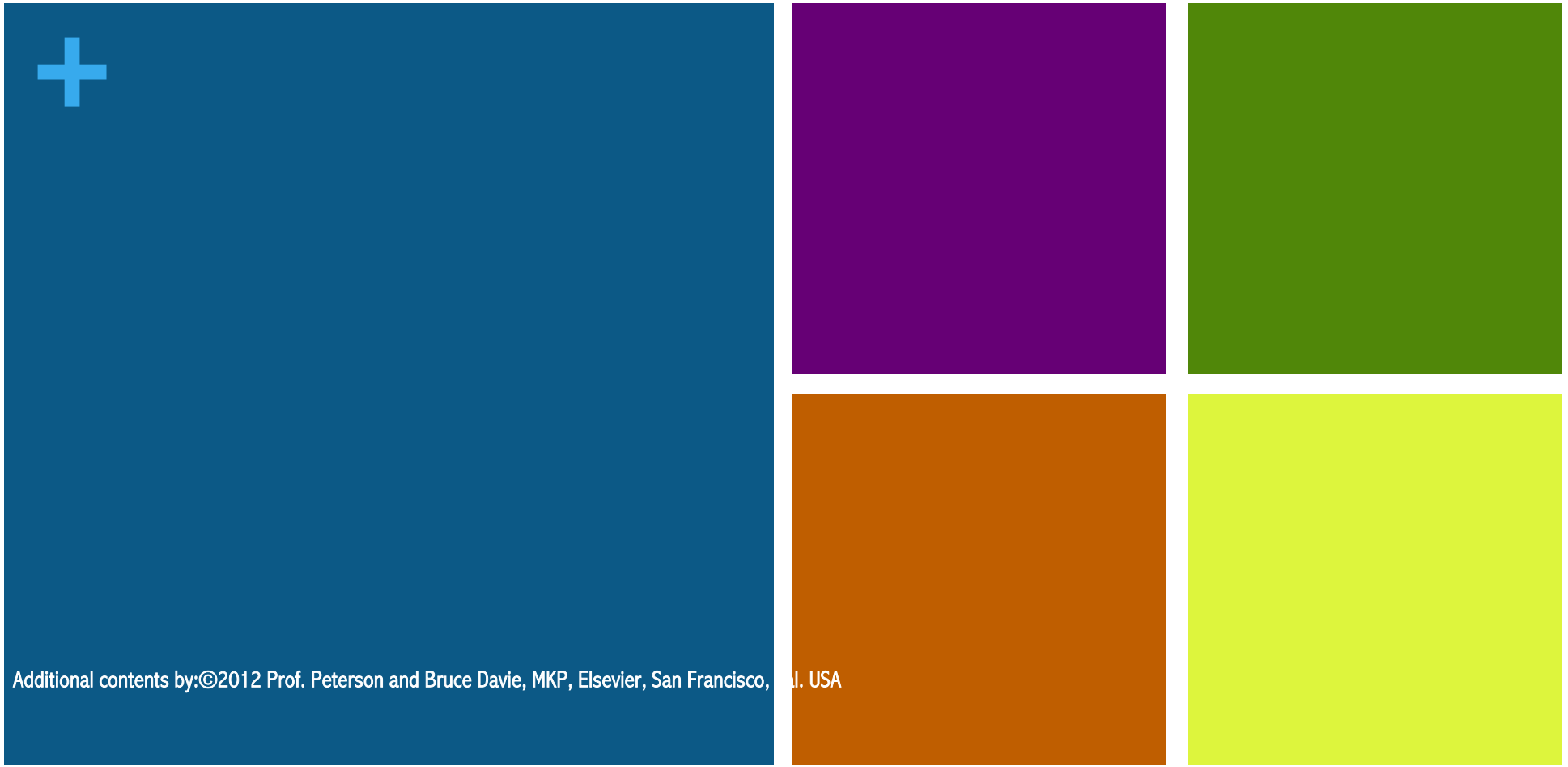


Introduction to UDP and TCP

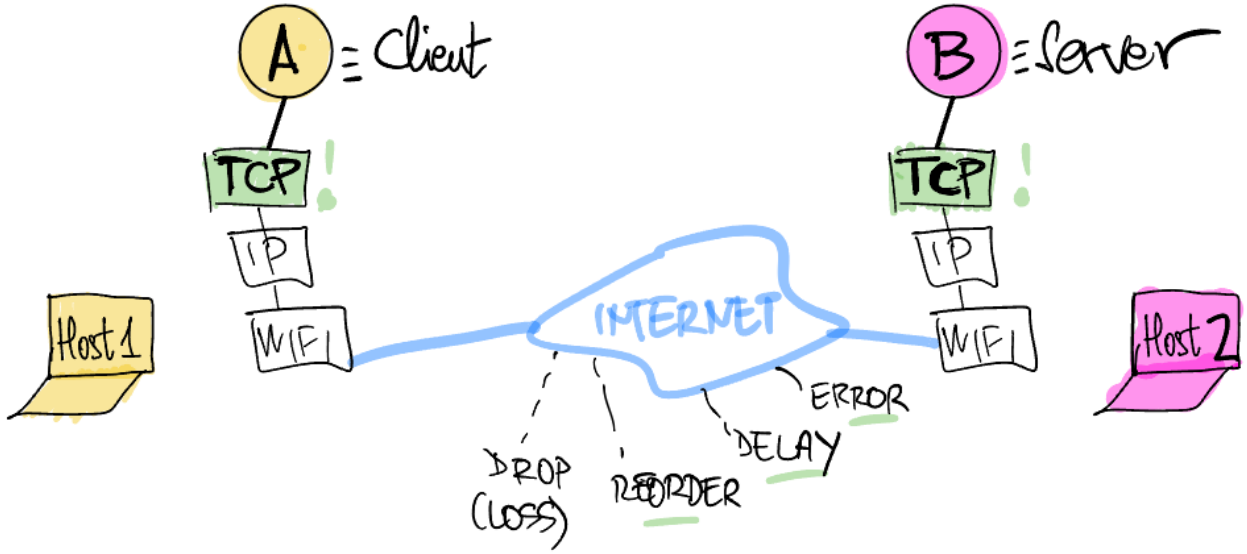
End-to-end process mapping, reliable transmission and end-to-end congestion control

Sliding Window Protocol and Flow Control

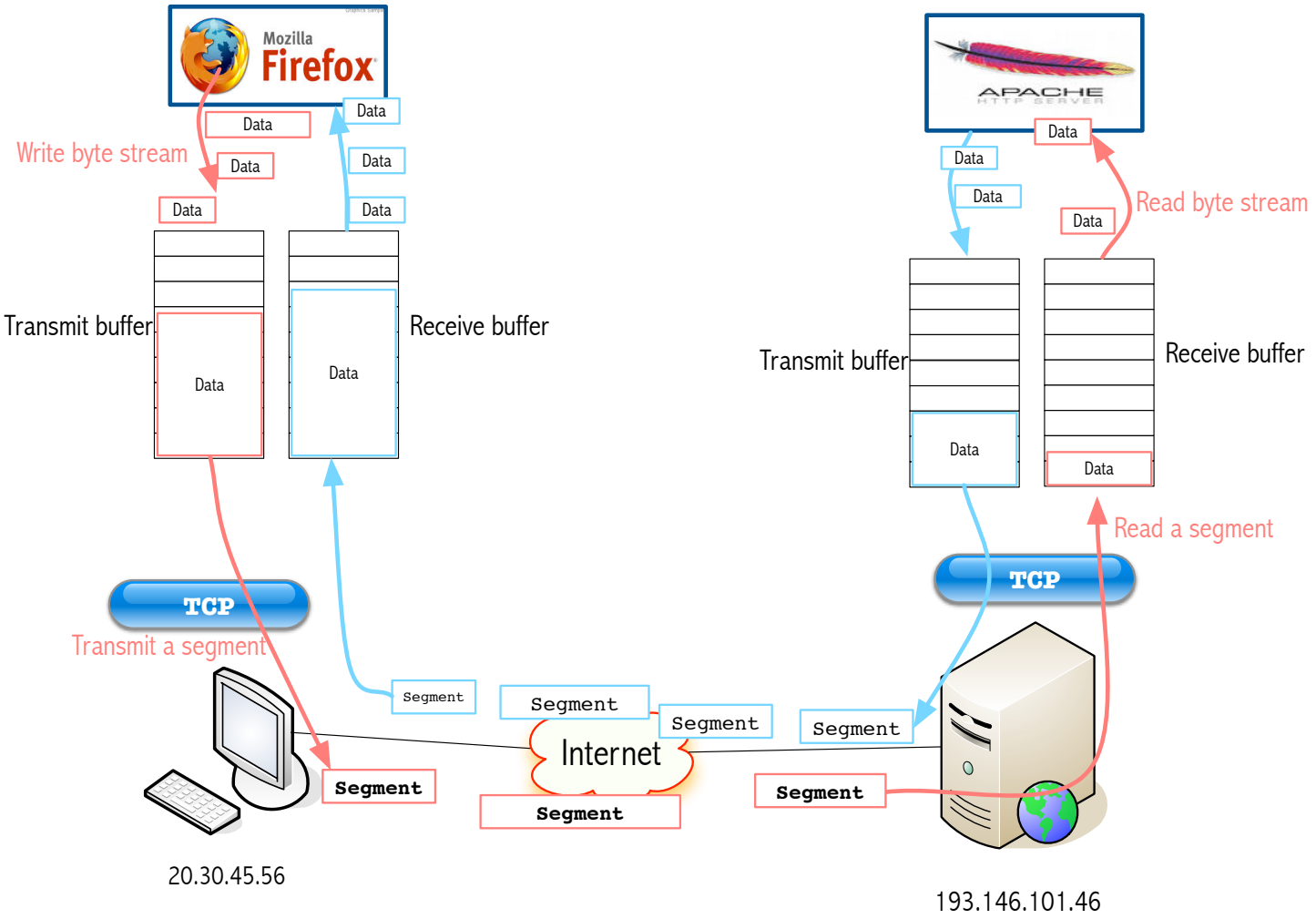
Efficient reliable transmission

Sliding Window Algorithm

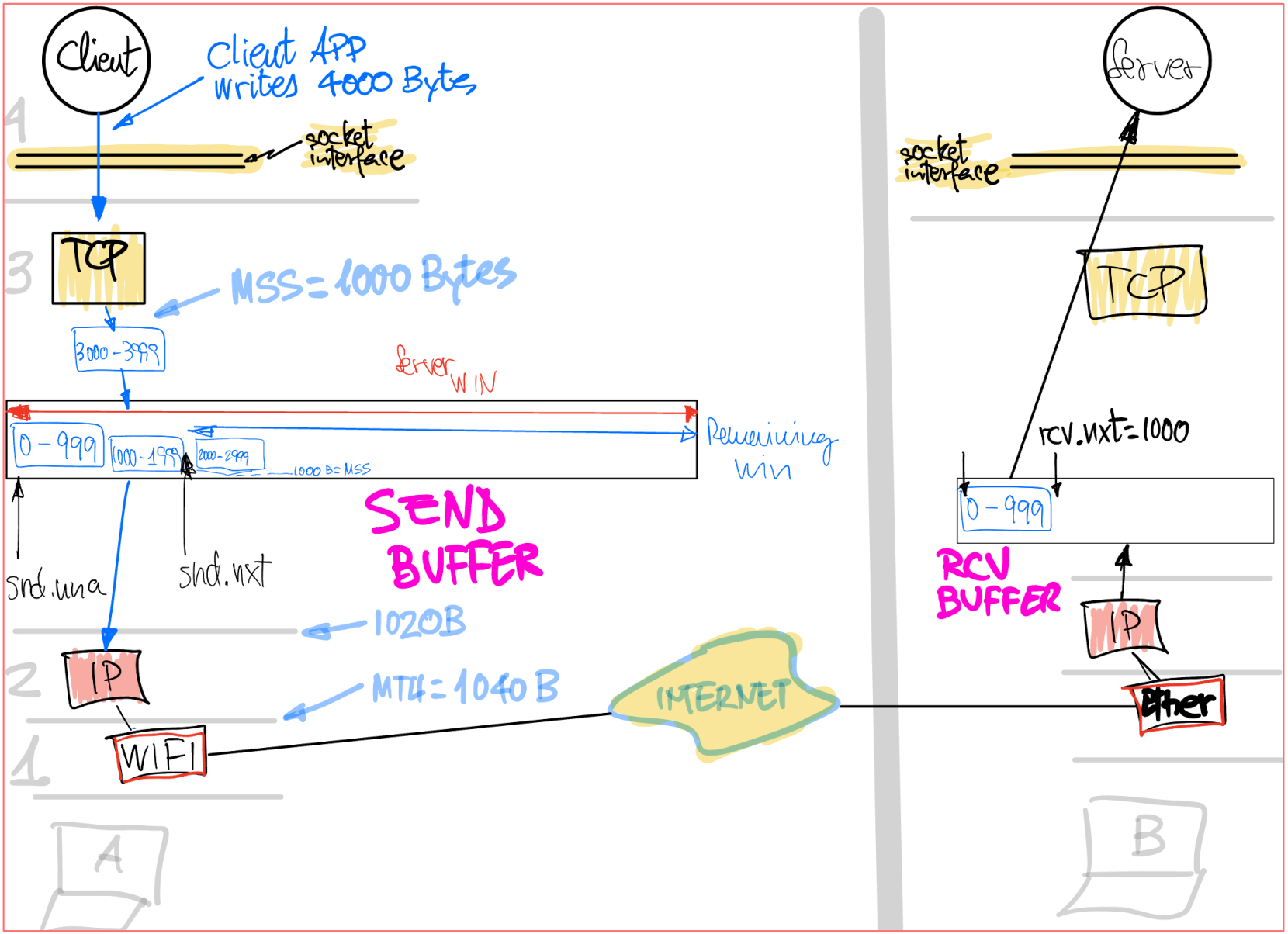
- This algorithm breaks down the would-be large data block submitted by an application into sub-blocks.
- Each sub-block becomes the payload to a new TCP segment
- Each segment is encapsulated into a new IP packet
- The payload encapsulated into a segment is a block of application data which initial byte is pointed to by the Sequence Number from the segment's header
- If a segment is successfully received at the destination TCP, then the receiver soon will send back a segment which ACK SN will represent that fact
  - Other wise, after a certain algorithmically calculated time elapses, the sender will resend the original segment (**ARQ**)
  - In-order delivery
  - Reliable delivery
  - Flow control



