# Foundations: Networking

## Software Performance Engineering: Theory & Practice

# Outline

- Bandwidth & Latency
- Packet Switching
- IP routing

- TCP
  - Sessions
  - Congestion control

# Computer networking

- Two or more computers exchanging messages over some transmission medium
  - structured messages governed by standard industry protocols
- LANs and WANs
  - Local Area Networks
    - Inexpensive and lightweight medium for transporting messages over short distances
    - e.g.,
      Ethernet 10baseT, 100baseT, Gigabit Ethernet, 10 Gigabit Ethernet, etc.
  - Wide Area Networks
    - Expensive, long distance telecommunication lines
    - e.g.,
      Frame Relay, ISDN, T1, T3, SONET

# Computer networking

- Protocols are generally packet-oriented and designed for serial interfaces.
- Performance metrics:

  - Latency
    - Round Trip Time (RTT) = 2 * latency
    - Signal propagation through a wire ≈ speed of light * 1/2
    - Signal propagation through fiber ≈ speed of light * 2/3

  - Bandwidth
    - Bits per second - Overhead

# Computer networking

- **Latency**
  - Signal propagation delay is a function of distance
  - ROT: 5 ms per 1000 km
  - Plus delay at each hop

$$Latency \cong {Distance}/{Signal\ propagtion\ delay} + (\#hops * router\ latency)$$

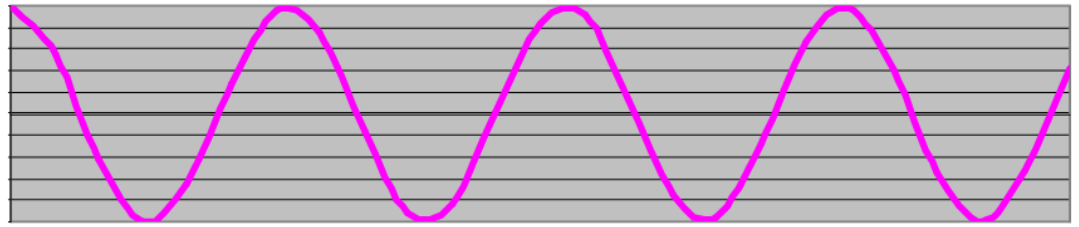
# Computer networking

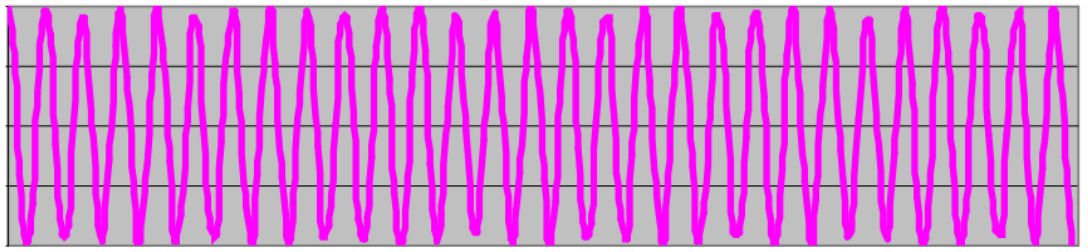
- **Latency**
  - Latency is the main performance consideration over long distances (WAN connections)
  - Latency is trivial in LAN technologies; frequently less than 5 μsecs

$$Latency \cong {}^{Distance}/_{Signal\ propagtion\ delay} + (\#hops * router\ latency)$$

# Bandwidth

- Bits per second:
- e.g.,
  - 10 Mb Ethernet
  - 100 Mb Ethernet
  - 1000 Mb Ethernet





| Circuit | Connection speed (bps) | Relative speed |
|---|---|---|
| **Modem** | **28,800** | **0.5** |
| **Frame Relay** | **56,000** | **1** |
| **T1/DS1** | **1,536,000** | **28** |
| **10 Mb Ethernet** | **10,000,000** | **180** |
| **11 Mb Wireless** | **11,000,000** | **196** |
| **T3/DS3** | **44,736,000** | **800** |
| **OC1** | **51,844,000** | **925** |
| **ATM** | **155,532,000** | **2800** |
| **LTE (4x4 MIMO)** | **326,000,000** | **6189** |
| **IEEE 802.11n** | **600,000,000** | **10,715** |
| **OC12** | **622,128,000** | **11,120** |
| **1-Gigabit Ethernet** | **1,000,000,000** | **18,000** |
| **10 Gigabit Ethernet** | **10,000,000,000** | **180,000** |
| **OC-768** | **40,000,000,000** | **720,000** |

see https://en.wikipedia.org/wiki/List_of_interface_bit_rates

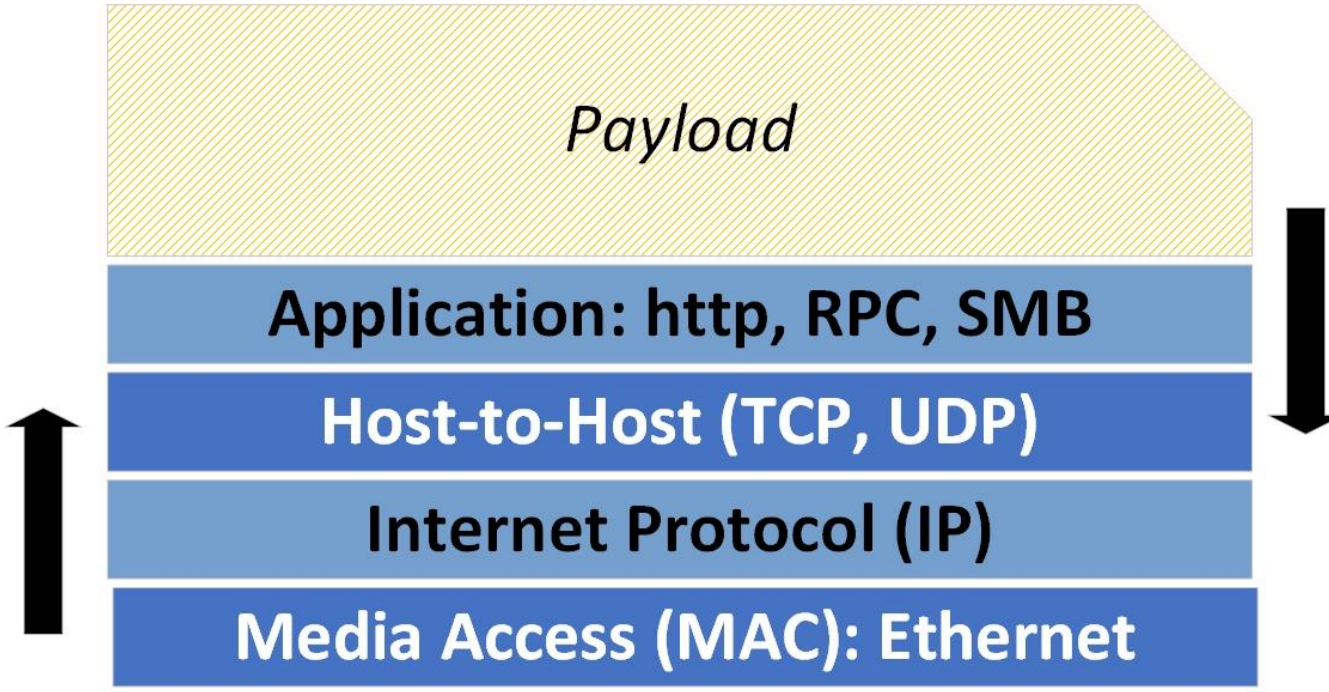
# Computer Networking

- **Latency**
  - Signal propagation delay is a function of distance
  - Round Trip Time (**RTT**) = 2 * Latency
- **Bandwidth**
  - Bits per second
- **Capacity**:
  - Latency * Bandwidth
- There are applications where ***both*** matter (file transfer, backup; digital video)

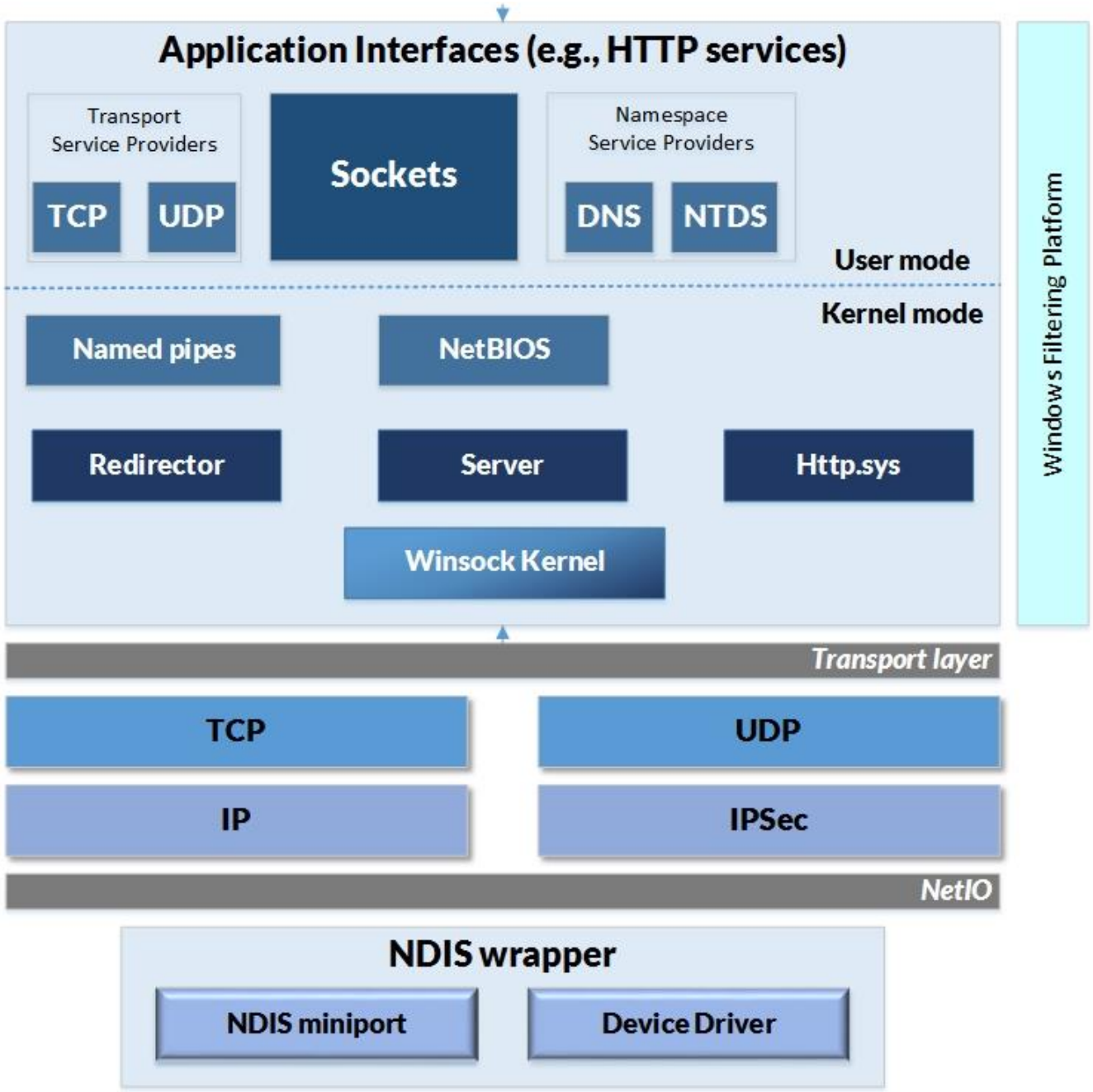
# Protocol Stack

- Internet Architecture

- MAC
  - physical transmission
- Internet Protocol (IP)
  - “Best Effort” packet delivery
- Host-to-Host
  - TCP, UDP
- Application
  - HTTP, etc.

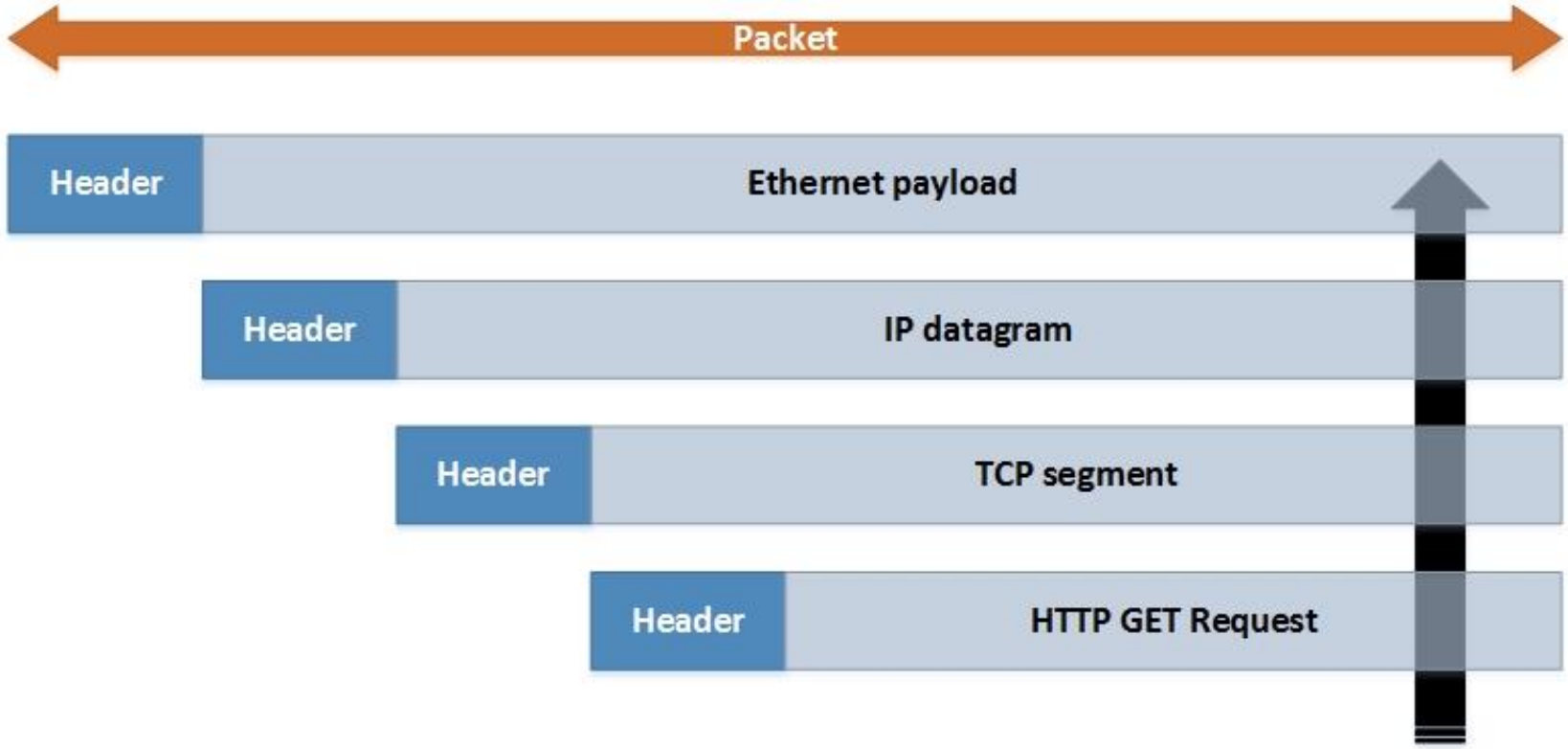


# Windows Networking Architecture



# Effective Bandwidth

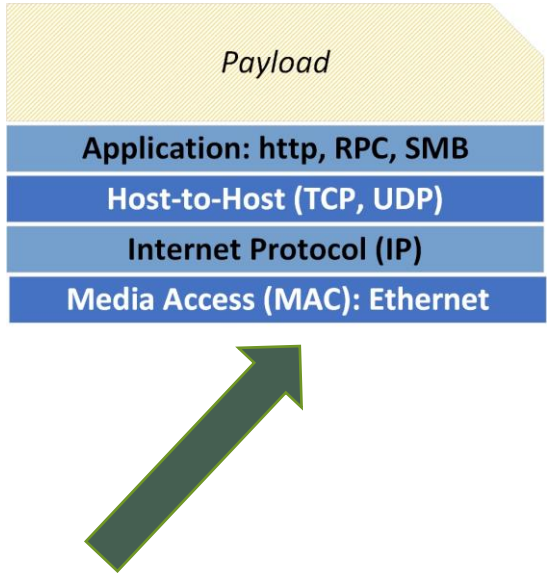
- 10/8 encoding:
  - 2 check bits associated with each 8 bits of data
  - 1000 Mb ⇨ 100 MB/sec
- Packet headers
  - MAC address: 48 bits
  - IPv4: 32 bits
  - IPv6: 128 bits
  - TCP Ports: 16 bits



# Ethernet

- **CSMA/CD:**
  **Carrier Sense Multiple Access / Collision detection**

- **No arbitration**
- **Performance characteristics**
  - **Distance limitations: 1500 meters using twisted pair**
  - **Bandwidth:**
    - **10, 100, and 1,000 Megabit**
  - **Latency:**
    - **51 μsecs at maximum cable length**
    - **most practical systems: < 5 μsecs**



# Ethernet

- Packet format
  - Transmit a standard 64 bit Preamble:
    - alternating 0,1 bits
  - Destination address: 48-bit unique MAC address
  - Source address: 48-bit unique MAC address
  - Flags
  - Maximum Transmission Unit (MTU)
    - Standard frame: Up to 1500 bytes of data, followed by CRC, Postamble
    - Jumbo Frames

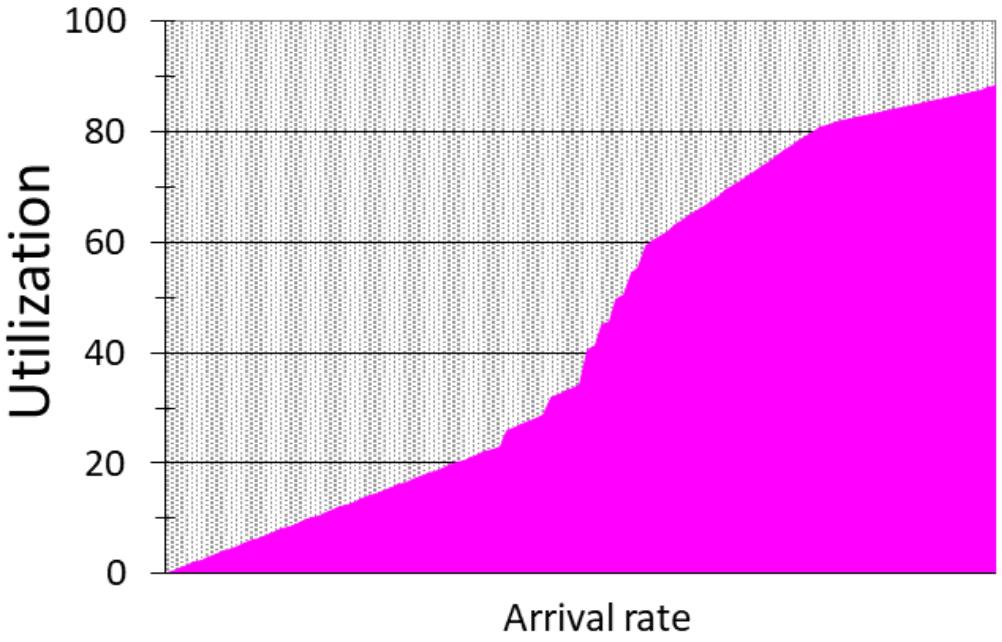
# Ethernet collisions

- Shared media multiple access protocol (10baseT)
  - Each station must wait 51 μsecs before sending the next packet,
  - A station waiting to send a packet transmits immediately.
  - When two or more stations attempt to transmit simultaneous, there is a **collision**.
  - Collisions are detected by the Network Interface card (NIC) before a limit of 512 bits is sent.

