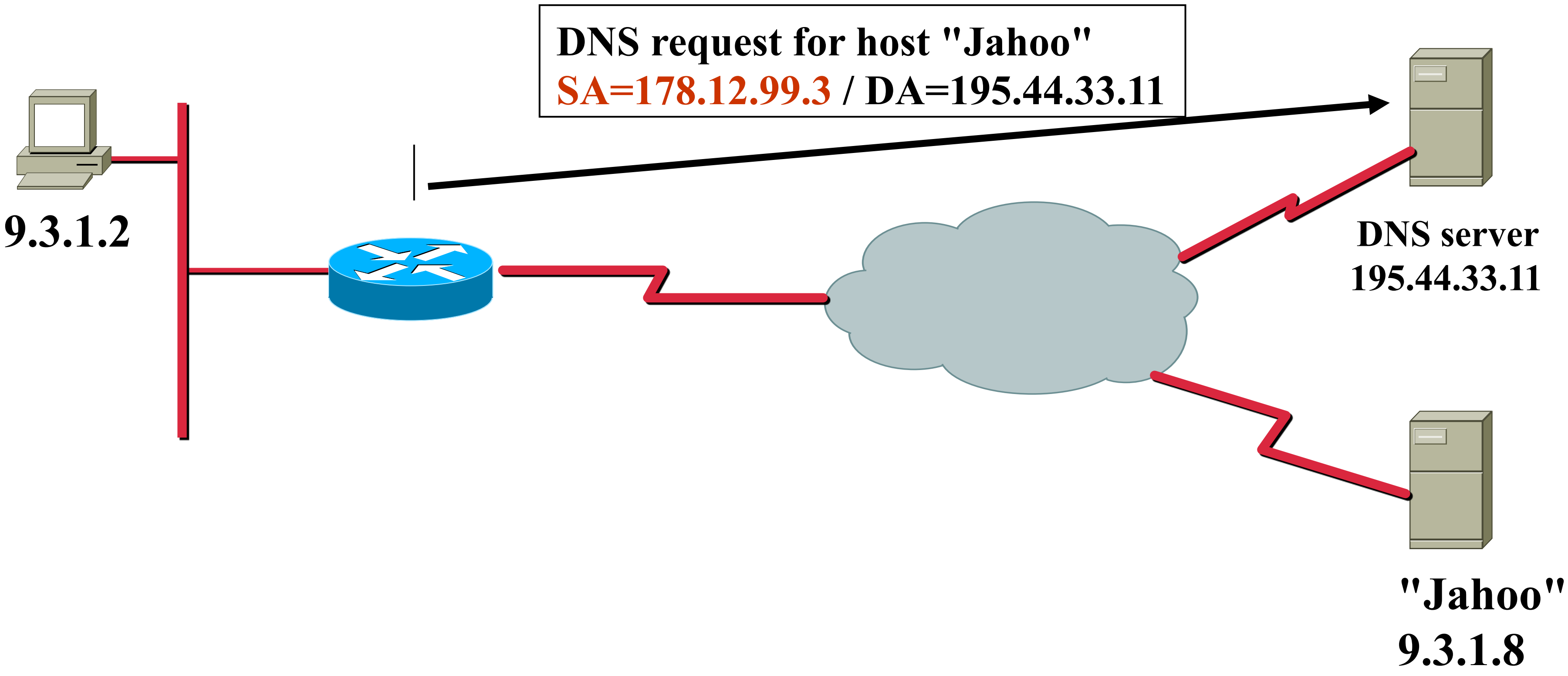
Agenda

- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

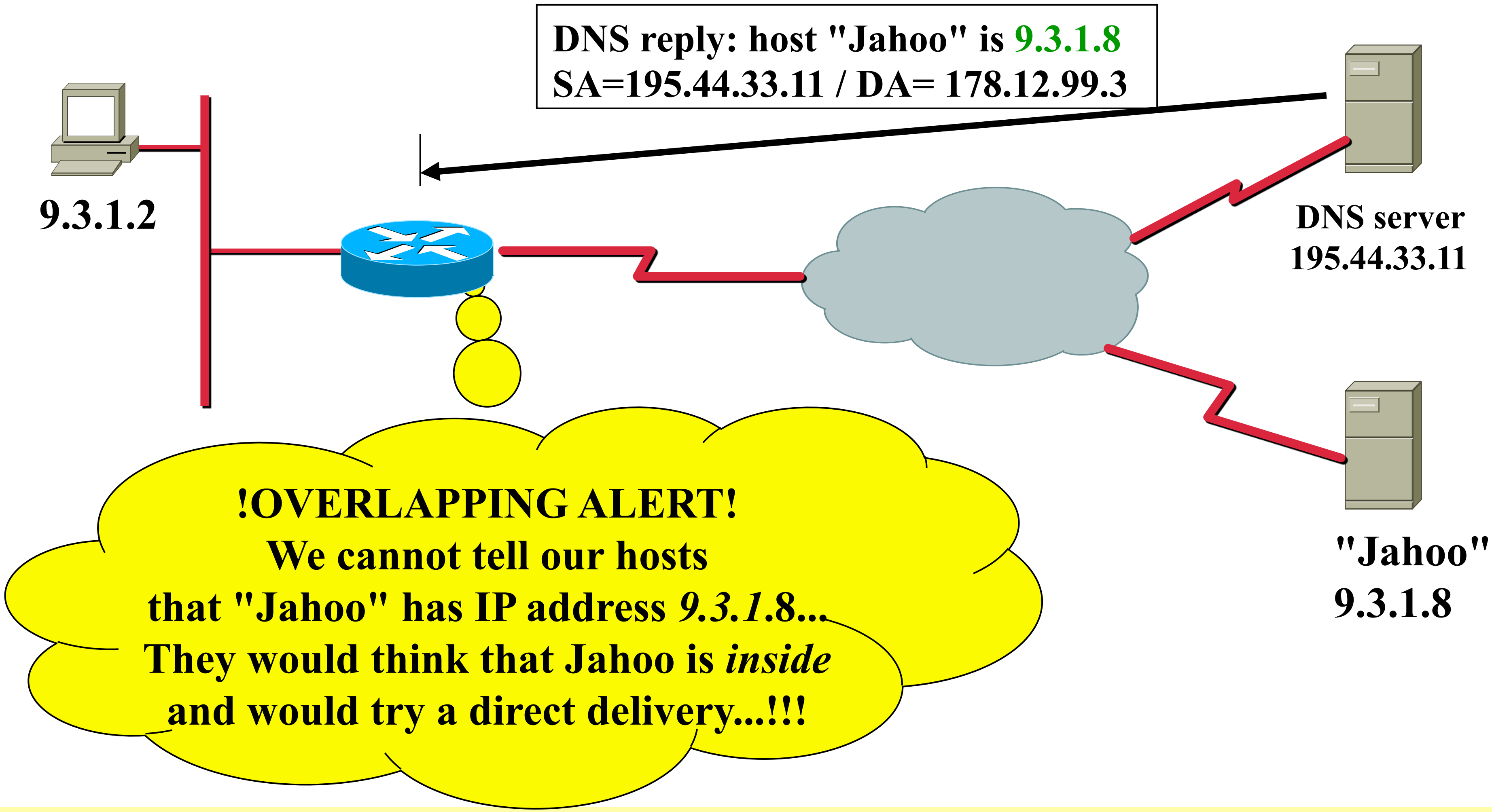
DNS Problem (1)