CONTEXT. A TCP connection Firefox (Client) Apache Web Server

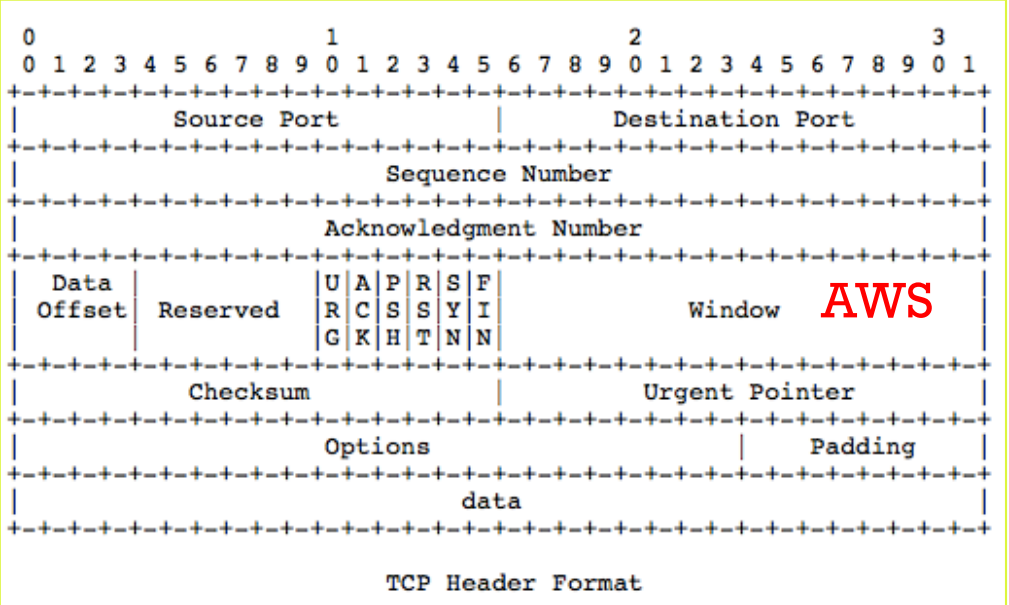


Example of *Sliding Window*



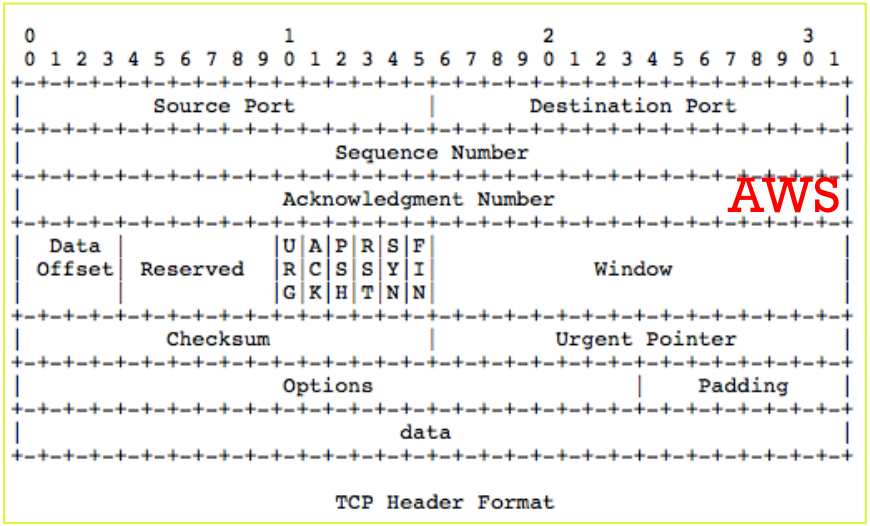
TCP Flow Control

- ~~Sending~~ Receiving side (S) informs ~~Receiving~~ Sending side (C) about its remaining buffer space
- In this situation, C will adapt its pace of transmission accordingly
  - The number of bytes sent within the current Rtt
- To this purpose, S uses a TCP header field known as *Advertised Window Size (AWS)* to inform C about its remaining buffer space
- AWS (Advertised Window Size) may reach 0
- In that situation C will send zero-data segments *from time to time* just for causing S to send an ACK containing an update of its AWS
  - Some implementations send ZeroWindowProbes periodically, every 5 sec
  - Others space ZeroWindowProbes exponentially

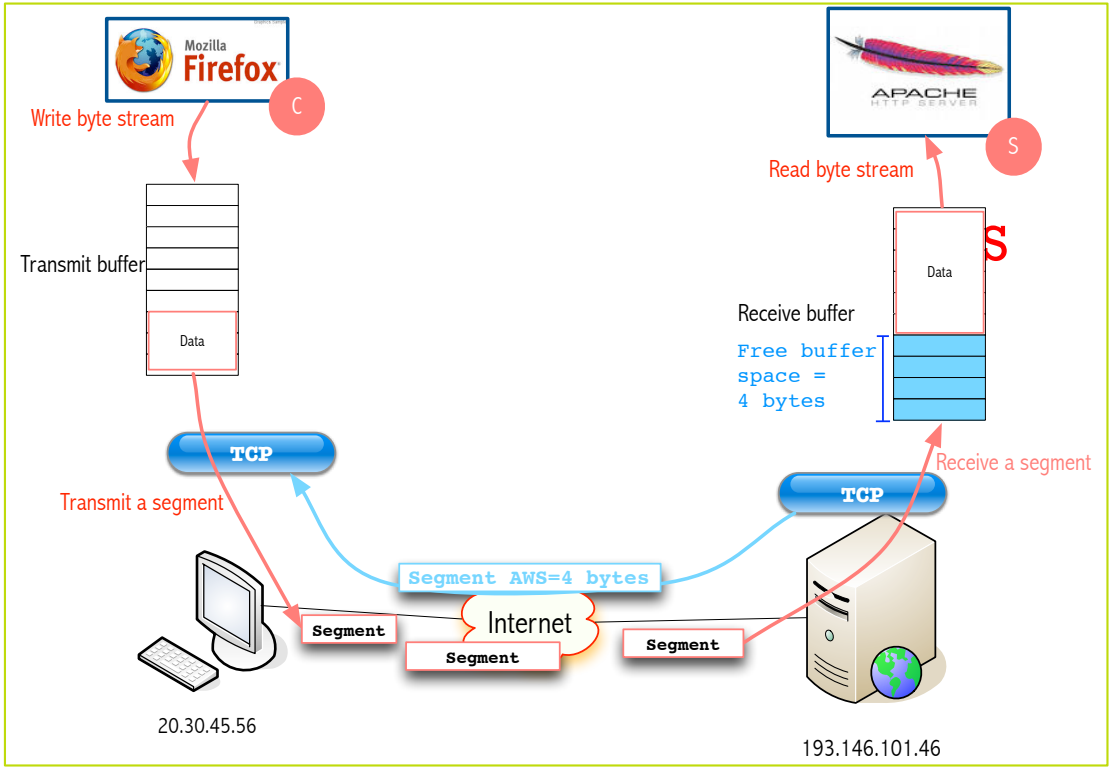


Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán
& José María Foces Vivancos

TCP Flow Control



- ~~Sending~~ Receiving side (S) informs ~~Receiving~~ Sending side (C) about its remaining buffer space
- In this situation, C will adapt its pace of transmission accordingly
  - The number of bytes sent within the current Rtt
- To this purpose, S uses a TCP header field known as *Advertised Window Size (AWS)* to inform C about its remaining buffer space
- AWS (Advertised Window Size) may reach 0
- In that situation C will send zero-data segments *from time to time* just for causing S to send an ACK containing an update of its AWS
  - Some implementations send ZeroWindowProbes periodically, every 5 sec
  - Others space ZeroWindowProbes exponentially

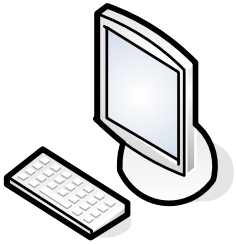


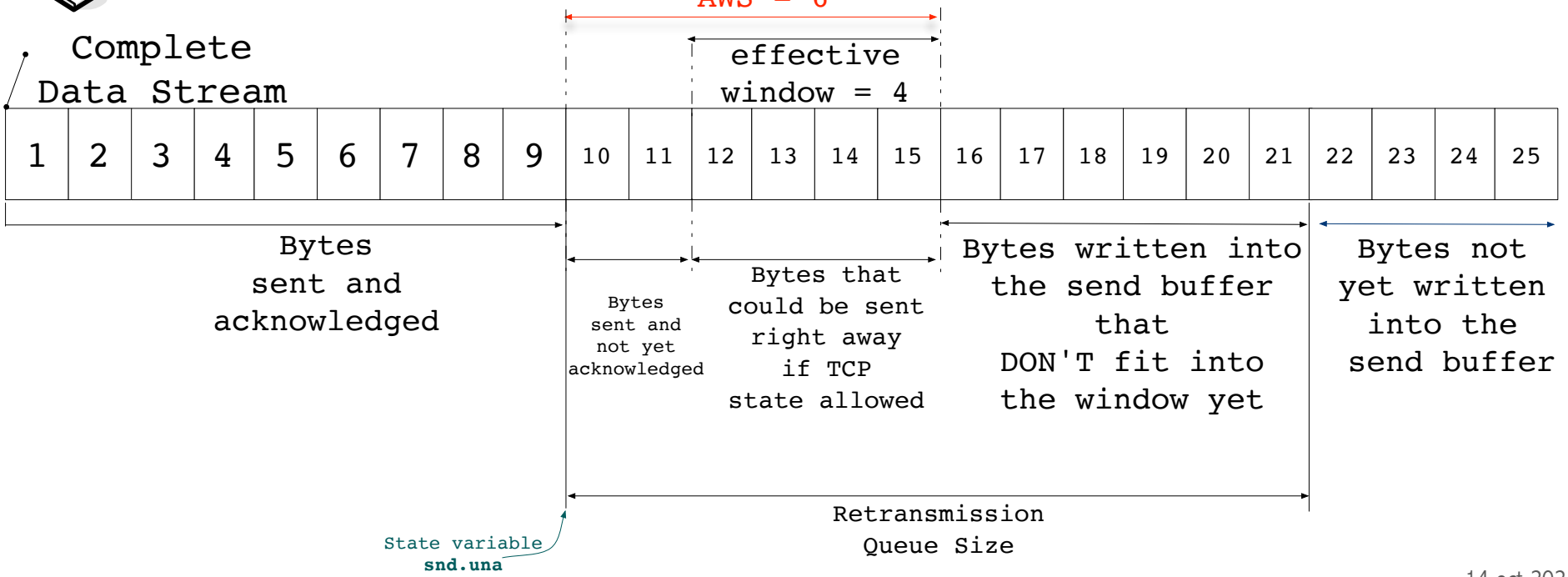
TCP sliding window

```
Send Sequence Space

                    1         2          3          4
               ----------|----------|----------|----------
                      SND.UNA    SND.NXT    SND.UNA
                                            +SND.WND

    1 - old sequence numbers which have been acknowledged
    2 - sequence numbers of unacknowledged data
    3 - sequence numbers allowed for new data transmission
    4 - future sequence numbers which are not yet allowed
```

© Send Sequence Space from RFC 793 (*A verbatim copy*).

transmitter



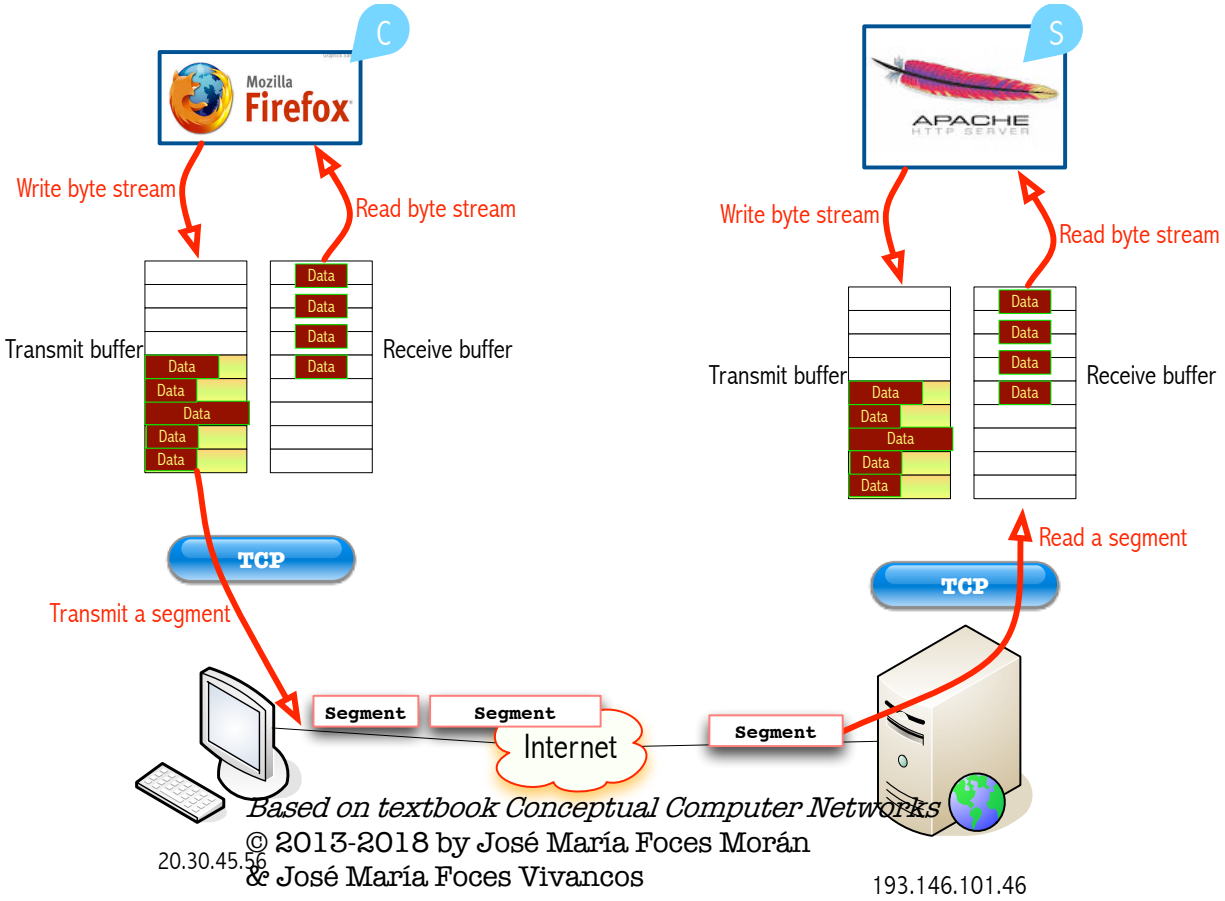


When does a TCP transmit?

