

Advanced Networking

TCP – Part II

Renato Lo Cigno
Renato.LoCigno@dit.unitn.it

Silly Window Syndrome

- The unnecessary splitting of the Tx window in many small segments due to protocol operation
- Caused either by
 - the receiver, solved by simple logic
 - the sender, solved by Nagle's Algorithm - RFC 896
- If not prevented it is a normal phenomenon and "kills" TCP performance



Renato.LoCigno@dit.unitn.it

Advanced Networking – TCP PII 2

Receiver

- Avoid setting $rcwnd < MSS$
 1. Try pushing data to the application in large chunks, this is a matter of socket management and process speeds
 2. If buffer space is $< MSS \rightarrow rcwnd=0$



Renato.LoCigno@dit.unitn.it

Advanced Networking – TCP PII 3

Sender: Nagle's Algorithm

```

if there is new data to send
  if the window size and available data is >= MSS
    send complete MSS size segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is
      received
    else send data immediately
  
```

- Again it has to do with socket management
- Works well for telnet or file transfers
- Interacts badly with delayed ACK on other applications (X, Web, ...)



Renato.LoCigno@dit.unitn.it

Advanced Networking – TCP PII 4

Effect of Window Size (reprise)

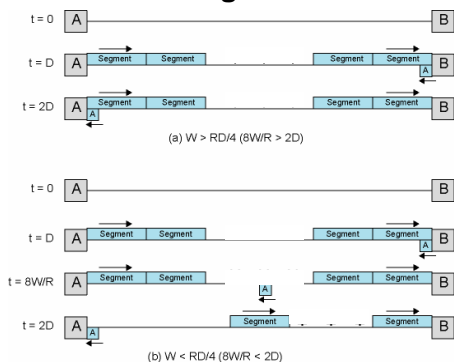
- W = TCP window size (octets)
- R = Data rate (bps) at TCP source
- D = End-to-End delay (seconds)
- After TCP source begins transmitting, it takes D seconds for first octet to arrive, and D seconds for acknowledgement to return
- TCP source should transmit $2RD$ bits, or $RD/4$ octets to "fill the pipe"



Renato.LoCigno@dit.unitn.it

Advanced Networking – TCP PII 5

Timing of TCP Flow Control



Renato.LoCigno@dit.unitn.it

Advanced Networking – TCP PII 6

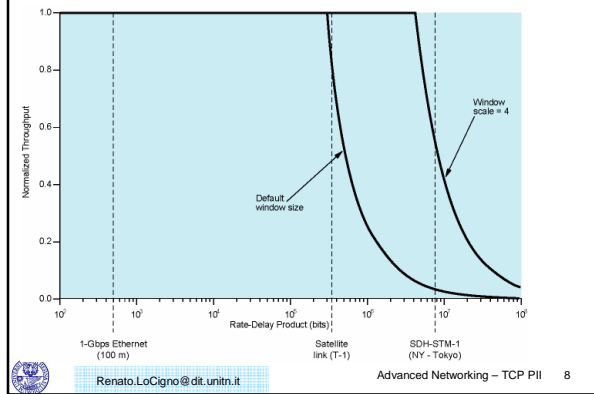
Normalized Throughput S

$$S = \begin{cases} 1 & W > RD/4 \\ \frac{4W}{RD} & W < RD/4 \end{cases}$$

Where are stored the $W - RD/4$ excessive bytes?



TCP Flow Control Performance



Complicating Factors

- Multiple TCP connections multiplexed over same network interface
 - Reducing R and efficiency
- For multi-hop connections, D is sum of delays across each network plus delays at each router
- If source data rate R exceeds data rate on a hop, that hop will be bottleneck
- Lost segments retransmitted, reducing throughput
 - Impact depends on retransmission policy



Retransmission Strategy

- TCP relies on positive acknowledgements
 - Retransmission on timeout
- No explicit negative acknowledgement
- Retransmission required when:
 - Segment arrives damaged
 - Checksum error
 - Receiver discards
 - Segment fails to arrive



Timers

- Timer (a single one per each TCP send process) initialized with each segment as it is sent
- If timer expires before acknowledgement, sender must retransmit
- Value of retransmission timer is key
 - Too small: many unnecessary retransmissions, wasting network bandwidth
 - Too large: delay in handling lost segment



