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 20:
TCP Bulk Data Flow

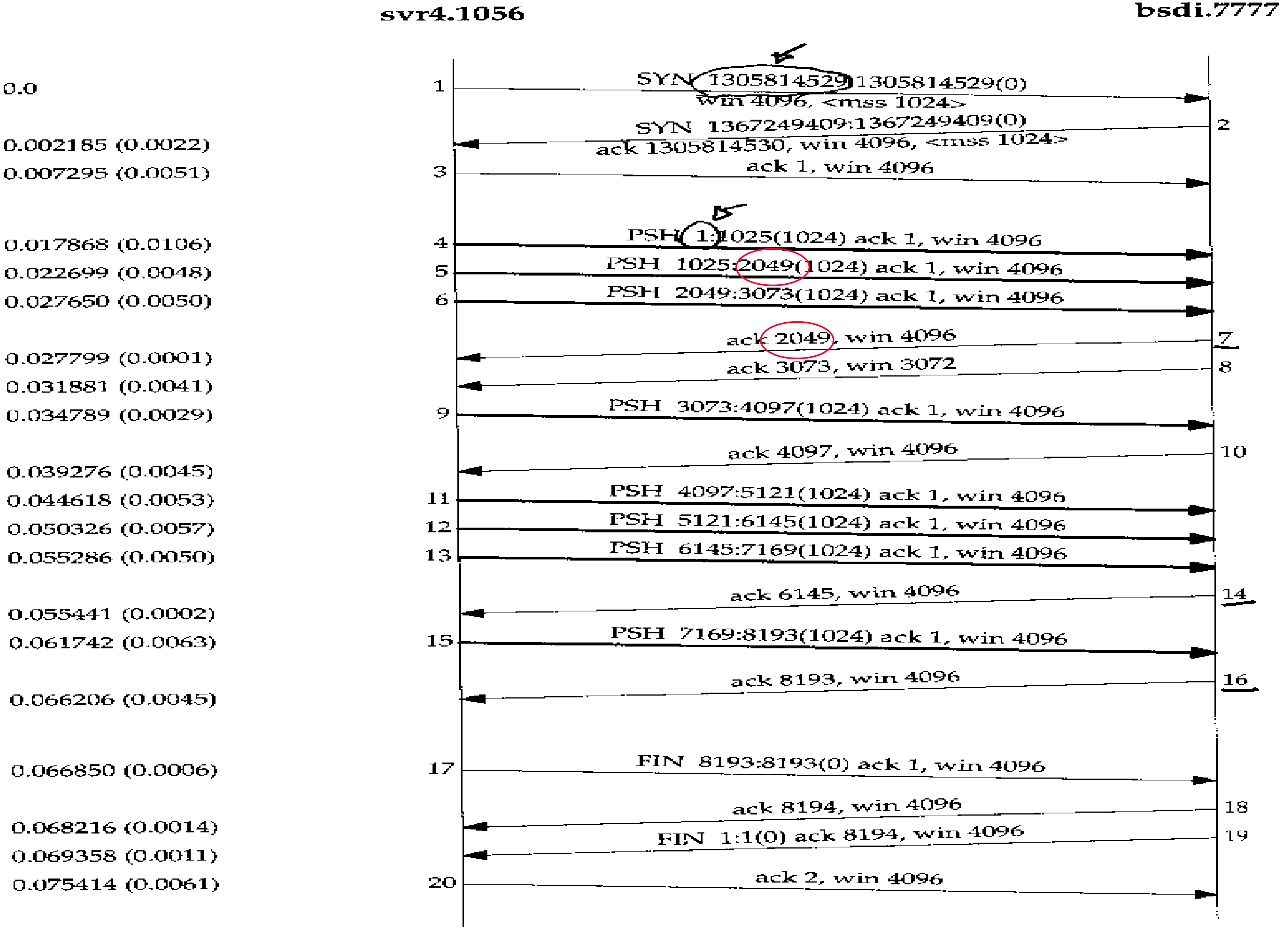
Introduction

- What we will see:
  - **sliding window protocol**: TCP uses a form of flow control called a sliding window protocol
  - **TCP's PUSH flag**
  - **slow start**: the technique used by TCP for getting the flow of data established on a connection, and then we examine bulk data throughput

Normal Data Flow

- Examination:
  - transfer 8192 bytes from svr4 to bsdi
  - on bsdi:
    - bsdi % sock -i -s 7777
  - on svr4:
    - svr4 % sock -i -n8 bsdi 7777
  - Flags:
    - -i : sink, read from the network and discard the data
    - -s : as a server
    - -n8 : perform 1024-byte writes to the network

Normal Data Flow



svr4.1056
bsdi.7777

0.0 — 1 — SYN 1305814529:1305814529(0) win 4096, <mss 1024>
0.002185 (0.0022) — 2 — SYN 1367249409:1367249409(0) ack 1305814530, win 4096, <mss 1024>
0.007295 (0.0051) — 3 — ack 1, win 4096
0.017868 (0.0106) — 4 — PSH 1:1025(1024) ack 1, win 4096
0.022699 (0.0048) — 5 — PSH 1025:2049(1024) ack 1, win 4096
0.027650 (0.0050) — 6 — PSH 2049:3073(1024) ack 1, win 4096
0.027799 (0.0001) — 7 — ack 2049, win 4096
0.031881 (0.0041) — 8 — ack 3073, win 3072
0.034789 (0.0029) — 9 — PSH 3073:4097(1024) ack 1, win 4096
0.039276 (0.0045) — 10 — ack 4097, win 4096
0.044618 (0.0053) — 11 — PSH 4097:5121(1024) ack 1, win 4096
0.050326 (0.0057) — 12 — PSH 5121:6145(1024) ack 1, win 4096
0.055286 (0.0050) — 13 — PSH 6145:7169(1024) ack 1, win 4096
0.055441 (0.0002) — 14 — ack 6145, win 4096
0.061742 (0.0063) — 15 — PSH 7169:8193(1024) ack 1, win 4096
0.066206 (0.0045) — 16 — ack 8193, win 4096
0.066850 (0.0006) — 17 — FIN 8193:8193(0) ack 1, win 4096
0.068216 (0.0014) — 18 — ack 8194, win 4096
0.069358 (0.0011) — 19 — FIN 1:1(0) ack 8194, win 4096
0.075414 (0.0061) — 20 — ack 2, win 4096

Figure 20.1 Transfer of 8192 bytes from svr4 to bsdi.

