TCP/IP 通訊協定及應用

Spring 2002

中央大學 吳曉光博士

http://wmlab.csie.ncu.edu.tw/course/tcp

Chapter 21: TCP Timeout and Retransmission

Introduction

- Two examples of timeout and retransmission had already seen:
  - 1. In the ICMP port unreachable (Section 6.5)
  - 2. In the ARP example to a nonexistent host (Section 4.5)
- TCP manage four different timers for each connection:
  - 1. A retransmission timer (this chapter)
  - 2. A persist timer (chapter 22)
  - 3. A keepalive timer (chapter 23)
  - 4. A 2MSL timer (section 18.6)
- Simple Timeout and Retransmission Example:
  - Line 4 is the transmission of "hello,world" and line 5 is its ACK.
  - Line 6 shows "and hi". Line 7-18 are 12 retransmissions
  - Line 19 is the TCP finally gives up and sends a reset

Simple Timeout and Retransmission

data, and watch what TCP does.

```
bsdi % telnet svr4 discard
Trying 140.252.13.34...
Connected to svr4.
Escape character is '^]'.
hello, world                          send this line normally
and hi                                disconnect cable before sending this line
Connection closed by foreign host.    output when TCP gives up after 9 minutes
```

Figure 21.1 shows the tcpdump output. (We have removed all the type-of-service information that is set by bsdi.)

```
 1    0.0                bsdi.1029 > svr4.discard: S 1747921409:1747921409(0)
                                                    win 4096 <mss 1024>
 2    0.004811 ( 0.0048) svr4.discard > bsdi.1029: S 3416685569:3416685569(0)
                                                    ack 1747921410
                                                    win 4096 <mss 1024>
 3    0.006441 ( 0.0016) bsdi.1029 > svr4.discard: . ack 1 win 4096

 4    6.102290 ( 6.0958) bsdi.1029 > svr4.discard: P 1:15(14) ack 1 win 4096
 5    6.259410 ( 0.1571) svr4.discard > bsdi.1029: . ack 15 win 4096

 6   24.480158 (18.2207) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
 7   25.493733 ( 1.0136) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
 8   28.493795 ( 3.0001) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
 9   34.493971 ( 6.0002) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
10   46.484427 (11.9905) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
11   70.485105 (24.0007) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
12  118.486408 (48.0013) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
13  182.488164 (64.0018) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
14  246.489921 (64.0018) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
15  310.491678 (64.0018) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
16  374.493431 (64.0018) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
17  438.495196 (64.0018) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
18  502.486941 (63.9917) bsdi.1029 > svr4.discard: P 15:23(8) ack 1 win 4096
19  566.488478 (64.0015) bsdi.1029 > svr4.discard: R 23:23(0) ack 1 win 4096
```

Figure 21.1 Simple example of TCP's timeout and retransmission.

Round-Trip Time Measurement

- Two methods of RTO calculate:
  - The original TCP specification method
    - $R \leftarrow \alpha R + (1 - \alpha) M$
    - $RTO = R\beta$
      - R=>smoothed RTT estimator, α is smoothing factor = 0.9
      - β is delay variance factor = 2
  - The Jacobson method
    - $Err = M - A$
    - $A \leftarrow A + g\,Err$
    - $D \leftarrow D + h(|Err| - D)$
    - $RTO = A + 4D$
      - A is the smoothed RTT average,D is mean deviation
      - The gain g is 1/8(0.125),h is 0.25

Round-Trip Time Measurement

- Karn's Algorithm:
  - A packet is transmitted,a timeout occurs,the packet is retransmitted with the longer RTO, and an acknowledgment is received
  - Is the ACK for the first transmission or the second?
  - This is called the retransmission ambiguity problem

- An RTT Example:
  - sent 32768 bytes of data from slip to vangogh.cs.berkeley.edu
    - slip% sock -D -i -n32 vangogh.cs.berkeley.edu. discard
  - slip is connected to the 140.252.1 Ethernet by two SLIP links
  - MTU between slip and bsdi is 296
  - 32 1024-byte=>128 segment with 256 bytes of user data