Two Strategies

- Timer should be longer than round-trip delay
- Delay is variable

- Strategies:
 - **Fixed timer**
 - **Adaptive**



Problems with Adaptive Scheme

- Peer TCP entity may accumulate acknowledgements and not acknowledge immediately
- For retransmitted segments, can't tell whether acknowledgement is response to original transmission or retransmission
- Network conditions may change suddenly



Average Round-Trip Time (ARTT)

- Take average of observed round-trip times over number of segments
- If average accurately predicts future delays, resulting retransmission timer will yield good performance

$$\text{ARTT}(K+1) = \frac{1}{K+1} \sum_{i=1}^{K+1} \text{RTT}(i)$$

- Use this formula to avoid recalculating sum every time

$$\text{ARTT}(K+1) = \frac{K}{K+1} \text{ARTT}(K) + \frac{1}{K+1} \text{RTT}(K+1)$$

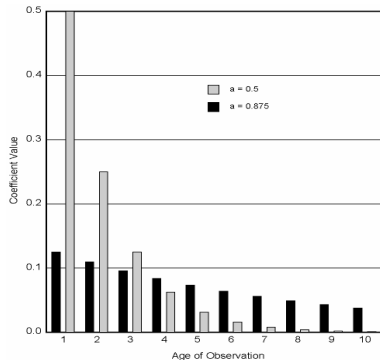


RFC 793 Exponential Averaging

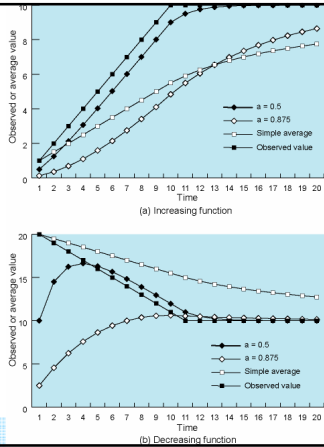
- Smoothed Round-Trip Time (SRTT)
 $\text{SRTT}(K+1) = \alpha * \text{SRTT}(K) + (1-\alpha) * \text{RTT}(K+1)$
- Gives greater weight to more recent values as shown by expansion of above:
 $\text{SRTT}(K+1) = (1-\alpha) \text{RTT}(K+1) + \alpha(1-\alpha) \text{RTT}(K) + \alpha^2(1-\alpha) \text{RTT}(K-1) + \dots + \alpha^K(1-\alpha) \text{RTT}(1)$
- α and $1-\alpha < 1$ so successive terms get smaller
- E.g. = 0.8
 $\text{SRTT}(K+1) = 0.2 \text{RTT}(K+1) + 0.16 \text{RTT}(K) + 0.128 \text{RTT}(K-1) + \dots$
- Smaller values of α give greater weight to recent values



Exponential Smoothing Coefficients



Behavior of exp. averaging with varying RTT values



RFC 793 Retransmission Timeout

- Equation above used in RFC 793 to estimate current round-trip time
- Retransmission timer set somewhat greater
- Could use a constant value:

$$RTO(K+1) = SRTT(K+1) + \Delta$$
- RTO is retransmission timer (or retransmission timeout)
- Δ is a constant
- Δ not proportional to SRTT
 - Large values of SRTT, Δ relatively small
 - Fluctuations in RTT result in unnecessary retransmissions
 - Small values of SRTT, Δ is relatively large
 - Unnecessary delays in retransmitting lost segments



RFC 793 Retransmission Timeout

- Use of timer value is proportional to SRTT, within limits

$$RTO(K+1) = \min(\text{UBOUND}, \max(\text{LBOUND}, \beta * \text{SRTT}(K+1)))$$

- UBOUND and LBOUND pre chosen fixed upper and lower bounds on timer value and β is a constant
- RFC 793 does not recommend values but gives "example values"
 - α between 0.8 and 0.9 and β between 1.3 and 2.0



Modern Retransmission Timeout

- Use of timer value is proportional to SRTT, within limits

$$RTO(K+1) = \min(\text{UBOUND}, \max(\text{LBOUND}, \beta * \text{SRTT}(K+1)))$$

- UBOUND and LBOUND pre chosen fixed upper and lower bounds on timer value and β is a constant
- RFC 793 does not recommend values but gives "example values"
 - α between 0.8 and 0.9 and β between 1.3 and 2.0
- **More on timeout setting while discussing congestion control**