Nagle's algorithm

Triggering Transmission

- How does TCP decide to transmit a segment?
  - TCP supports a byte stream abstraction
  - Application programs write bytes into streams
  - It is up to TCP to decide that it has enough bytes to send a segment



Triggering Transmission

Assuming that the other end's window is sufficiently large. TCP transmits the next segment available in the transmit buffer if any one of these conditions holds true. TCP will always attempt coalescing bytes from the transmission buffer into **full segments** (MSS)

a. The bytes in the send buffer are >= MSS even if no ACK pending

b. Push operation

c. An ACK that advances snd.una is received

   - The segment is transmitted even if the resulting segment length is < MSS

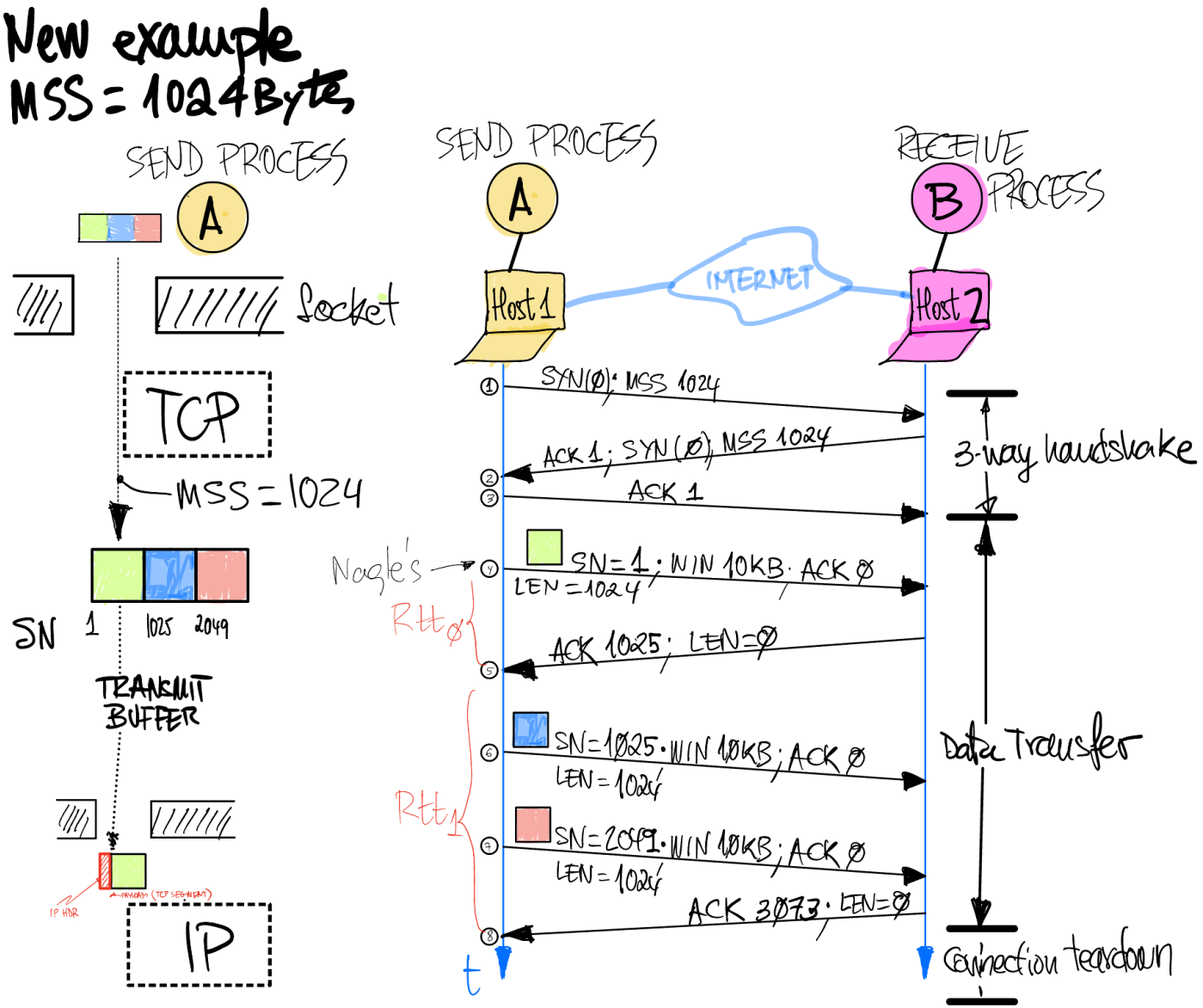
Silly Window Syndrome

- If TCP makes up **arbitrarily small segments**, the receiver will receive many of them and then it will be forced to provide an ACK to each, thereby coaxing the transmitter to transmit the next (snd.nxt) bunch of bytes of whatever length
- This **vicious circle** of **send small segment/ack** squanders network resources because the encapsulating segments/packets header lengths remain constant (at least 20 + 20 bytes) despite the payloads being ever tiny: Inefficiency
- Known as the Silly Window Syndrome
- Can be avoided by:
  - Nagle's algorithm (Send side)
  - Window control (Receive side)

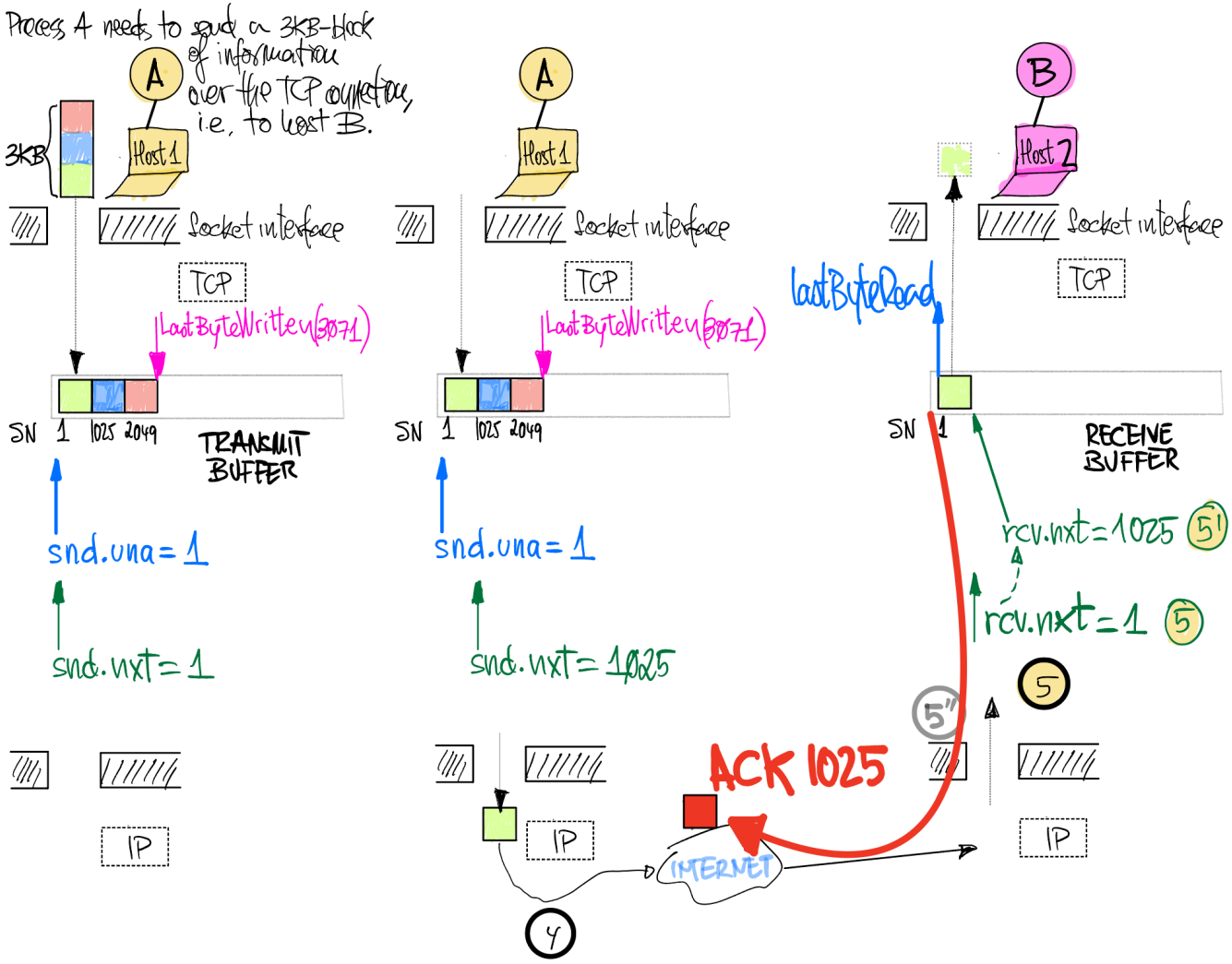
Nagle's Algorithm

- If no ACK is expected, then transmit any byte size available
- Transmit again only when an advancing ACK is received