An RTT Example

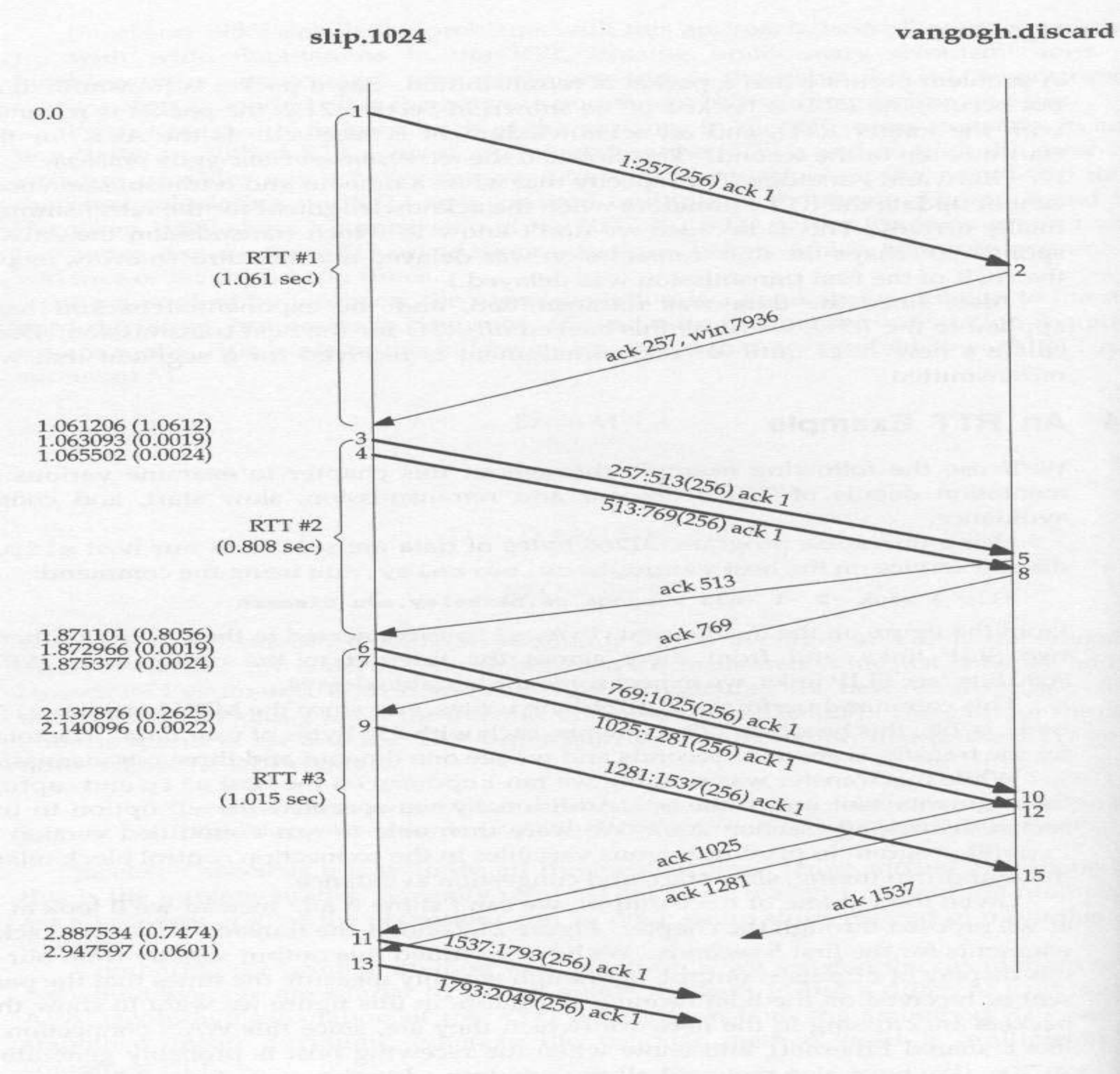

Figure 21.2 Packet exchange and RTT measurement.

An RTT Example

- Round-Trip Time Measurements:
  - RTT#1 is 1.061 seconds => 3 clock ticks
  - RTT#2 is 0.808 seconds => 1 clock tick
  - RTT#3 is 1.015 seconds => 2 clock ticks
  - Segment 4,7,9 cannot be timed,since the timer is already being used by segment 3 and 6