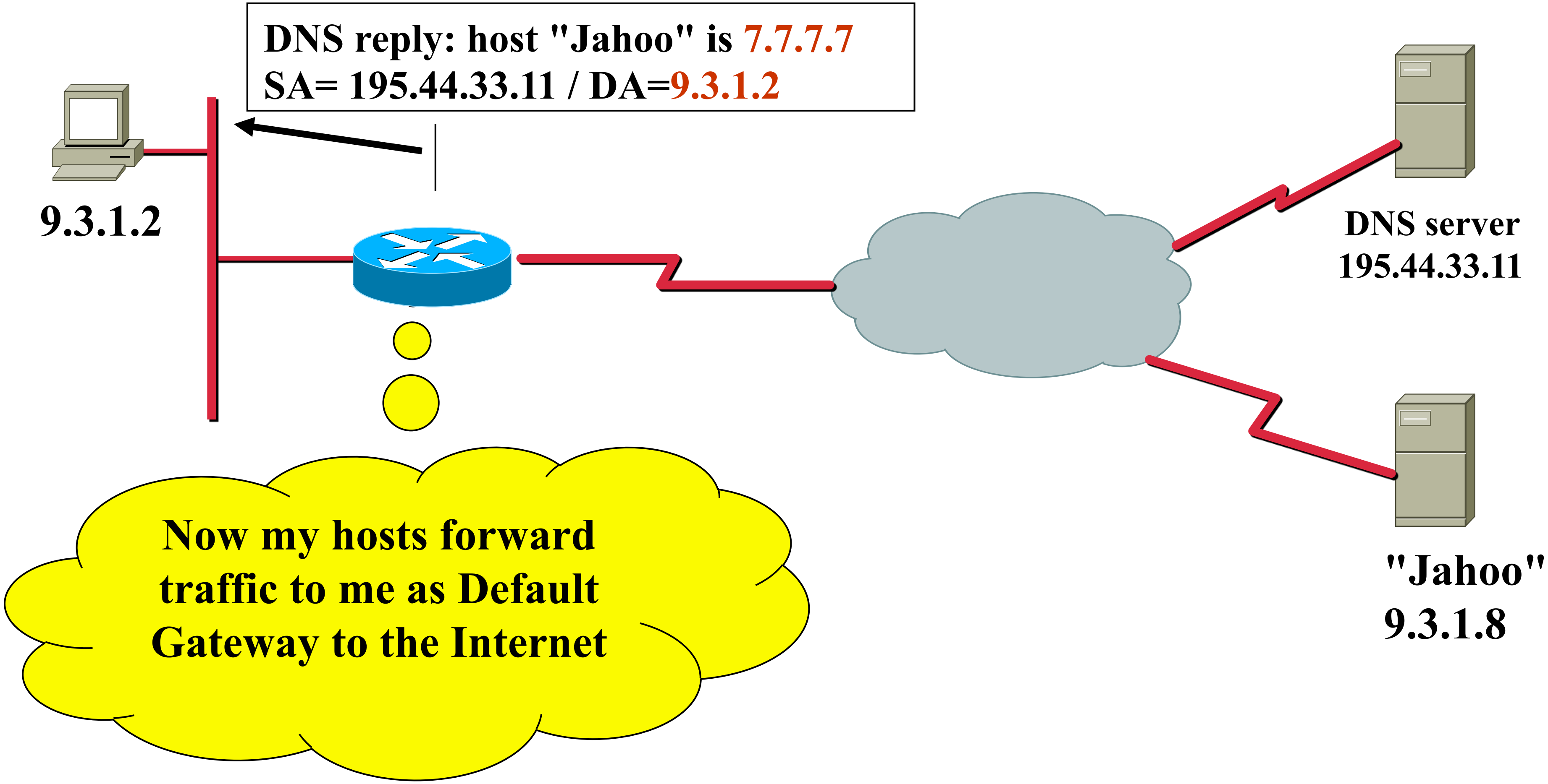
DNS Problem (2)