Normal Data Flow

- (continued...)
- Segment 7 ACKs 2049, not 3073:
  - segment 4, 5, 6 arrives, IP passes them to TCP in the same order
  - TCP processes segment 4, the connection is marked to generate a delayed ACK
  - TCP processes segment 5: since TCP now has two outstanding segments to ACK, the ACK of 2049 is generated (segment 7), and the delayed ACK flag for this connection is turned off
  - TCP processes segment 6: the connection is again marked for a delayed ACK. But before segment 9 arrives, the delayed ACK timer goes off, then segment 8 ACKs 3073

Normal Data Flow

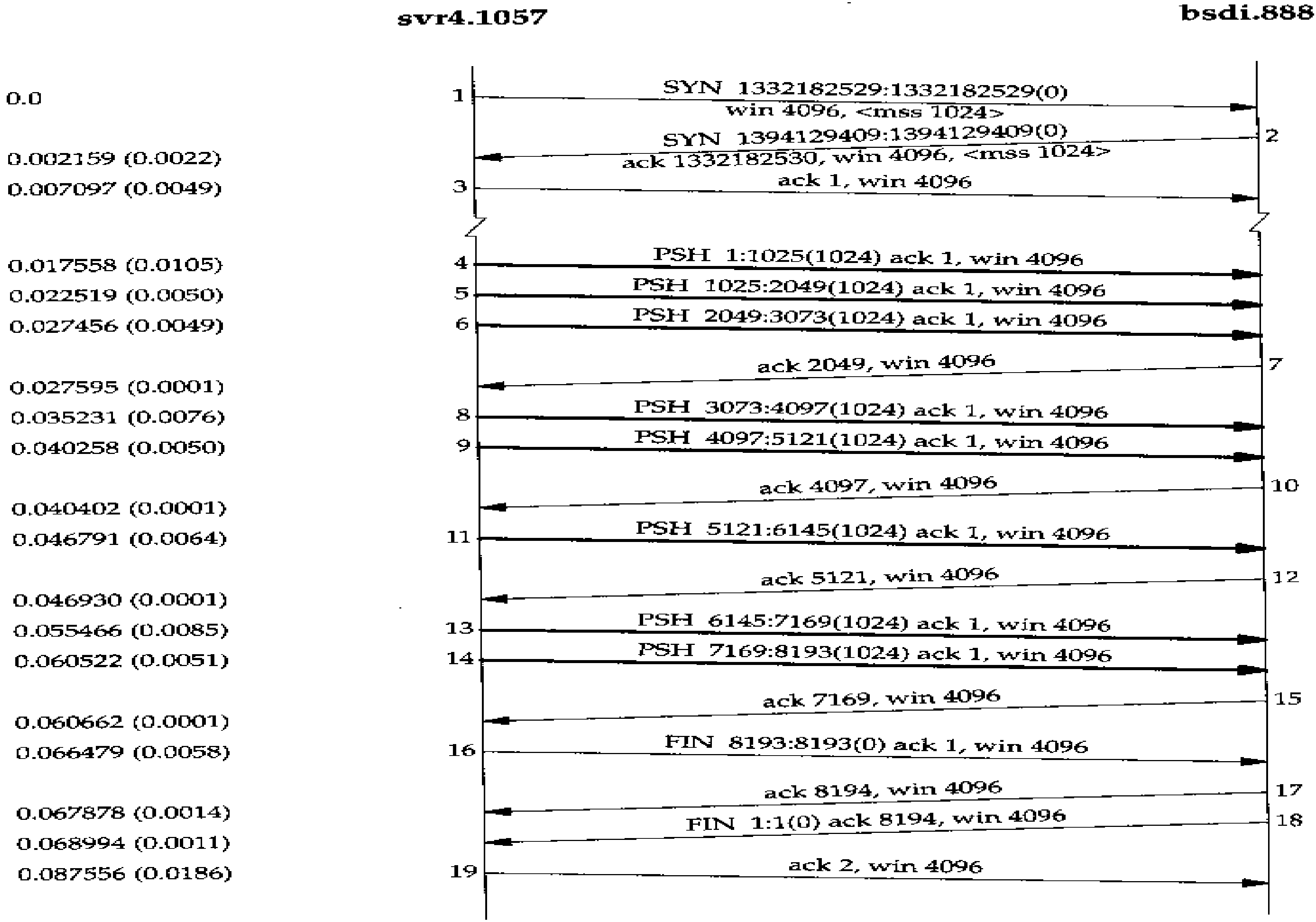


Figure 20.2 Another transfer of 8192 bytes from `svr4` to `bsdi`.

Fast Sender. Slow Receiver

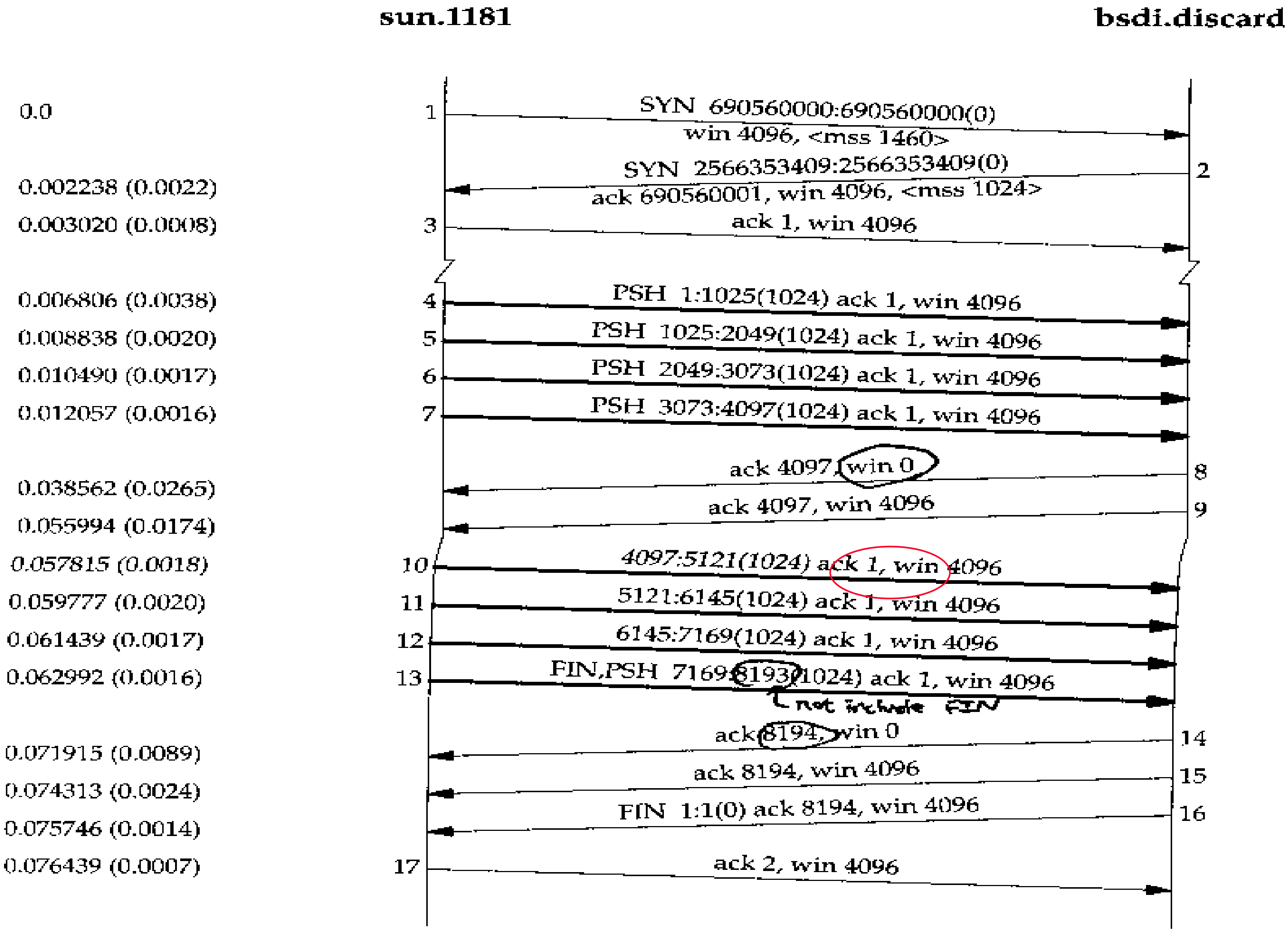


Figure 20.3 Sending 8192 bytes from a fast sender to a slow receiver.