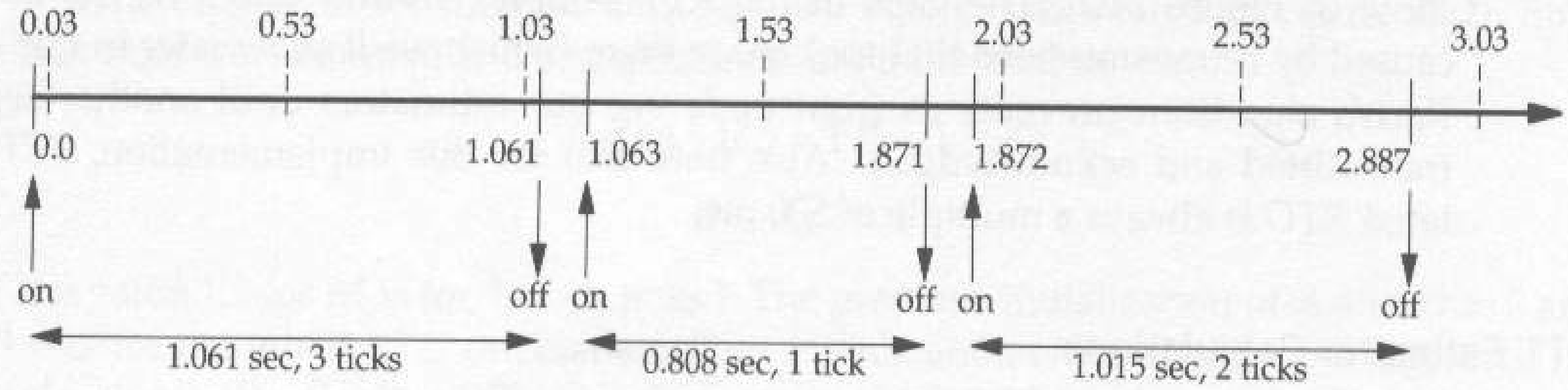

Figure 21.3 RTT measurements and clock ticks.

An RTT Example

- In this complete example 18 RTT samples were collected

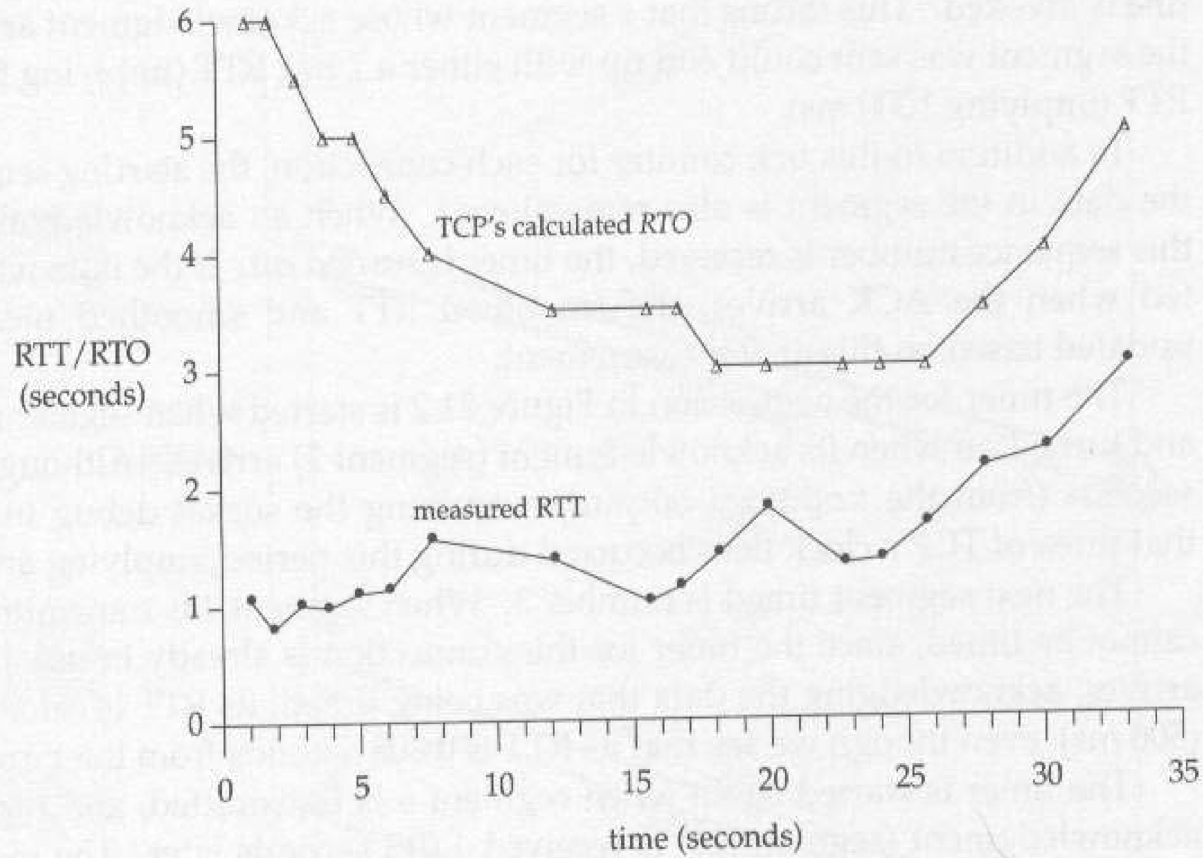

Figure 21.4 Measured RTT and TCP's calculated *RTO* for example.