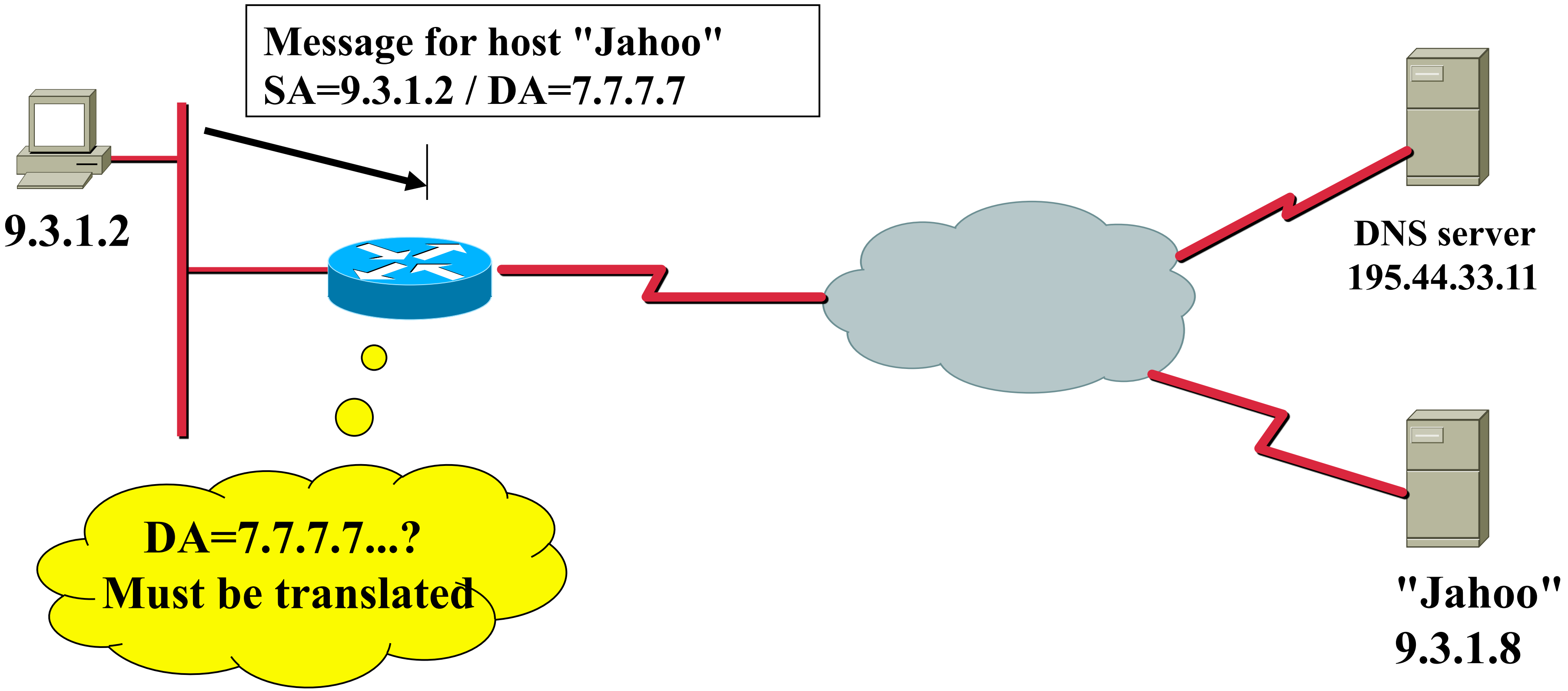
DNS Problem (3)



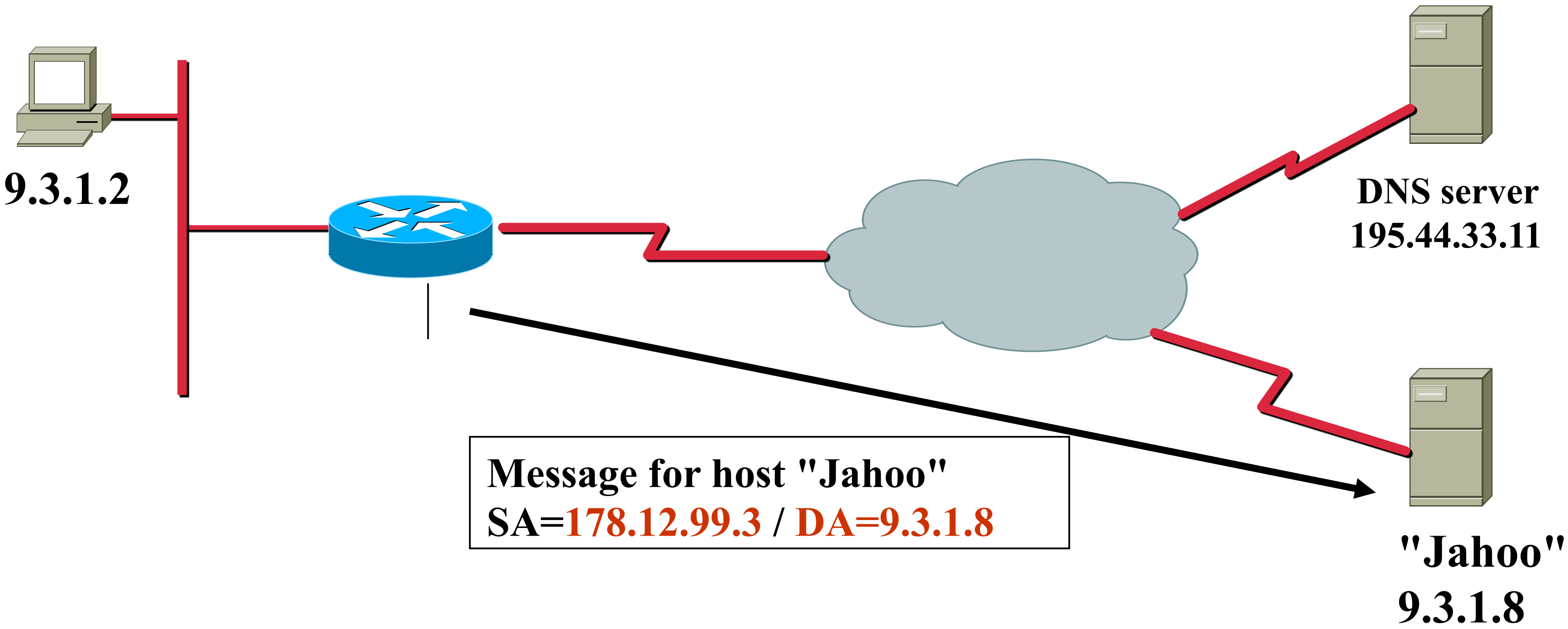
DNS Problem (4)