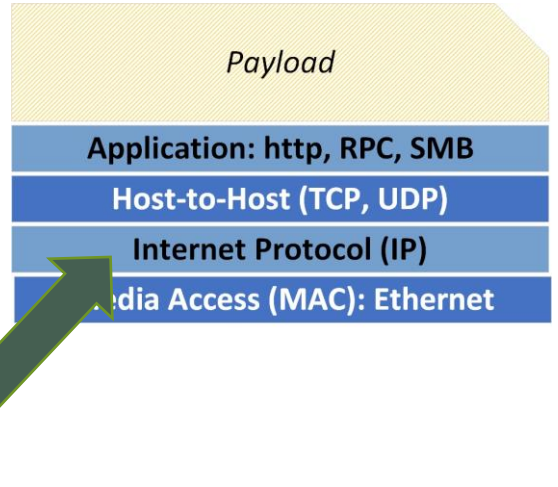
- Switches
  - create dedicated point-to-point links (star topology)
  - but collisions will still occur on the same segment for endpoints that are being accessed concurrently

# Ethernet collisions

- Back-off and retry
  - Each adapter retires following a collision
    - First retry: randomly distributed between 0 and 51 μsecs
    - Exponential backoff:
      - Subsequent retries based on doubling, quadrupling, etc., the retry interval until successful
  - On a shared loop, collisions tend to occur if utilization > 0.30 **and** multiple stations are transmitting.
  - Using Switches, 100% utilization is achievable point-to-point



# IP (Internet Protocol)




- Dynamic routing protocol
  - A "Best effort" service model that is both unreliable and connectionless
    - The host connection layer above IP is responsible for reliable, in-order delivery of packets (TCP).
  - Time To Live (TTL) counter prevents packet from circulating indefinitely
- Packet fragmentation and reassembly
  - Application packets that are too big for the MAC layer

# IP Routing

- Routing protocol
  - Each packet contains destination and source address
  - IP makes a "best effort" to deliver packets, but is not responsible for notifying the sender upon either successful or unsuccessful transmission.
  - Routers forward packets to either a directly connected address or to the **next hop**
    - Router overhead ≈ 5 - 25 microseconds
    - Many routers **drop** packets, rather than queue them
      - deterministic service time

- IP Header:
  - Length
  - Checksum
  - Reassembly instructions
  - Hop count
  - Source
  - Destination

# IP Routing

- Routing protocol
  - Each packet is transmitted independently.
  - "Best effort"
    - Different packets can take different routes
    - Packets can arrive at their destination out of order
    - Packets might not arrive at their destination at all!
- Two technical challenges
  - How is knowledge of the network topology stored?
    - It is not. Only information about direct-attached connections is maintained locally in ***Routing Tables***
  - How to prevent a very fast router from overloading a slow receiver?
    - Flow control in IP is the responsibility of the upper layer Host-to-Host connection (i.e., TCP)

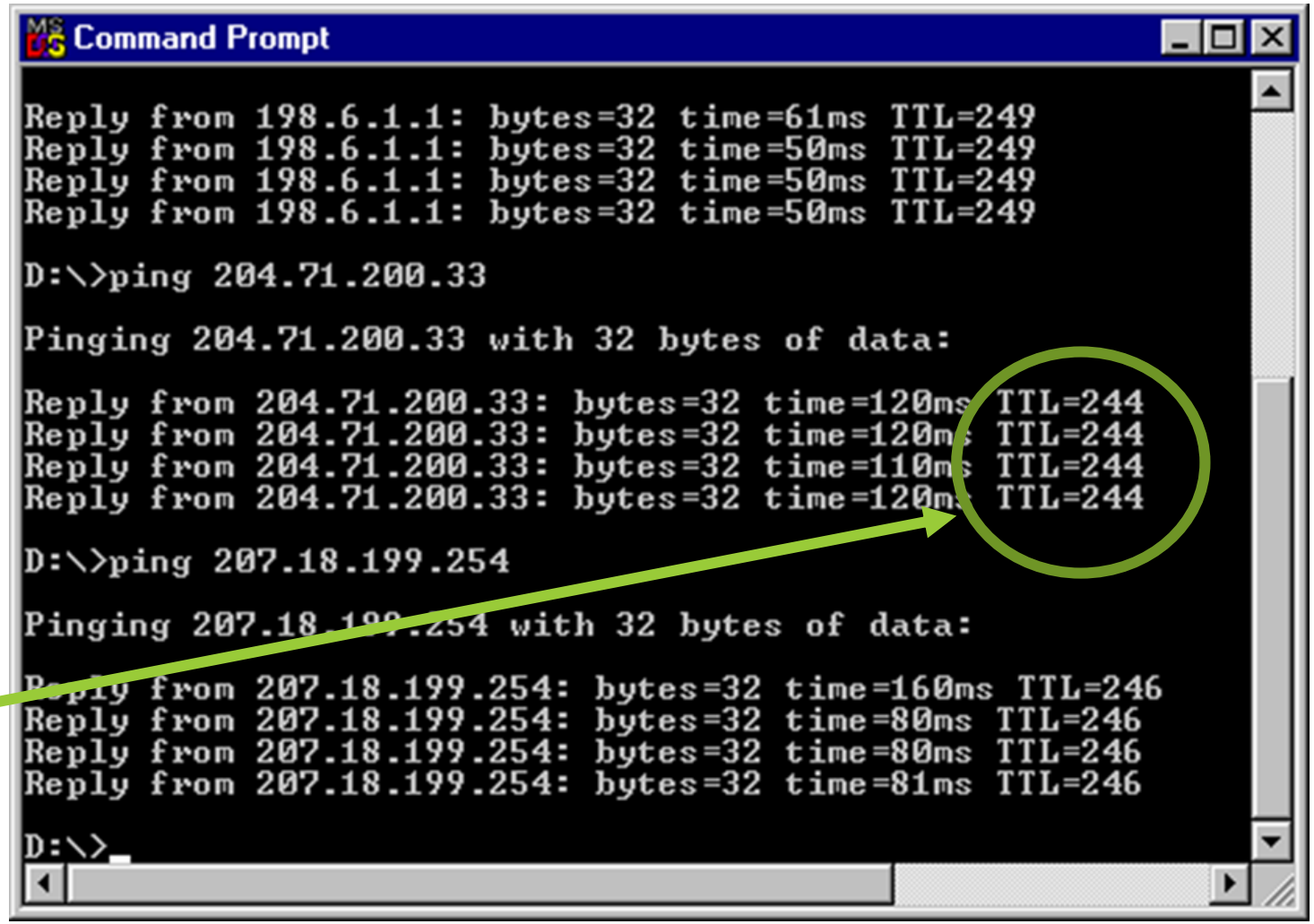
- Sessionless
- Connectionless

- No state!

```
route print

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.153     50
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    172.31.104.80  255.255.255.240         On-link     172.31.104.81   5256
    172.31.104.81  255.255.255.255         On-link     172.31.104.81   5256
    172.31.104.95  255.255.255.255         On-link     172.31.104.81   5256
      192.168.0.0    255.255.255.0         On-link     192.168.0.153    306
    192.168.0.153  255.255.255.255         On-link     192.168.0.153    306
    192.168.0.255  255.255.255.255         On-link     192.168.0.153    306
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link     172.31.104.81   5256
        224.0.0.0        240.0.0.0         On-link     192.168.0.153    306
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link     172.31.104.81   5256
  255.255.255.255  255.255.255.255         On-link     192.168.0.153    306
===========================================================================
Persistent Routes:
  None
```

# IP tools

- Ping
- FreePing



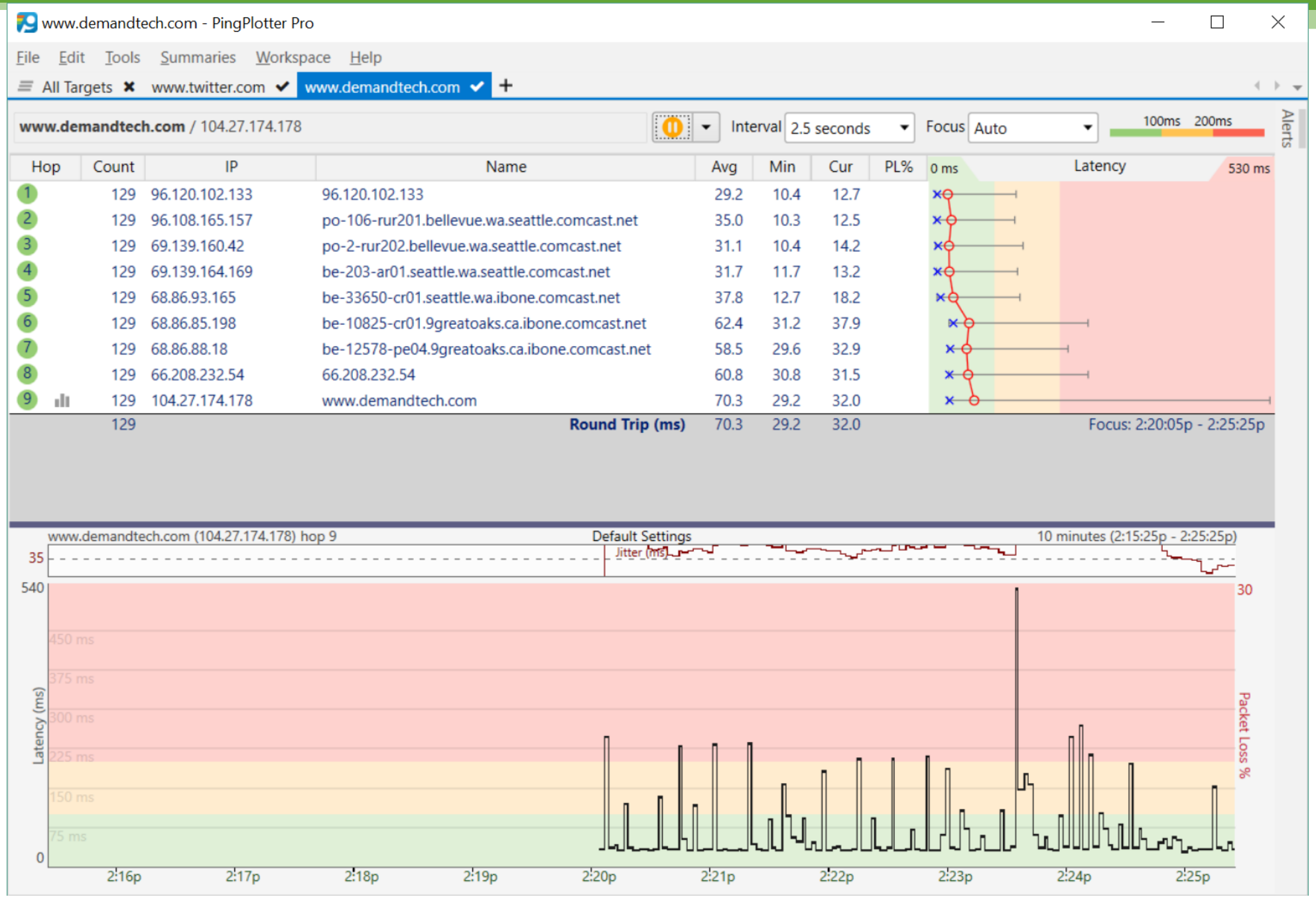

- TTL = 255 - # of hops

# Tracert

```
C:\>tracert 104.244.42.129

Tracing route to 104.244.42.129 over a maximum of 30 hops

  1    14 ms    13 ms    13 ms  96.120.102.133
  2    14 ms    11 ms    13 ms  po-106-rur201.bellevue.wa.seattle.comcast.net [96.108.165.157]
  3    16 ms    12 ms    12 ms  po-2-rur202.bellevue.wa.seattle.comcast.net [69.139.160.42]
  4    13 ms    18 ms    12 ms  be-203-ar01.seattle.wa.seattle.comcast.net [69.139.164.169]
  5    15 ms    22 ms    16 ms  be-33650-cr01.seattle.wa.ibone.comcast.net [68.86.93.165]
  6    12 ms    13 ms    14 ms  be-10846-pe01.seattle.wa.ibone.comcast.net [68.86.86.90]
  7    32 ms   123 ms    12 ms  66.208.232.186
  8    13 ms    15 ms    13 ms  twitter-ic-305911-sea-b1.c.telia.net [62.115.41.158]
  9     *        *        *     Request timed out.
 10   216 ms    37 ms    80 ms  104.244.42.129

Trace complete.
```

# Ping Plotter:

# visual trace route



# ICMP

- Internet Control Message Protocol
  - Error messages: e.g., **Destination Unreachable**
  - ***ping*** uses Echo Reply and Echo Request messages
  - ***tracert*** uses Echo Reply and Echo Request messages with TTL starting at 1, then increments TTL until the final destination is reached
    - That is why it sometimes takes less time to reach longer intermediate locations!
  - ***PMTU*** turns on Don't Fragment bit using progressively shorter messages

# TCP

- Transmission Control Protocol
  - Connection-oriented
  - Ensures in-order delivery of all packets
  - Provides flow control
    - "Sliding window" algorithm
    - Reactive ***Congestion Window***
  - Adaptive retransmission (time out values)
  - *de facto* standard due to Internet apps
    - Sockets interface
    - supports http, ftp, smtp, telnet, etc.

# TCP Header

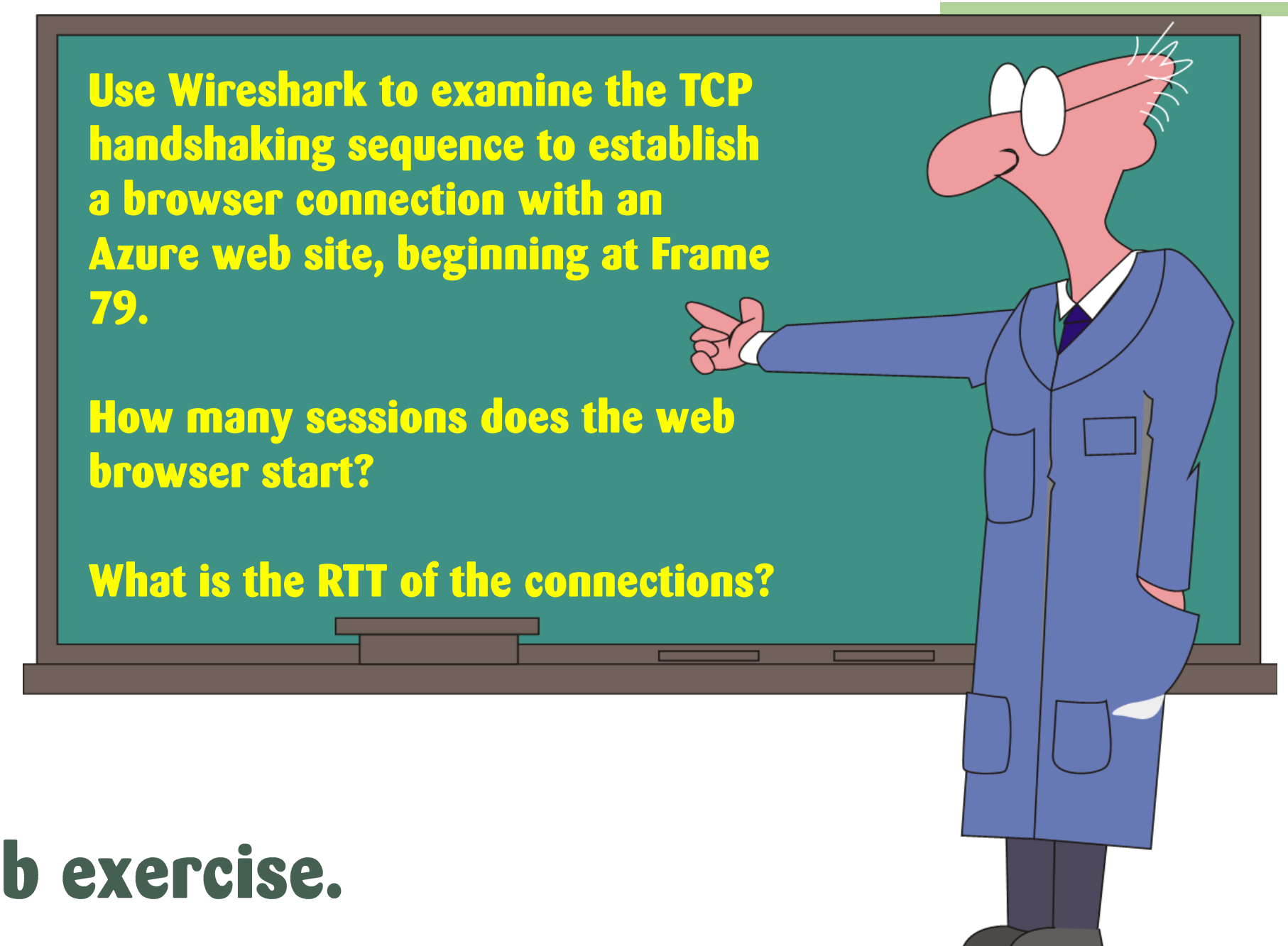

- Byte-oriented
- Session-oriented
- In order delivery
- Flow control:
  - Advertised window
  - How much data on the wire can be transmitted at a time before an ACK is required
- Options
  - Timestamps
  - SACK