An RTT Example

- RTT Estimator Calculations:
  - The initial RTO=A+2D => 0＋2x3 = 6 seconds
  - After 5.802 seconds RTO=A+4D => 0＋4x3 = 12 seconds
  - The ACK arrives 467 ms after the retransmission.The A and D are not updated because of retransmission ambiguity
  - The ACK on line 4 is not timed since it is only an ACK

shows the first four lines from the tcpdump output file.

```
1  0.0                 slip.1024 > vangogh.discard: S 35648001:35648001(0)
                                                    win 4096 <mss 256>
2  5.802377 (5.8024)   slip.1024 > vangogh.discard: S 35648001:35648001(0)
                                                    win 4096 <mss 256>
3  6.269395 (0.4670)   vangogh.discard > slip.1024: S 1365512705:1365512705(0)
                                                    ack 35648002
                                                    win 8192 <mss 512>
4  6.270796 (0.0014)   slip.1024 > vangogh.discard: . ack 1 win 4096
```

Figure 21.5 Timeout and retransmission of initial SYN.

An RTT Example

- RTO caculations
  - first segment arrives => RTO=6 seconds
  - second segment arrives => RTO=6.3125 seconds
  - Fixed-point calculations that are actually used =>RTO is 6 seconds (not 6.3125)
- Slow Start
  - See the slow start algorithm in Section 20.6
- Congetion Example:
  - Retransmission will appear as motion down and to the right
  - total time was 45 sec,35 sec for send data segments only,first data segment was sent until 6.3 sec,final took 4.0 sec to receive ACKs

Congetion Example

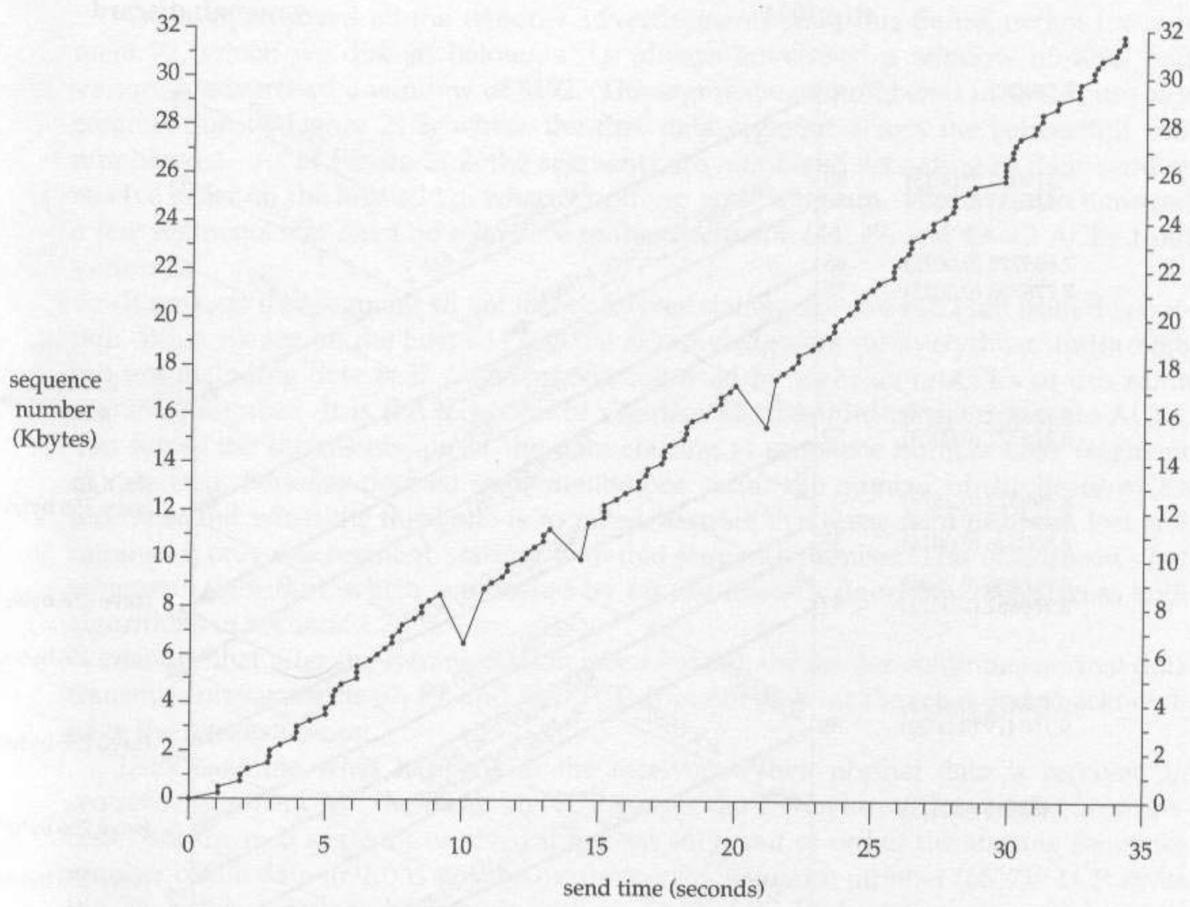

Figure 21.6 Sending of 32768 bytes of data from slip to vangogh.