DNS Problem (5)



DNS Problem (6)



NAT Table

Inside Local	Inside Global	Outside Global	Outside Local
9.3.1.2	178.12.99.3	9.3.1.8	7.7.7.7

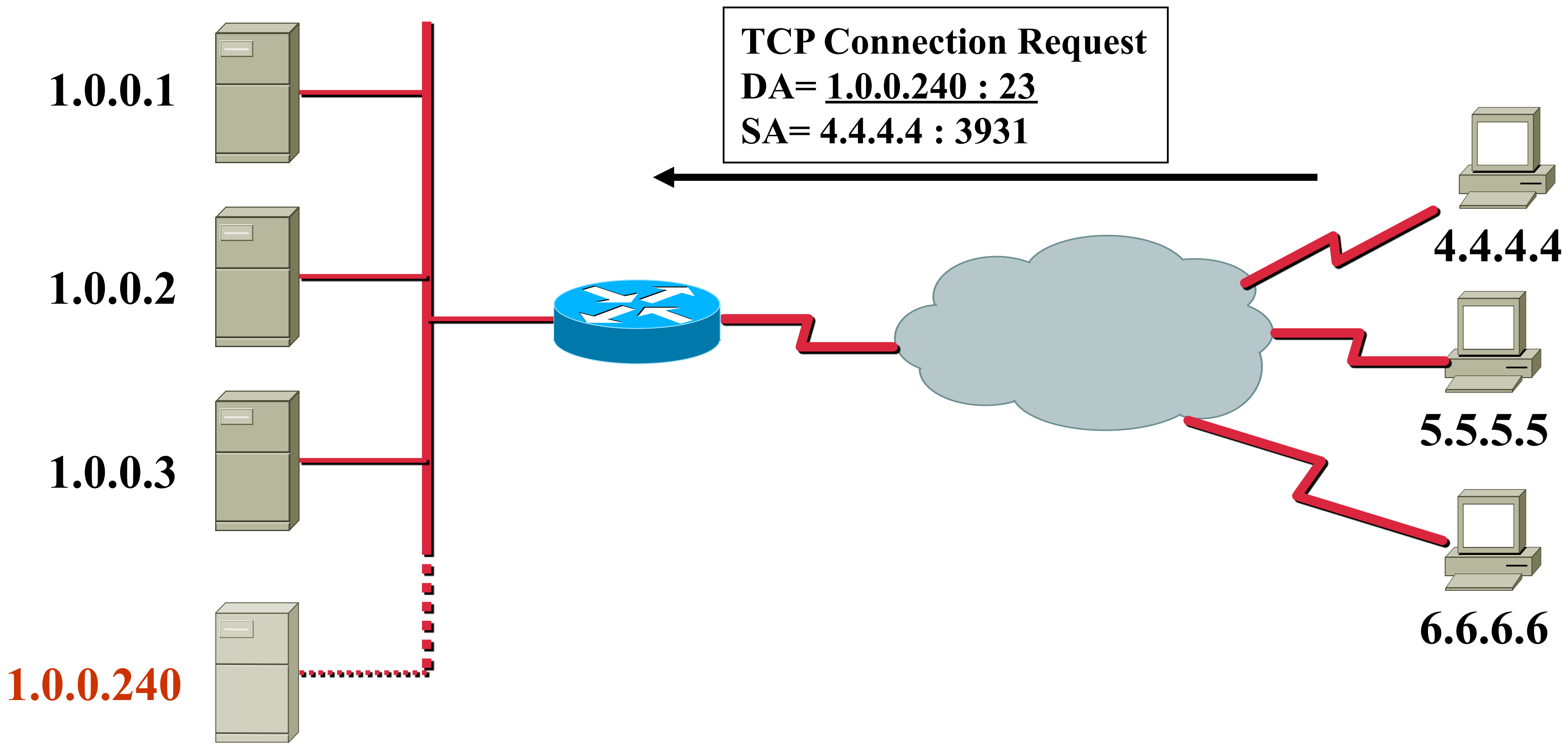
Agenda

- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  - NAT Basics
  - NAPT
  - Virtual Server
  - Complex NAT
  - DNS Aspects
  - Load Balancing
  - RFCs