# TCP Ports

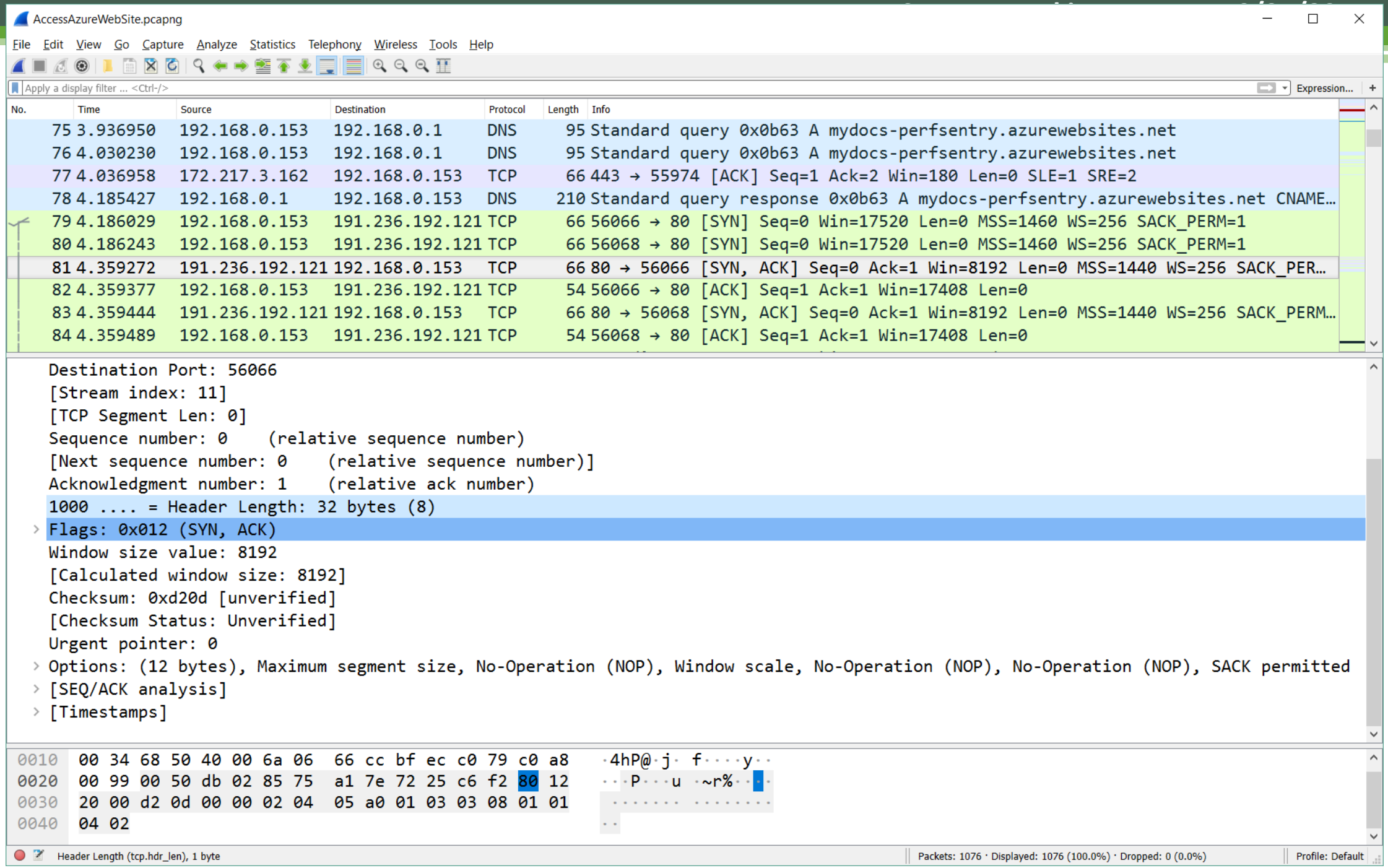
| | |
|---|---|
| 20 | FTP data |
| 21 | FTP Control |
| 23 | Telnet |
| 25 | Simple Mail Transport Protocol (SMTP) |
| 66 | Oracle SQL*NET |
| 70 | Gopher |
| 80 | Hypertext Transfer Protocol (HTTP) |
| 160 | SNMP Traps |
| 161 | SNMP |
| 179 | BGP |
| 389 | Lightweight Directory Access Protocol (LDAP) |
| 443 | Secure HTTP |
| 445 | Server Message Block |

# TCP Connections

- Handshaking to:
  - Establish a connection
    - **SYN**, **SYN-ACK**, and **ACK** messages for the exchange of byte Sequence Numbers
    - 32-bit Byte Sequence Numbers identify where any specific packet belongs in the byte-stream that reflects the exchange of messages between Host applications
    - "Negotiate" session parameters – more like a Least Common Denominator
      - **AdvertisedWindow** and Options
  - Terminate a connection
    - Two sets of FIN, FIN-ACK message pairs required




# Lab exercise.


AccessAzureWebSite.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
75 3.936950 192.168.0.153 192.168.0.1 DNS 95 Standard query 0x0b63 A mydocs-perfsentry.azurewebsites.net
76 4.030230 192.168.0.153 192.168.0.1 DNS 95 Standard query 0x0b63 A mydocs-perfsentry.azurewebsites.net
77 4.036958 172.217.3.162 192.168.0.153 TCP 66 443 → 55974 [ACK] Seq=1 Ack=2 Win=180 Len=0 SLE=1 SRE=2
78 4.185427 192.168.0.1 192.168.0.153 DNS 210 Standard query response 0x0b63 A mydocs-perfsentry.azurewebsites.net CNAME…
79 4.186029 192.168.0.153 191.236.192.121 TCP 66 56066 → 80 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
80 4.186243 192.168.0.153 191.236.192.121 TCP 66 56068 → 80 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
81 4.359272 191.236.192.121 192.168.0.153 TCP 66 80 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PER…
82 4.359377 192.168.0.153 191.236.192.121 TCP 54 56066 → 80 [ACK] Seq=1 Ack=1 Win=17408 Len=0
83 4.359444 191.236.192.121 192.168.0.153 TCP 66 80 → 56068 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PERM…
84 4.359489 192.168.0.153 191.236.192.121 TCP 54 56068 → 80 [ACK] Seq=1 Ack=1 Win=17408 Len=0
Destination Port: 56066
[Stream index: 11]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
[Next sequence number: 0 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, ACK)
Window size value: 8192
[Calculated window size: 8192]
Checksum: 0xd20d [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
[SEQ/ACK analysis]
[Timestamps]
0010 00 34 68 50 40 00 6a 06 66 cc bf ec c0 79 c0 a8 ·4hP@·j· f····y··
0020 00 99 00 50 db 02 85 75 a1 7e 72 25 c6 f2 80 12 ···P···u ·~r%····
0030 20 00 d2 0d 00 00 02 04 05 a0 01 03 03 08 01 01 ···· ···· ····
0040 04 02 ··
Header Length (tcp.hdr_len), 1 byte
Packets: 1076 · Displayed: 1076 (100.0%) · Dropped: 0 (0.0%)
Profile: Default

# TCP Sessions: in-order delivery

- Protocol guarantees that all bytes sent are received
  - Each byte sent must be specifically ***acknowledged*** by the Receiver
    - ACK indicator and Byte number in the TCP header
    - Option to use 32-bit ACK + 32-bit TimeStamp in tandem
  - Initial handshaking establishes the first byte of data to be transmitted by the Sender (**SYN-ACK** reply packet)
  - ACK packets frequently consist of header data only (although full duplex operations – ***piggybacking*** – is supported)
  - Selective Acknowledgement (SACK) allows for larger Receive Windows
  - Any bytes not acknowledged must be ***retransmitted*** (after a suitable delay, based on the Session **RTT**)

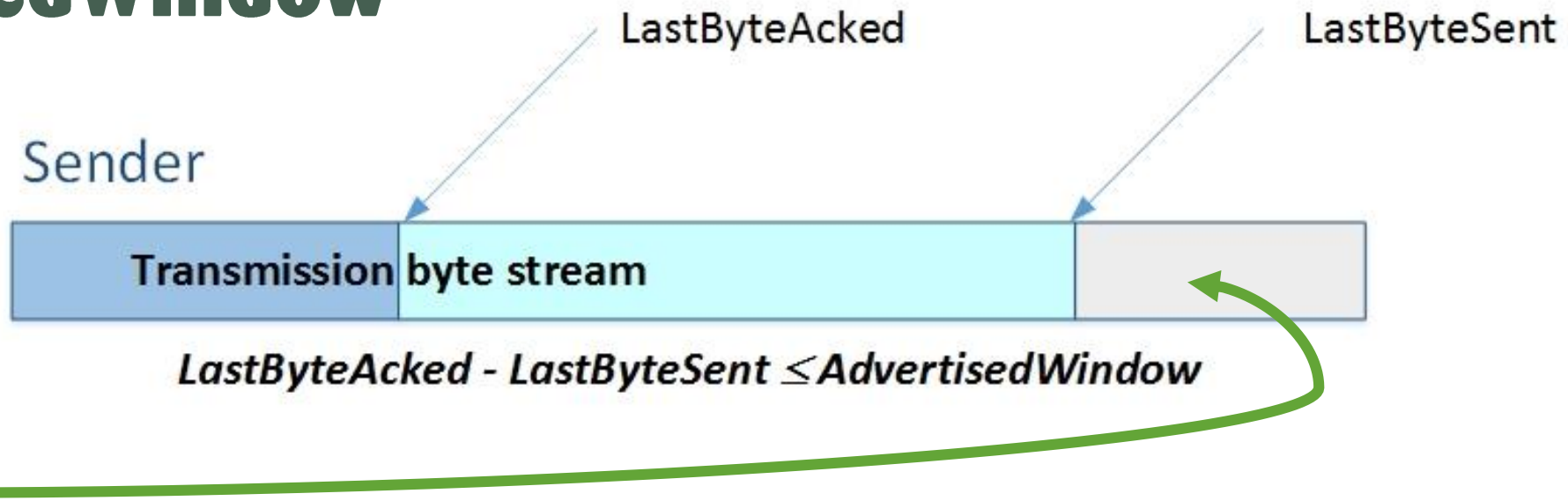
# TCP Sessions

- Performance implications of ACKs
  - **RTT** limits throughput over long distances
  - Efficiency
    - Receiver can defer individual acknowledgements. While each byte sent must be acknowledged, each packet sent does *not* require an ACK packet in return.
  - ***Flow control***:
    - Sender must wait for an ACK once it has sent a **window**'s worth of data
  - ***Congestion Window***:
    - TCP detects and dynamically reacts to network congestion
  - ***Timeout & retransmission***:
    - Determine if a Un-ACKed packet was damaged in transmission or the network is just slow?