Congetion Example

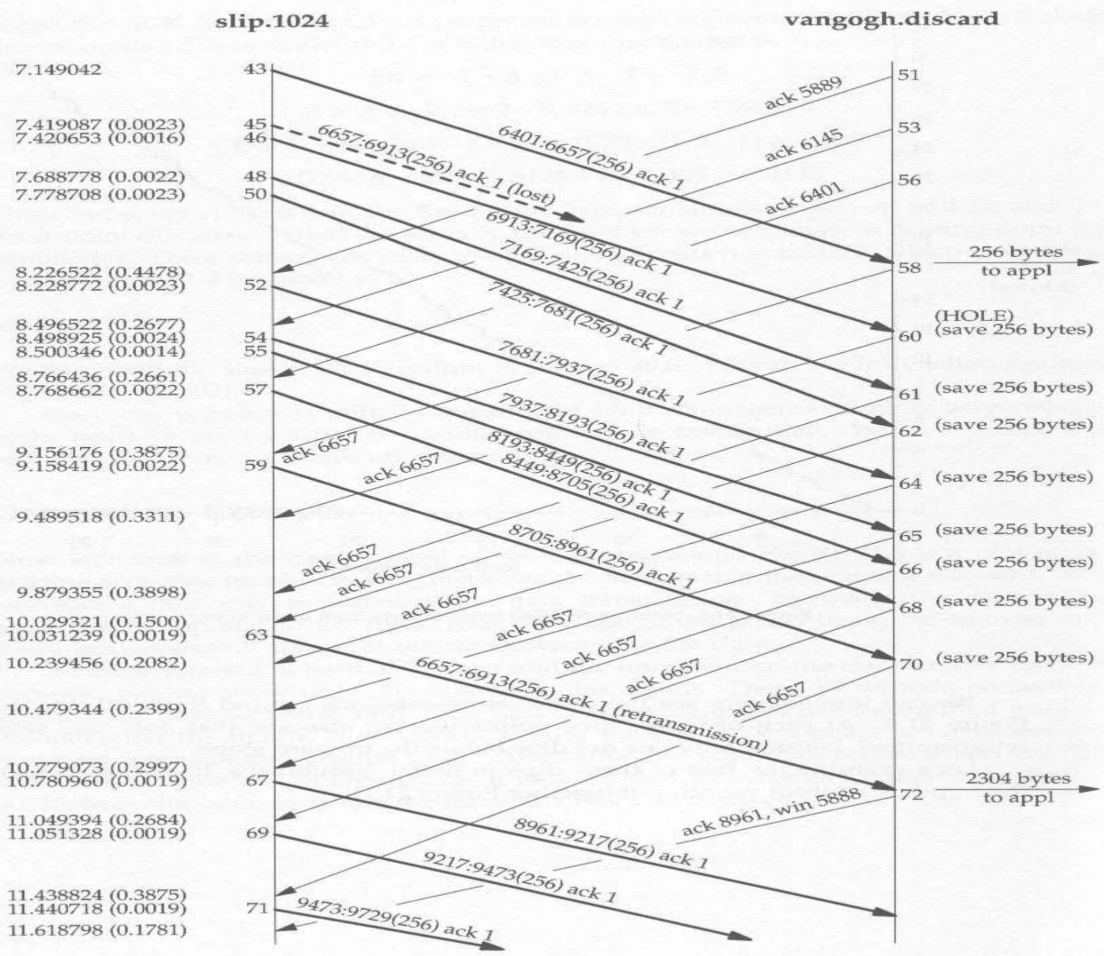

Figure 21.7 Packet exchange for retransmission around the 10-second mark.

Congetion Example

- The Jacobson's fast retransmit algorithm:
  - It is followed by his fast recovery algorithm.The third of the duplicate ACKs was received that forces to retransmit
  - Berkeley-derived implementation when the third one is received, assume that a segment has been lost and retransmit only one segment

- When the missing data arrives(segment 63):
  - The receiving TCP now has data bytes 6657-8960 in its buffer, and passes these 2304 bytes to the user process.
  - All 2304 bytes are acknowledged in segment 72

Congestion Avoidance Algorithm

- What's congestion avoidance?
  - It is a way to deal with lost packets
- Two indication of packet loss:
  - 1. A timeout occurring
  - 2. The receipt of duplicate ACKs
- Congestion avoidance algorithm operates:
  - 1.Initialization=>cwnd is one segment,ssthresh is 65535 bytes
  - 2.TCP output never sends more than the minimum of cwnd and the receiver's advertised window
  - 3.When congestion occurs,one-half of the current window size is saved in ssthresh.If timeout,cwnd is set to one segment
  - 4.When new data is acknowledged by the other end,increase cwnd If cwnd is less than or equal to ssthresh,doing slow start

