CPE348: Introduction to Computer Networks

Lecture #16: Chapter 5.2

Jianqing Liu
Assistant Professor of Electrical and Computer Engineering, University of Alabama in Huntsville

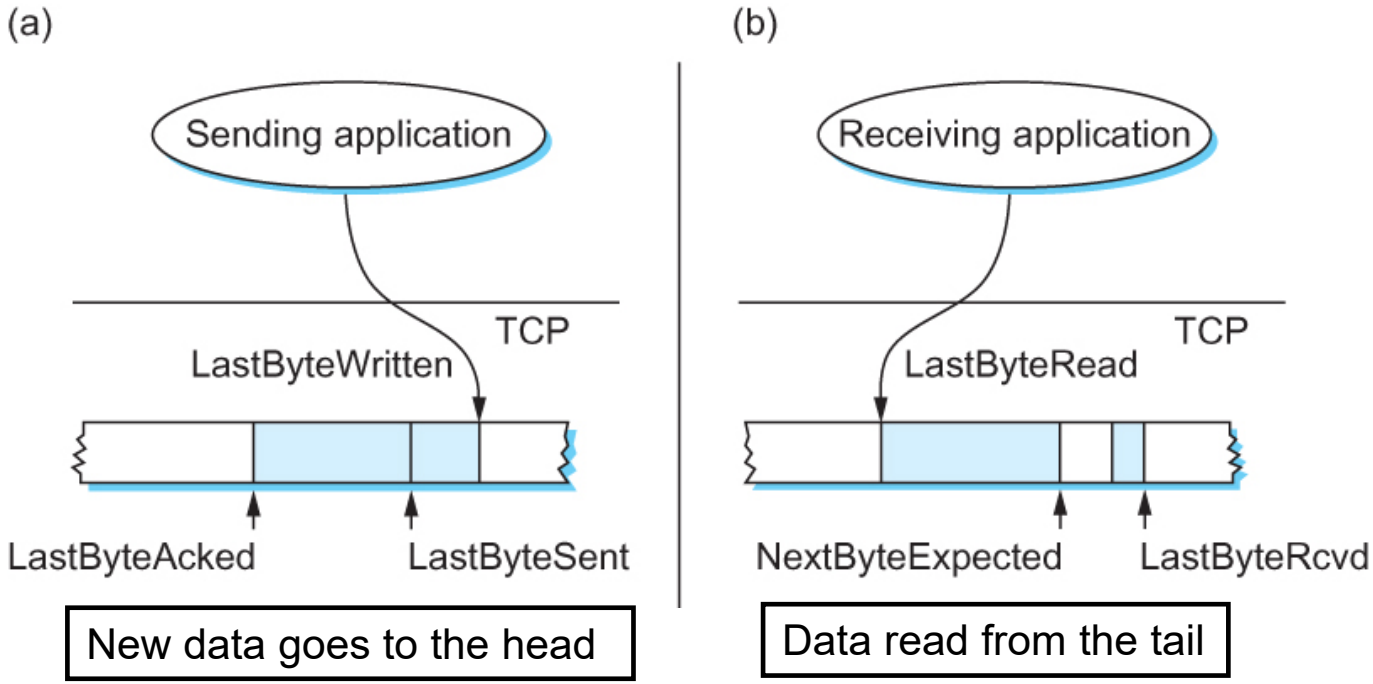
jianqing.liu@uah.edu
http://jianqingliu.net

Sliding Window Revisited

- TCP's variant of the sliding window algorithm, which serves several purposes:
  - (1) it guarantees the reliable delivery of data,
  - (2) it ensures that data is delivered in order, and
  - (3) it enforces flow control between the sender and the receiver.