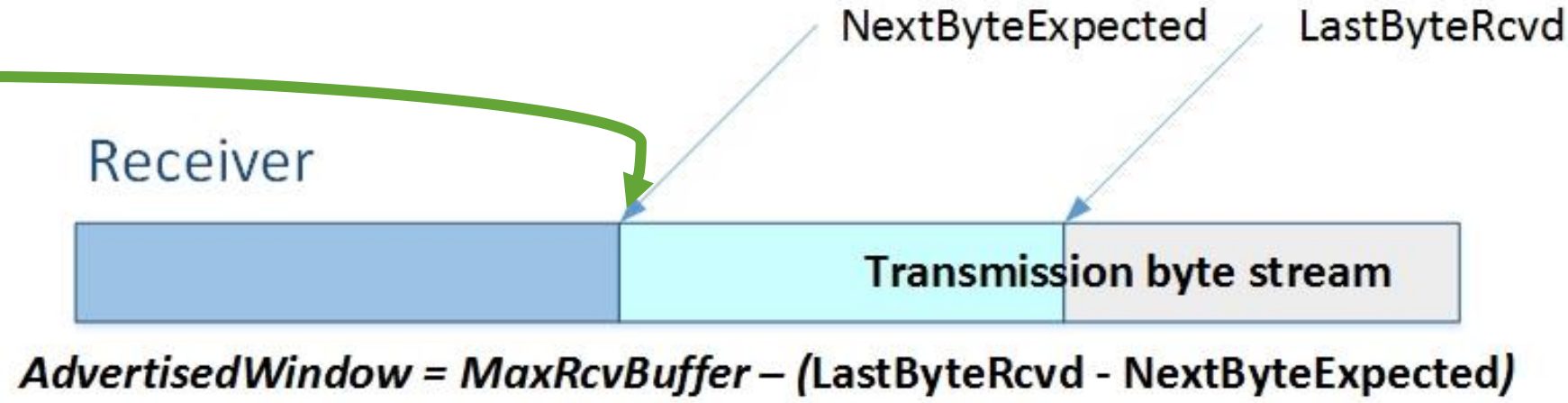
# TCP AdvertisedWindow

**if**
*NextPacket + (LastByteAcked – LastByteSent) > AdvertisedWindow*

**Sender is forced to Wait**

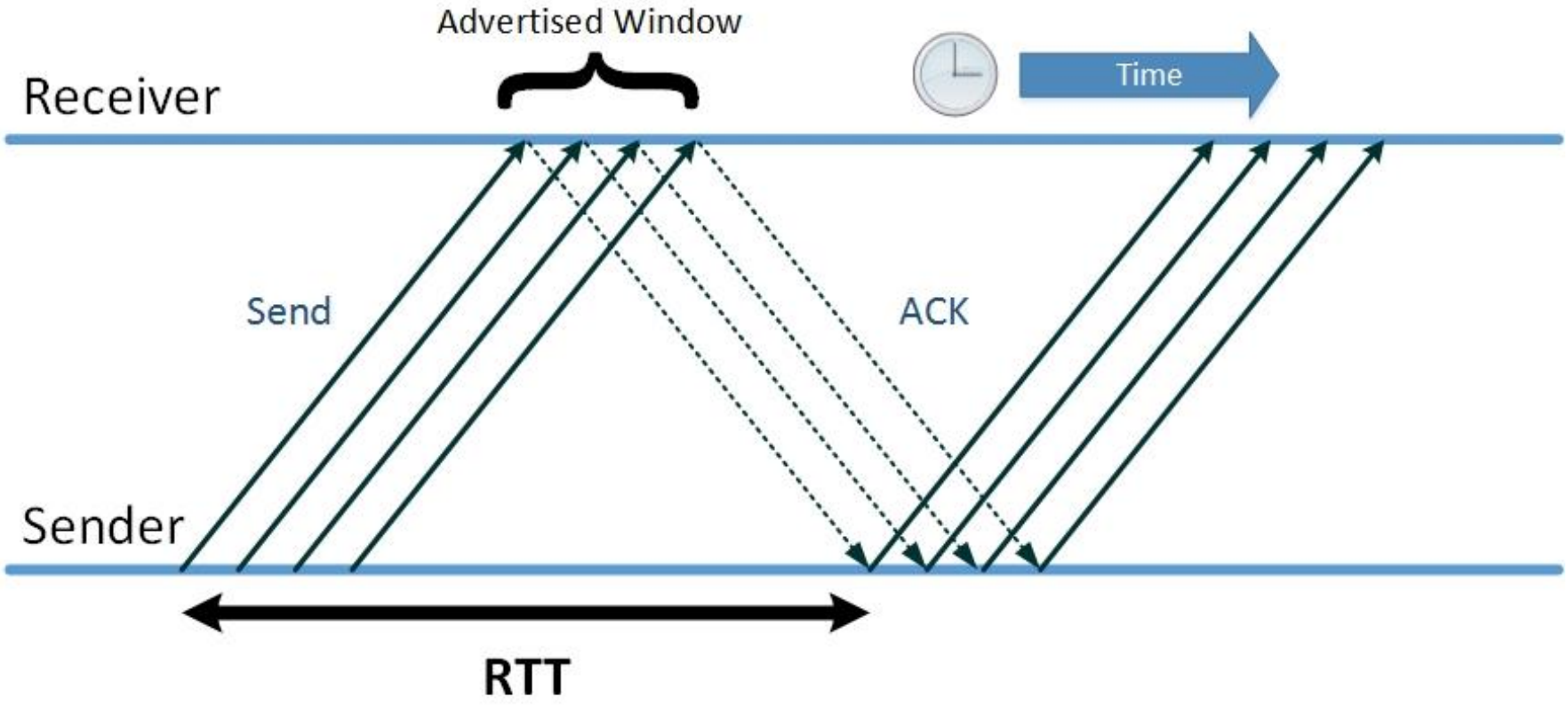


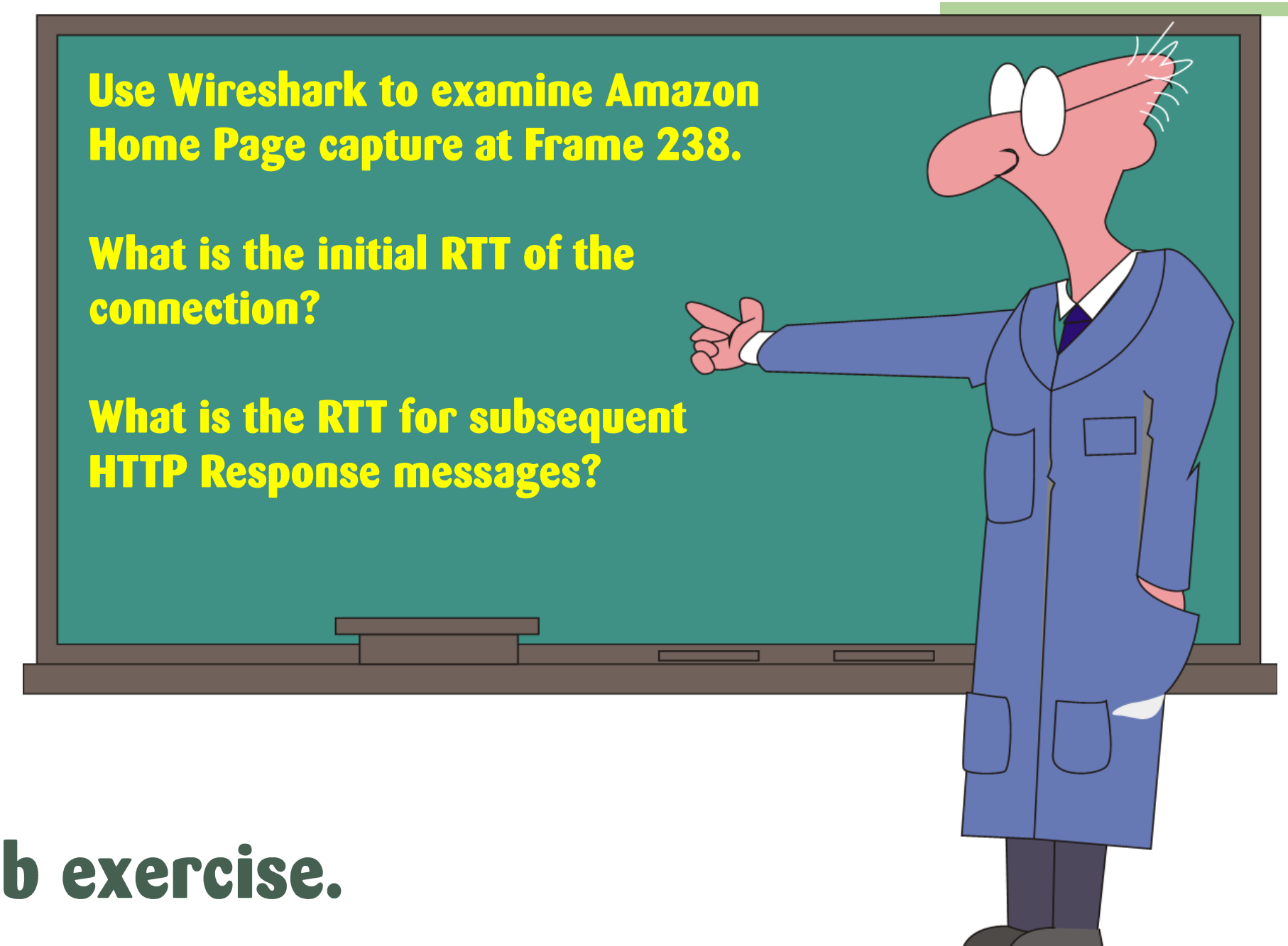

**While a packet in the middle remains missing, when a later one is received, Receiver sends a duplicate ACK for NextByteExpected (unless SACK is enabled).**



# TCP Sliding Window

- Once the Advertised Window is full, the Sender must wait for an ACK
- This pause is also regarded as an implicit **congestion signal**.

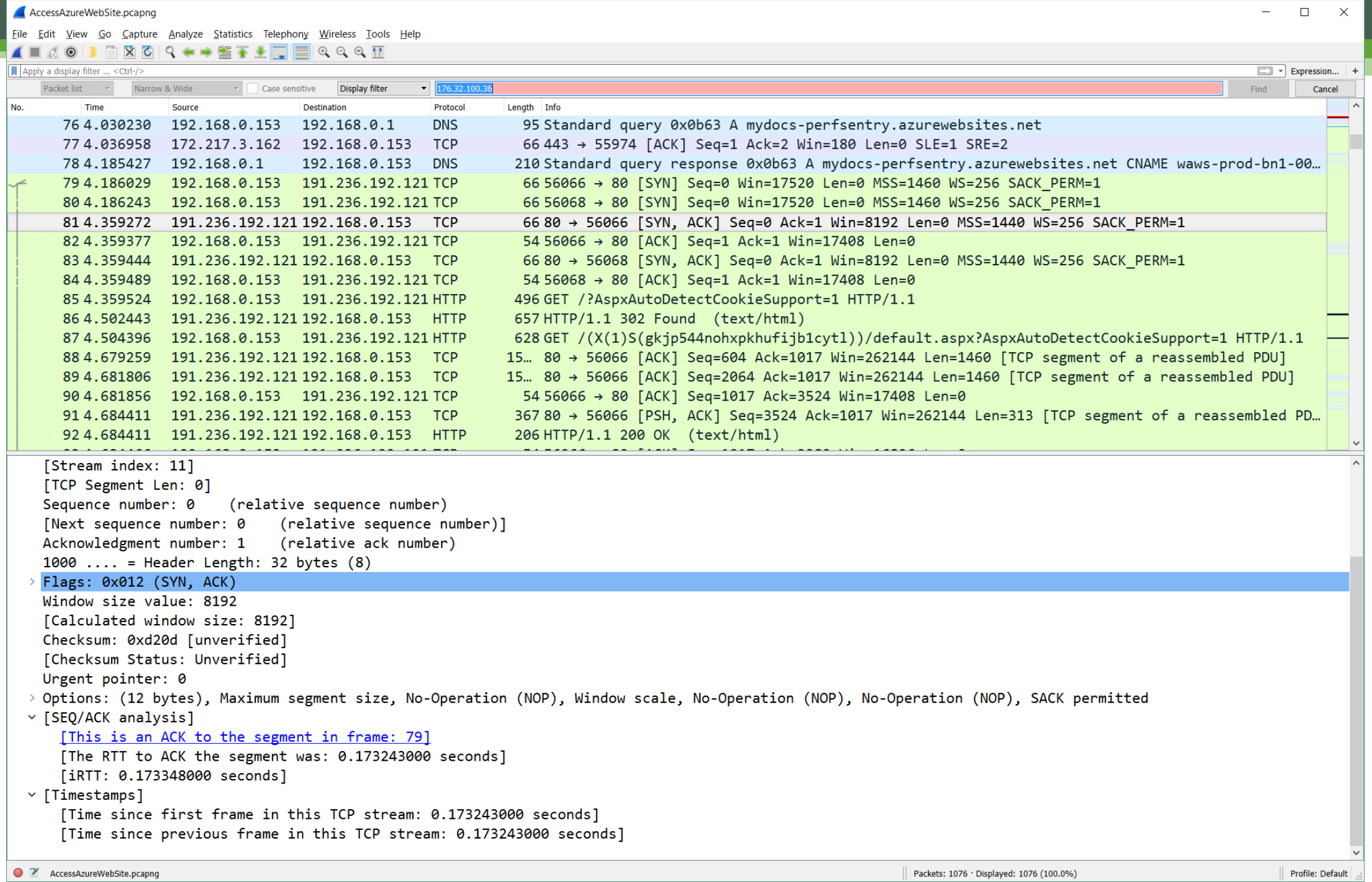





# Lab exercise.

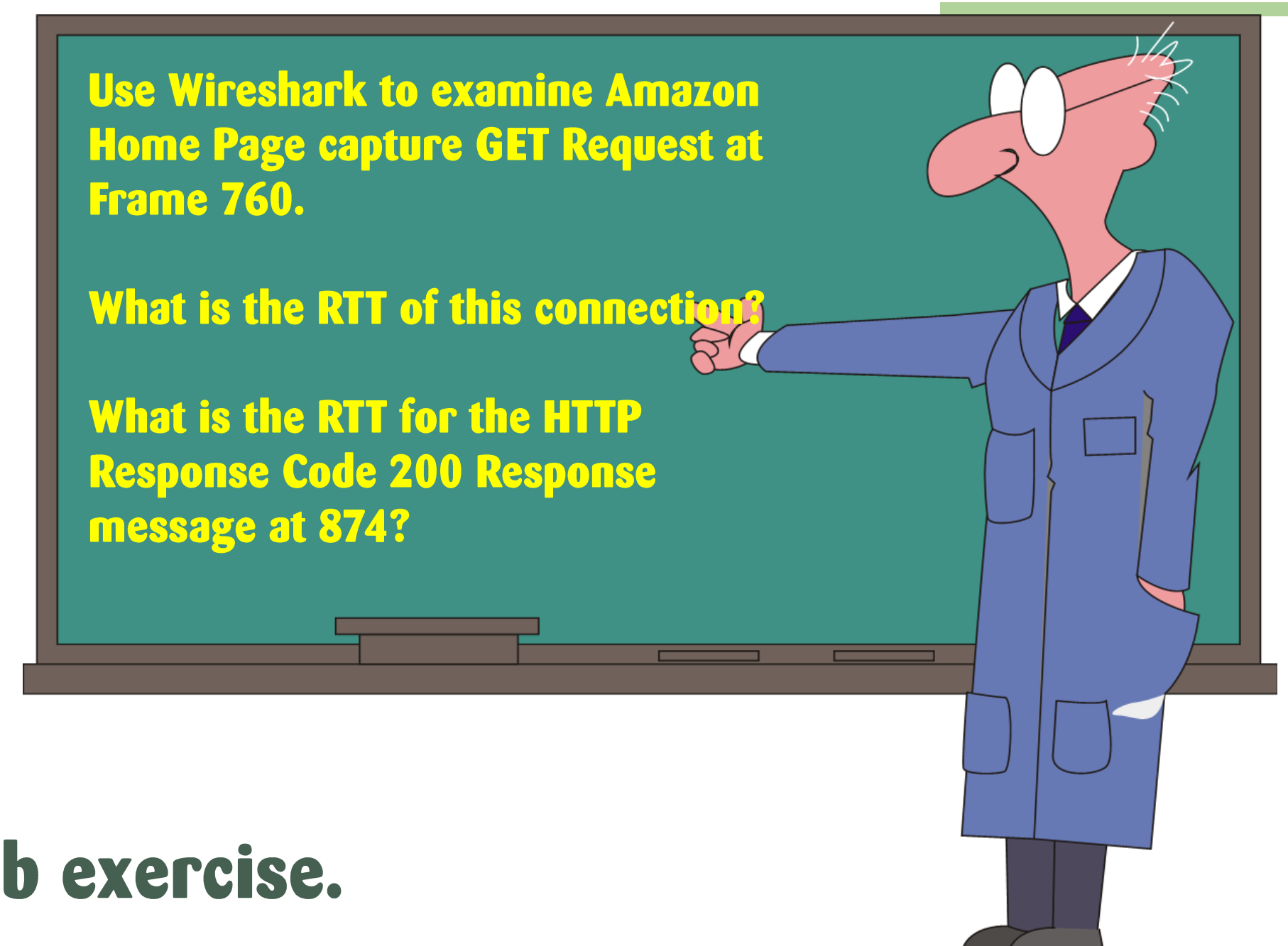
# TCP Sessions

- RTT is maintained per Session
  - Initial the Session RTT is set based on the TCP ACK
    - e.g.,
      - initial SYN-ACK response packet is received at #81 in the Trace
      - initial RTT ≅ 175 ms.
  - Subsequently, the RTT includes the Receiver Host application latency!
    - e.g.,

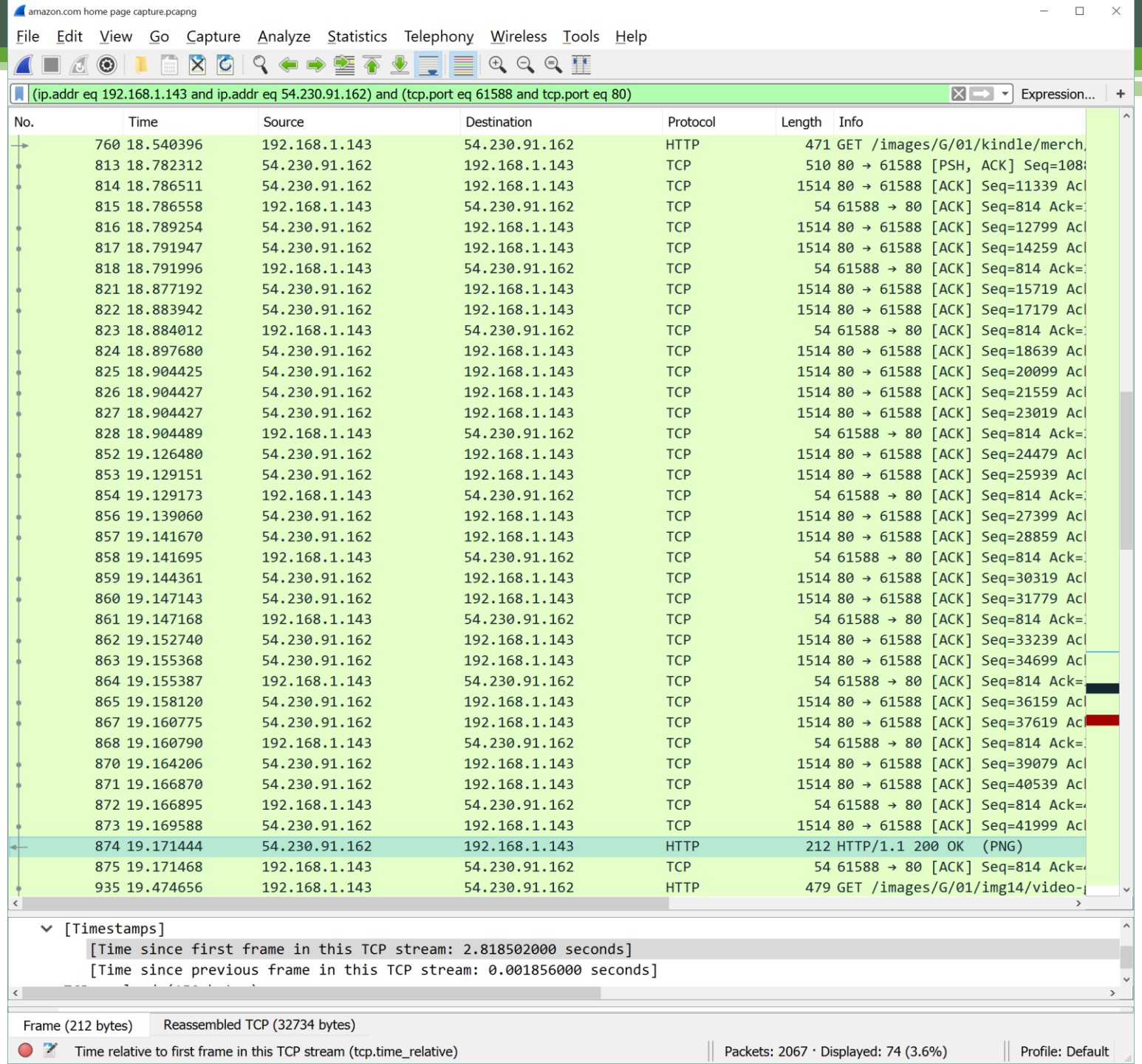

AccessAzureWebSite.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Packet list
Narrow & Wide
Case sensitive
Display filter
176.32.100.36
Find
Cancel
No. Time Source Destination Protocol Length Info
76 4.030230 192.168.0.153 192.168.0.1 DNS 95 Standard query 0x0b63 A mydocs-perfsentry.azurewebsites.net
77 4.036958 172.217.3.162 192.168.0.153 TCP 66 443 → 55974 [ACK] Seq=1 Ack=2 Win=180 Len=0 SLE=1 SRE=2
78 4.185427 192.168.0.1 192.168.0.153 DNS 210 Standard query response 0x0b63 A mydocs-perfsentry.azurewebsites.net CNAME waws-prod-bn1-00…
79 4.186029 192.168.0.153 191.236.192.121 TCP 66 56066 → 80 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
80 4.186243 192.168.0.153 191.236.192.121 TCP 66 56068 → 80 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
81 4.359272 191.236.192.121 192.168.0.153 TCP 66 80 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PERM=1
82 4.359377 192.168.0.153 191.236.192.121 TCP 54 56066 → 80 [ACK] Seq=1 Ack=1 Win=17408 Len=0
83 4.359444 191.236.192.121 192.168.0.153 TCP 66 80 → 56068 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256 SACK_PERM=1
84 4.359489 192.168.0.153 191.236.192.121 TCP 54 56068 → 80 [ACK] Seq=1 Ack=1 Win=17408 Len=0
85 4.359524 192.168.0.153 191.236.192.121 HTTP 496 GET /?AspxAutoDetectCookieSupport=1 HTTP/1.1
86 4.502443 191.236.192.121 192.168.0.153 HTTP 657 HTTP/1.1 302 Found (text/html)
87 4.504396 192.168.0.153 191.236.192.121 HTTP 628 GET /(X(1)S(gkjp544nohxpkhufijb1cytl))/default.aspx?AspxAutoDetectCookieSupport=1 HTTP/1.1
88 4.679259 191.236.192.121 192.168.0.153 TCP 15… 80 → 56066 [ACK] Seq=604 Ack=1017 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
89 4.681806 191.236.192.121 192.168.0.153 TCP 15… 80 → 56066 [ACK] Seq=2064 Ack=1017 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
90 4.681856 192.168.0.153 191.236.192.121 TCP 54 56066 → 80 [ACK] Seq=1017 Ack=3524 Win=17408 Len=0
91 4.684411 191.236.192.121 192.168.0.153 TCP 367 80 → 56066 [PSH, ACK] Seq=3524 Ack=1017 Win=262144 Len=313 [TCP segment of a reassembled PD…
92 4.684411 191.236.192.121 192.168.0.153 HTTP 206 HTTP/1.1 200 OK (text/html)
[Stream index: 11]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
[Next sequence number: 0 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, ACK)
Window size value: 8192
[Calculated window size: 8192]
Checksum: 0xd20d [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
[SEQ/ACK analysis]
[This is an ACK to the segment in frame: 79]
[The RTT to ACK the segment was: 0.173243000 seconds]
[iRTT: 0.173348000 seconds]
[Timestamps]
[Time since first frame in this TCP stream: 0.173243000 seconds]
[Time since previous frame in this TCP stream: 0.173243000 seconds]
AccessAzureWebSite.pcapng
Packets: 1076 · Displayed: 1076 (100.0%)
Profile: Default

# TCP Sessions

- RTT is maintained per Session
  - RTT is used to figure out how long to Wait before re-transmitting an unACKed packet
  - Initially, the Session RTT is set based on the TCP reply ACK from the connection SYN request
    - e.g.,
      - initial SYN-ACK response packet is received at #247 in the Trace
      - initial RTT $\cong$ 175 ms.