Fast Sender, Slow Receiver

- (continued...)
- Things we noticed:
  - the sender transmits segment 4 to 7 to fill the receiver's window and then waits for an ACK (since segment 2 telled the host sun the window size is 4096)
  - the receiver sends the ACK(segment 8) but the advertised window is 0: the receiver has all the data, but it's all in the receiver 's TCP buffer
  - another ACK (called a **window update**) is sent later, announcing that the receiver can now receive another 4096 bytes: it does not acknowledge any new data, it just advances the right edge of the window
  - segment 13 contains 2 flag bits: PUSH and FIN.

Sliding Windows

- Illustration of sliding windows:

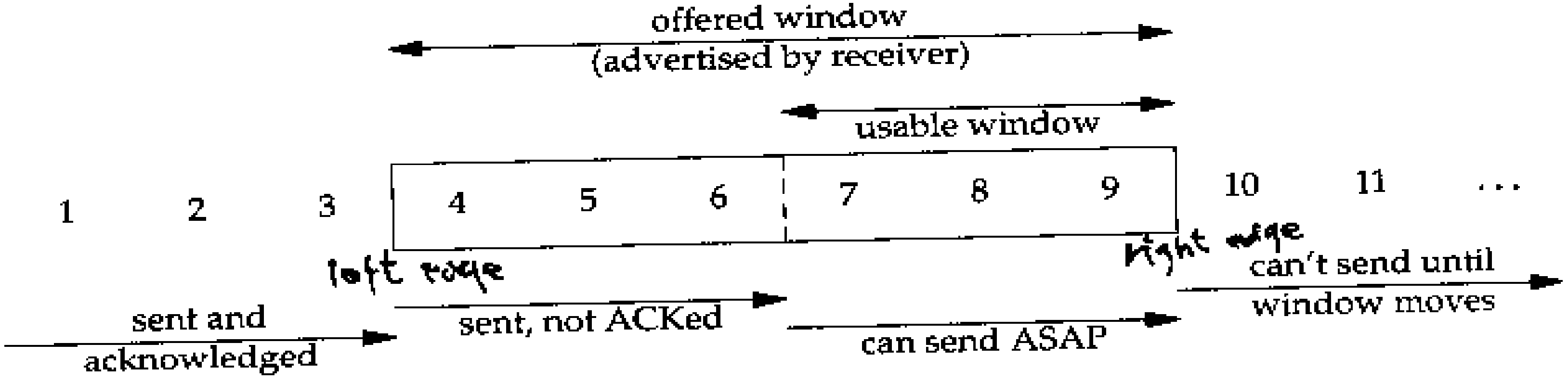


Figure 20.4 Visualization of TCP sliding window.

- offered window: 4 to 9
- usable window: 7 to 9 (computed by the sender)

Sliding Windows

- (continued...)
- the window closes: the left edge advances to right
- the window opens: the right edge moves to the right
- the window shrinks: the right edge moves to the left; The Host Requirement RFC strongly discourages this
- NEVER moves the left edge to the left: if an ACK were received that implied moving the left edge to the left, it is a duplicate ACK, and discarded

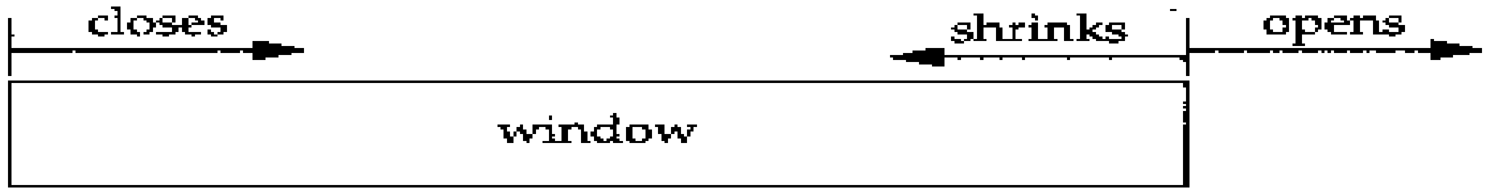


Figure 20.5 Movement of window edges.

Sliding Windows

- An Example for the data transfer in Figure 20.1:

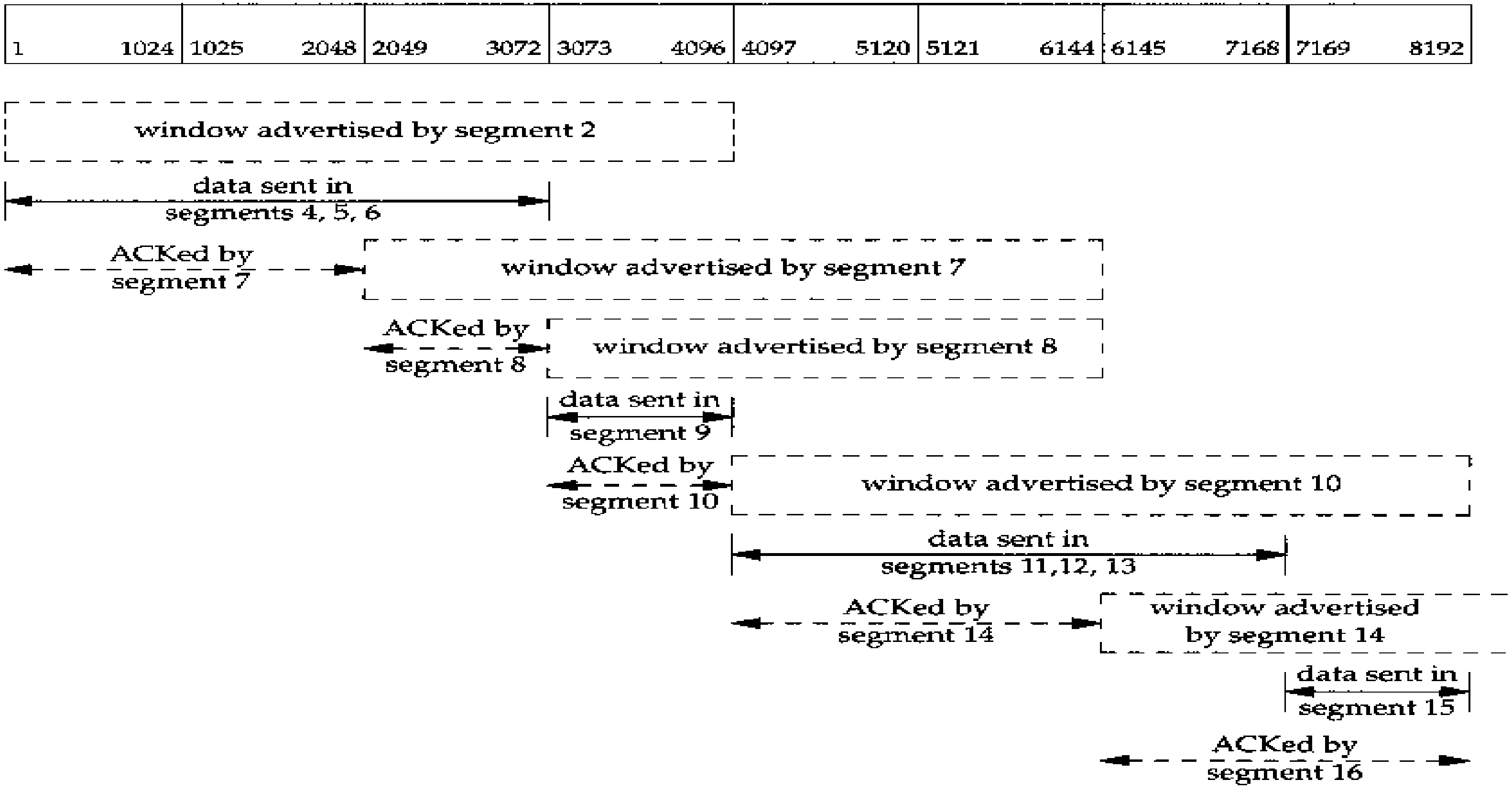

1
1024
1025
2048
2049
3072
3073
4096
4097
5120
5121
6144
6145
7168
7169
8192
window advertised by segment 2
data sent in segments 4, 5, 6
ACKed by segment 7
window advertised by segment 7
ACKed by segment 8
window advertised by segment 8
data sent in segment 9
ACKed by segment 10
window advertised by segment 10
data sent in segments 11,12, 13
ACKed by segment 14
window advertised by segment 14
data sent in segment 15
ACKed by segment 16

Figure 20.6 Sliding window protocol for Figure 20.1.

Window Size

- The size of window offered by the receiver can usually be controlled by the receiving process. This can affect the TCP performance.
- For file transfer between two workstations on a Ethernet, the common default of 4096 bytes for both is not optimal: An approximate 40% increase in throughput is seen by just increasing both buffer to 16384 bytes.

Window Size

- An Example: from sun (the client) to bsdi (the server)
  - bsdi % sock -i -s -R6144 5555
  - sun % sock -i -n1 -w8192 bsdi 5555
  - Flag:
    - -R6144 : set the size of the receive buffer (as 6144 bytes)
    - -n1 -w 8192 : perform one write of 8192 bytes

Window Size

- (continued...)

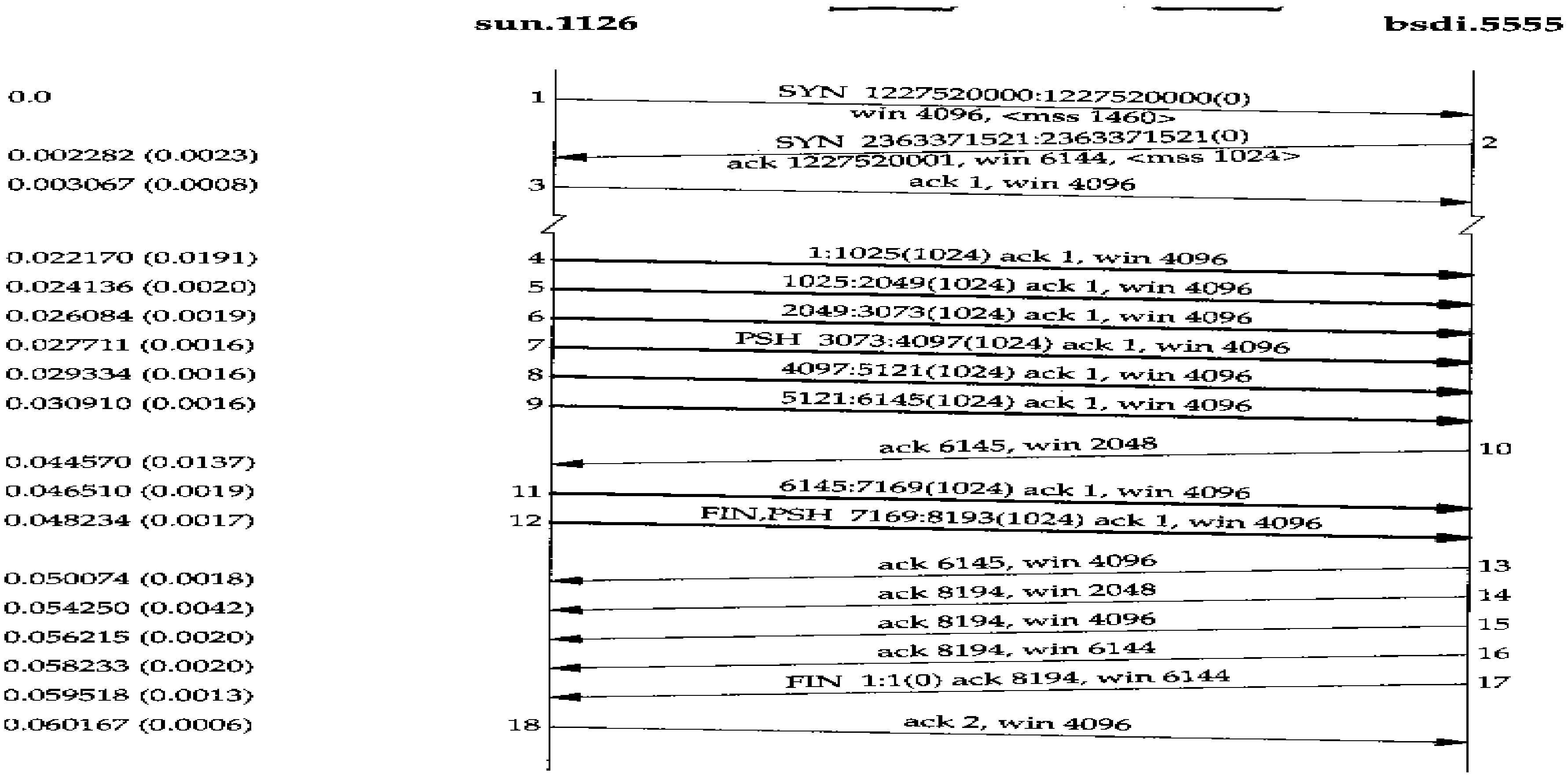


Figure 20.7 Data transfer with receiver offering a window size of 6144 bytes.

PUSH Flag

- PUSH flag: a notification from the sender to the receiver for the receiver to pass all the data that it has to the receiving process
- In the original TCP specification, it was assumed that the programming interface would allow the sending process to tell its TCP when to set the PUSH flag.
- Today, however, most APIs don't provide a way for the application to tell its TCP to set the PUSH flag. Indeed, many implementors feel the need for the PUSH flag is outdated, and a good TCP implementation can determine when to set the flag by itself.
- Most Berkeley-derived implementations automatically set the PUSH flag if the data in the segment being sent empties the send buffer.