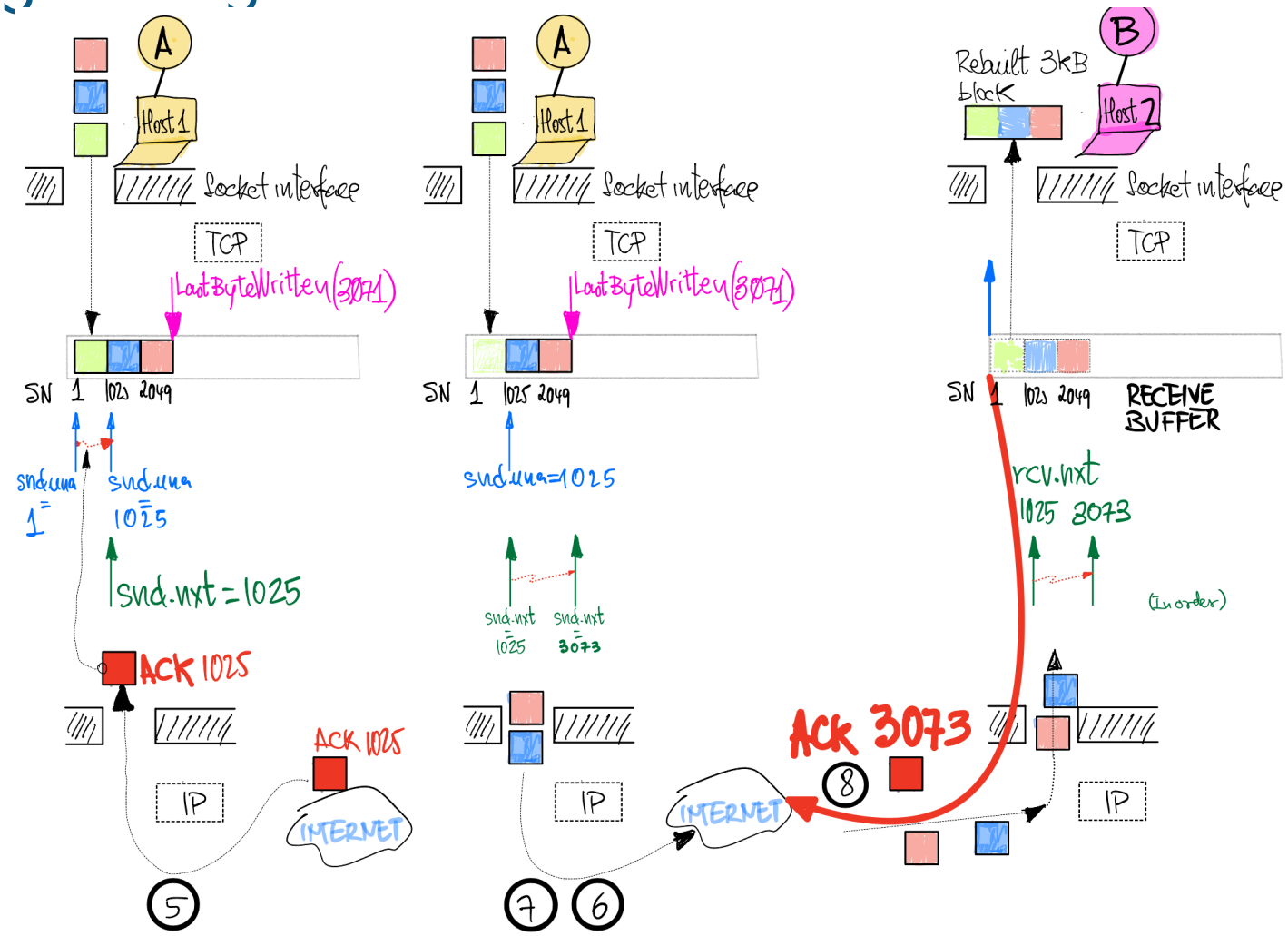
Nagle's Algorithm



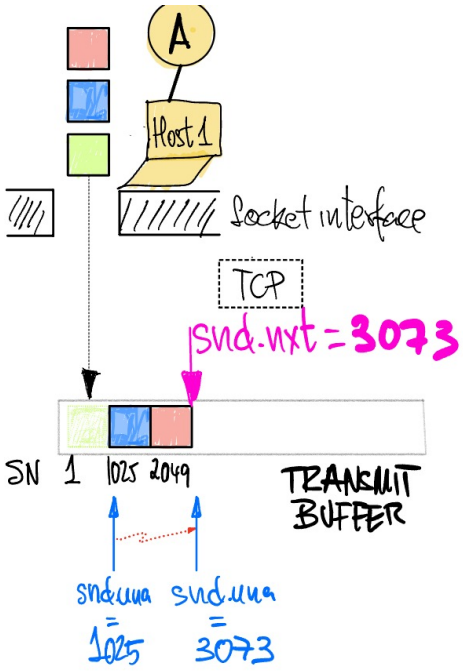
Nagle's Algorithm

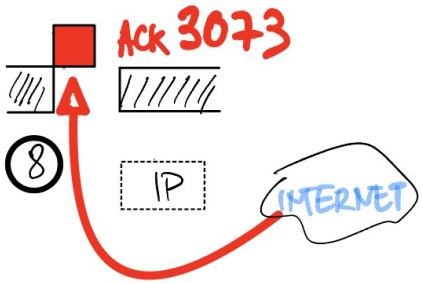


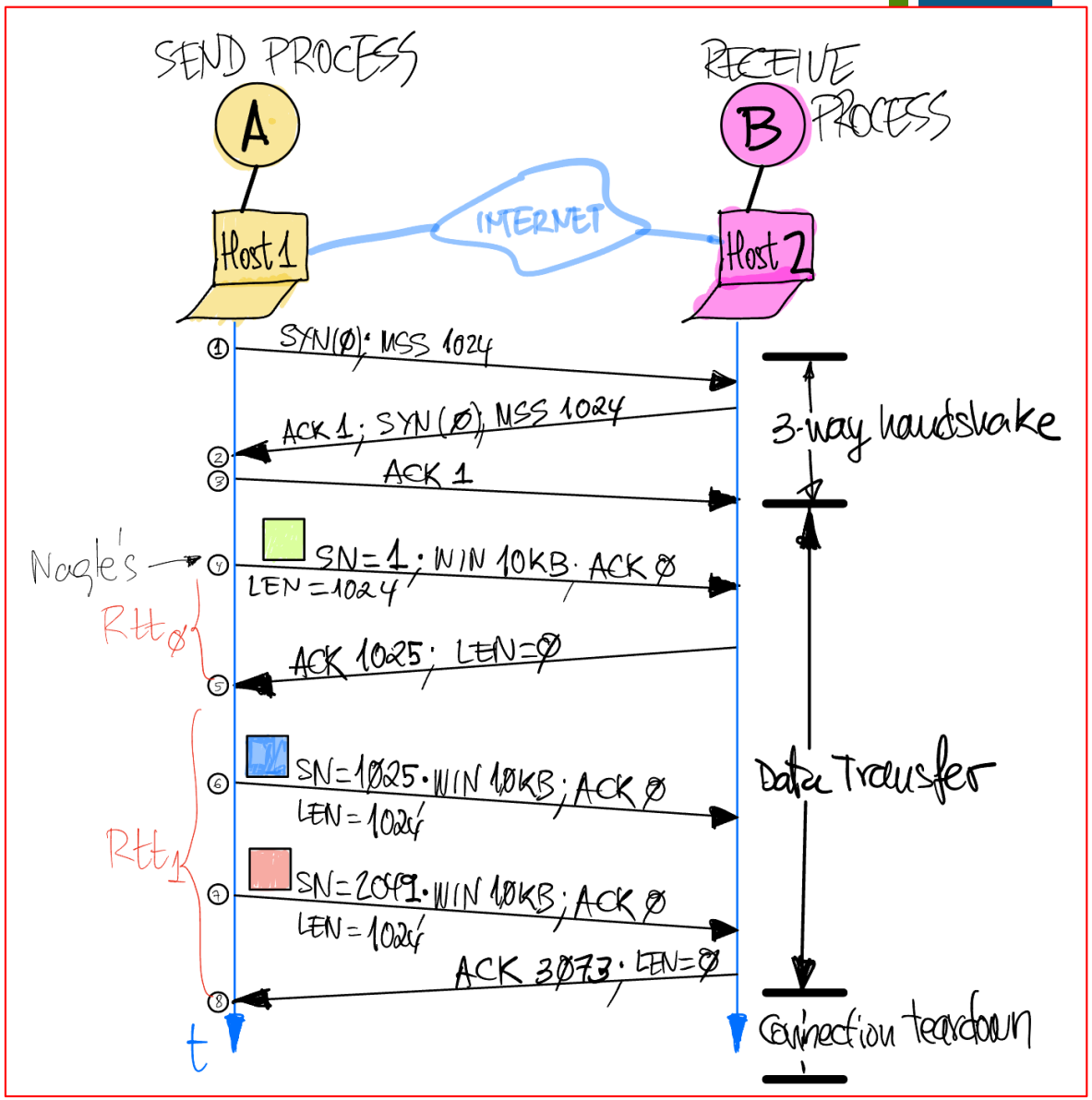
Nagle's Algorithm



Nagle's alg.





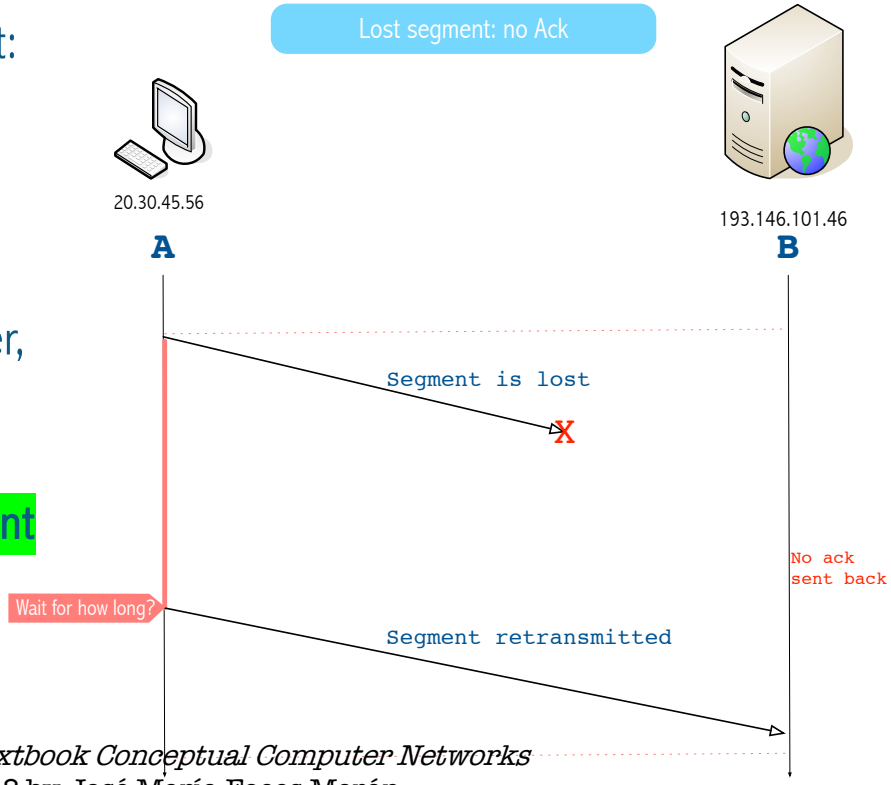


TCP retransmissions

Adaptive RTO estimation

If a segment is lost, it must be retransmitted

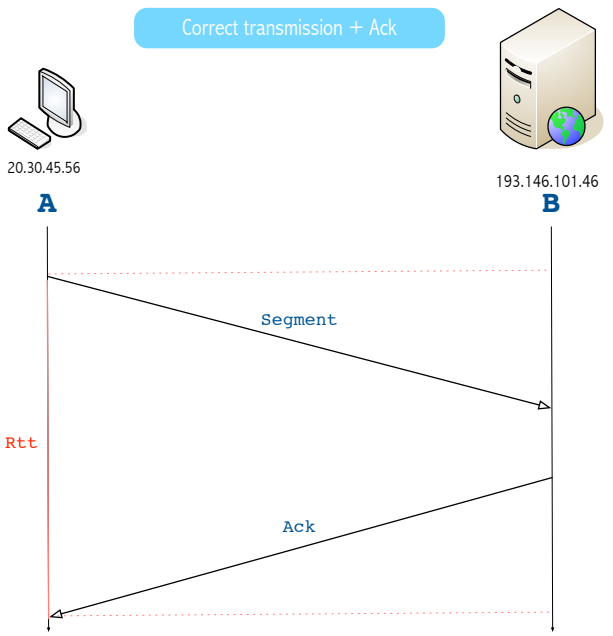
- When should the segment be retransmitted?
- TCP uses two complementary retransmission mechanisms
- 1. Schedule a timer when the first segment is to be sent: Retransmission TimeOut (**RTO**)
  - After RTO expires: Retransmit all segments from snd.una
- 2. If 3 ACKs are received for the same sequence number, the segment at snd.una was lost and must be retransmitted immediately: **3-DUP**
  - After 3-DUP: Fast Retransmit of only the lost segment
- RTO and 3-DUP are complementary mechanisms

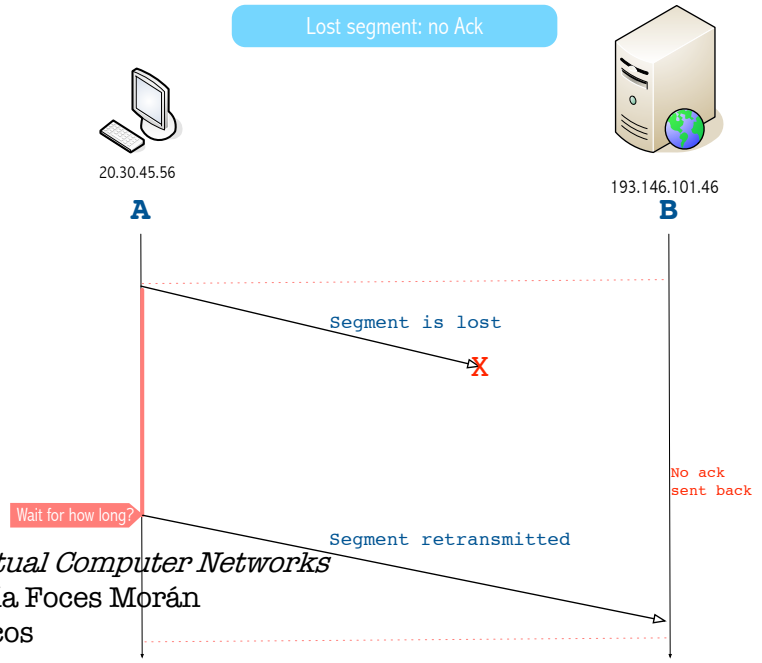


Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán
& José María Foces Vivancos

Retransmissions in TCP

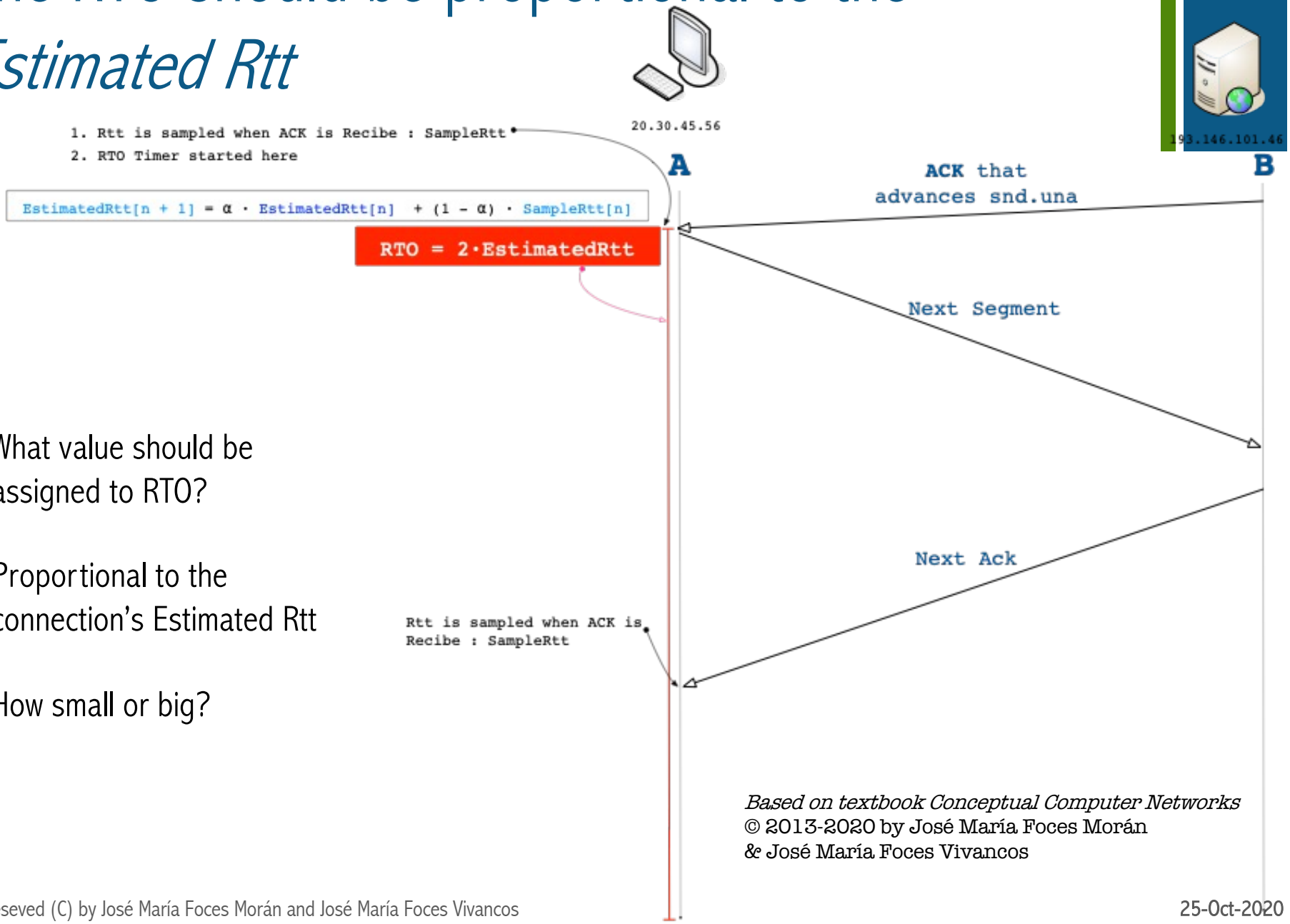
- If a segment is lost, it will have to be retransmitted
- If the ACK to a segment is lost, the original segment would have to be retransmitted, however, if more segments follow the ACK, their cumulative ACK will acknowledge the data acknowledged by the missing ACK thereby rendering the retransmission of the original data needless.
- Upon transmission of a segment, a *Retransmission Timer* is started with a countdown value of RTO sec
- RTO must be set so that the stability of Internet is honored and unnecessary transmissions are avoided as long as it is possible
  - What value should be assigned to RTO (Retransmission TimeOut)?





Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán
& José María Foces Vivancos

The RTO should be proportional to the *Estimated Rtt*

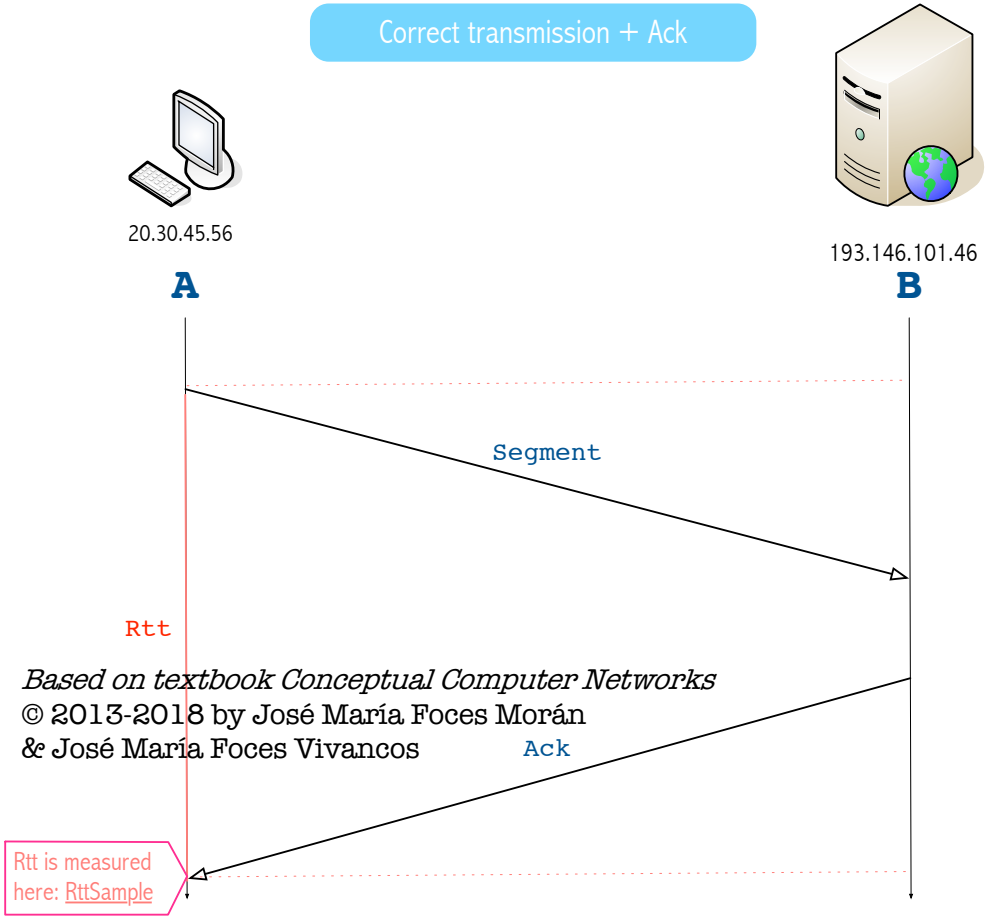


- What value should be assigned to RTO?
- Proportional to the connection's Estimated Rtt
- How small or big?

Based on textbook Conceptual Computer Networks
© 2013-2020 by José María Foces Morán & José María Foces Vivancos

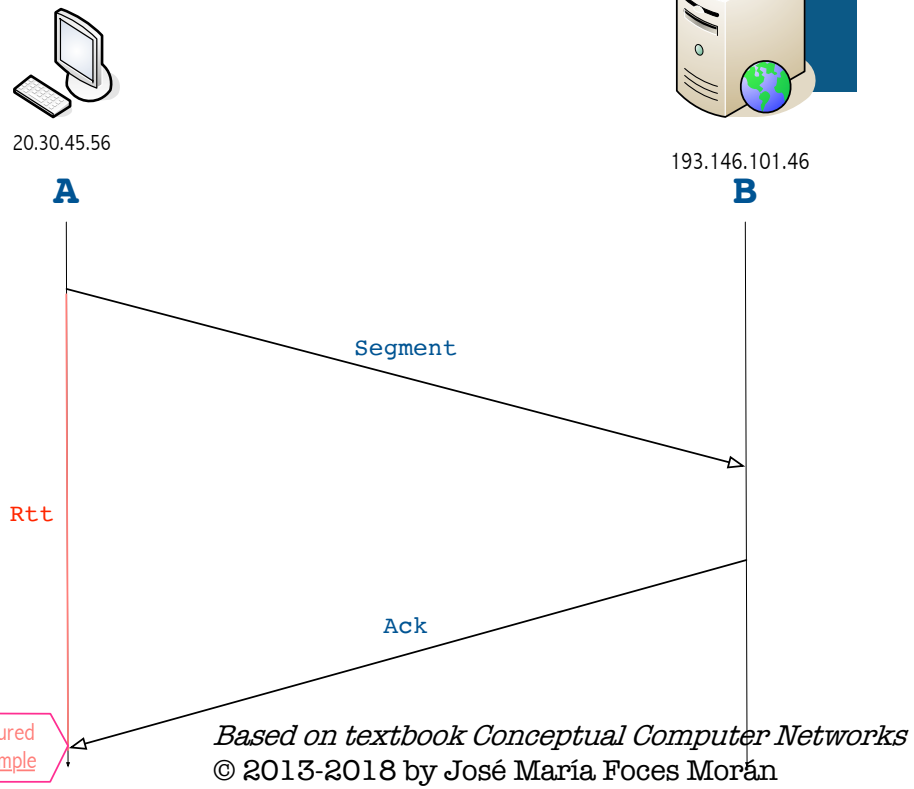
Then, how is EstimatedRtt computed?

- EstimatedRtt depends on:
  - Past values of EstimatedRtt (80-90%)
  - Last RttSample taken (20-10%)
- When an Ack is received, a new RttSample is taken
  - New EstimatedRtt is computed



Then, how is EstimatedRtt computed?

- The recursive formula assigns a higher weight to the past history of EstimatedRtt
  - Weighted Average
  - $\alpha = 0.8 - 0.9$
  - $1 - \alpha = 0.2 - 0.1$
- This function behaves like a Low Pass Digital Filter
  - Will somewhat suppress the highest samples of Rtt (RttSample)



Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán & José María Foces Vivancos

$$EstimatedRtt[n+1] = \alpha \cdot EstimatedRtt[n] + (1 - \alpha) \cdot SampleRtt[n]$$

Current value of EstimatedRtt

New value of EstimatedRtt[n + 1]

$RTO = 2 \cdot EstimatedRtt[n+1]$

Computing EstimatedRtt: an example

- Assume a TCP connection in a specific time point has a value of EstimatedRtt of 150ms and that the next three samples of Rtt (SampleRTT in ms) are: 130, 180 and 39,2 ms. What's the value of SRTT (The next value of EstimatedRtt, also known as Smoothed RTT)? Assume parameter $\alpha = 0{,}9$ and that the original TCP adaptive retransmission algorithm holds.

  - a. 150 ms
  - b. 180 ms
  - c. 140 ms
  - d. 135 ms
  - e. A value other than those above

Computing EstimatedRtt: an example

- SampleRTT in ms = {130, 180, 39.2}
- $\alpha = 0{,}9$
- Initial EstimatedRtt = 150 ms

$$\text{EstimatedRtt}[n+1] = \alpha \cdot \text{EstimatedRtt}[n] + (1 - \alpha) \cdot \text{SampleRtt}[n]$$

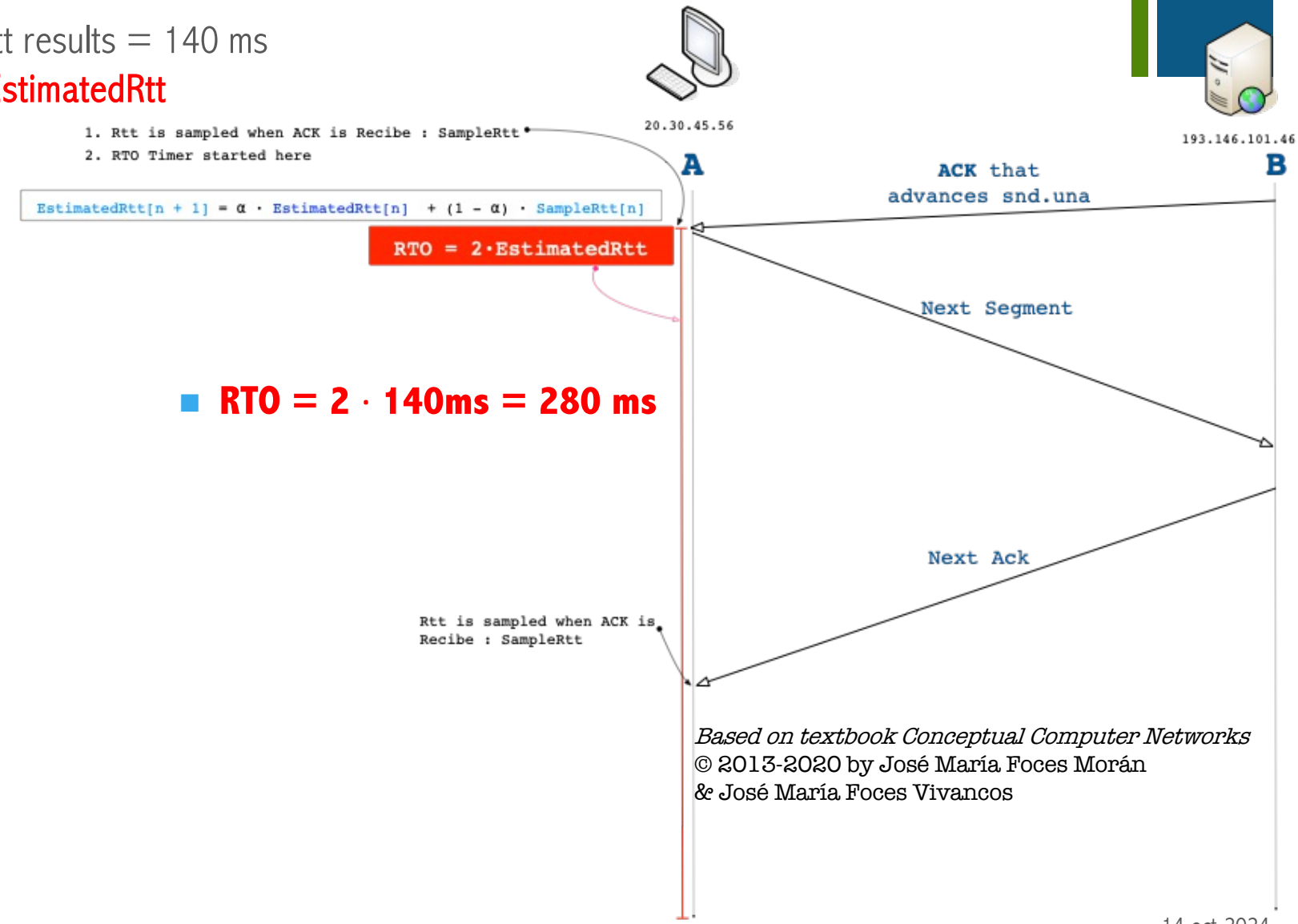
  - $\text{EstimatedRtt}[1] = 0.9 \cdot 150 + 0.1 \cdot 130 = 148$ ms
  - $\text{EstimatedRtt}[2] = 0.9 \cdot 148 + 0.1 \cdot 180 = 151{,}2$ ms
  - $\text{EstimatedRtt}[3] = 0.9 \cdot 151{,}2 + 0.1 \cdot 39.2 \approx 140$ ms

- **Tick answer c. (140 ms)**

Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán
& José María Foces Vivancos

Computing RTO = 2 · EstimatedRtt: an example

- Estimated Rtt results = 140 ms
  RTO = 2 · EstimatedRtt

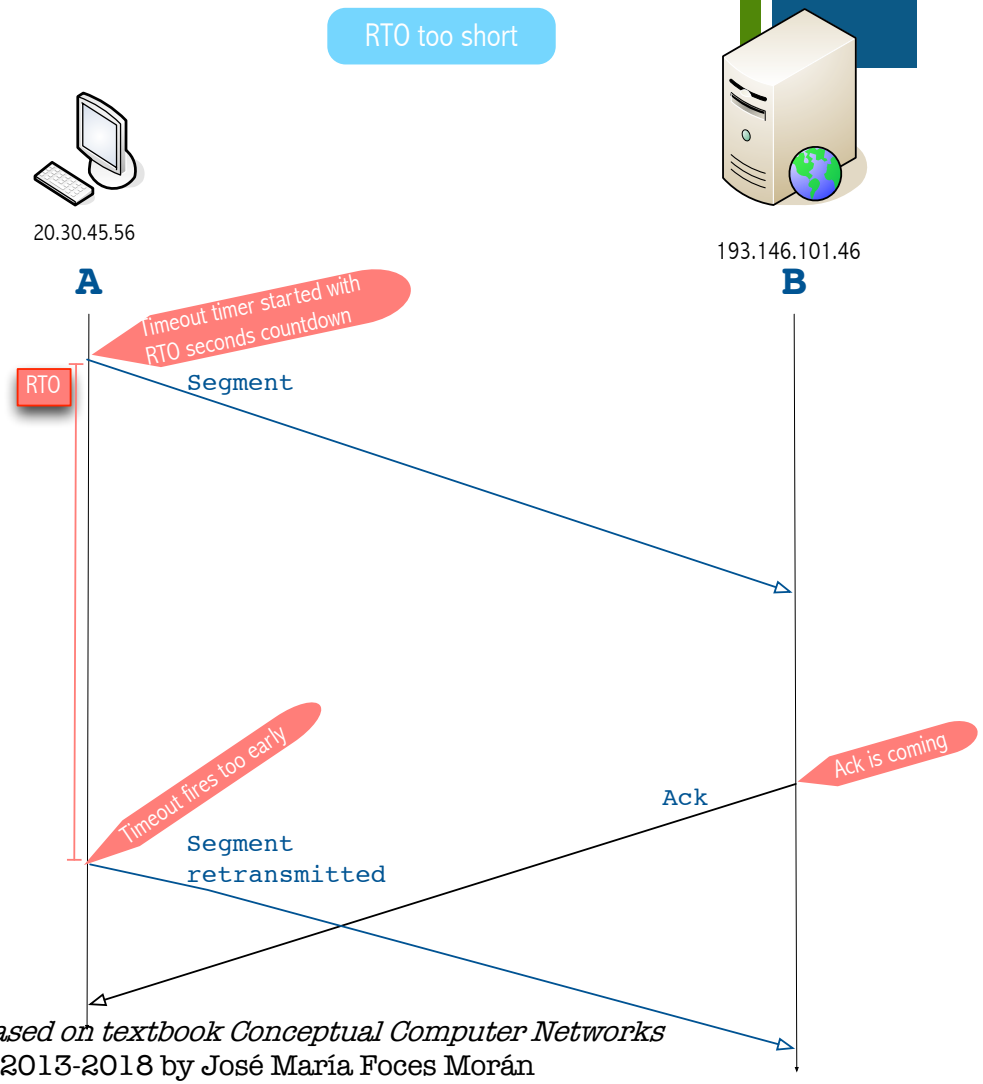


- **RTO = 2 · 140ms = 280 ms**

Based on textbook Conceptual Computer Networks
© 2013-2020 by José María Foces Morán & José María Foces Vivancos

What happens if RTO is not properly estimated?