# Lab exercise.

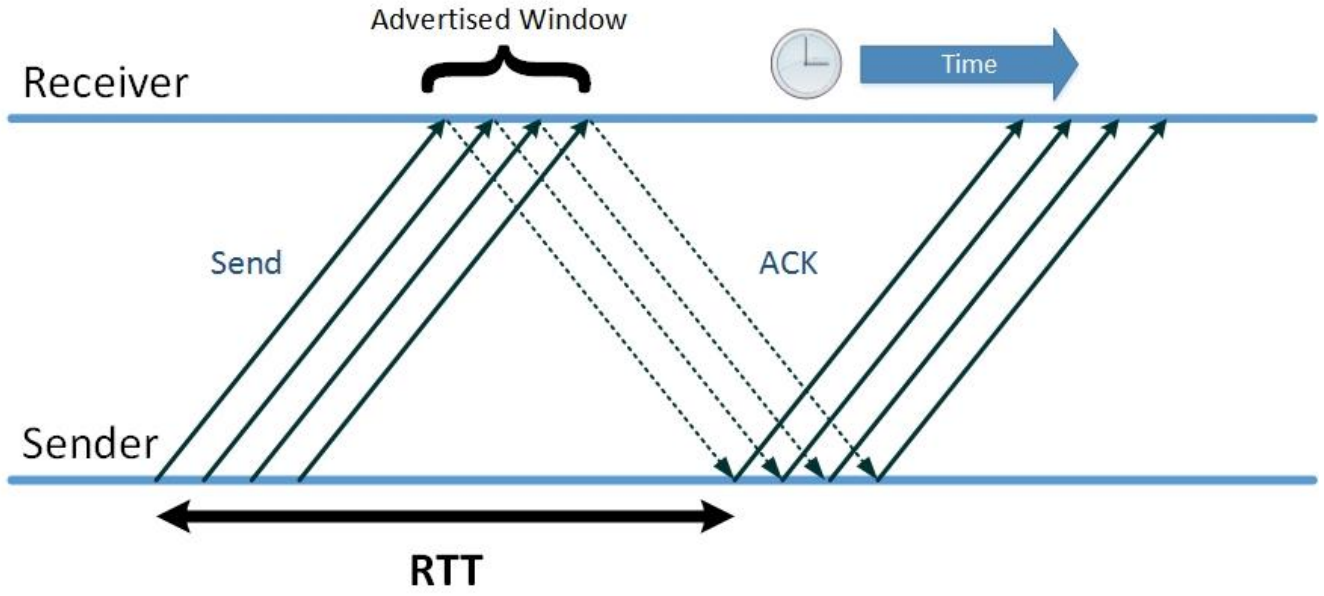

amazon.com home page capture.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
(ip.addr eq 192.168.1.143 and ip.addr eq 54.230.91.162) and (tcp.port eq 61588 and tcp.port eq 80)
Expression...
No. Time Source Destination Protocol Length Info
760 18.540396 192.168.1.143 54.230.91.162 HTTP 471 GET /images/G/01/kindle/merch,
813 18.782312 54.230.91.162 192.168.1.143 TCP 510 80 → 61588 [PSH, ACK] Seq=108
814 18.786511 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=11339 Acl
815 18.786558 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
816 18.789254 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=12799 Acl
817 18.791947 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=14259 Acl
818 18.791996 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
821 18.877192 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=15719 Acl
822 18.883942 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=17179 Acl
823 18.884012 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
824 18.897680 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=18639 Acl
825 18.904425 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=20099 Acl
826 18.904427 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=21559 Acl
827 18.904427 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=23019 Acl
828 18.904489 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
852 19.126480 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=24479 Acl
853 19.129151 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=25939 Acl
854 19.129173 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
856 19.139060 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=27399 Acl
857 19.141670 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=28859 Acl
858 19.141695 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
859 19.144361 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=30319 Acl
860 19.147143 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=31779 Acl
861 19.147168 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
862 19.152740 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=33239 Acl
863 19.155368 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=34699 Acl
864 19.155387 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
865 19.158120 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=36159 Acl
867 19.160775 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=37619 Acl
868 19.160790 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
870 19.164206 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=39079 Acl
871 19.166870 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=40539 Acl
872 19.166895 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
873 19.169588 54.230.91.162 192.168.1.143 TCP 1514 80 → 61588 [ACK] Seq=41999 Acl
874 19.171444 54.230.91.162 192.168.1.143 HTTP 212 HTTP/1.1 200 OK (PNG)
875 19.171468 192.168.1.143 54.230.91.162 TCP 54 61588 → 80 [ACK] Seq=814 Ack=
935 19.474656 192.168.1.143 54.230.91.162 HTTP 479 GET /images/G/01/img14/video-
[Timestamps]
[Time since first frame in this TCP stream: 2.818502000 seconds]
[Time since previous frame in this TCP stream: 0.001856000 seconds]
Frame (212 bytes)
Reassembled TCP (32734 bytes)
Time relative to first frame in this TCP stream (tcp.time_relative)
Packets: 2067 · Displayed: 74 (3.6%)
Profile: Default

# TCP Sessions

- Subsequently, the RTT includes the Receiver Host application latency!
  - e.g.,
    - HTTP GET Request at #760
    - First Response packet received at #813
    - when it includes the Application layer, RTT ≅ 240 ms.
    - HTTP Response Code 200 (“OK”) Reply message received at #874
    - HTTP Request RTT ≅ 2820 ms.

# TCP flow control

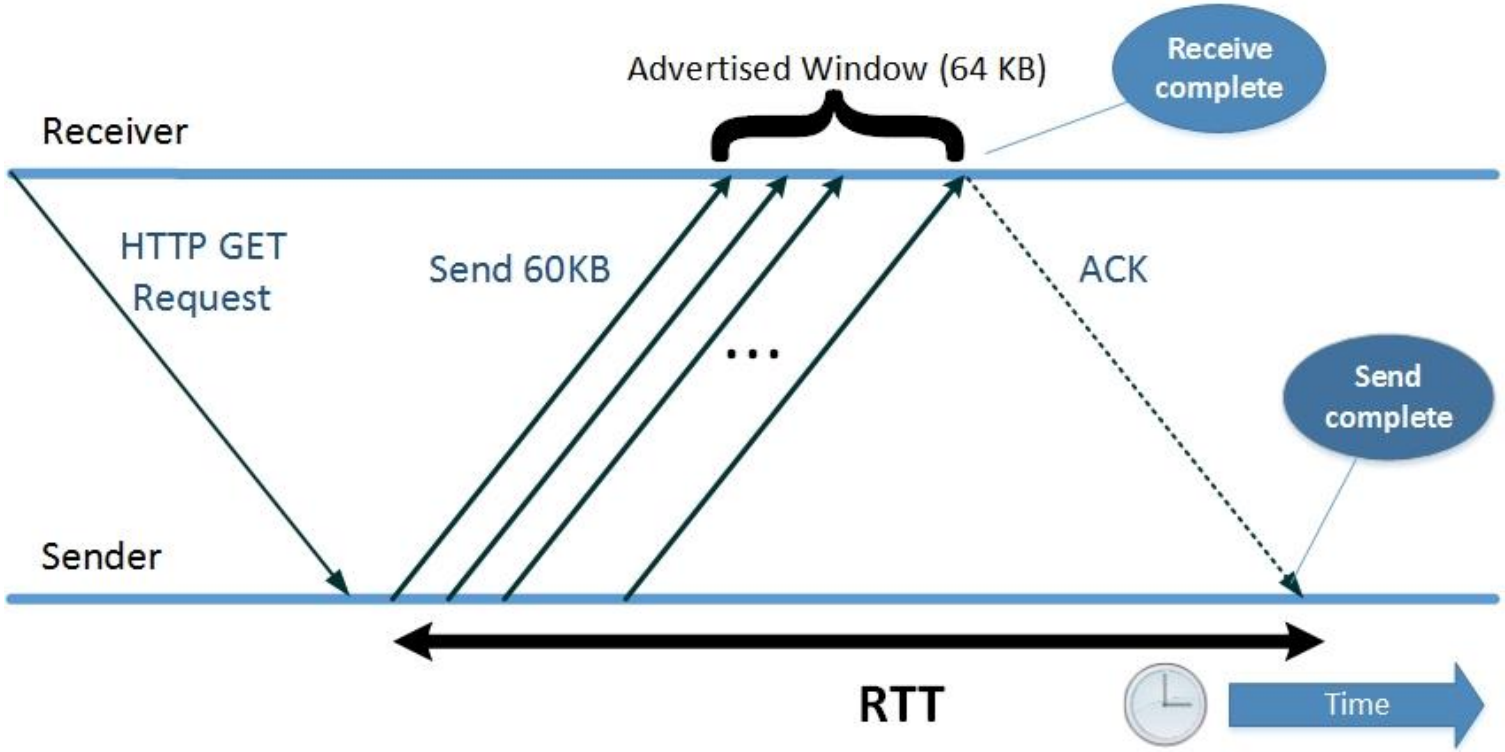
- How does RTT & the Sliding Window limit throughput per connection?




**Max Thruput per connection**
**= Window size / (signal propagation delay * 2)**

# TCP flow control

- How does RTT & the Sliding Window limit throughput per connection?
- Example: Get Request for a 60 KB image file

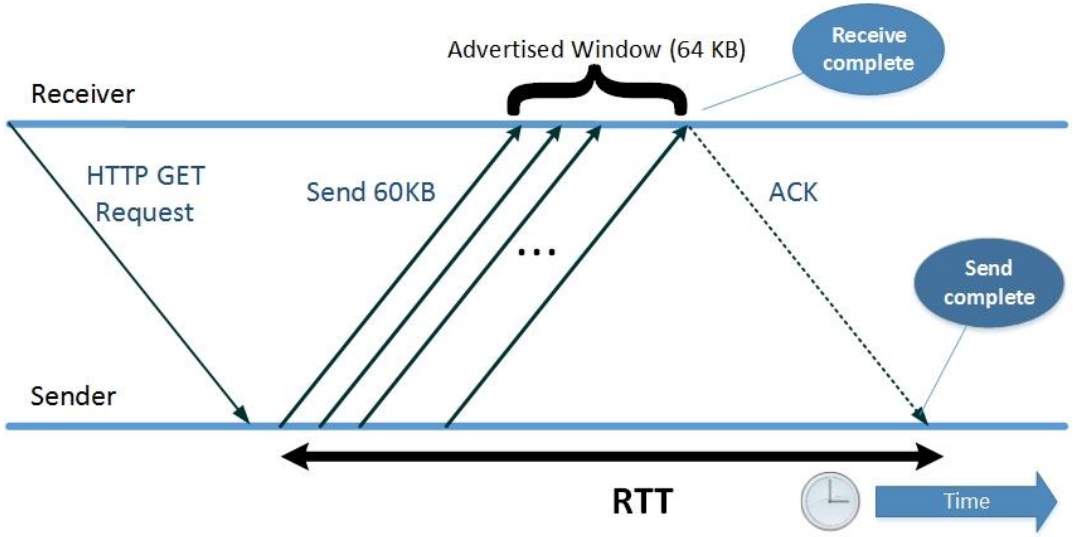



Max Thruput/connection = AdvertisedWindow / RTT

- How does RTT & the Sliding Window limit throughput per connection?

- Example:
  - RTT for amazon.com = 30 ms
  - Max thruput =
    33 * 64 KB ≅ 2.1 MB

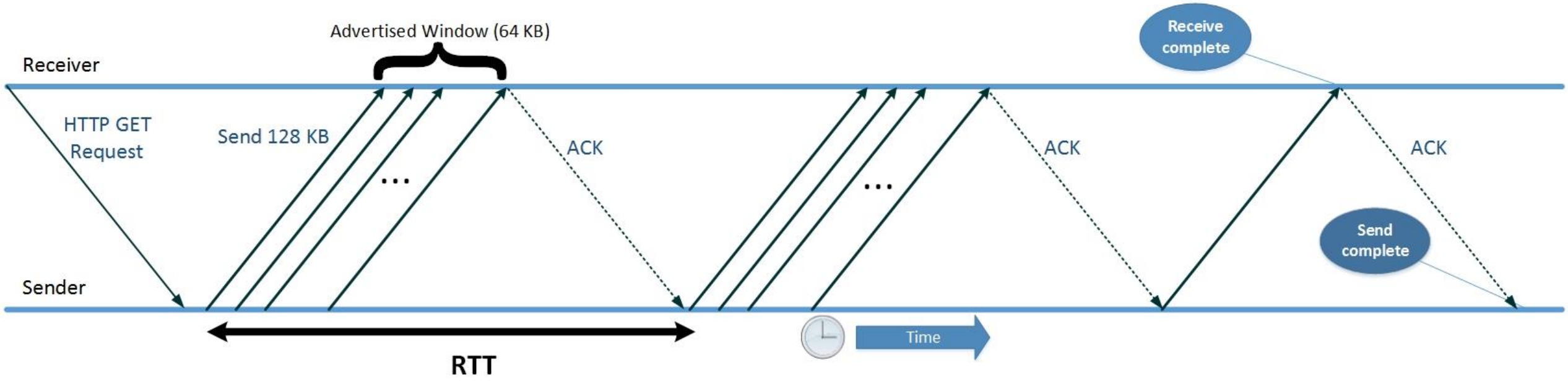
  - RTT for amazon.com = 250 ms
  - Max thruput =
    4 * 64 KB ≅ 256 KB




Max Thruput/connection = AdvertisedWindow / RTT

# TCP flow control

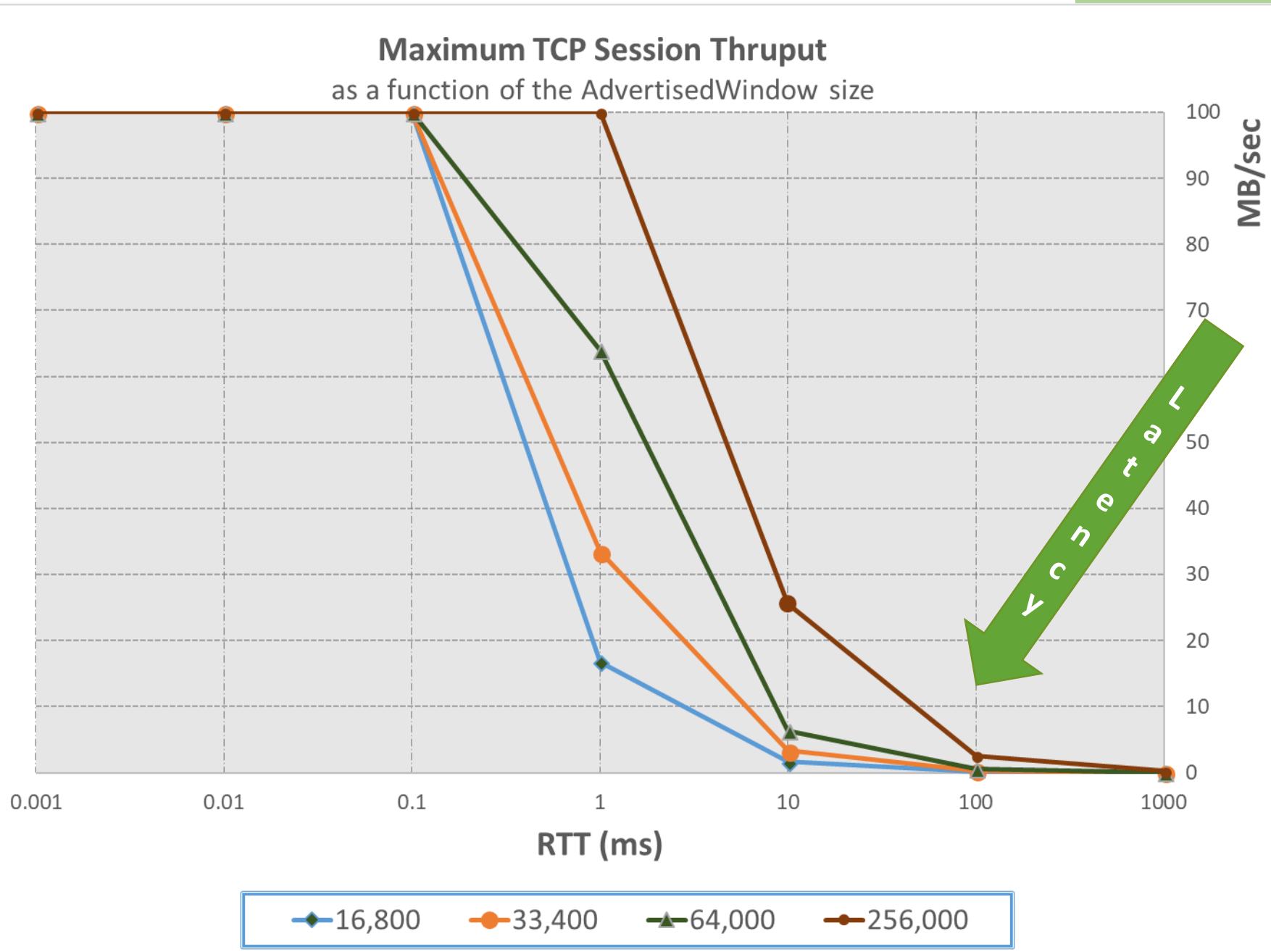
- **Example: Get Request for a 120 KB image file**