Slow Start

- As we’ve seen for the sender starts off by injecting multiple segments into the network, up to the windows size advertised by the receiver. This naive approach is OK when two hosts are on the same LAN but reduce the throughput drastically of TCP connections between many routers and slow links.
- TCP is now required to support an algorithm called slowstart:
  - it operates by observing that the rate at which new packets should be injected into the network is the rate at which the ACKs are returned by the other end

Slow Start

- Slow start adds another window to the sender’s TCP: the congestion window, or, cwnd
- win: the window size advertised by the other end
- Operations of slow start:
  - 1. initial connection, cwnd = 1 segment (announced by the other end)
  - 2. the sender transmit up to the minimum(cwnd, win)
  - 3. Each time an ACK is received, the congestion window is increased by one segment
  - 3. the sender starts by transmitting one segment and waiting for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is ACKed, the congestion window is increased to four. This provides an exponential increase.

Slow Start

- An Example:

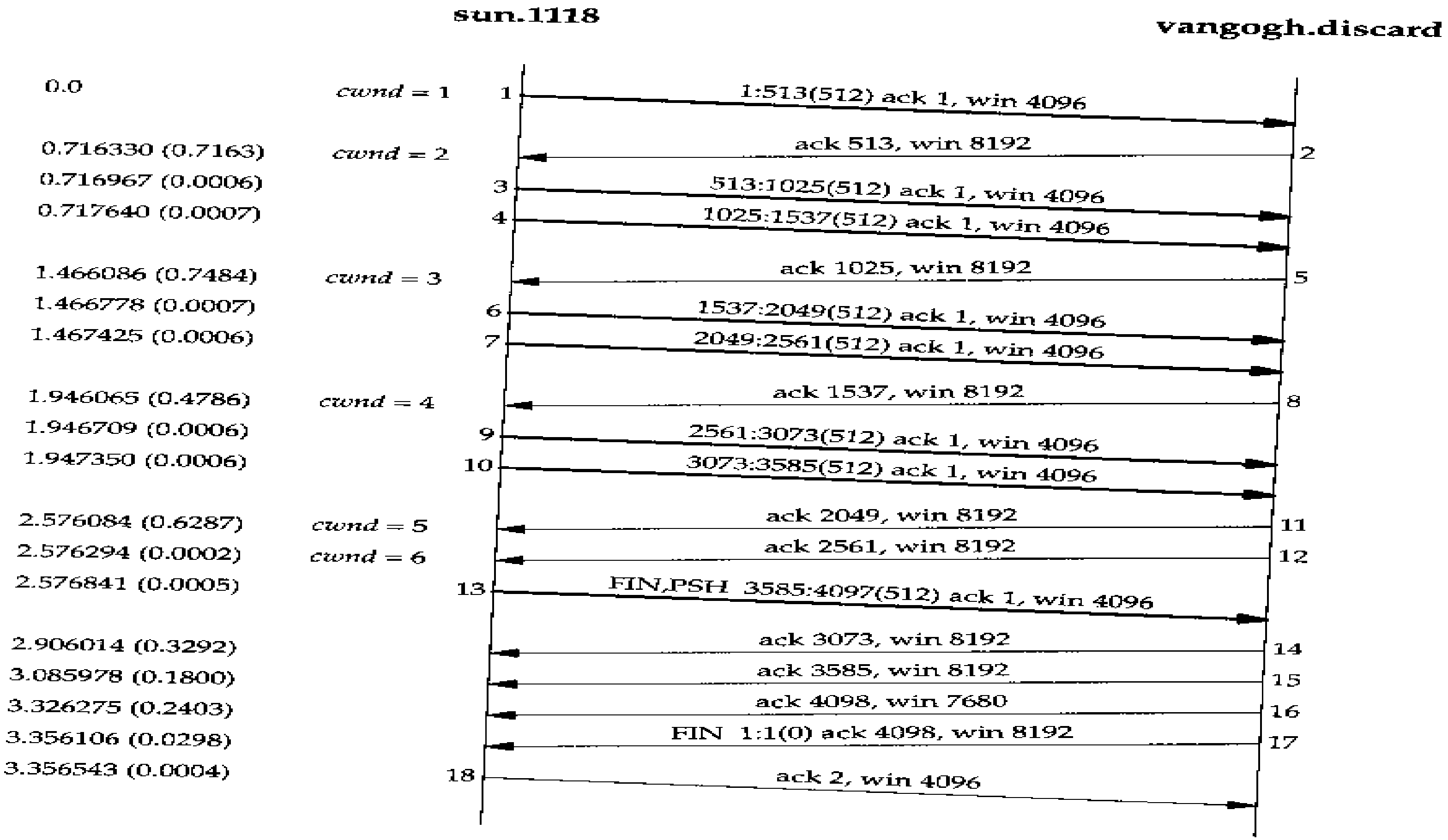
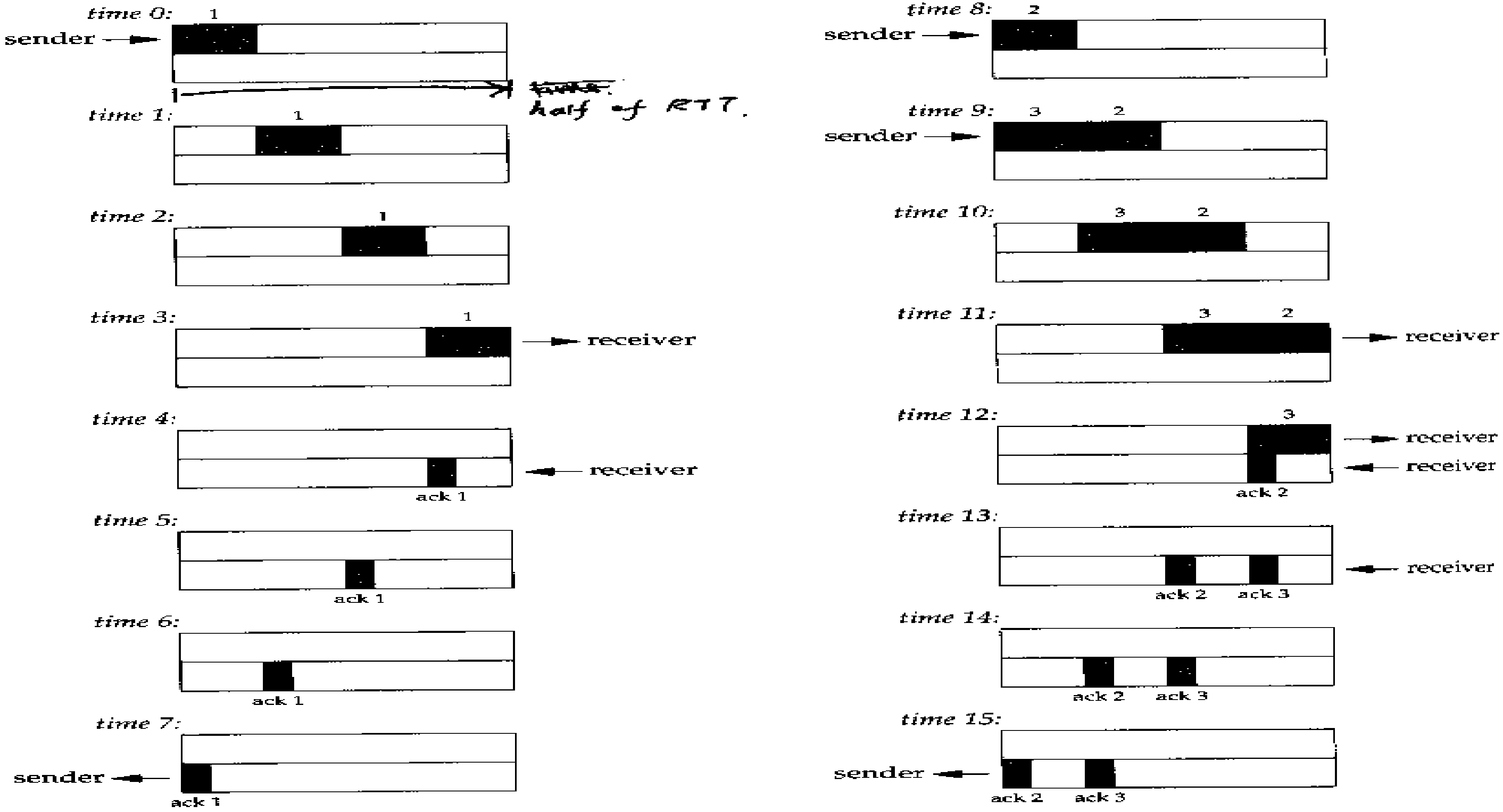