- Window size is not fixed
  - Receiver advertises a window size to the sender

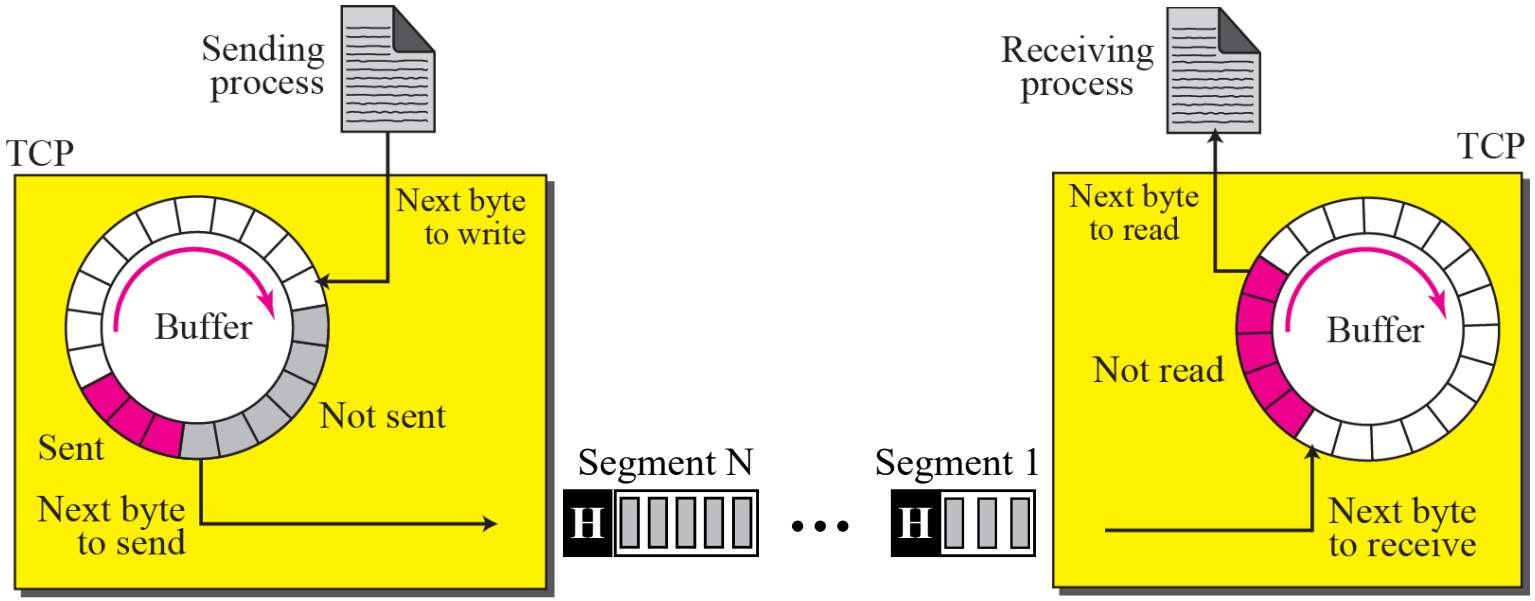
Sliding Window Revisited



Relationship between TCP send buffer (a) and receive buffer (b).