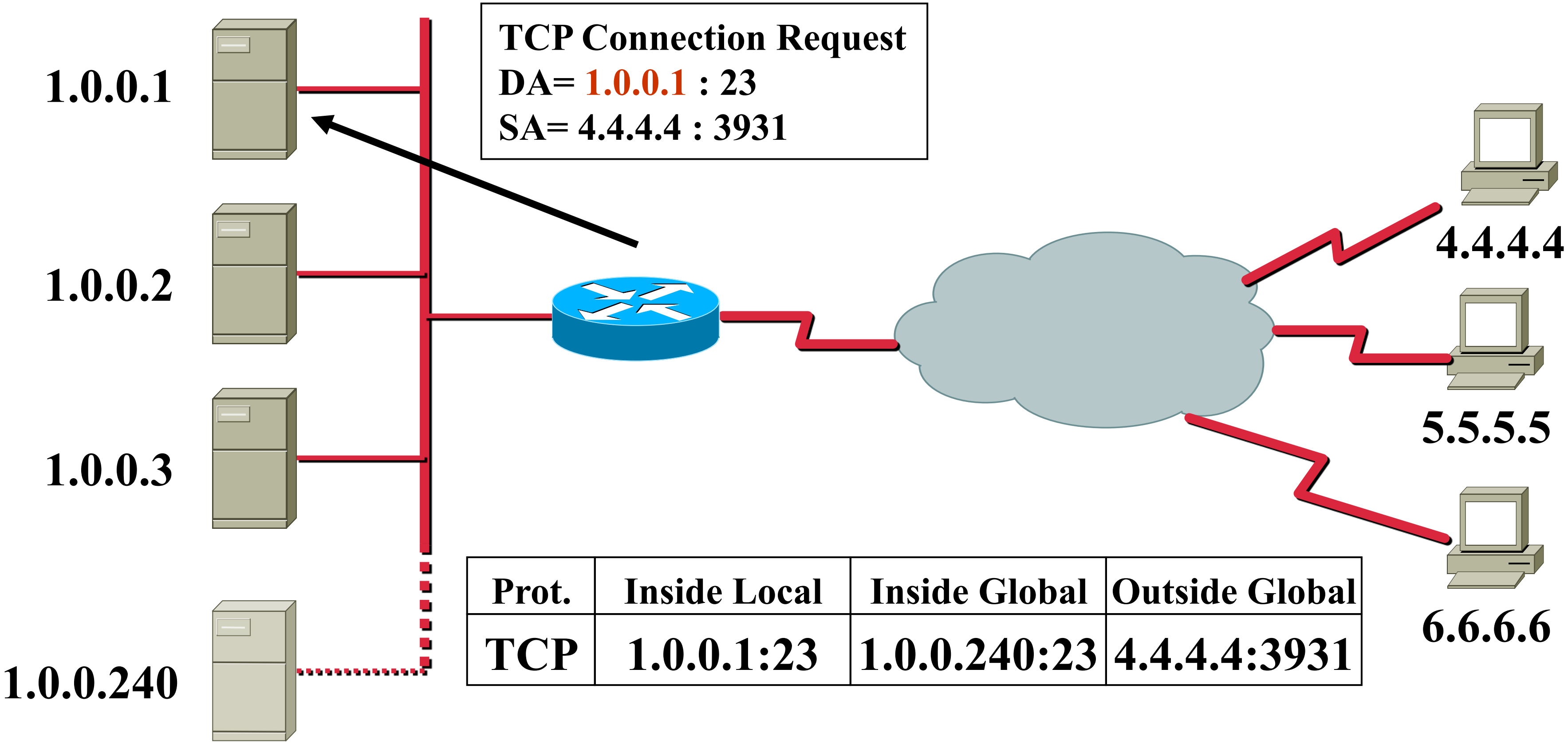
TCP Load Sharing (1)

- **Multiple servers represented by a single inside-global IP address**
  - *Virtual host* address
- **New TCP session requests to the Virtual Host are forwarded to one of a group of real hosts**
  - *Rotary group*

TCP Load Sharing (2)