Congestion Avoidance Algorithm

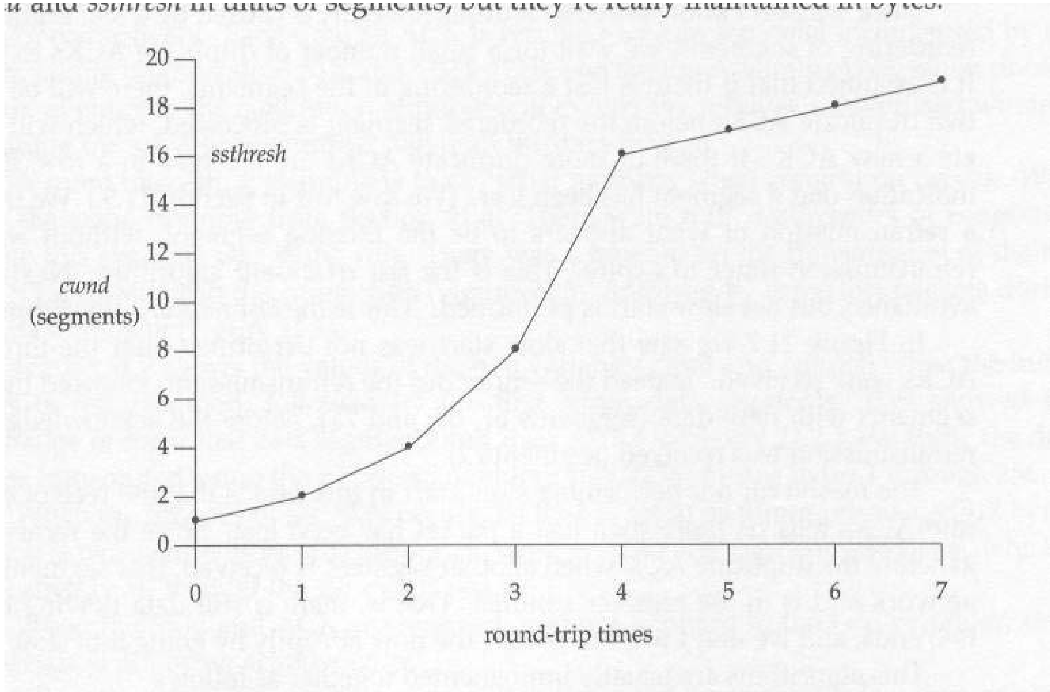

Figure 21.8 Visualization of slow start and congestion avoidance.

Fast Retransmit and Fast Recovery Algorithms

- Fast retransmit algorithm:
  - If three or more duplicate ACKs are received in a row,indicate a segment has been lost,then retransmission the missing segment
- Fast recovery algorithm:
  - Next,congestion avoidance,but not slow start is performed

- Congestion Example (Continued)
  - In congestion avoidance:
    - cwnd←cwnd＋(egsize×segsize)／cwmd＋ segsize ／ 8
  - By fast retransmit and fast recovey,we can sed a new data segment when cwnd > unacknowledged bytes

Congestion Example (Continued)

Segment# (Figure 21.2)	Action			Variable	
	Send	Receive	Comment	*cwnd*	*ssthresh*
			initialize	256	65535
	SYN				
			timeout	256	512
	SYN		retransmit		
		SYN, ACK			
	ACK				
1	1:257(256)				
2		ACK 257	slow start	512	512
3	257:513(256)				
4	513:769(256)				
5		ACK 513	slow start	768	512
6	769:1025(256)				
7	1025:1281(256)				
8		ACK 769	cong. avoid	885	512
9	1281:1537(256)				
10		ACK 1025	cong. avoid	991	512
11	1537:1793(256)				
12		ACK 1281	cong. avoid	1089	512

Figure 21.9 Example of congestion avoidance.

Segment# (Figure 21.7)	Action			Variable	
	Send	Receive	Comment	*cwnd*	*ssthresh*
58		ACK 6657	ACK of new data	2426	512
59	8705:8961(256)				
60		ACK 6657	duplicate ACK #1	2426	512
61		ACK 6657	duplicate ACK #2	2426	512
62		ACK 6657	duplicate ACK #3	1792	1024
63	6657:6913(256)		retransmission		
64		ACK 6657	duplicate ACK #4	2048	1024
65		ACK 6657	duplicate ACK #5	2304	1024
66		ACK 6657	duplicate ACK #6	2560	1024
67	8961:9217(256)				
68		ACK 6657	duplicate ACK #7	2816	1024
69	9217:9473(256)				
70		ACK 6657	duplicate ACK #8	3072	1024
71	9473:9729(256)				
72		ACK 8961	ACK of new data	1280	1024

Figure 21.11 Example of congestion avoidance (continued).