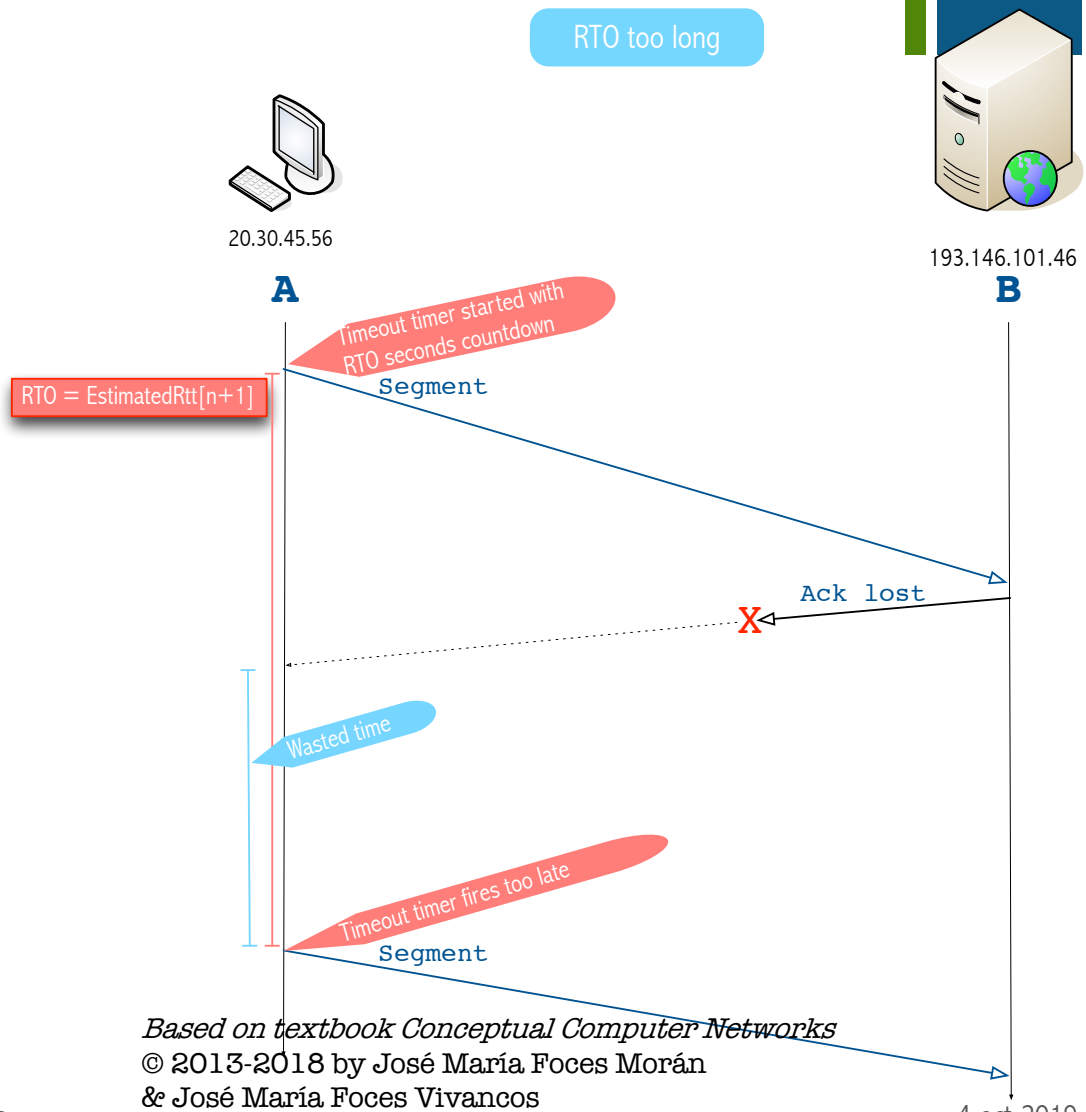
- RTO is too short
  - Timer will fire too soon
  - Transmitter will retransmit a segment needlessly
- If RTO had been long enough, retransmitting the segment would have not been necessary



Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán & José María Foces Vivancos

What happens if RTO is not properly estimated?

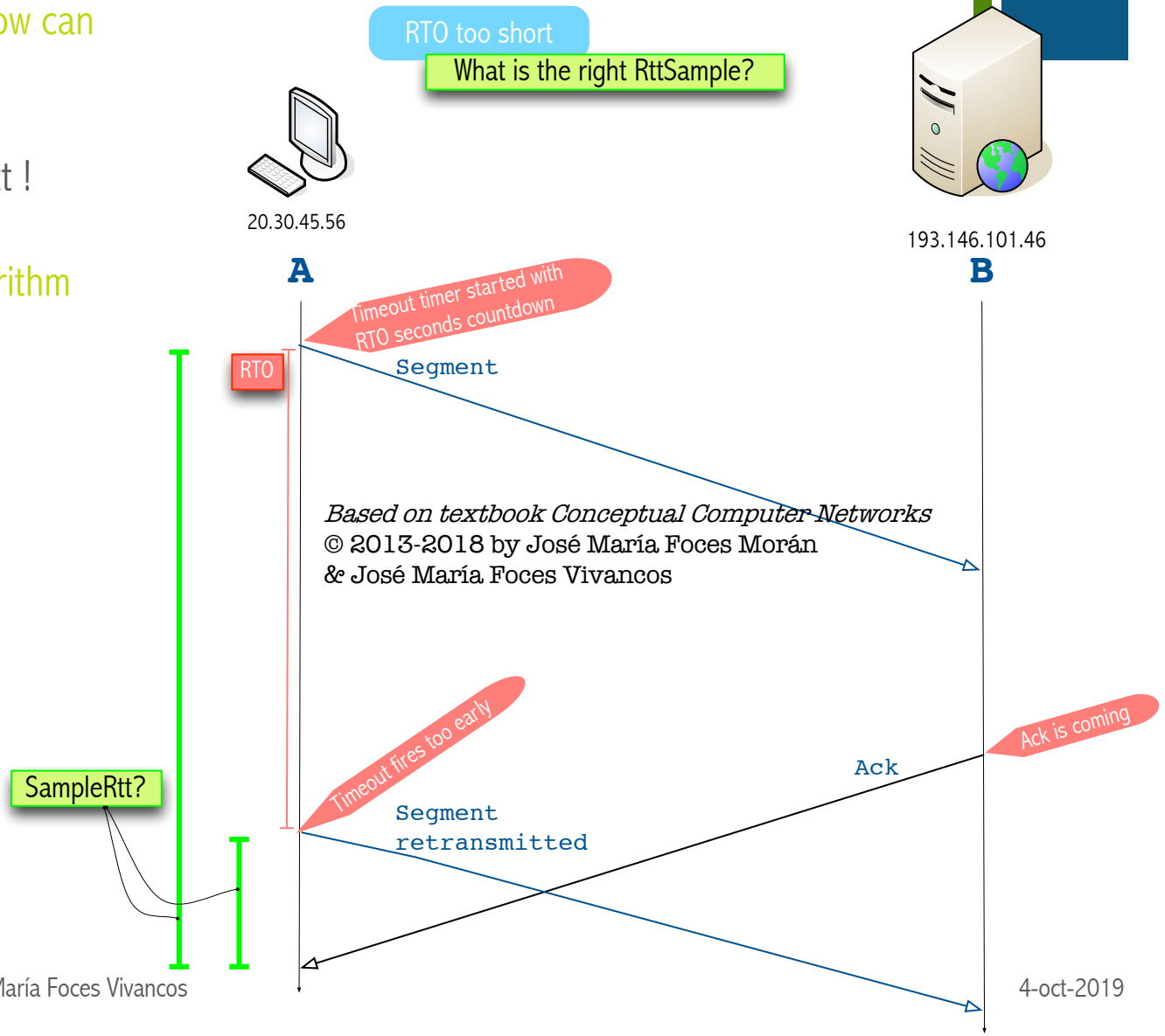
- RTO is too long
  - Timer will fire too late
  - Transmitter will retransmit a segment after an execessively long time
  - Receiver will receive the segment too late
  - Performance will suffer
- If RTO had been shorter, the receiver would have not wasted so much time waiting for the missing segment



Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán & José María Foces Vivancos

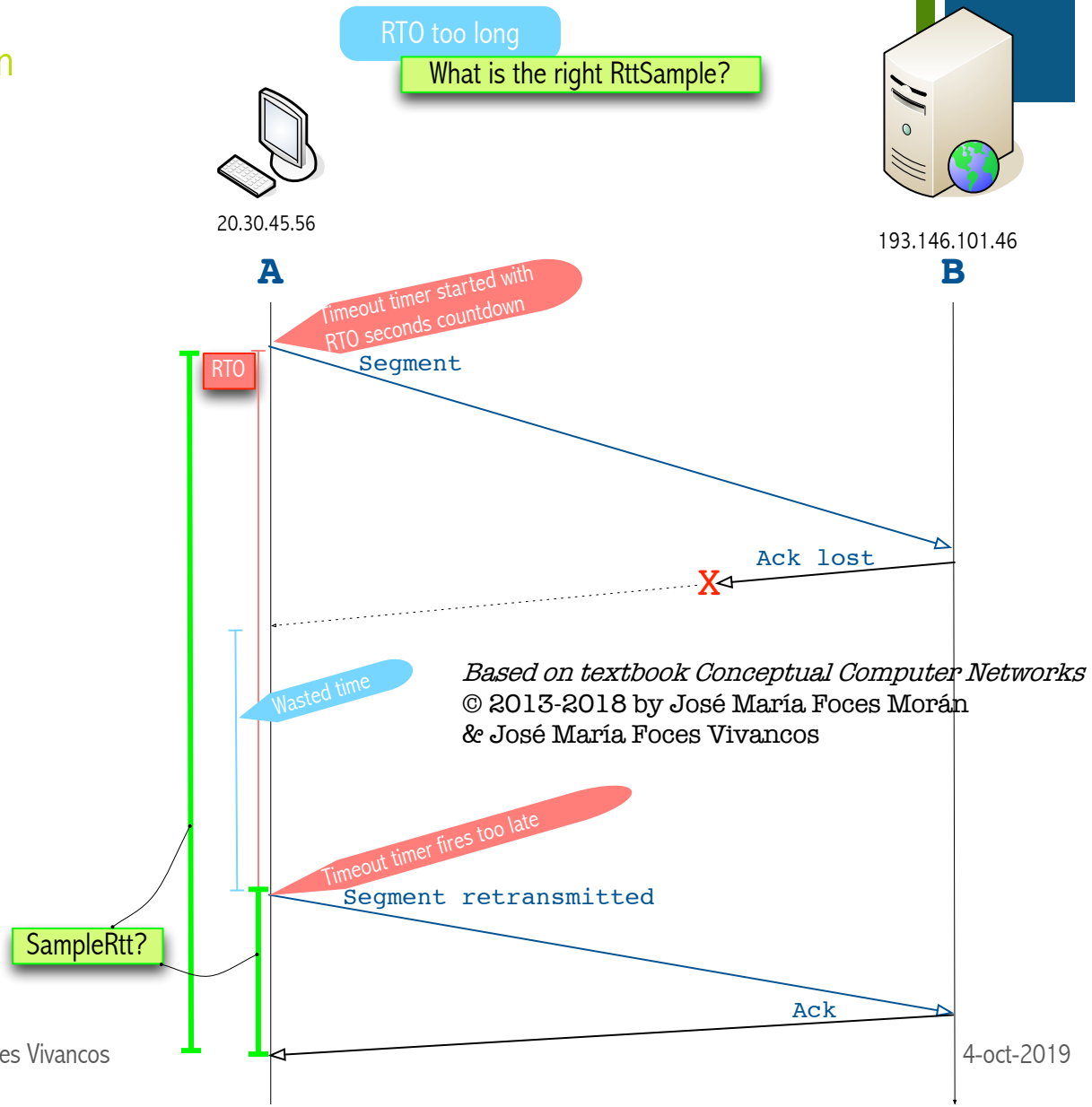
What's the Rtt when retransmissions occur?