# Max Thruput/connection = AdvertisedWindow / RTT

RTT begins to constrain thruput

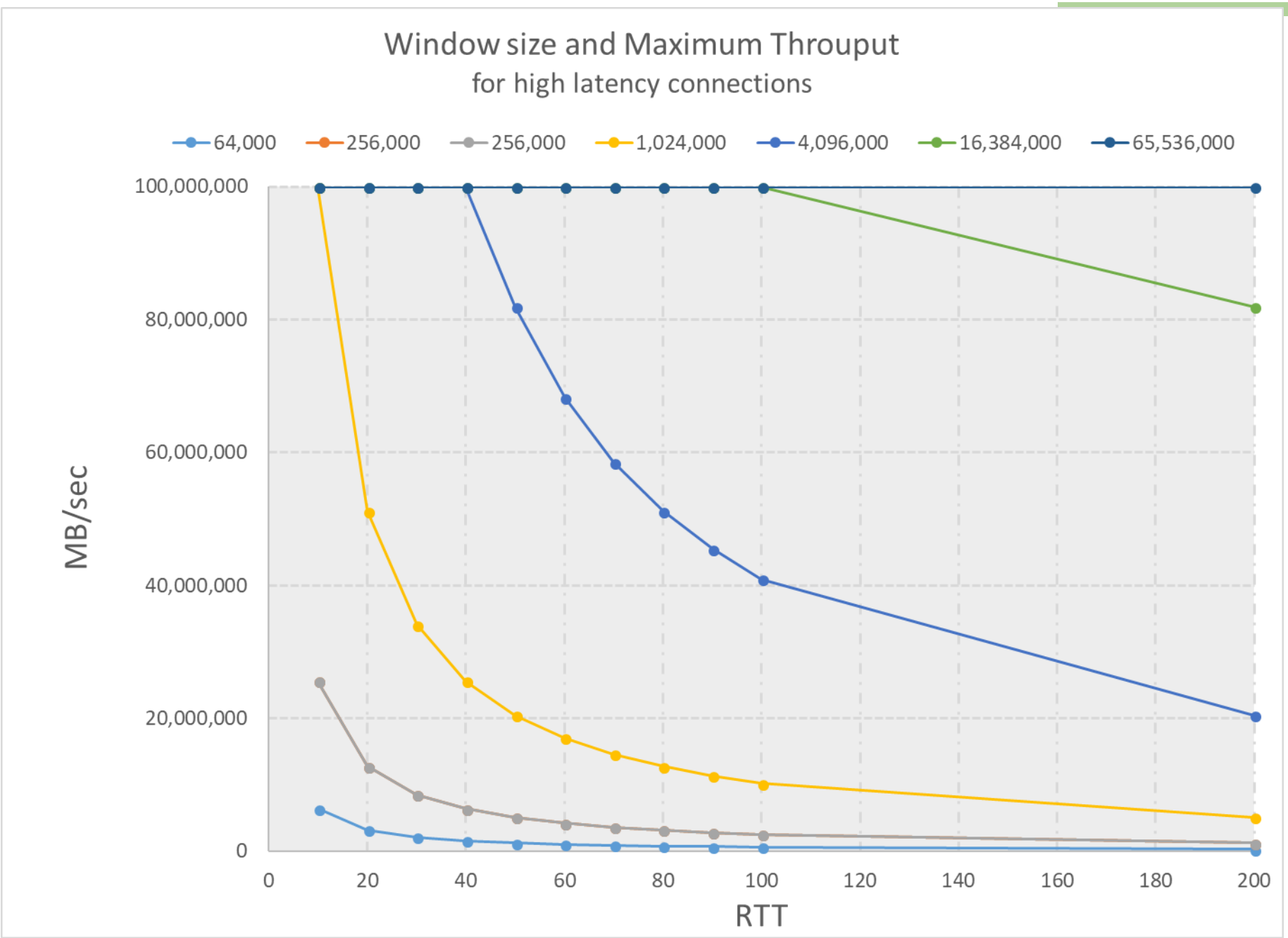
| RTT (ms) | Max Windows/sec | Max Throughput (bytes/sec) |
|---|---|---|
| | | 1000BaseT (~ 100 MB/sec) |
| | | 64 KB Window |
| 0.001 | 1,000,000 | 100,000,000 |
| 0.01 | 100,000 | 100,000,000 |
| 0.1 | 10,000 | 100,000,000 |
| 1 | 1,000 | 64,000,000 |
| 10 | 100 | 6,400,000 |
| 100 | 10 | 640,000 |
| 1000 | 1 | 64,000 |




# What about other Advertised Window* sizes?

- Scaling factor Option allows for larger windows

* Memory requirements are one reason why very large TCP ***AdvertisedWindows*** are not used very frequently




# What about other Advertised Window sizes?

```
Max Thruput =
AdvertisedWindow / RTT
```

**is actually a Best Case because**

$$Cwin \leq AdvertisedWin$$

# TCP Adaptive Retransmission

- Packet retransmission is the ***only*** error recovery mechanism available for TCP to use!
- ***How long*** should a TCP Sender wait for an unacknowledged packet before retransmitting it?

  Too short: increases network load unnecessarily

  Too long: increases network response time needlessly

# TCP Adaptive Retransmission

- TCP measures RTT at the connection level to figure out whether unacknowledged packets are lost
- Possible causes:
  - Sent packet was damaged during transmission
  - Sent packet was *dropped* by a busy router
  - ACK packet was damaged during transmission
  - ACK packet was *dropped* by a busy router
- Due to the reliability of modern networks, *dropped* is the most likely reason (with the exception of cellular/mobile transmissions)

# TCP Adaptive Retransmission

- Calculating Retransmission Timeout (**RTO**):
  - Initial RTT of the SYN-ACK exchange is used at the outset of a connection
    - Defaults to 3 seconds for initial connection hand-shaking
  - Subsequently, RTT is ***sampled*** at the start of each Window send,
  - Unless optional ***Timestamps*** are used: Sender returns the TimeStamp field it received in the ACK packet
  - Calculate a smoothed RTT, a moving average using ***Jacobson's*** or ***Karn's algorithm***

# TCP Adaptive Retransmission

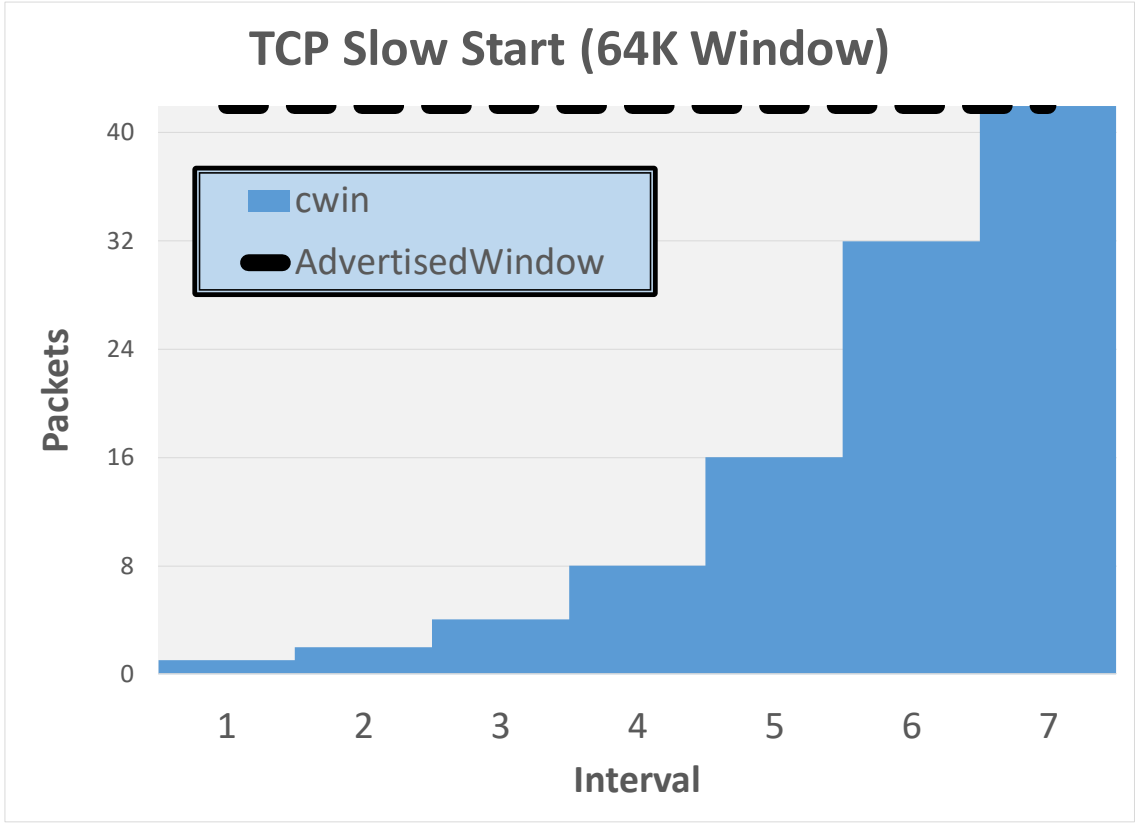
- Suppose packets are received out of order…
- **Fast Retransmit**
  - When an out-of-order packet is received, Receiver responds by sending a duplicate ACK.
  - As soon as Sender receives *TcpMaxDupAcks* duplicate ACKs, it retransmits unacknowledged packets immediately
  - If **SACK** enabled (a requirement for large AdvertisedWindows), only missing packets need be retransmitted.

# TCP Congestion Window (*cwin*)

- TCP Standard (other variants available, see, e.g., TCP Vegas)
  - Implemented the recommendations from a seminal paper by Jacobson and Karels, published in 1988.
  - Strategy is to use **ACK**s to pace the session: ***self-clocking***
  - ***Slow Start***
    - Start with one packet and then expand the *cwin* exponentially
  - TCP recognizes and adapts to ***congestion signals***
    - Window Full conditions
    - packet loss
  - ***Additive Increase/Multiplicative Decrease***

# TCP Congestion Window (*cwin*)

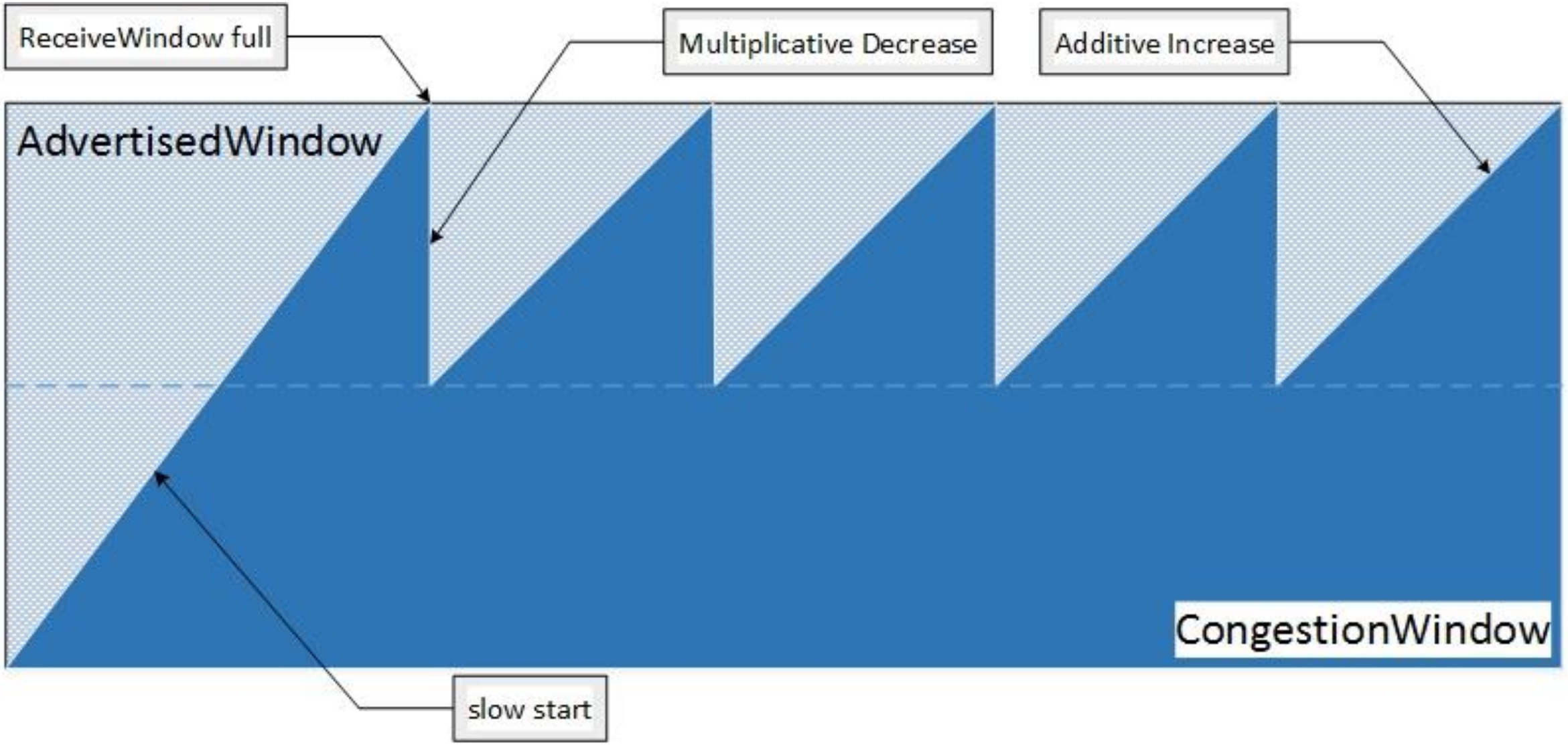
- ***Slow Start***
  - Grow the **CongestionWindow** exponentially until it equals the **AdvertisedWindow**
    - Initially, send 1, 2, 4, 8, then 16 packets



# TCP Congestion Window (*cwin*)

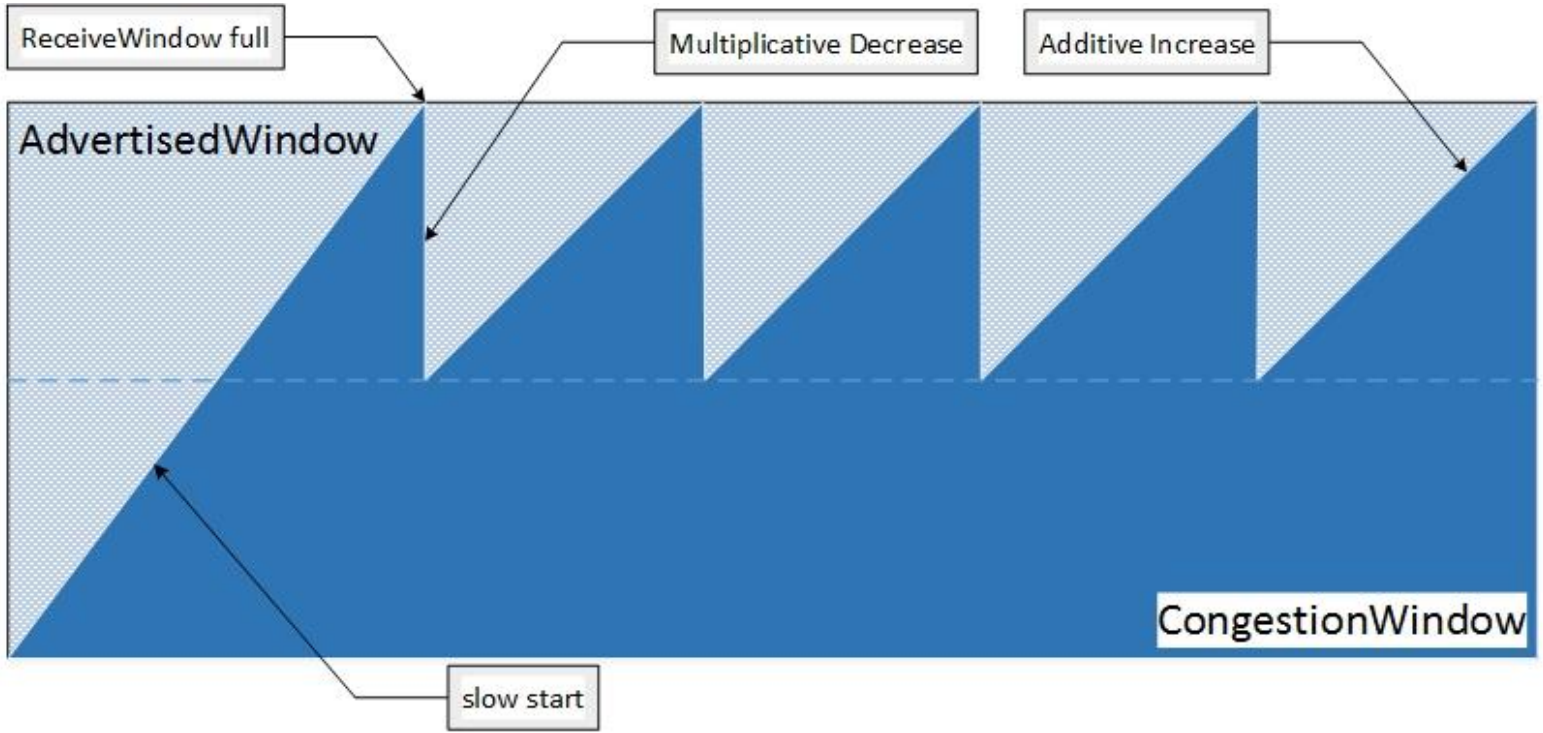
- ***Additive Increase/Multiplicative Decrease***
  - ***CongestionWindow*** = ***CongestionWindow*** / 2 when the ***AdvertisedWindow*** is full
  - ***CongestionWindow*** = ***CongestionWindow*** + 1 until ***CongestionWindow*** = ***AdvertisedWindow***
  - Return to Slow Start when RTO occurs!