Congestion Example (Continued)

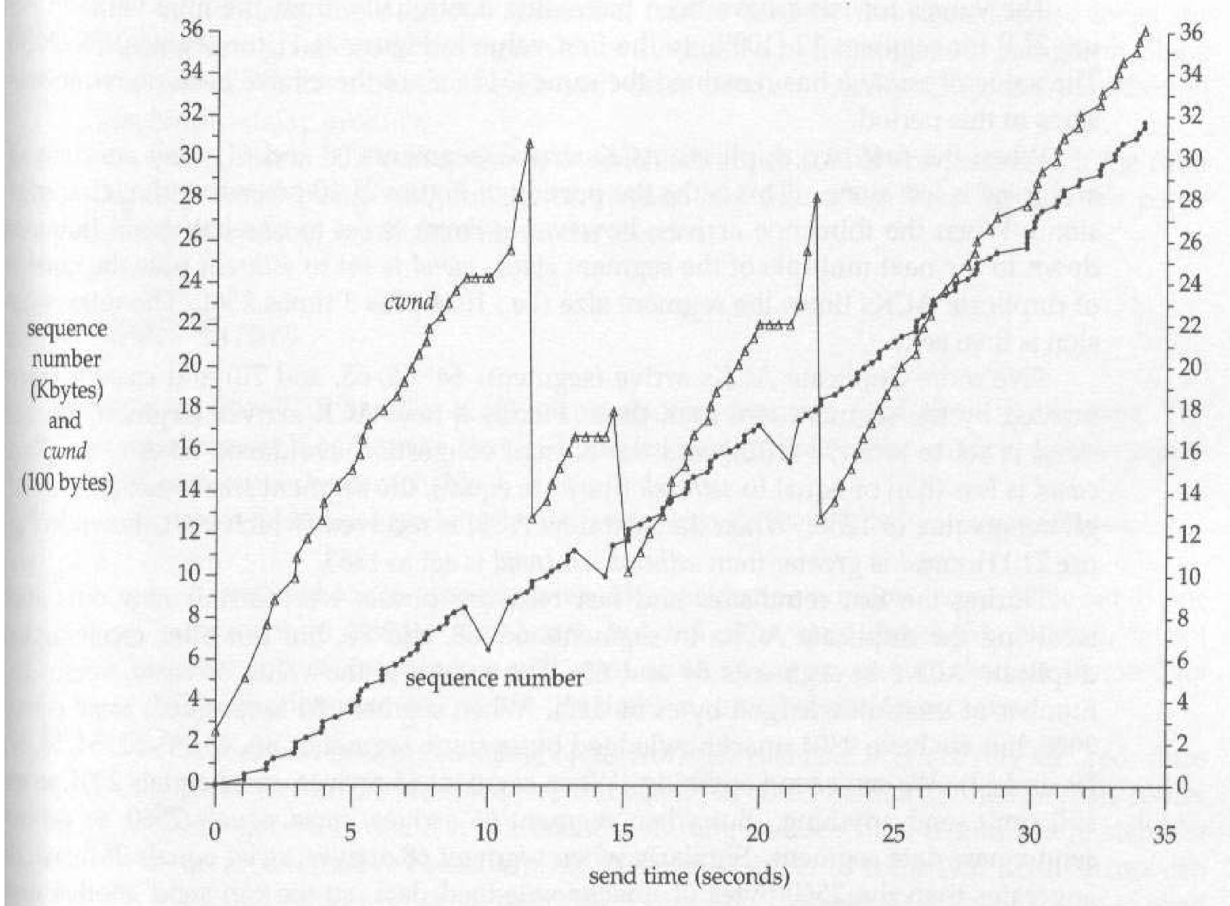

Figure 21.10 Value of *cwnd* and send sequence number while data is being transmitted.

ICMP Errors

- Berkeley-based implementations handle ICMP errors as follows:
  - A received source quench cause the cwnd set to one segment to initiate slow start,but the ssthresh is not changed
  - A received host unreachable or network unreachable is effectively ignored, since these two errors are considered transient
- An Example
  - slip% sock aix echo
  - test line
  - test line
  - SLIP link is brought down here
  - another line — type this line and retransmissions
  - SLIP link is reestablished here
  - …….. — after the last line,SLIP link is brought down
  - read error:No route to host — TCP finally gives up