Figure 20.8 Example of slow start.

Bulk Data Throughput



half of RTT.

Figure 20.9 Times 0–15 for bulk data throughput example.

Bulk Data Throughput

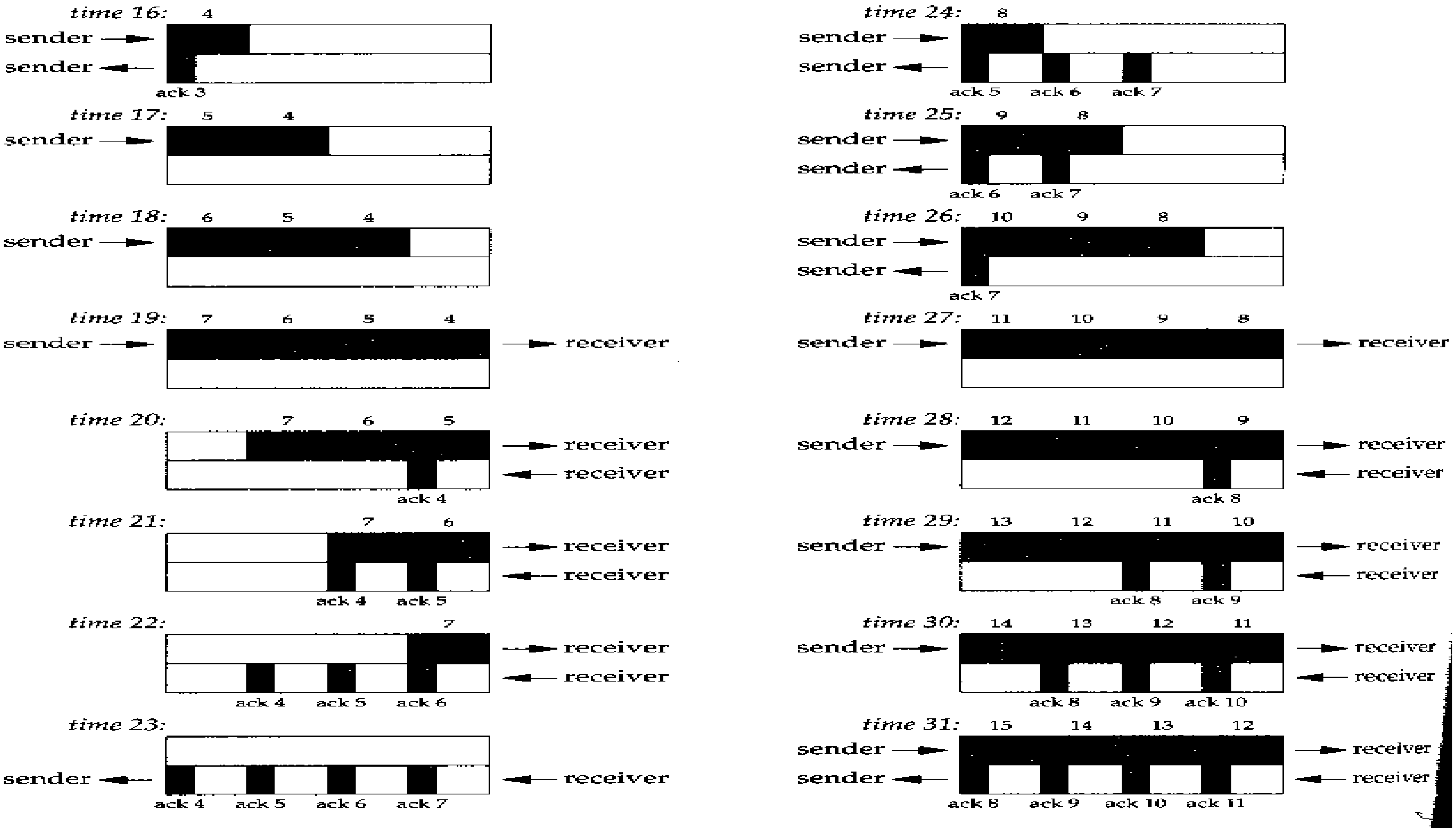


Figure 20.10 Times 16–31 for bulk data throughput example.

Bulk Data Throughput

- How big should the window be?
  - min(cwnd, advertised window)
- Bandwidth-Delay Product.:
  - Capacity (bits) = bandwidth (bits/sec) * round-trip time (sec)
- Examples:
  - T1 line (1,544,000 bits/sec) across US (about 60ms) => bandwidth-delay product (capacity) = 11,580 bytes
  - T3 line (45,000,000 bits/sec) across US (about 60ms) => bandwidth delay product (capacity)=337,500 bytes (>65,535) => Need new TCP window scale option

Bulk Data Throughput

- Relation between capacity, bandwidth and RTT:

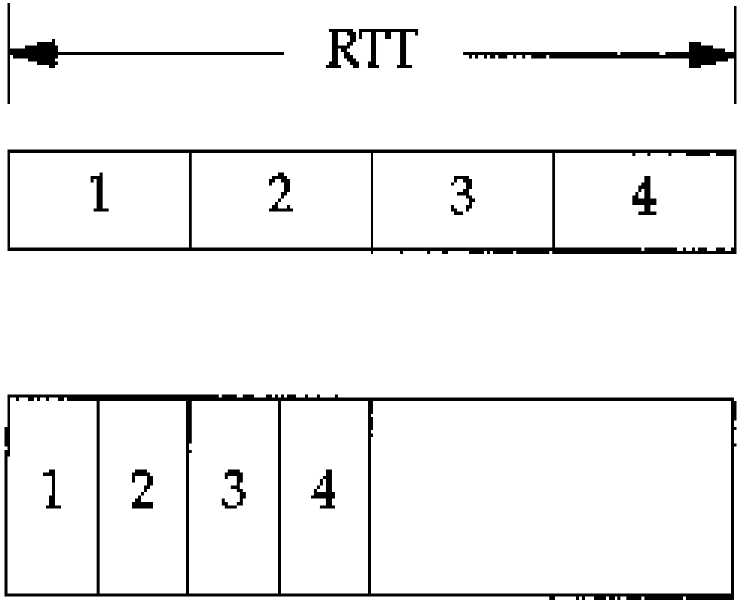


Figure 20.12 Doubling the bandwidth doubles the capacity of the pipe.