# TCP Congestion Window (*cwin*)



# TCP Congestion Window (*cwin*)

- ***Additive Increase/Multiplicative Decrease***
  - **ROT: reduces effective capacity by ~25%**

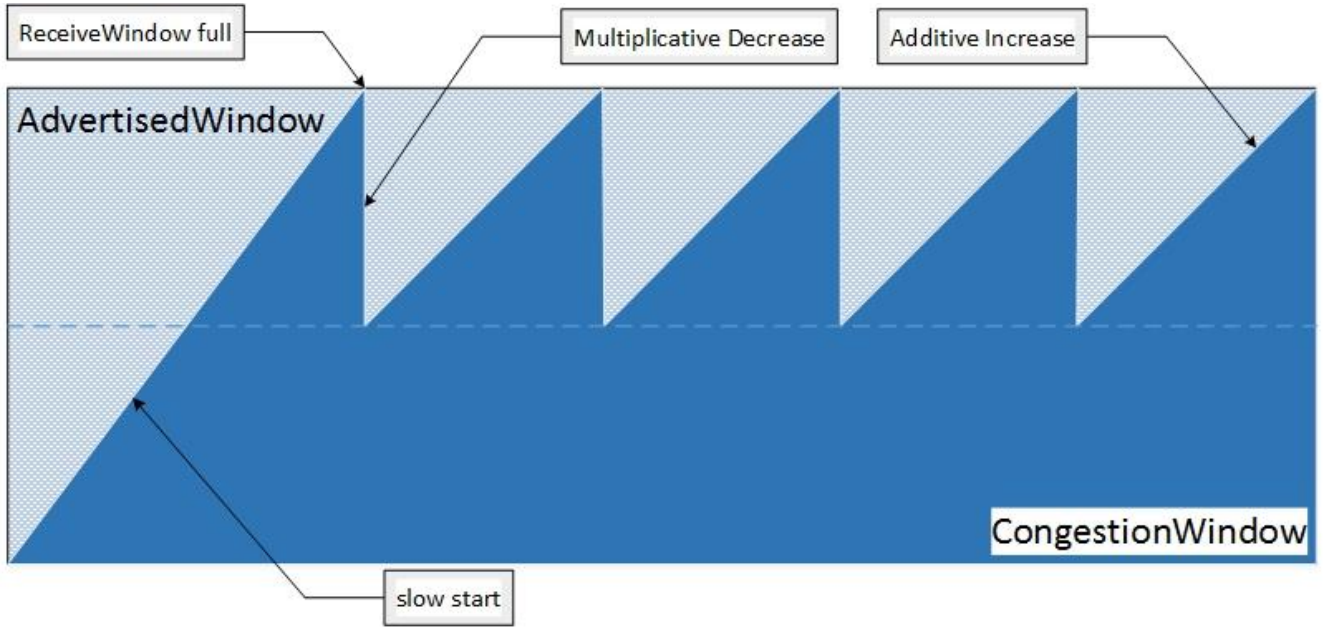


# TCP Congestion Window (*cwin*)

- ***Additive Increase/Multiplicative Decrease***

$$\textit{Effective TCP Bandwidth} \cong (\text{MSS}/\text{RTT}) * (1/\sqrt{p})$$

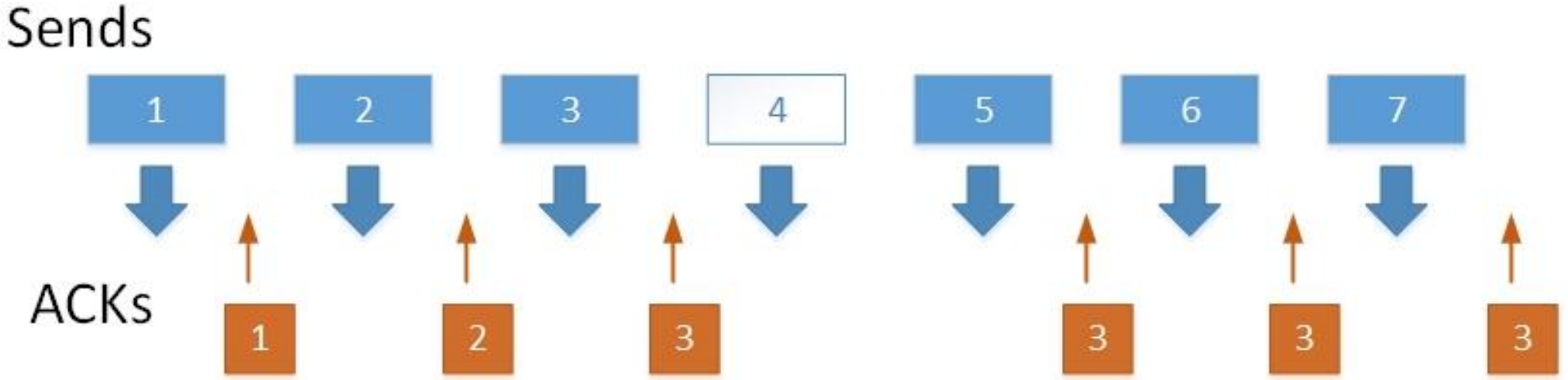
where $p$ is the number of congestion signals per acknowledged packet*




* See Mathis, Ott, et. al., "The macroscopic behavior of the TCP congestion avoidance algorithm", *Computer Communications Review*, July 1997

# TCP Congestion Window (*cwin*)

- **Duplicate ACKs and Fast Retransmit**




- **Three duplicate ACKs triggers immediate retransmission without waiting for the RTO timer to expire!**

# TCP Congestion Window (*cwin*)

- Known issues (standard TCP)
  - Poor utilization of expensive links by large bandwidth delay product (BDP) data flows
    - e.g., satellite links
  - When congestion occurs, sessions over long latency links are penalized disproportionately
  - Packet loss is incorrectly interpreted as a congestion signal on lossy networks
    - e.g., wireless

# TCP Congestion Window (*cwin*)

- Router participation
  - Random Early Detection/Drop (**RED**)
    - powerful routers on links with large delay-bandwidth products (BDP) maintain large Queues to accommodate bursty traffic
    - Current average queue length > threshold, the router will drop random packets to send an ***implicit*** congestion signal to Senders
  - Explicit Congestion Notification (**ECN**)
    - Router sets the ECN bits in the IP Type of Service (ToS) field, which the Receiver copies into its ACK packet

# TCP Congestion Window/Avoidance

- TCP Reno uses **Fast Recovery**
  - ***ssthresh*** determines whether to use **Slow Start** or less aggressive **Congestion Avoidance** following a congestion signal

- Protocol stack increasingly complex
  - see **Set-NetTCPSetting** options in Windows
    - e.g., `AutoTuningLevelLocal`

Trace Event Sequence View

**Trace Event Sequence** Events Displayed: 2169 Total Events: 2169

| Number | Time Stamp | Delay | Protocol | Event Type | Source Addr | Port | Dest Addr | Port | SeqNo | Ack | Bytes | Resp Time (ms) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2119 | 22:36:29.3097898 | 00:00:00.0000496 | TCP/IPv4 | TcpSendTransmitted | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090448 | 3711584863 | 108 | 0 |
| 2122 | 22:36:29.3098737 | 00:00:00.0000839 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090556 | 3711584943 | 80 | 0 |
| 2124 | 22:36:29.3099296 | 00:00:00.0000559 | TCP/IPv4 | TcpSendTransmitted | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090556 | 3711584943 | 108 | 0 |
| 2127 | 22:36:29.3100123 | 00:00:00.0000827 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 100 | 0 |
| 2130 | 22:36:29.3100446 | 00:00:00.0000323 | TCP/IPv4 | TcpSendTransmitted | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 117 | 0 |
| 2131 | 22:36:29.3250281 | 00:00:00.0149835 | TCP/IPv4 | TcpTimerExpired | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 0 | 0 |
| 2132 | 22:36:29.3250289 | 00:00:00.0000008 | TCP/IPv4 | TcpTimerExpired | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 0 | 0 |
| 2133 | 22:36:29.3250313 | 00:00:00.0000024 | TCP/IPv4 | TcpDataTransferRetransmit | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 0 | 0 | 0 | 0 |
| 2134 | 22:36:29.3250321 | 00:00:00.0000008 | TCP/IPv4 | TcpDataTransferTimeout | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 0 | 0 |
| 2135 | 22:36:29.3250352 | 00:00:00.0000031 | TCP/IPv4 | TcpSendTransmitted | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 117 | 0 |
| 2136 | 22:36:29.3250970 | 00:00:00.0000618 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090664 | 3711585043 | 0 | 0 |
| 2140 | 22:36:29.3267240 | 00:00:00.0016270 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2141 | 22:36:29.3267347 | 00:00:00.0000107 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2142 | 22:36:29.3267492 | 00:00:00.0000145 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2143 | 22:36:29.3267579 | 00:00:00.0000087 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2144 | 22:36:29.3268367 | 00:00:00.0000788 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2145 | 22:36:29.3268477 | 00:00:00.0000110 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2146 | 22:36:29.3268611 | 00:00:00.0000134 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2147 | 22:36:29.3268697 | 00:00:00.0000086 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2148 | 22:36:29.3268823 | 00:00:00.0000126 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2149 | 22:36:29.3269347 | 00:00:00.0000524 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2150 | 22:36:29.3269465 | 00:00:00.0000118 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2151 | 22:36:29.3269686 | 00:00:00.0000221 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2152 | 22:36:29.3269808 | 00:00:00.0000122 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2153 | 22:36:29.3270229 | 00:00:00.0000421 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2154 | 22:36:29.3270352 | 00:00:00.0000123 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2155 | 22:36:29.3270698 | 00:00:00.0000346 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2156 | 22:36:29.3270726 | 00:00:00.0000028 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2157 | 22:36:29.3270856 | 00:00:00.0000130 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2158 | 22:36:29.3271076 | 00:00:00.0000220 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2159 | 22:36:29.3271171 | 00:00:00.0000095 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |
| 2160 | 22:36:29.3271553 | 00:00:00.0000382 | TCP/IPv4 | TcpReceiveData | 10.50.0.25 | 49161 | 10.50.0.24 | 445 | 1892090781 | 3711585127 | 1460 | 0 |

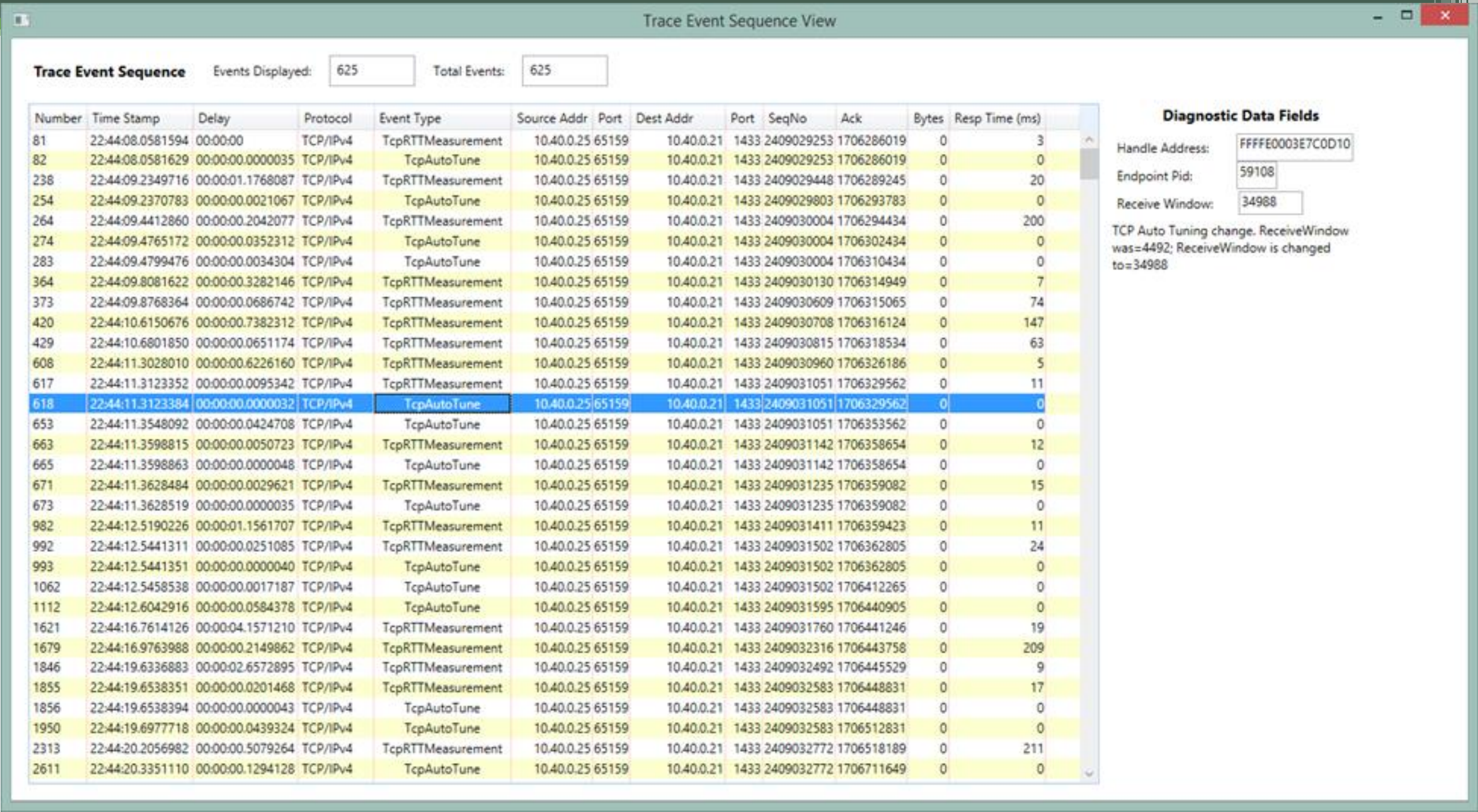
**Diagnostic Data Fields**

Handle Address: FFFFE0003CED09F0

Endpoint Pid: 4

Receive Window: 0

Data Transfer Timeout updates CongestionWindow=1460 and Slow Start Threshold=3060


Trace Event Sequence View
Trace Event Sequence
Events Displayed: 625
Total Events: 625
Number Time Stamp Delay Protocol Event Type Source Addr Port Dest Addr Port SeqNo Ack Bytes Resp Time (ms)
81 22:44:08.0581594 00:00:00 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409029253 1706286019 0 3
82 22:44:08.0581629 00:00:00.0000035 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409029253 1706286019 0 0
238 22:44:09.2349716 00:00:01.1768087 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409029448 1706289245 0 20
254 22:44:09.2370783 00:00:00.0021067 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409029803 1706293783 0 0
264 22:44:09.4412860 00:00:00.2042077 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030004 1706294434 0 200
274 22:44:09.4765172 00:00:00.0352312 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409030004 1706302434 0 0
283 22:44:09.4799476 00:00:00.0034304 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409030004 1706310434 0 0
364 22:44:09.8081622 00:00:00.3282146 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030130 1706314949 0 7
373 22:44:09.8768364 00:00:00.0686742 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030609 1706315065 0 74
420 22:44:10.6150676 00:00:00.7382312 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030708 1706316124 0 147
429 22:44:10.6801850 00:00:00.0651174 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030815 1706318534 0 63
608 22:44:11.3028010 00:00:00.6226160 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409030960 1706326186 0 5
617 22:44:11.3123352 00:00:00.0095342 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031051 1706329562 0 11
618 22:44:11.3123384 00:00:00.0000032 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031051 1706329562 0 0
653 22:44:11.3548092 00:00:00.0424708 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031051 1706353562 0 0
663 22:44:11.3598815 00:00:00.0050723 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031142 1706358654 0 12
665 22:44:11.3598863 00:00:00.0000048 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031142 1706358654 0 0
671 22:44:11.3628484 00:00:00.0029621 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031235 1706359082 0 15
673 22:44:11.3628519 00:00:00.0000035 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031235 1706359082 0 0
982 22:44:12.5190226 00:00:01.1561707 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031411 1706359423 0 11
992 22:44:12.5441311 00:00:00.0251085 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031502 1706362805 0 24
993 22:44:12.5441351 00:00:00.0000040 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031502 1706362805 0 0
1062 22:44:12.5458538 00:00:00.0017187 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031502 1706412265 0 0
1112 22:44:12.6042916 00:00:00.0584378 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409031595 1706440905 0 0
1621 22:44:16.7614126 00:00:04.1571210 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409031760 1706441246 0 19
1679 22:44:16.9763988 00:00:00.2149862 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409032316 1706443758 0 209
1846 22:44:19.6336883 00:00:02.6572895 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409032492 1706445529 0 9
1855 22:44:19.6538351 00:00:00.0201468 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409032583 1706448831 0 17
1856 22:44:19.6538394 00:00:00.0000043 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409032583 1706448831 0 0
1950 22:44:19.6977718 00:00:00.0439324 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409032583 1706512831 0 0
2313 22:44:20.2056982 00:00:00.5079264 TCP/IPv4 TcpRTTMeasurement 10.40.0.25 65159 10.40.0.21 1433 2409032772 1706518189 0 211
2611 22:44:20.3351110 00:00:00.1294128 TCP/IPv4 TcpAutoTune 10.40.0.25 65159 10.40.0.21 1433 2409032772 1706711649 0 0
Diagnostic Data Fields
Handle Address: FFFFE0003E7C0D10
Endpoint Pid: 59108
Receive Window: 34988
TCP Auto Tuning change. ReceiveWindow was=4492; ReceiveWindow is changed to=34988

# TCP Overhead

- Maintaining session state is resource-intensive
  - cwin calculations
  - RTO timers
  - buffer pools for large AdvertisedWindows
  - interactions with parallel sessions in HTTP

- How resource intensive?