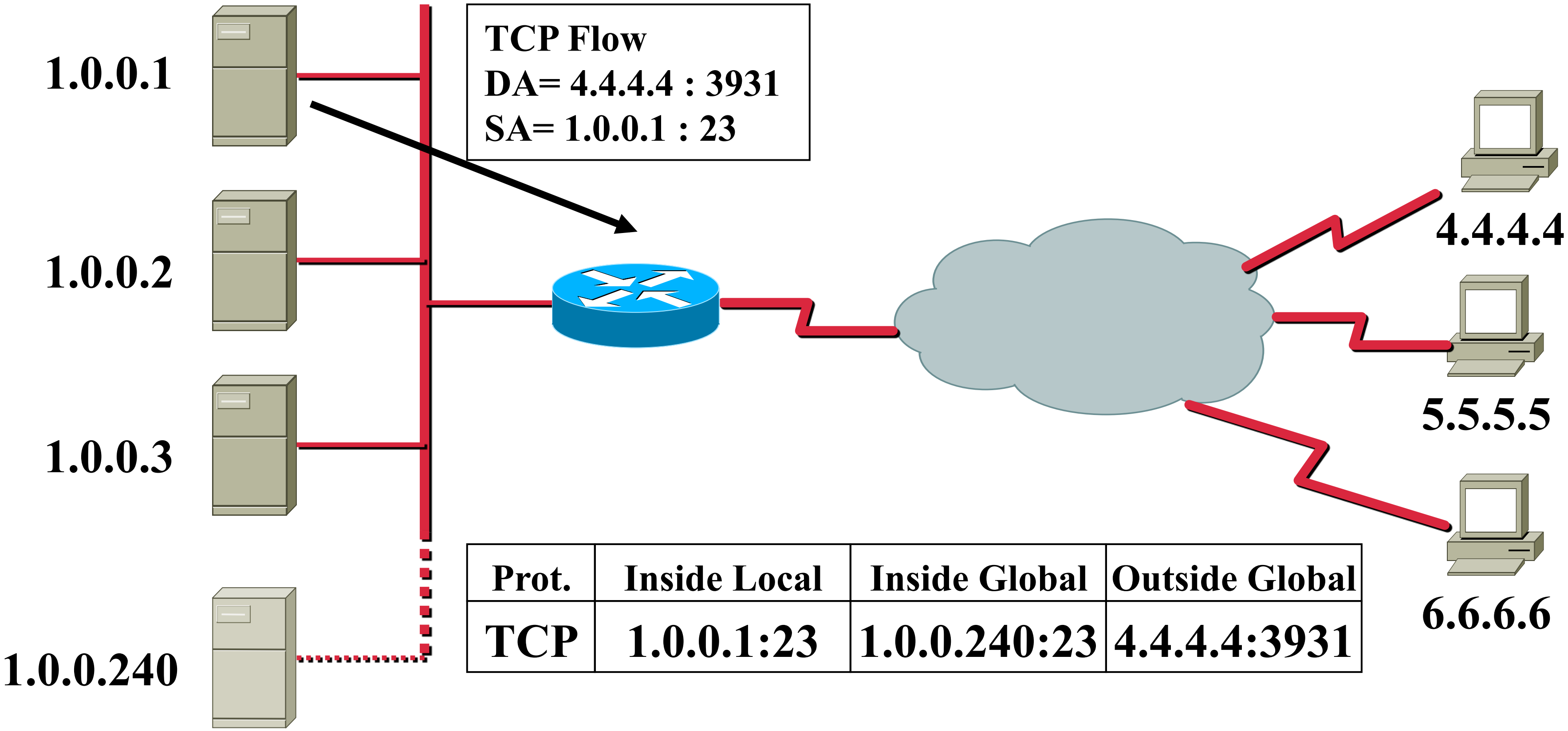


TCP Load Sharing (3)



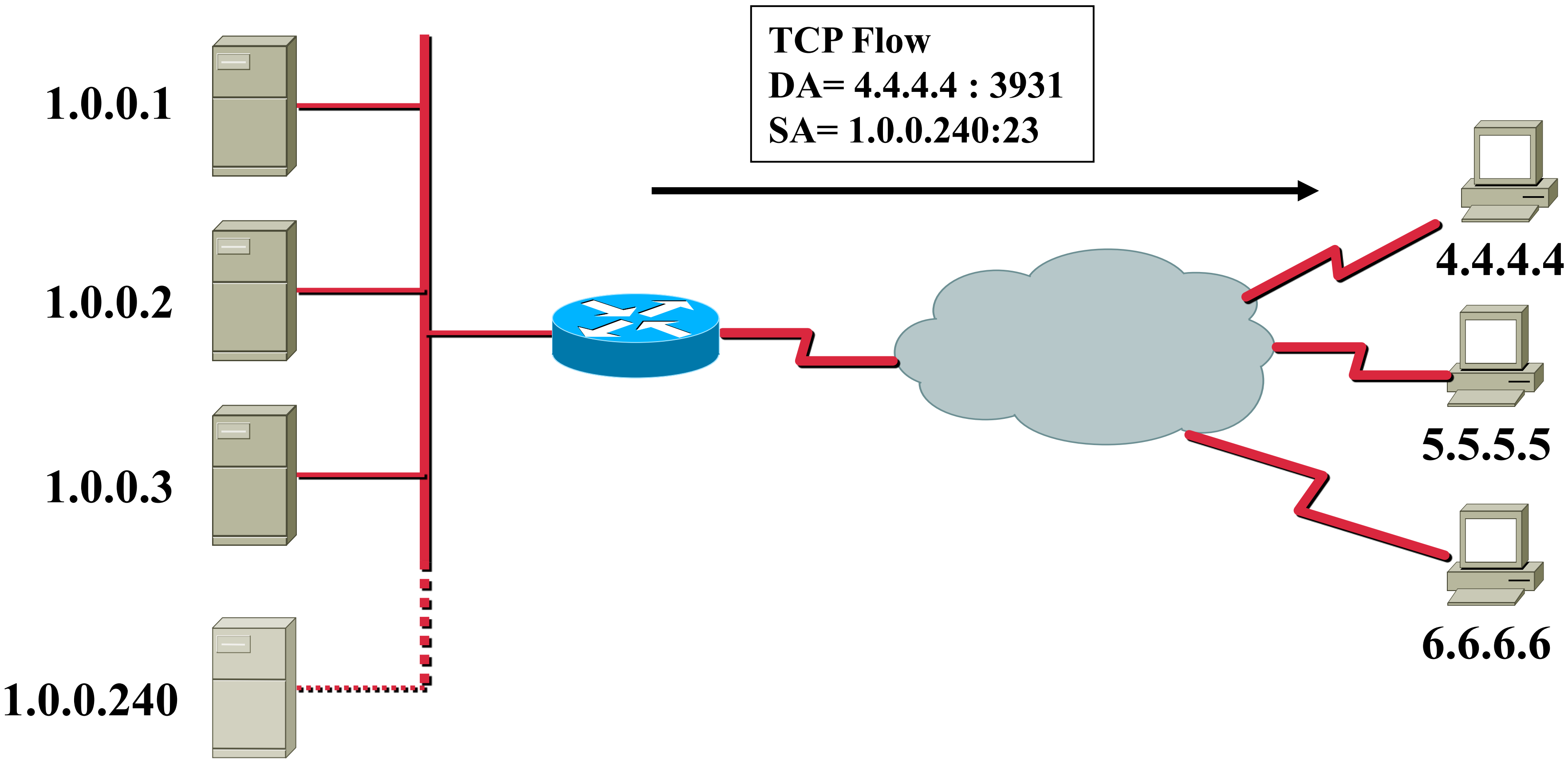
Prot.	Inside Local	Inside Global	Outside Global
TCP	1.0.0.1:23	1.0.0.240:23	4.4.4.4:3931