- PROBLEM: In this situation, how can EstimatedRtt be calculated?
- Recall: RTO = 2 · EstimatedRtt !
- Solution: Karn/Partridge algorithm



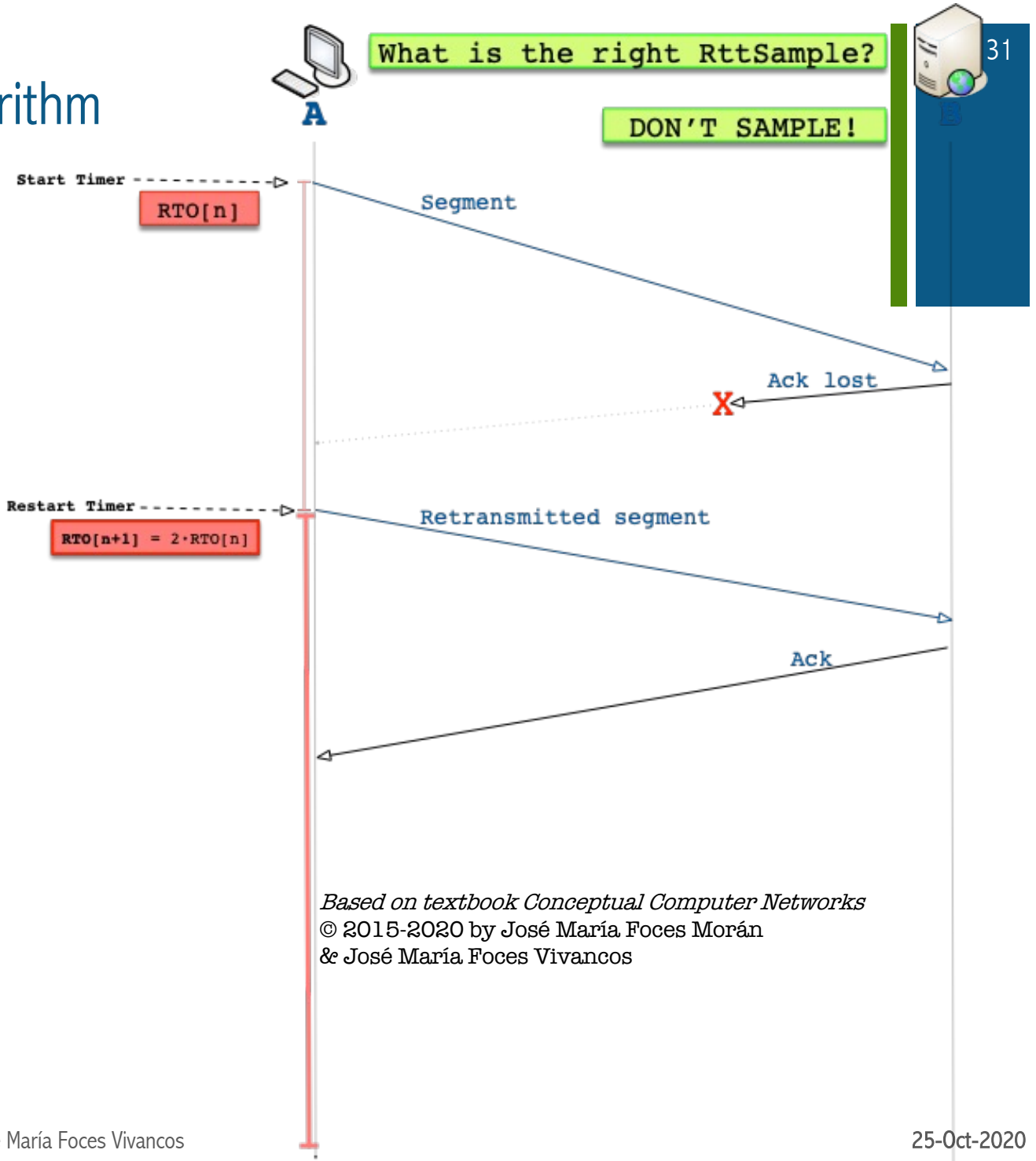
What's the Rtt when retransmissions occur?

- PROBLEM: In this situation, how can EstimatedRtt be calculated?
- Recall: RTO = 2 · EstimatedRtt !
- Solution: Karn/Partridge algorithm



Karn/Partridge Algorithm

- Do not sample RTT when retransmitting
- Simply, double timeout (TO) after each retransmission:
- RTO[n+1] = 2 · RTO[n]



Based on textbook Conceptual Computer Networks
© 2015-2020 by José María Foces Morán
& José María Foces Vivancos

Karn/Partridge Algorithm

- An ACK represents the correct receipt of a past segment
  - It does not mean "*this transmission was correct*"

Karn/Partridge Algorithm: complications

- KP does not consider the variance of SampleRtt
- If variance is low, then EstimatedRtt can be better trusted
  - And RTO can be equated to EstimatedRtt, up front
  - RTO ≈ EstimatedRtt
- However, if variance is high:
  - RTO != f(EstimatedRtt)

Jacobson/Karels Algorithm: Account for variance in SampleRtt

- SampleRTT is same as before
- Difference = SampleRTT − EstimatedRTT
- EstimatedRTT = EstimatedRTT + (× Difference)
- Deviation = Deviation + (|Difference| − Deviation)
- TimeOut = μ × EstimatedRTT + × Deviation
  - where based on experience, μ is typically set to 1 and is set to 4. Thus, when the variance is small, TimeOut is close to EstimatedRTT; a large variance causes the deviation term to dominate the calculation.

Protecting against Wraparound

- SequenceNum: 32 bits long
- AdvertisedWindow: 16 bits long
  - TCP has satisfied the requirement of the sliding window algorithm that the sequence number space be twice as big as the window size
  - $2^{32} >> 2 \times 2^{16}$

Protecting against Wraparound

- 32-bit sequence number space
  - The sequence number used on a given connection might wraparound
    - A byte with sequence number x could be sent at one time
    - Later, a second byte with the same sequence number x could be sent
- Packets cannot survive in the Internet for longer than the MSL (Max. Segment Lifetime)
  - MSL is set to 120 sec
  - We need to make sure that the sequence number does not wrap around within a 120-second period of time
  - Depends on how fast data can be transmitted over the Internet

Protecting against Wraparound

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

Keeping the Pipe Full

- 16-bit AdvertisedWindow field must be big enough to allow the sender to keep the pipe full
- Clearly the receiver is free not to open the window as large as the AdvertisedWindow field allows
- If the receiver has enough buffer space
  - The window needs to be opened far enough to allow a full delay × bandwidth product's worth of data
  - Assuming an RTT of 100 ms

Keeping the Pipe Full

Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

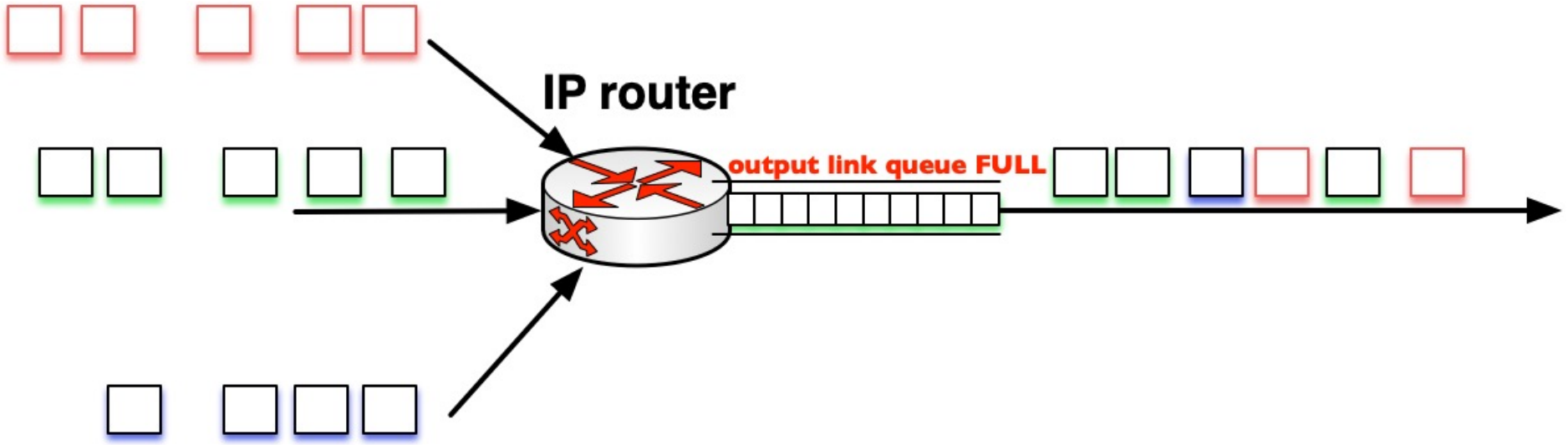
Required window size for 100-ms RTT

The basics of Internet congestion

When excessive network delay compromises service

Basic structure of an IP Router

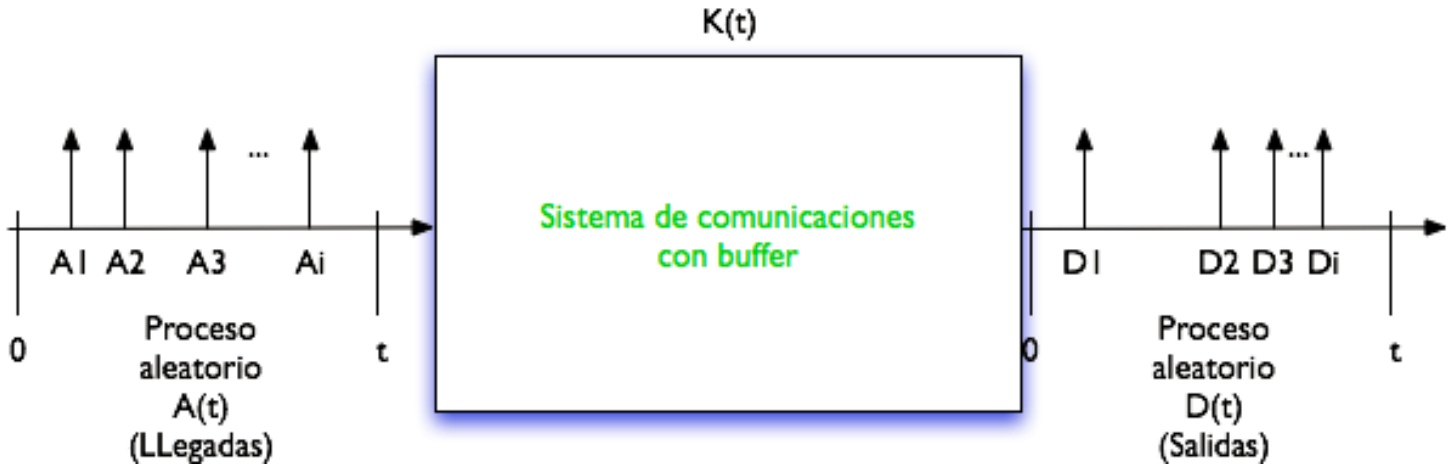
- At this moment, *the output link,* receives traffic from three *input links*
- The output link, when demand is high, queues packets in a buffer
  - Increases the delay undergone by each packet
  - In the limit, when the link is congested, it begins to drop packets (Packets get lost)



Queue length: Little's Law

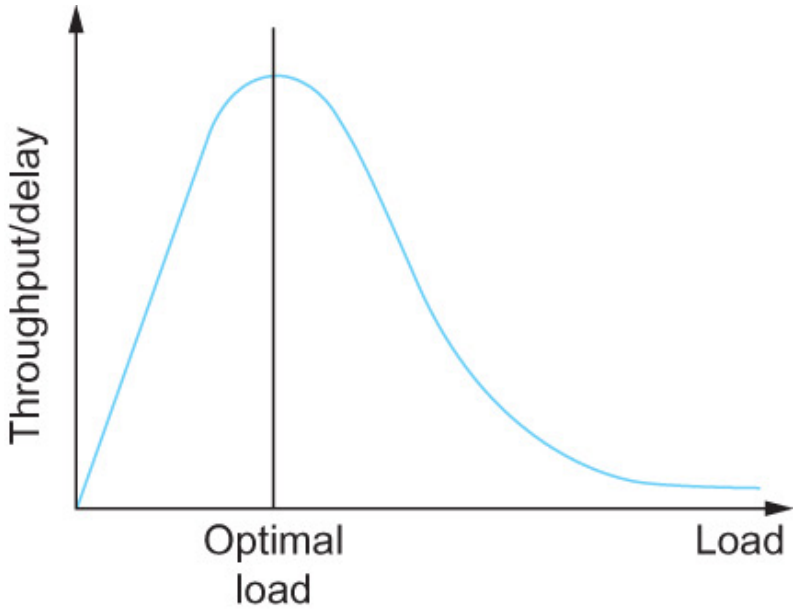
If t is sufficiently large:

- $N = \lambda \cdot T_a$
  - The *average* queue length is given by the product of average packet rate and the average residence time
  - **The interarrival time is given a by a Poisson probability distribution A(t) = P(interravival time <= t)**
- Applies to a variety of queue disciplines, not only FIFO
- Input probability distribution