# TCP Overhead

- Host processing requirements is a long standing problem for high-speed networking

- Concentrate all interrupts from a device on **a single host CPU**
  - cache warm start & locality
- MTU for Ethernet packets is 1500 bytes

**Assume:**

- average segment size ≅ 1K bytes
- Sends = Receives
- 10K cycles per interrupt

| Interface | Interrupts/sec | Cycles |
|---|---|---|
| 10 Mb | 500 | 5 MHz |
| 100 Mb | 5K | 50 MHz/sec |
| 1 Gb | 50K | 500 MHz/sec |
| 10 Gb | 500K | 5 GHz/sec |

# TCP Overhead

- **NTttcp** testing tool
- Environment:
  - 2.2 GHz dual core processor
  - 1 Gb Ethernet adaptor on a dedicated network segment
  - Windows 6 SP1
    - (Vista and WS 2K8)
  - 512-byte packets

| | |
|---|---|
| Throughput(KB/s) | 16,475.553 |
| Throughput(Mbit/s) | 131.804 |
| Average Frame Size | 764.394 |
| Packets Sent | 1,309,892 |
| Packets Received | 2,586,923 |
| Packets received/Int | 2 |
| Interrupts/sec | 9,494.04 |
| Cycles/Byte | 129.3 |

*2.2 GHz/sec ÷ 9500 interrupts/sec ≈* ***200,000 clocks/interrupt***

*500,000 interrupts/sec * 200,000 cycles ≈ 100,000,000,000 cycles/second*

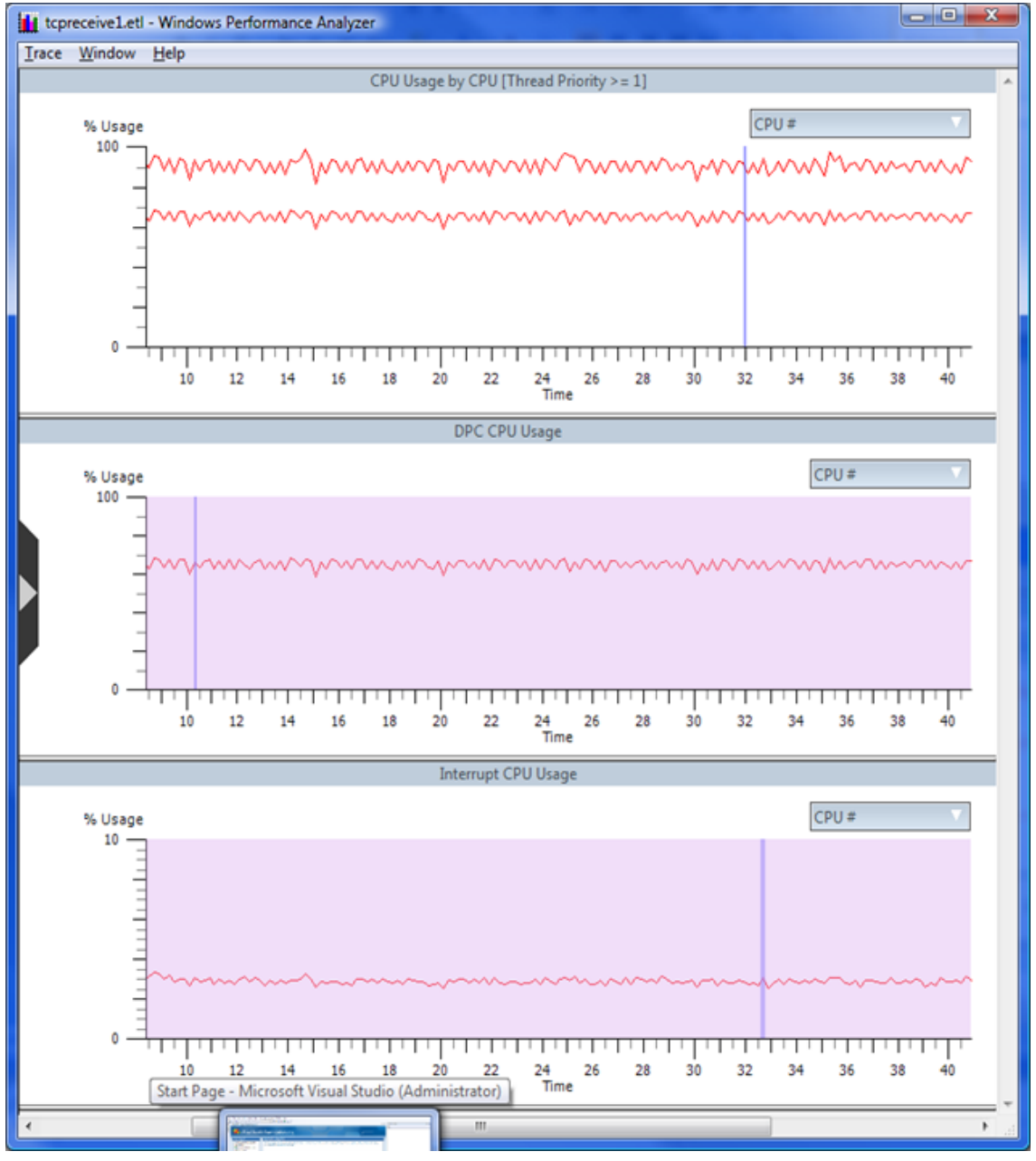
# TCP Overhead

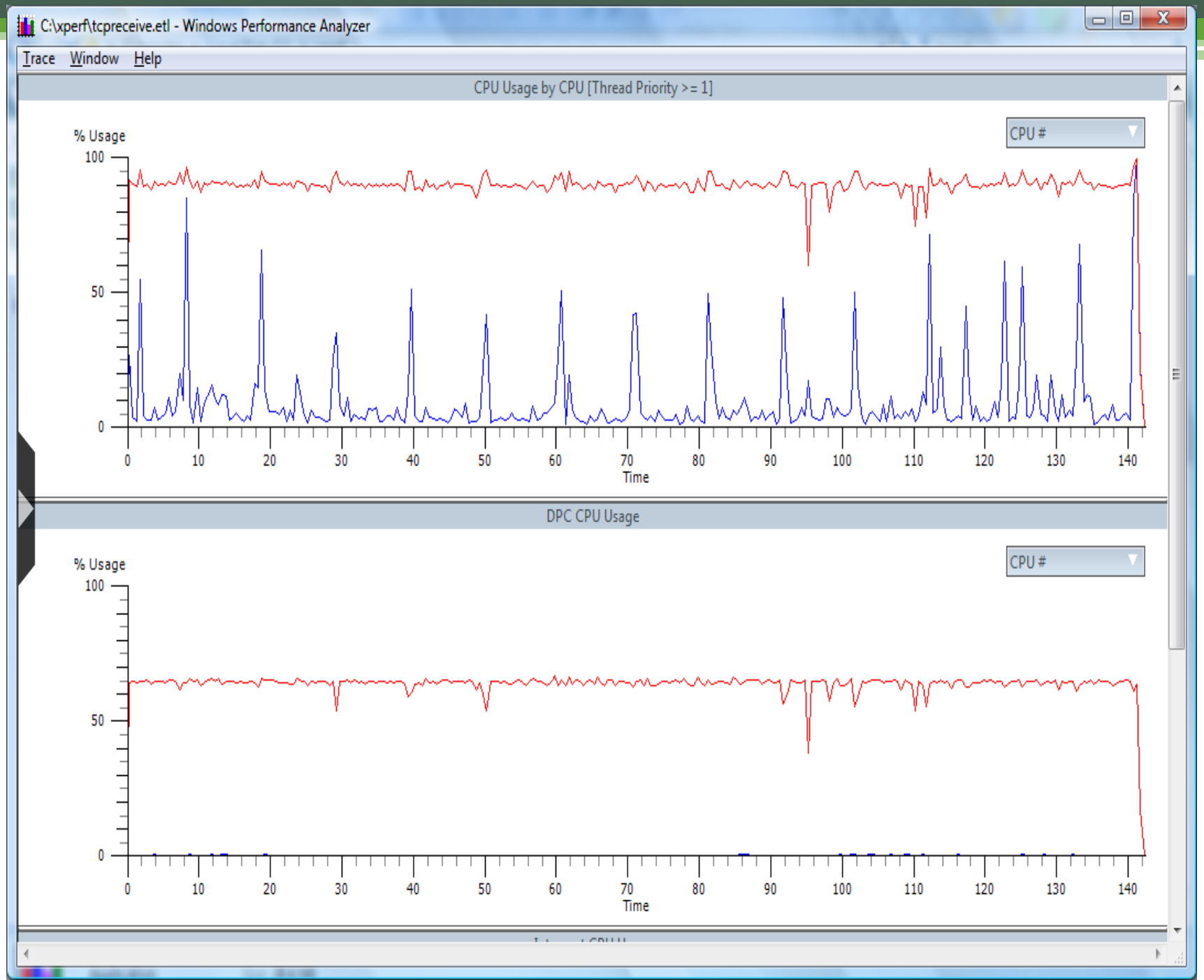
- Repeat the experiment, collecting Trace data:

```
xperf -on LATENCY
      -f tcpreceive1.etl
      -ClockType Cycle
```

**LATENCY** flags request trace data:

- CPU context switches
- interrupts
  - Interrupt Service Routines (ISRs)
    Deferred Procedure calls (DPCs)




C:\xperf\tcpreceive.etl - Windows Performance Analyzer
Trace
Window
Help
CPU Usage by CPU [Thread Priority >= 1]
% Usage
CPU #
100
50
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Time
DPC CPU Usage
% Usage
CPU #
100
50
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Time

# TCP Overhead

## xperfview Summary Table display showing processor utilization by DPC.

| Module | Function | Count | Max Duration [ms] | Avg Duration [ms] | Duration [ms] |
|---|---|---|---|---|---|
| ndis.sys | | 425125 | 1.116595 | 0.075376 | 32044.44967 |
| | 0x8ac79237 | 423800 | 0.797987 | 0.075552 | 32019.18583 |
| | 0x8ad38209 | 207 | 1.116595 | 0.11752 | 24.326752 |
| | 0x8ad3892f | 1117 | 0.012439 | 0.000837 | 0.935867 |
| | 0x8ad399b3 | 1 | 0.001213 | 0.001213 | 0.001213 |
| USBPORT.SYS | | 8312 | 0.064506 | 0.011802 | 98.100399 |
| tcpip.sys | | 4154 | 0.551394 | 0.009585 | 39.817004 |
| dxgkrnl.sys | 0x8f34e09b | 3039 | 0.528346 | 0.012848 | 39.047187 |
| iastor.sys | | 1221 | 0.033546 | 0.015061 | 18.390545 |

# TCP Overhead

## xperfview Summary Table display showing processor utilization by ISR.

| Module | Function | Count | Max Duration [ms] | Avg Duration [ms] | Duration [ms] |
|---|---|---|---|---|---|
| ntkrnlpa.exe | 0x828d6fa2 | 423803 | 0.023875 | 0.003173 | 1345.147547 |
| dxgkrnl.sys | 0x8f3630ea | 3040 | 0.096759 | 0.039523 | 120.151742 |
| USBPORT.SYS | 0x8f4098c2 | 5529 | 0.025199 | 0.007241 | 40.038229 |
| pcmcia.sys | 0x82f8deea | 4968 | 0.02401 | 0.006754 | 33.558272 |
| iastor.sys | 0x8aaa7f6c | 4968 | 0.016345 | 0.005103 | 25.353482 |

**Total Interrupt Time ≈ 80 µsecs**

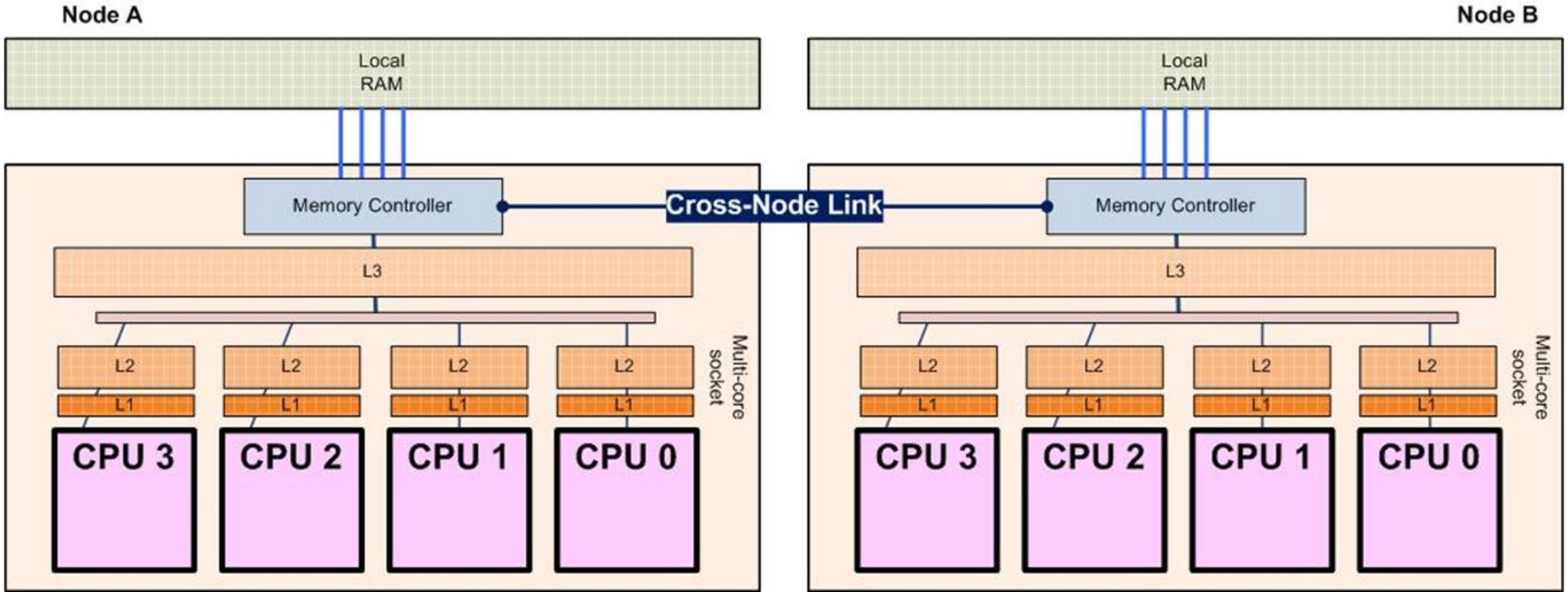
*500,000 interrupts/sec * 175,000 cycles = 88 GHz*

# TCP Overhead

- TCP Interrupt processing can scale (theoretically) by using multiple processors to minimize pending Interrupt delays
- Concentrating Interrupt processing on fewer processors boosts CPU efficiency by maximizing cache warm starts
- What are the potential NUMA impacts when the application layer processes the packet on a different node than the DPC/ISR?

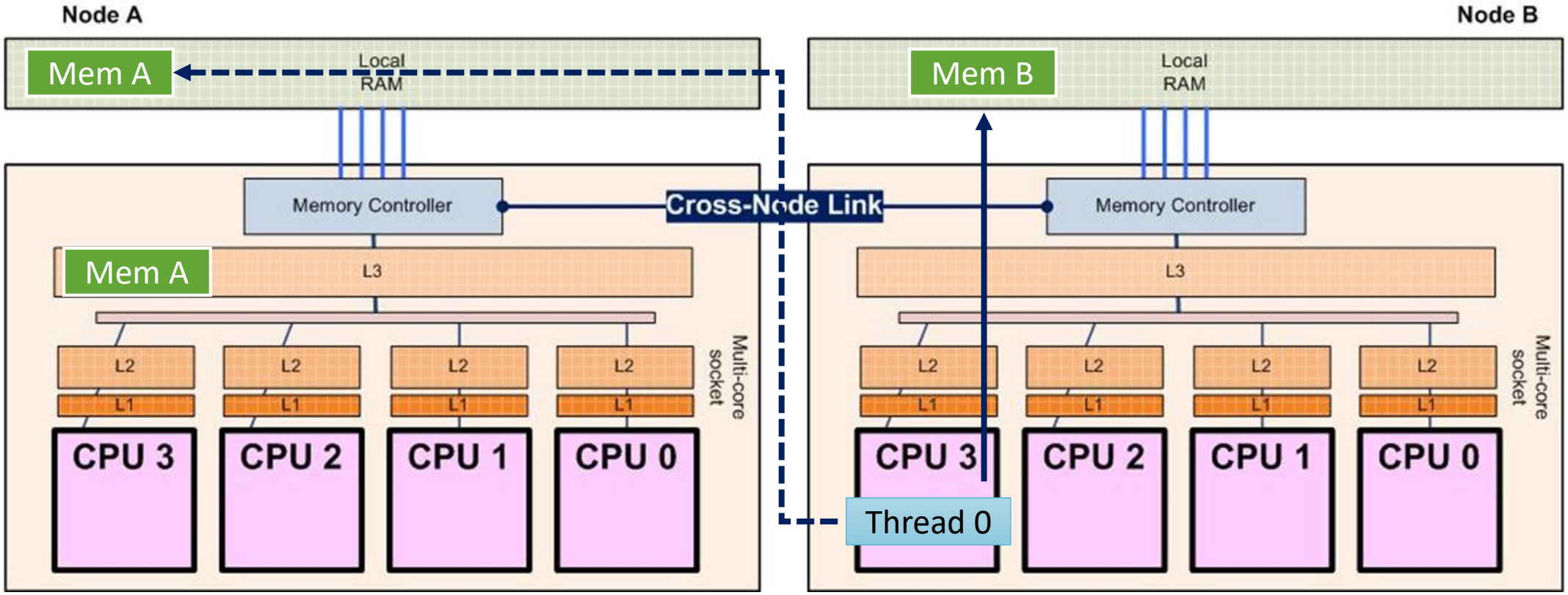
# TCP Overhead

- What are the potential NUMA impacts when the application layer processes the packet on a different node than the DPC/ISR?



# TCP Overhead

- What are the potential NUMA impacts when the application layer processes the packet on a different node than the DPC/ISR?



# References


- Jacobson and Karels, "Congestion avoidance and control," August 1988. Available here.
- Floyd and Jacobson, "Random early detection gateways for congestion avoidance," August 1993. Available here.
- Friedman, "Mainstream NUMA and the TCP/IP stack", Dec. 2010.

- Peterson and Davis, *Computer Networks: a systems approach*, 2011
- Hassan and Jain, *High Performance TCP/IP Networking*, 2003.
- Ross and Kurose, *Computer Networking: a Top-Down approach*, 2016.