TCP Load Sharing (4)

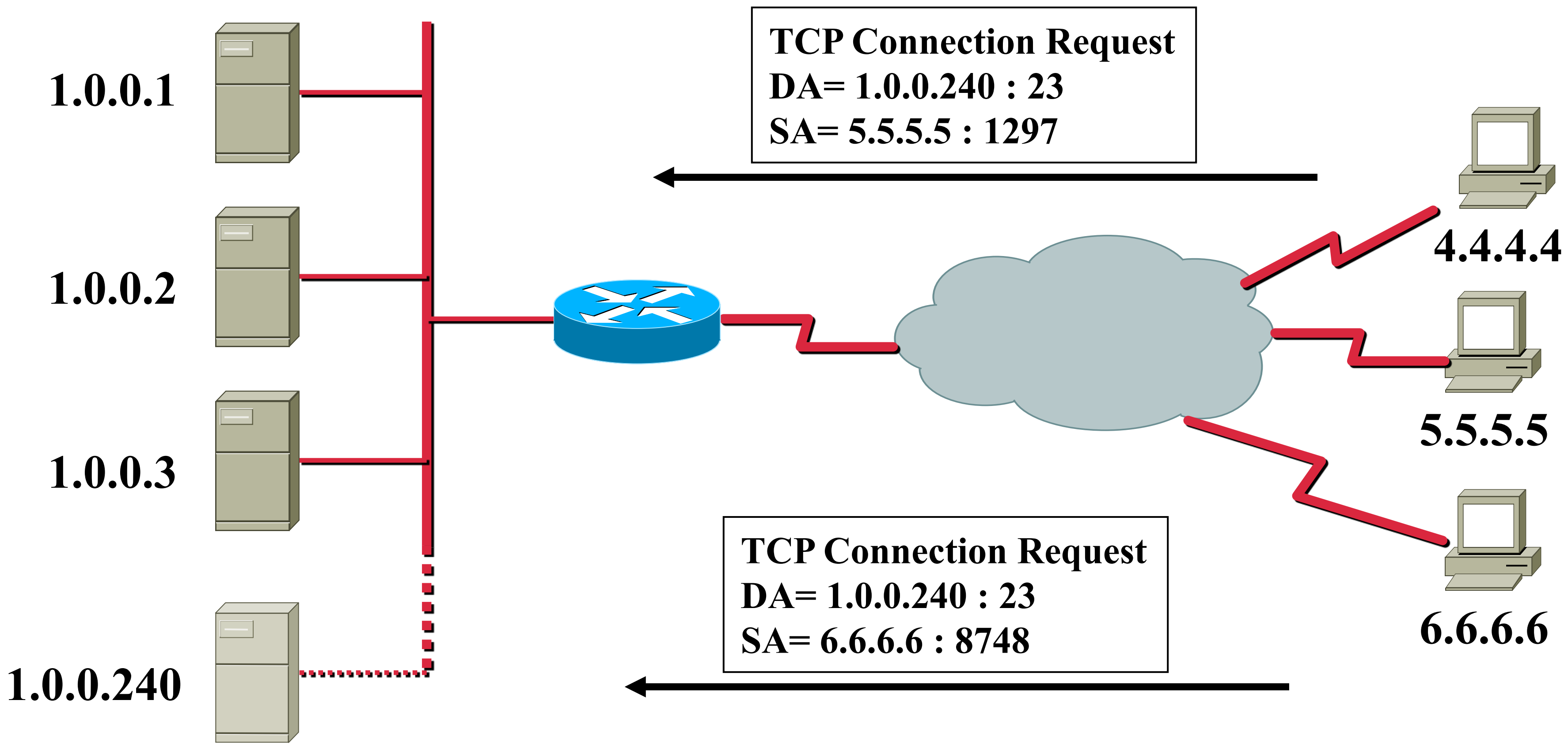


Prot.	Inside Local	Inside Global	Outside Global
TCP	1.0.0.1:23	1.0.0.240:23	4.4.4.4:3931