Power curve of a netowrk

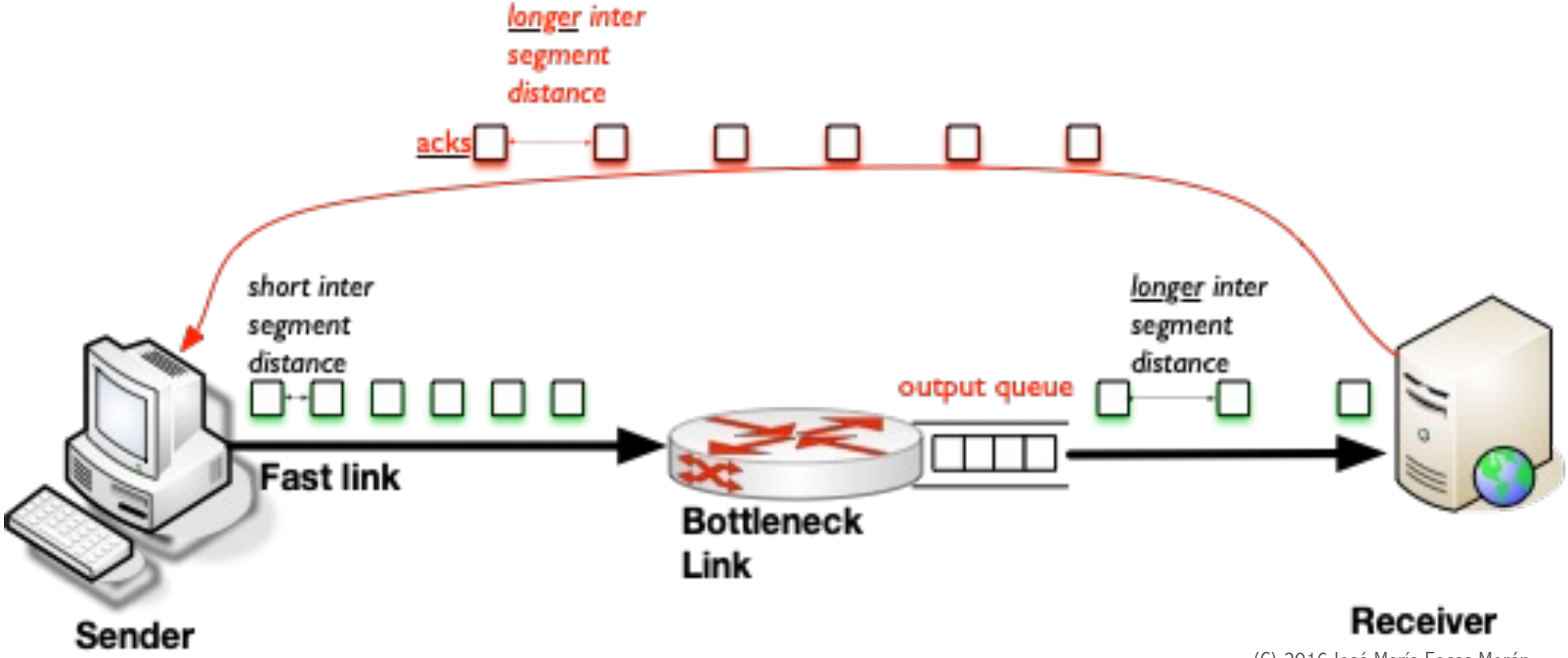
- As the offered load increases, the ratio Throughput/delay also increases
  - Offered load
  - Achieved throughput/delay (T/d)
- A time point comes when the T/d flats and the begins to decrease as offered load keeps increasing
  - This is due to the increasing delay at each router



© Morgan-Kaufmann 2012, Prof. L. Peterson and Bruce Davie

Bottleneck link at an IP router

- The bottleneck link limits the maximum number of segments present in the network
- Product 2BD: B is the bandwidth of the bottleneck link and 2D is the Rtt

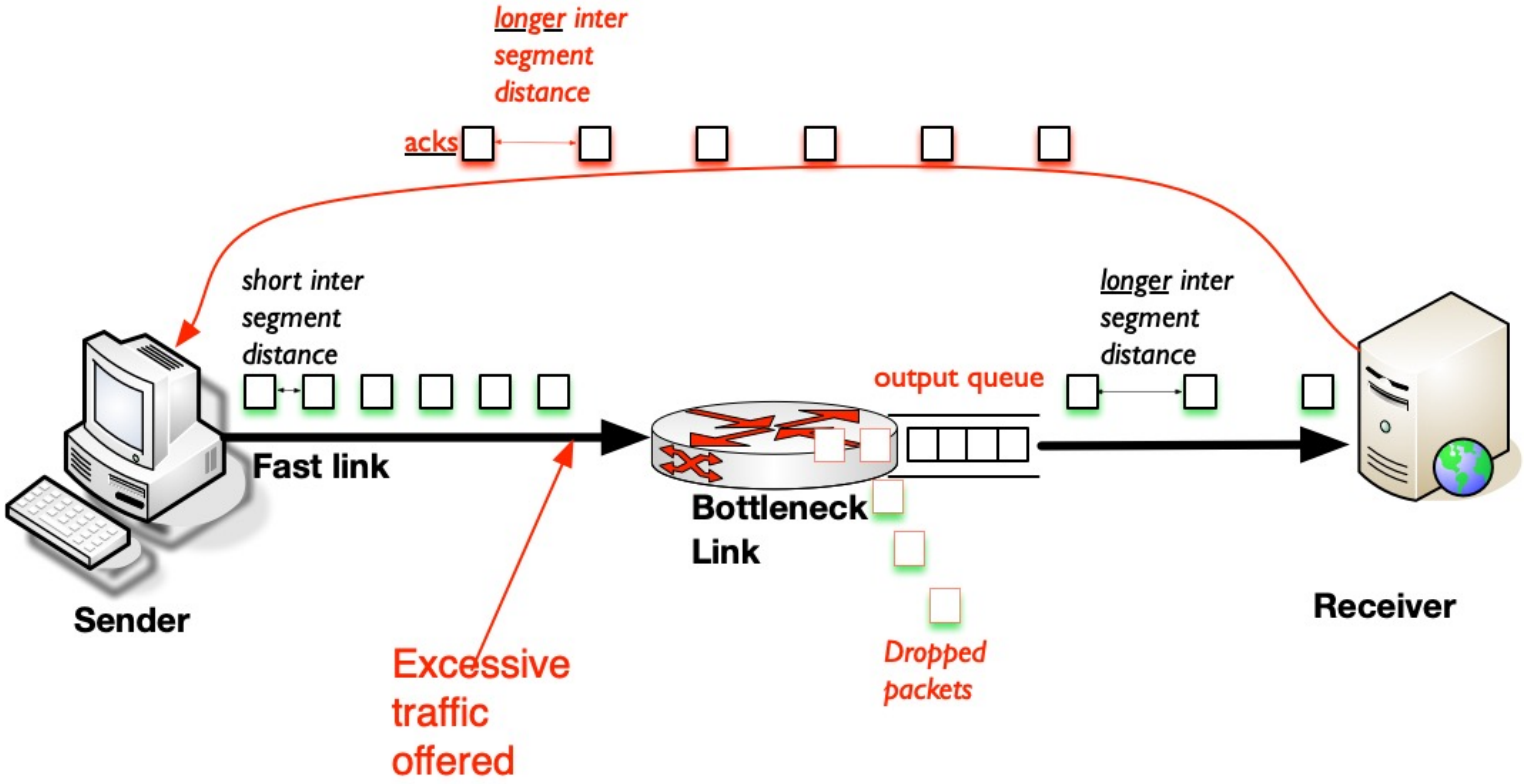


TCP, congestion control

14-oct-2024

How TCP discovers the end-to-end capacity of a TCP connection

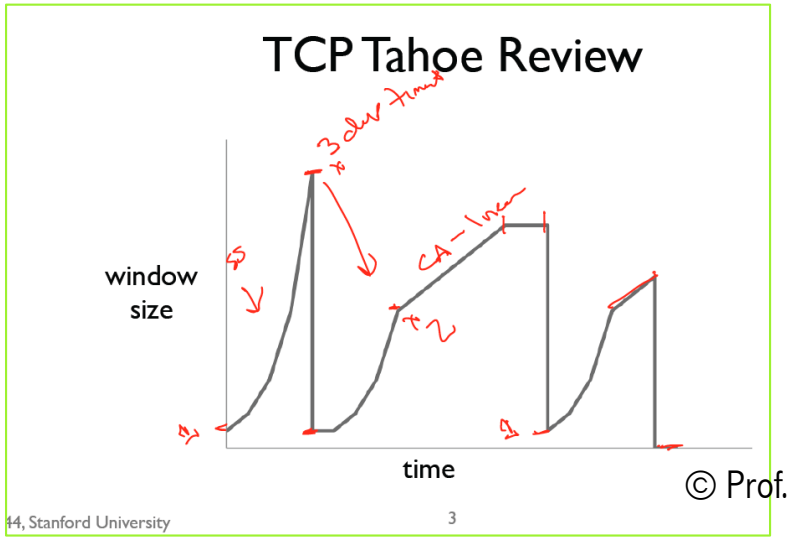
- TCP needs to discover how many packets/sec can be injected into the network, *safely*
- With a limited packet loss

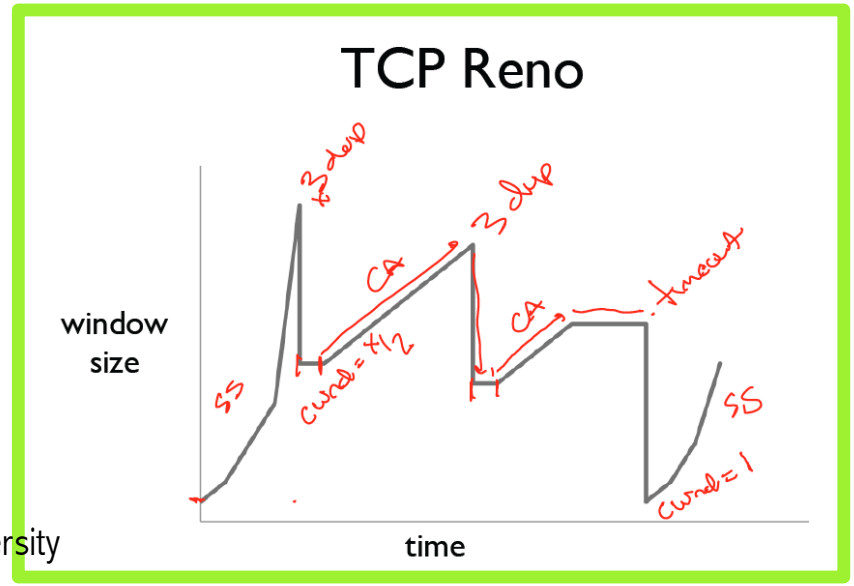


AIMD: Additive Increase, Multiplicative Decrease

14-oct-2024

- TCP needs to discover how many packets/sec can be injected into the network, safely
- Without packet loss
- The effective TCP's transmit window becomes =MIN (CongestionWindow, AdvWindow)
- CW = CongestionWindow

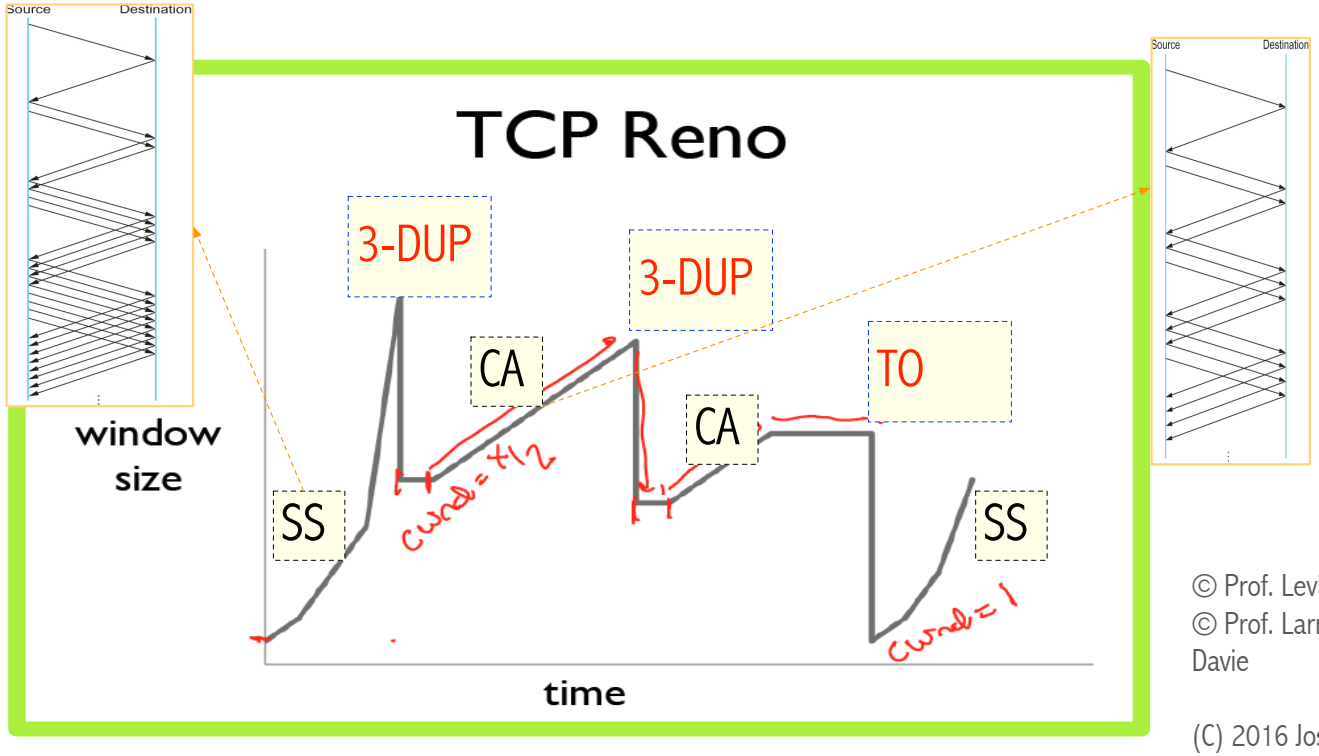




© Prof. Levis, Stanford University

How discovers network capacity

- Slow Start (SS)
  - Probe for network capacity by growing CW (Congestion window)
    CW = 2 * CW each Rtt
  - Initially, CW = 1
- 3-DUP causes transition to CA (Congestion Avoidance) with CW = SSthrsh / 2
- TO (Timeout) causes SS to start again
  - Linux implements TCP Reno and CUBIC congetion control



Reno, Fast Retransmit and Fast Recovery

- Fast Retransmit:
  - Upon a 3-DUP the transmitter will retransmit the missing segment, only
- Fast Recovery
  - Also, artificially increase CW = CW + 3 to compensate for the 3-DUP that didn't advance LastByteAcked and which, therefore, could not be used to spur the transmitter to transmit 3 new segments
  - Use the remaining, upcoming ACKS to keep the transmission pace
  - NO Slow Start in Reno upon 3-DUP