Congestion

- Congestion:
  - can occur when data arrives on a big pipe (a fast LAN) and gets sent out a smaller pipe (a slower WAN)
  - can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs

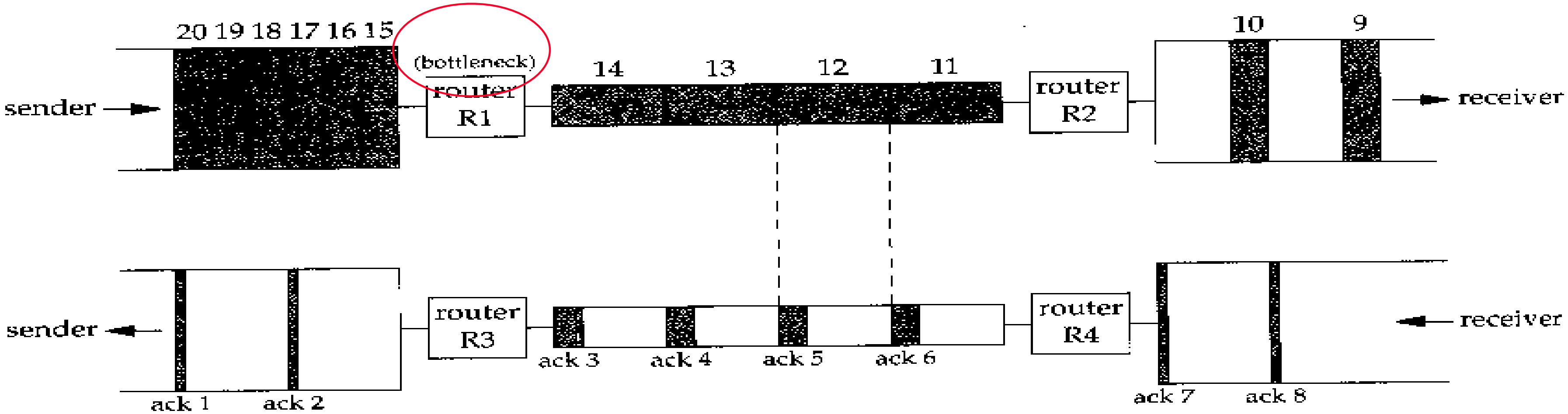


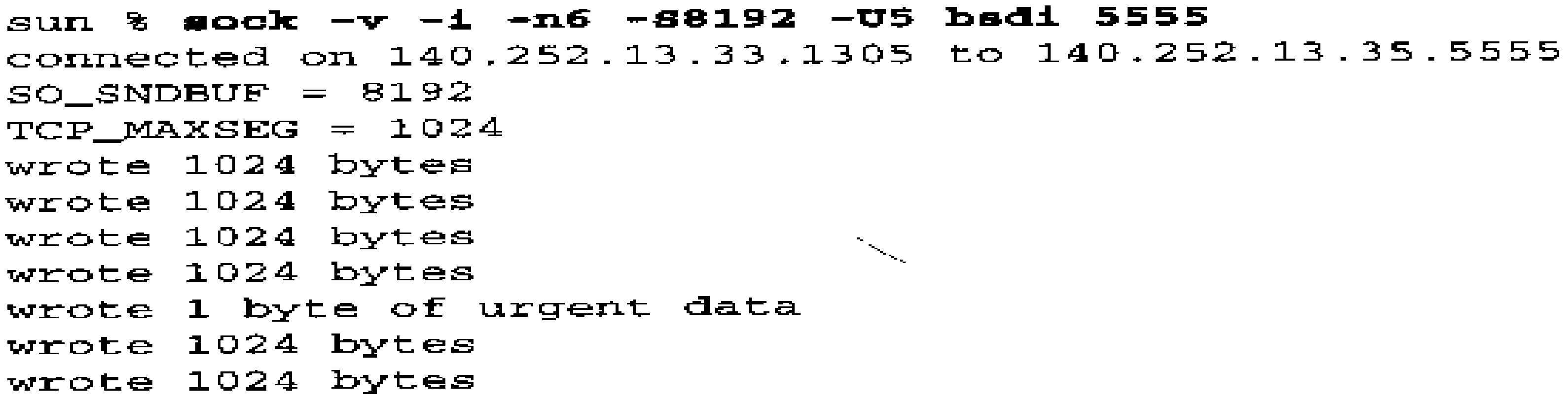
Figure 20.13 Congestion caused by a bigger pipe feeding a smaller pipe.

Urgent Mode

- TCP provides what it calls urgent mode, allowing one end to tell the other end that “urgent data” of some form has been placed into the normal stream of data.
- Telnet and Rlogin use urgent mode from thee server to he client because it’s possible for this direction of data flow to be stopped by the client TCP (i.e., it advertises a window of 0). But if the server process enter urgent mode , the server TCP immediately sends the urgent pointer and the URG flag, even though it can’t send any data.
- The Urgent pointer just advances in the data stream, and its previous position at the receiver is lost.