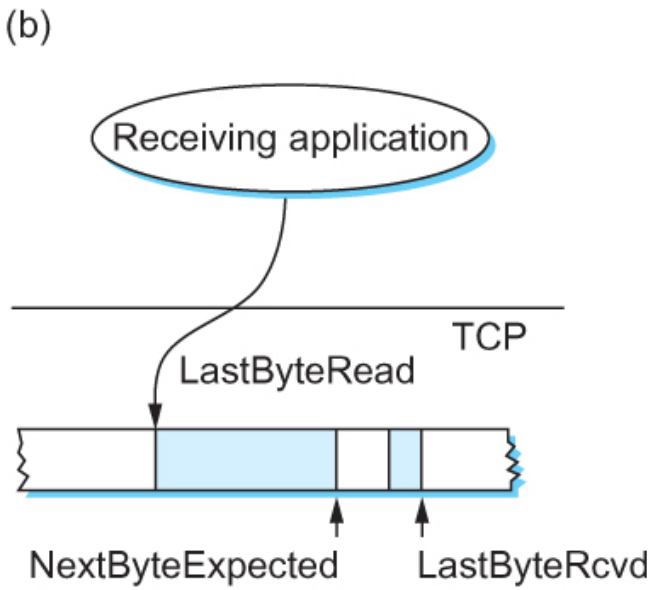
Sliding Window Revisited

- Another angle to look at the TCP sliding window protocol



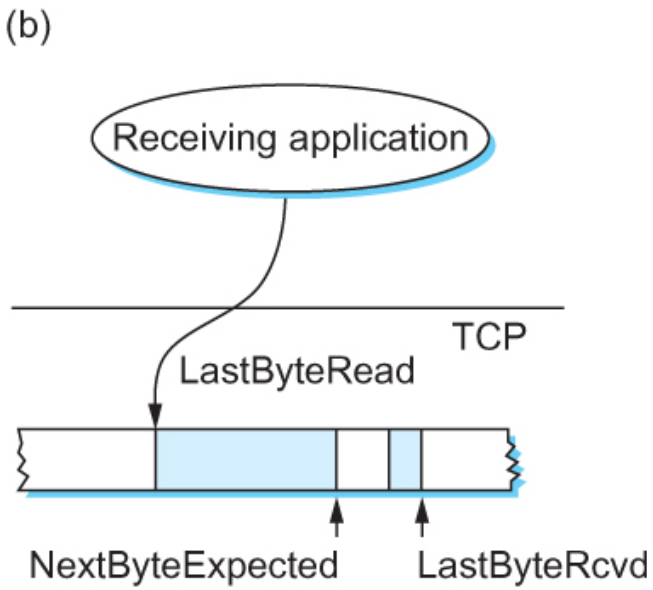
TCP Sliding Window – Rcvr

- Three pointers into receive buffer
  - LastByteRead, NextByteExpected and LastByteRcvd
  - LastByteRead < NextByteExpected
  - NextByteExpected ≤ LastByteRcvd + 1
  - NextByteExpected points to next byte to be received
  - Bytes to left of LastByteRead are dropped from buffer



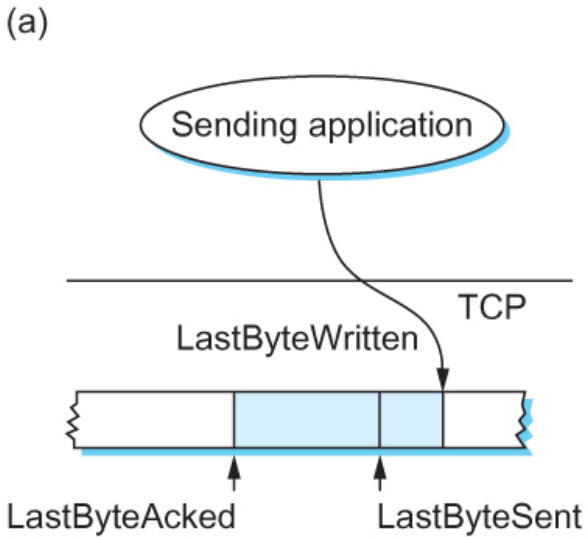
TCP Sliding Window – Rcvr

- Three pointers into receive buffer
  - LastByteRcvd − LastByteRead ≤ MaxRcvBuffer
  - AdvertisedWindow = MaxRcvBuffer − ((NextByteExpected − 1) − LastByteRead)
  - As data is received, the advertised window decreases
  - As data is read by the process, the advertised window increases



TCP Sliding Window – Tx

- Three pointers into sending buffer
  - LastByteAcked, LastByteSent and LastByteWritten
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
  - Acked bytes of data are dropped out of the buffer

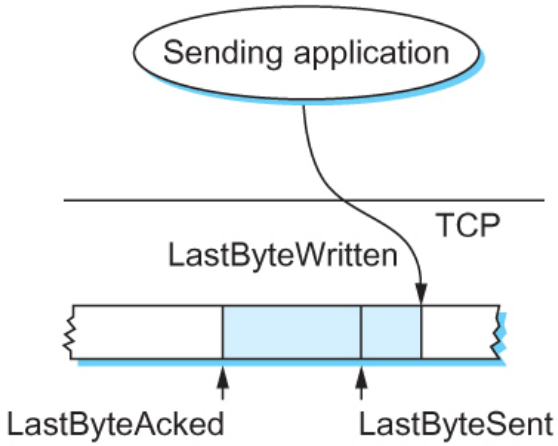


TCP Sliding Window – Tx

- Three pointers into sending buffer
  - LastByteSent − LastByteAcked ≤ AdvertisedWindow
  - LastByteWritten − LastByteAcked ≤ MaxSendBuffer
  - EffectiveWindow = AdvertisedWindow − (LastByteSent − LastByteAcked)
  - If the sending process tries to write y bytes to TCP, but (LastByteWritten − LastByteAcked) + y > MaxSendBuffer then TCP blocks the sending process and does not allow it to generate more data.