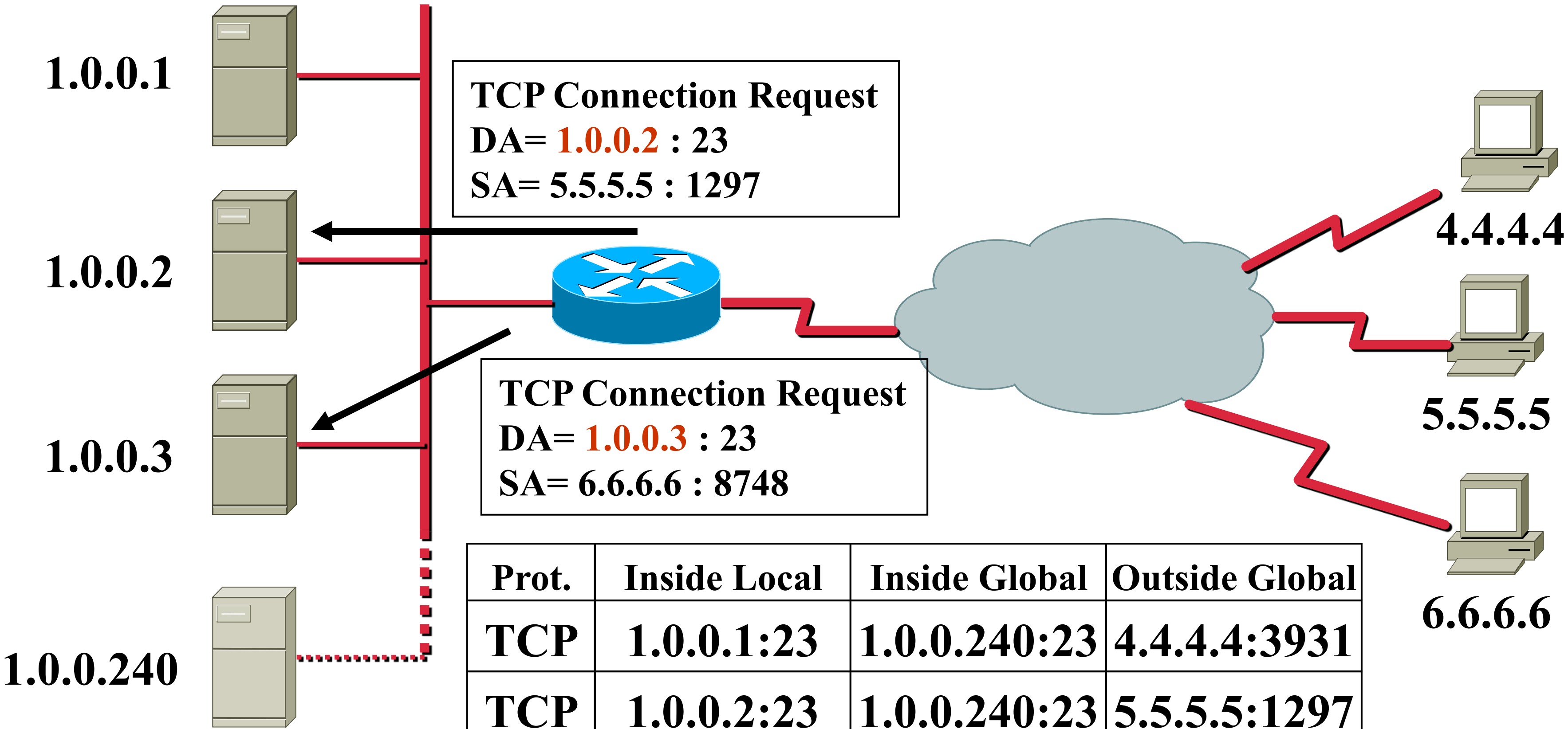
TCP Load Sharing (5)



TCP Load Sharing (6)



TCP Load Sharing (7)



Prot.	Inside Local	Inside Global	Outside Global
TCP	1.0.0.1:23	1.0.0.240:23	4.4.4.4:3931
TCP	1.0.0.2:23	1.0.0.240:23	5.5.5.5:1297
TCP	1.0.0.3:23	1.0.0.240:23	6.6.6.6:8748

Agenda

- **TCP Fundamentals**
- **TCP Performance**
- **UDP**
- **RFC Collection**
- **NAT**
  – NAT Basics
  – NAPT
  – Virtual Server
  – Complex NAT
  – DNS Aspects
  – Load Balancing
  – RFCs

Further Information

- **RFC 1631**
  - NAT
- **RFC 2391**
  - Load Sharing Using IP Network Address Translation (LSNAT)
- **RFC 2666**
  - IP Network Address Translator (NAT) Terminology and Considerations
- **RFC 2694**
  - DNS ALG
- **RFC 2776**
  - Network Address Translation Protocol Translation (NAT-PT)
- **RFC 2993**
  - Architectural Implications of NAT
- **RFC 3022**
  - Traditional IP Network Address Translator (Traditional NAT)

Further Information

- **RFC 3027**
  - Protocol Complications with the IP Network Address Translator,
- **RFC 3235**
  - Network Address Translator (NAT)-Friendly Application Design Guidelines
- **RFC3303**
  - Middlebox Communication Architecture and Framework
- **RFC 3424**
  - IAB Considerations for Unilateral Self Address Fixing (UNSAF) Across Network Address Translation
- **RFC 3715**
  - IPsec—Network Address Translation (NAT) Compatibility Requirements

Further Information

- **RFC 3489 STUN**
  – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs) March 2003 (Obsoleted by RFC5389)
- **RFC 5389**
  – Session Traversal Utilities for NAT (STUN) October 2008 (Obsoletes RFC3489) (Status: PROPOSED STANDARD)

- **Internet Protocol Journal**
  – www.cisco.com/ipj
    - Issue Volume 3, Number 4 (December 2000)
    - „The Trouble with NAT“
    - Issue Volume 7, Number 3 (September 2004)
    - „Anatomy (of NAT)“