ICMP Errors

```
 1      0.0                    slip.1035 > aix.echo: P 1:11(10) ack 1
 2      0.212271 (   0.2123)   aix.echo > slip.1035: P 1:11(10) ack 11
 3      0.310685 (   0.0984)   slip.1035 > aix.echo: . ack 11

                               SLIP link brought down here

 4    174.758100 (174.4474)    slip.1035 > aix.echo: P 11:24(13) ack 11
 5    174.759017 (   0.0009)   sun > slip: icmp: host aix unreachable

 6    177.150439 (   2.3914)   slip.1035 > aix.echo: P 11:24(13) ack 11
 7    177.151271 (   0.0008)   sun > slip: icmp: host aix unreachable

 8    182.150200 (   4.9989)   slip.1035 > aix.echo: P 11:24(13) ack 11
 9    182.151189 (   0.0010)   sun > slip: icmp: host aix unreachable

10    192.149671 (   9.9985)   slip.1035 > aix.echo: P 11:24(13) ack 11
11    192.150608 (   0.0009)   sun > slip: icmp: host aix unreachable

12    212.148783 ( 19.9982)    slip.1035 > aix.echo: P 11:24(13) ack 11
13    212.149786 (   0.0010)   sun > slip: icmp: host aix unreachable

                               SLIP link brought up here

14    252.146774 ( 39.9970)    slip.1035 > aix.echo: P 11:24(13) ack 11
15    252.439257 (   0.2925)   aix.echo > slip.1035: P 11:24(13) ack 24
16    252.505331 (   0.0661)   slip.1035 > aix.echo: . ack 24

17    261.977246 (   9.4719)   slip.1035 > aix.echo: P 24:38(14) ack 24
18    262.158758 (   0.1815)   aix.echo > slip.1035: P 24:38(14) ack 38
19    262.305086 (   0.1463)   slip.1035 > aix.echo: . ack 38

                               SLIP link brought down here

20    458.155330 (195.8502)    slip.1035 > aix.echo: P 38:52(14) ack 38
21    458.156163 (   0.0008)   sun > slip: icmp: host aix unreachable

22    461.136904 (   2.9807)   slip.1035 > aix.echo: P 38:52(14) ack 38
23    461.137826 (   0.0009)   sun > slip: icmp: host aix unreachable

24    467.136461 (   5.9986)   slip.1035 > aix.echo: P 38:52(14) ack 38
25    467.137385 (   0.0009)   sun > slip: icmp: host aix unreachable

26    479.135811 ( 11.9984)    slip.1035 > aix.echo: P 38:52(14) ack 38
27    479.136647 (   0.0008)   sun > slip: icmp: host aix unreachable

28    503.134816 ( 23.9982)    slip.1035 > aix.echo: P 38:52(14) ack 38
29    503.135740 (   0.0009)   sun > slip: icmp: host aix unreachable

                               14 lines of output deleted here

44   1000.219573 ( 64.0959)    slip.1035 > aix.echo: P 38:52(14) ack 38
45   1000.220503 (   0.0009)   sun > slip: icmp: host aix unreachable

46   1064.201281 ( 63.9808)    slip.1035 > aix.echo: R 52:52(0) ack 38
47   1064.202182 (   0.0009)   sun > slip: icmp: host aix unreachable
```

Figure 21.12 TCP handling of received ICMP host unreachable error.

Repacketization

- TCP is allowed to perform repacketization which can increase performance
- notice on bytes of line 3 and 6 in following illustration

```
1   0.0                  bsdi.1032 > svr4.discard: P 1:13(12) ack 1
2   0.140489 ( 0.1405)   svr4.discard > bsdi.1032: . ack 13

                         Ethernet cable disconnected here

3  26.407696 (26.2672)   bsdi.1032 > svr4.discard: P 13:27(14) ack 1
4  27.639390 ( 1.2317)   bsdi.1032 > svr4.discard: P 13:27(14) ack 1
5  30.639453 ( 3.0001)   bsdi.1032 > svr4.discard: P 13:27(14) ack 1

                         third line typed here

6  36.639653 ( 6.0002)   bsdi.1032 > svr4.discard: P 13:33(20) ack 1
7  48.640131 (12.0005)   bsdi.1032 > svr4.discard: P 13:33(20) ack 1

                         Ethernet cable reconnected here

8  72.640768 (24.0006)   bsdi.1032 > svr4.discard: P 13:33(20) ack 1
9  72.719091 ( 0.0783)   svr4.discard > bsdi.1032: . ack 33
```

Figure 21.13 Repacketization of data by TCP.

Summary

- TCP calculates a smoothed RTT estimator and a smoothed mean deviation estimator.Then use these two estimators to calculate the next retransmission timeout value.
- We see many of TCP's algorithm in action:
  - slow start
  - congestion avoidance
  - fast retransmit
  - fast recovery
- We see effect various ICMP errors have on a TCP connection
- We see how TCP is allowed to repacketize its data