If advertised window drops to zero, the sender periodically sends 1 byte segments until a non-zero advertised window size is returned in the ack

(a)



TCP Flow Control

Recall what flow control is.

AdvertisedWindow is for flow control purpose!

If AdvertisedWindow → 0, the sender periodically sends 1 byte segments until a non-zero AdvertisedWindow size is returned in the Ack

TCP Flow Control

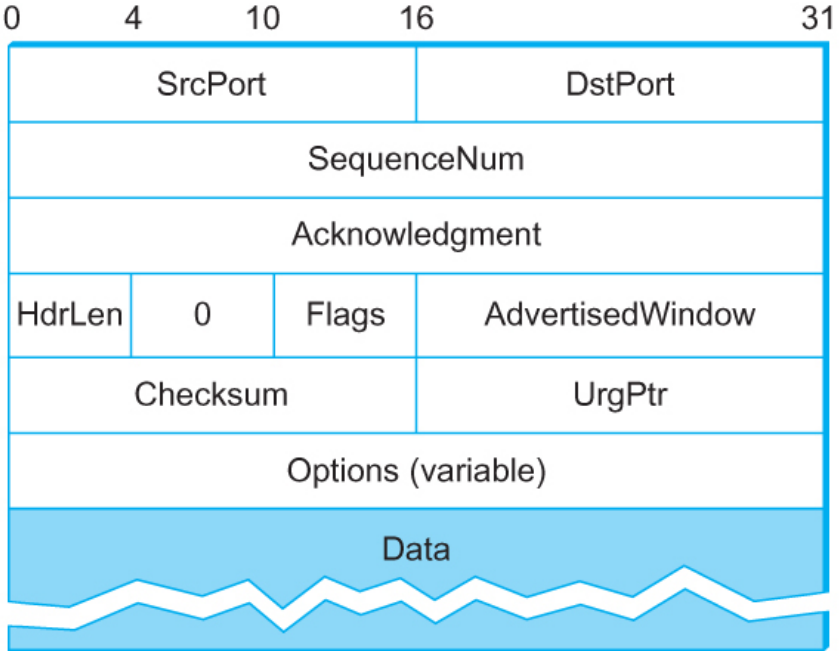
Recall the sliding window protocol on Layer 2.

What is the **relationship** between SWS/RWS and the Sequence Number?

TCP – Sequence Number

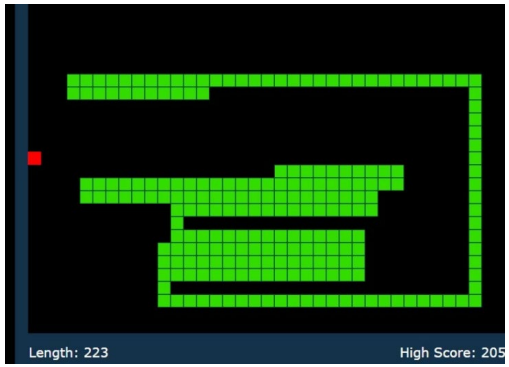
- SequenceNum: 32 bits longs
- AdvertisedWindow: 16 bits long
- TCP satisfies the sliding window algorithm requirement that the the sequence number space be twice as big as the window size

$2^{32} >> 2 \times 2^{16} = 2^{17}$



TCP – Sequence Number

- TCP has 32-bit sequence number space
  - The sequence number used on a given connection might wraparound.
  - How much time does it take for TCP SN to wraparound? Depending on the data rate!



TCP – Sequence Number 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

Time until 32-bit sequence number space wraps around.