Urgent Mode

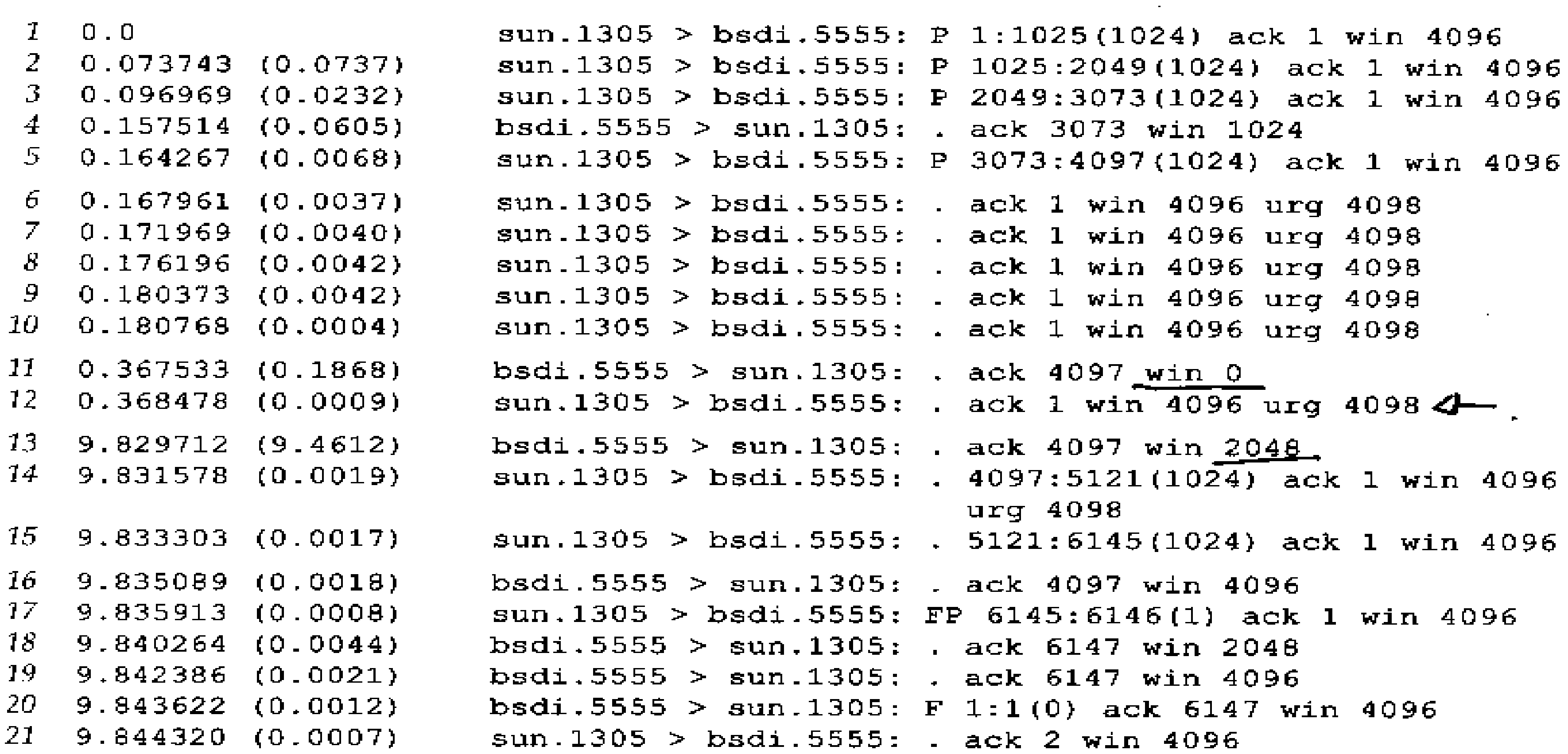
- An Example:
- bsdi % sock -i -s -P10 5555
  - -i : sink
  - -P : pause 10 secs
  - -S8192 : using a send buffer of 8192 bytes
  - -n6 : write six 1024-byte writes
  - -U5: write 1 byte of data and enter urgent mode before writing the fifth buffer to network



```
sun % sock -v -i -n6 -S8192 -U5 bsdi 5555
connected on 140.252.13.33.1305 to 140.252.13.35.5555
SO_SNDBUF = 8192
TCP_MAXSEG = 1024
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1 byte of urgent data
wrote 1024 bytes
wrote 1024 bytes
```

Urgent Mode

- (continued...)

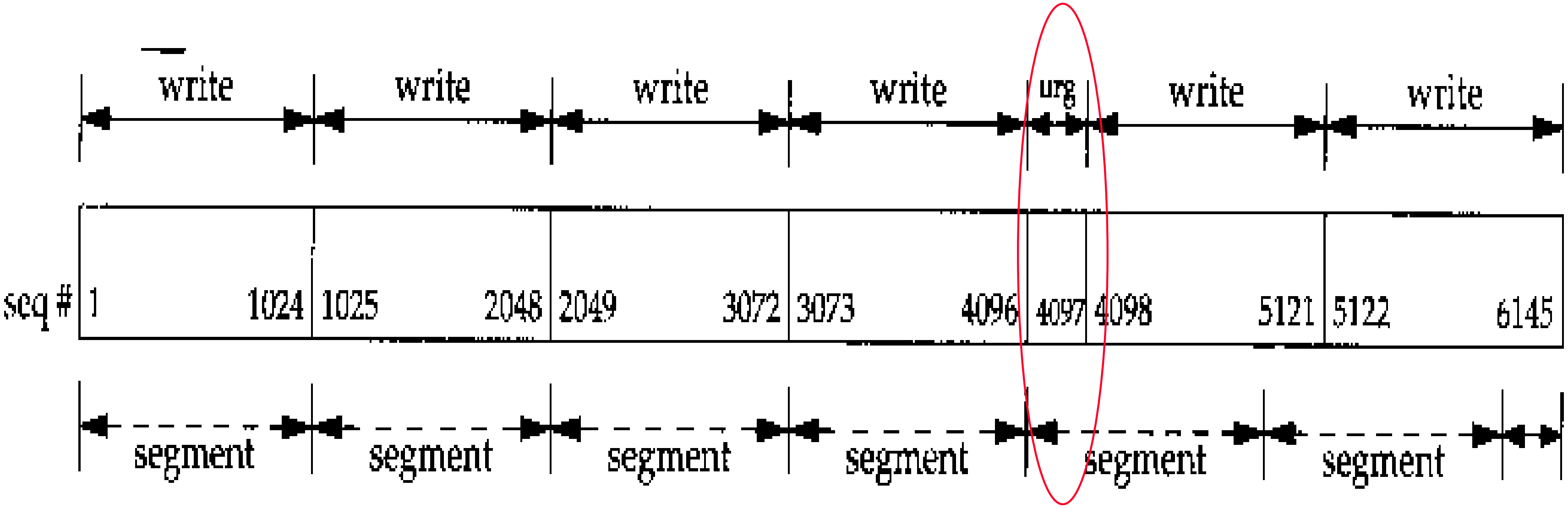


```
 1  0.0                  sun.1305 > bsdi.5555: P 1:1025(1024) ack 1 win 4096
 2  0.073743 (0.0737)    sun.1305 > bsdi.5555: P 1025:2049(1024) ack 1 win 4096
 3  0.096969 (0.0232)    sun.1305 > bsdi.5555: P 2049:3073(1024) ack 1 win 4096
 4  0.157514 (0.0605)    bsdi.5555 > sun.1305: . ack 3073 win 1024
 5  0.164267 (0.0068)    sun.1305 > bsdi.5555: P 3073:4097(1024) ack 1 win 4096

 6  0.167961 (0.0037)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098
 7  0.171969 (0.0040)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098
 8  0.176196 (0.0042)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098
 9  0.180373 (0.0042)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098
10  0.180768 (0.0004)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098

11  0.367533 (0.1868)    bsdi.5555 > sun.1305: . ack 4097 win 0
12  0.368478 (0.0009)    sun.1305 > bsdi.5555: . ack 1 win 4096 urg 4098 <--

13  9.829712 (9.4612)    bsdi.5555 > sun.1305: . ack 4097 win 2048
14  9.831578 (0.0019)    sun.1305 > bsdi.5555: . 4097:5121(1024) ack 1 win 4096
                                                urg 4098
15  9.833303 (0.0017)    sun.1305 > bsdi.5555: . 5121:6145(1024) ack 1 win 4096

16  9.835089 (0.0018)    bsdi.5555 > sun.1305: . ack 4097 win 4096
17  9.835913 (0.0008)    sun.1305 > bsdi.5555: FP 6145:6146(1) ack 1 win 4096
18  9.840264 (0.0044)    bsdi.5555 > sun.1305: . ack 6147 win 2048
19  9.842386 (0.0021)    bsdi.5555 > sun.1305: . ack 6147 win 4096
20  9.843622 (0.0012)    bsdi.5555 > sun.1305: F 1:1(0) ack 6147 win 4096
21  9.844320 (0.0007)    sun.1305 > bsdi.5555: . ack 2 win 4096
```

Figure 20.14 tcpdump output for TCP urgent mode.

Urgent Mode

- (continued...)



Summary

- TCP Bulk Data Flow
- push flag
- slow start
- bandwidth-delay product
- urgent mode