

# TCP: Transmission Control Protocol

## Part II : Protocol Mechanisms

*Surasak Sanguanpong*

*nguan@ku.ac.th*

*<http://www.cpe.ku.ac.th/~nguan>*

Last updated: May 24, 1999

# Agenda

- **TCP timers**
- **Nagle's algorithm**
- **Silly Window Syndrome**
- **Delay acknowledgment**
- **Congestion control**

---

# TCP timers

- **Retransmission timer**- expecting acknowledgment time
- **Persist timer**- keeps window size information flowing
- **Keepalive timer**- detect idle connection due to crashing or reboot
- **2MSL**- duration of TIME\_WAIT state

---

# Retransmission timer

- **Fixed time-out is unacceptable because:**
  - impossible to support both LAN/WAN
  - if too short, retransmission problem
  - if too long, decrease throughput
- **TCP uses adaptive retransmission timer**
  - learning by measurement of round-trip time (RTT) experiences
  - track these changes to adjust its time-out

# Retransmission timer: RTO

- **Jacobson's retransmission time-out (RTO):**

$$RTO = A + 4D$$

$$D \leftarrow D + g(|Err| - D)$$

$$A \leftarrow A + hERR$$

$$Err = M - A$$

- $A$  = smoothed RTT (estimator of the average)
- $D$  = smoothed mean deviation
- $Err$  = diff between last RTT and current RTT estimator
- $g$  = gain; normally set to 0.125
- $h$  = deviation gain; normally set to 0.25

---

## Retransmission timer: backoff

- RTO changed in exponential like 1,3,6,12,24,48, 64 secs
- RTO is doubled for each retransmission with an upper limit of 64 secs
- this called **exponential backoff**
- retry retransmission until 9 minutes, then reset

---

## Retransmission timer: Karn's algo.

- Consider a case : a packet is transmitted, a time out occurs, the packet is retransmitted, an ACK is received. **Is ACK for the first or the second?**
- Karn's algorithm specifies when time-out, do not update the RTT estimator
- new RTO is calculated only for a not-retransmitted segment

---

# Persist timer

- **one end advertise window=0 to stop transferring**
- **later, it send a segment with window advertisement, but a segment is lost!**
- **if no any mechanism, transferring is stopped**
- **other end set a persist timer ~500 ms to ask for a new window updated**
- **send 1 byte of data to probe**

---

# Persist timer: Silly Window Syndrome

- **small amount of data are exchanged, instead of full-sized segments**
- **cause:**
  - **receiver advertise small windows, instead of waiting for a bigger one**
  - **sender transmit small chunk, instead of waiting for a bigger one**
- **solve:**
  - **receiver must not advertise small windows**
  - **sender try to transmit a full-sized segment of a half of window advertisement buffer**

---

# Keepalive timer

- **periodically sent TCP segment to confirm the connection**
- **if no acknowledge after a number of retries, the connection is reset.**
- **keep-alive segment should not be passed to the application layer**

---

# Delayed Acknowledgment

- **not send ACK immediately after receiving data,**
- **delayed ACK typically .2 ms, then send win size, ACK and echo data together (piggyback)**
- **most implementations use a 200 ms delay**
- **Host requirements RFC specifies delay ACK should be implemented with max 500 ms delay**

---

# Nagel's algorithm

- **RFC 896: prevent sending of small segments which cause congestion in WAN**
- **TCP can have only one outstanding unack small segment. Can't send more until ack arrives**
- **this collects more data before next sending**
- **self-clocking :go as fast as the small latency**
- **not for applications that send small data chunks e.g. X windows, a mouse click has to be sent as real time as possible**

# Congestion control

- **congestion : outgoing way has less capacity to send data**
  - faster LAN to slower WAN
  - multiple input go to router's less capacity output
- **what's then? -packet dropped; if more data sent, more bad situation**
- **How does a host know that lost packets are from congestion or damage packet?**
  - No way!, but our assumption is, lost packets cause by damage is very small (<1%)
  - We assume that the loss come from congestion!

---

# Congestion control

- **How to solve congestion?**
  - not easy to direct solve, but we can avoid
  - Use Jacobson's Congestion Avoidance and Control Algorithm
- **Jacobson's algorithm: 2 parts**
  - slow start
  - congestion avoidance

---

# Slow start

- **Slow start:**
  - set congestion window (*cwnd*) to one segment
  - data sent no more than *cwnd* and receiver's windows advertisement
  - double *cwnd* each sending until reach receiver's windows advertisement
  - if congestion occurs, perform slow start with congestion avoidance

---

# Slow start with congestion avoidance

- **new slow start:**
  - **slow start until cwnd reaches a half of old *cwnd* at congestion point**
  - **perform congestion avoidance: increasing *cwnd* by  $1/cwnd$  for each ack**
  - **lead to linear increasing of *cwnd***