(2^{32} Bytes = 2^{35} bits = 34.3597 Gbits)

TCP - Keeping the Pipe Full

- 16-bit AdvertisedWindow field
  - Theoretically allows for at most 64KB data
- What does delay x bandwidth tell us?

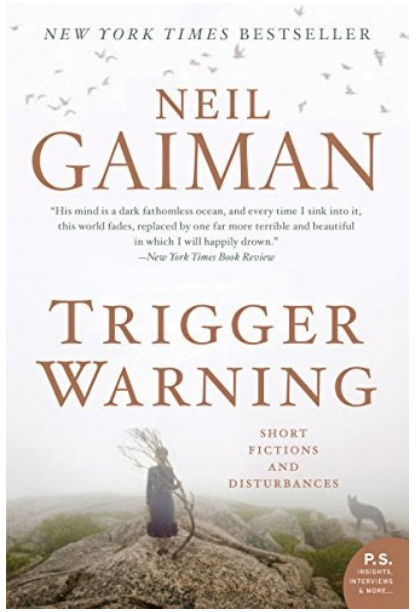
Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

Required window size for 100-ms RTT.

Extensions to TCP allow for the advertised window to indicate X number of bytes

TCP - Triggering Transmission

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



TCP - Triggering Transmission

- What factors governs the decision to transmit
  - TCP has three mechanism to trigger the transmission of a segment
    - 1) TCP maintains a variable for Maximum Segment Size (MSS)
      - TCP sends a segment as soon as it has collected MSS bytes from the sending process
      - MSS is usually set to the size of the largest segment TCP can send without causing local IP to fragment.
      - MSS: use MTU of directly connected network

        MSS = MTU - (TCP header + IP header)
    - 2) Sending process has explicitly asked TCP to send it
      - TCP supports push operation
    - 3) When a timer fires
      - Resulting segment contains as many bytes as are currently buffered for transmission

Silly Window Syndrome

Silly Window Syndrome

Silly Window Syndrome

- Sender Aggressively fills any available advertised window size
- If segment size sent is smaller than MSS, that smaller size remains
  - Receiver acks the smaller size
  - Receiver opens up a small window size after a 0 advertised window size
- Only a problem when sender sends a small segment or receiver opens a small window.

Nagle's Algorithm

- If there is data to send but the window is open less than MSS, wait!
- But how long?
  - If we wait too long, long delay!
  - If we wait too short, sending a bunch of tiny packets → waste of resources
- The solution is to introduce a timer and to transmit when the timer expires

Nagle's Algorithm

- We could use a clock-based timer;
- Nagle introduced an elegant self-clocking solution
- Key Idea
  - As long as TCP has any data in flight, the sender will eventually receive an ACK
  - This ACK can be treated like a timer firing, triggering the transmission of more data

Nagle's Algorithm

```
When the application produces data to send
    if both the available data and the window ≥ MSS
        send a full segment
    else
        if there is unACKed data in flight
                buffer the new data until an ACK arrives
        else
                send all the new data now
```

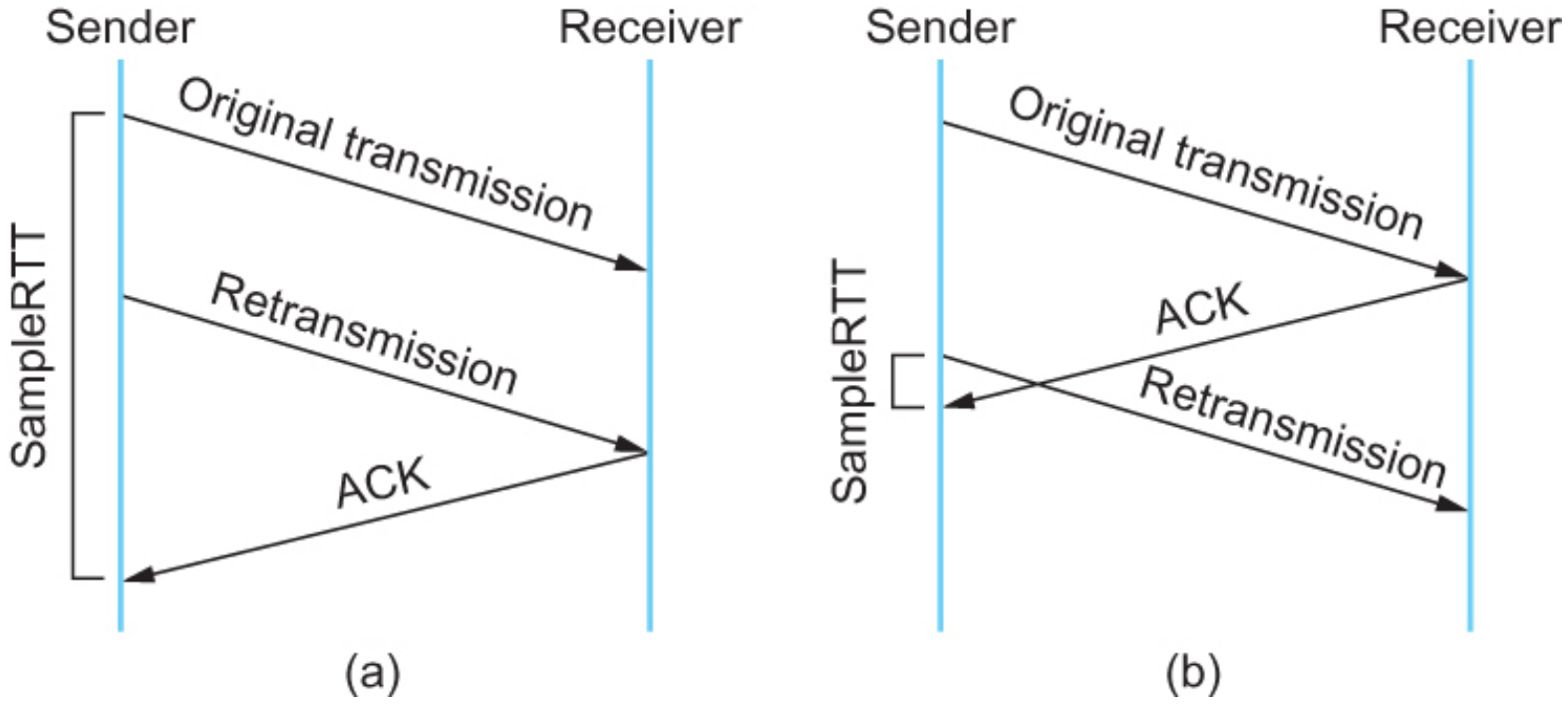
TCP - Adaptive Retransmission

- TCP retransmits a segment if an ACK is not received before a timeout occurs
- Timeout is based on RTT
- RTT can vary, so an adaptive retransmission mechanism is used.

TCP - Adaptive Retransmission

- Original Algorithm
  - Measure **`SampleRTT`** for each segment/ACK pair
  - Compute weighted average of RTT
    - **`EstRTT`** = α x **`EstRTT`** + (1 - α)x **`SampleRTT`**
    - α between 0.8 and 0.9
  - Set timeout based on **`EstRTT`**
    - **`TimeOut`** = 2 x **`EstRTT`**

TCP - Adaptive Retransmission –Problem



Sample RTT too large

Sample RTT too small

Associating the ACK with (a) original transmission versus (b) retransmission

Karn/Partridge Algorithm

- Measure sample RTT for segments sent only once
- Do not sample RTT for retransmitted segments
- Double timeout after each retransmission – exponential backoff

Jacobson/Karels Algorithm

- Main problem with the original computation is that it does not take variance of Sample RTTs into consideration.
- If the variance among Sample RTTs is small then
  - Estimated RTT can be better trusted
  - No need to multiply this by 2 to compute the timeout

Jacobson/Karels Algorithm

Difference = SampleRTT − EstimatedRTT

EstimatedRTT = EstimatedRTT + (δ× Difference)

Deviation = Deviation + δ (|Difference| − Deviation)

TimeOut = μ × EstimatedRTT + (Φ × Deviation)

- where based on experience, μ is typically set to 1
- Φ is set to 4, and δ is a fraction between 0 and 1
- Thus, when the variance is small, TimeOut is close to EstimatedRTT; a large variance causes the deviation term to dominate the calculation.