Chapter 19:
TCP Interactive Data Flow

Introduction

- Examining the transfer of interactive data using TCP
- Using delayed acknowledges
- Using Nagle algorithm

Interactive Input

- **Examining interactive data on Rlogin connections:**

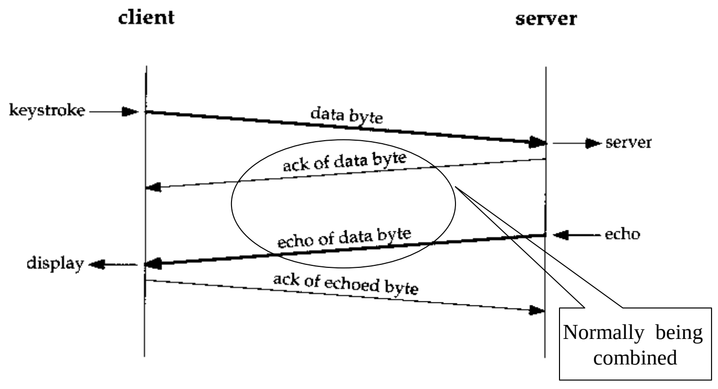


Figure 19.1 One possible way to do remote echo of interactive keystroke.

Interactive Input (Cont.)

- **A Rlogin Example:**

```
 1  0.0                  bsdi.1023 > svr4.login: P 0:1(1) ack 1 win 4096
 2  0.016497 (0.0165)    svr4.login > bsdi.1023: P 1:2(1) ack 1 win 4096
 3  0.139955 (0.1235)    bsdi.1023 > svr4.login: . ack 2 win 4096

 4  0.458037 (0.3181)    bsdi.1023 > svr4.login: P 1:2(1) ack 2 win 4096
 5  0.474386 (0.0163)    svr4.login > bsdi.1023: P 2:3(1) ack 2 win 4096
 6  0.539943 (0.0656)    bsdi.1023 > svr4.login: . ack 3 win 4096

 7  0.814582 (0.2746)    bsdi.1023 > svr4.login: P 2:3(1) ack 3 win 4096
 8  0.831108 (0.0165)    svr4.login > bsdi.1023: P 3:4(1) ack 3 win 4096
 9  0.940112 (0.1090)    bsdi.1023 > svr4.login: . ack 4 win 4096

10  1.191287 (0.2512)    bsdi.1023 > svr4.login: P 3:4(1) ack 4 win 4096
11  1.207701 (0.0164)    svr4.login > bsdi.1023: P 4:5(1) ack 4 win 4096
12  1.339994 (0.1323)    bsdi.1023 > svr4.login: . ack 5 win 4096

13  1.680646 (0.3407)    bsdi.1023 > svr4.login: P 4:5(1) ack 5 win 4096
14  1.697977 (0.0173)    svr4.login > bsdi.1023: P 5:7(2) ack 5 win 4096
15  1.739974 (0.0420)    bsdi.1023 > svr4.login: . ack 7 win 4096

16  1.799841 (0.0599)    svr4.login > bsdi.1023: P 7:37(30) ack 5 win 4096
17  1.940176 (0.1403)    bsdi.1023 > svr4.login: . ack 37 win 4096
18  1.944338 (0.0042)    svr4.login > bsdi.1023: P 37:44(7) ack 5 win 4096
19  2.140110 (0.1958)    bsdi.1023 > svr4.login: . ack 44 win 4096
```

Figure 19.2 TCP segments when date typed on Rlogin connection.

Delayed Acknowledge

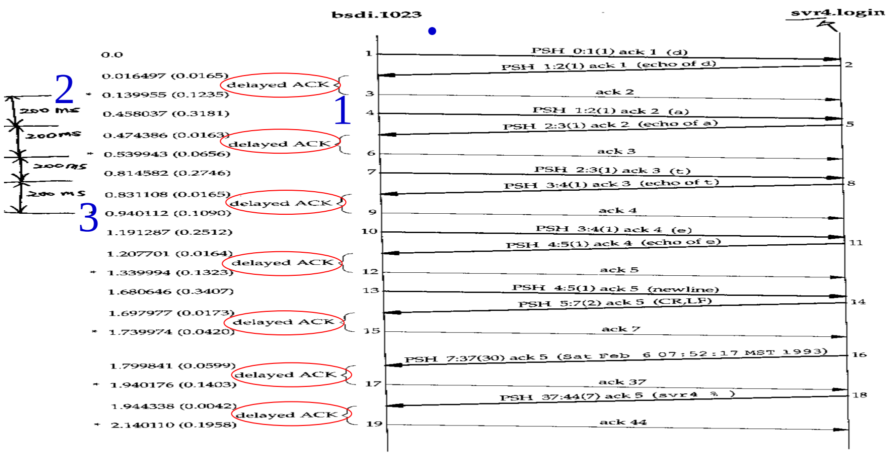


Figure 19.3 Time line of data flow for date command typed on an rlogin connection.

Delayed Acknowledge (Cont.)

- **Delayed ACKs:**
  - Normally TCP does not send an ACK the instant it receives data. Instead, it delays the ACK, hoping to have data going in the same direction as the ACK, so the ACK can be sent along with the data. This is sometimes called having the ACK *piggyback* with the data.
  - Most implementations use a 200-ms delay -- that is, TCP will delay an ACK up to 200ms to see if there is data to send with the ACK.
  - Every 200 ms relative to when the kernel was bootstrapped, NOT relative to when the data received.

Nagle Algorithm

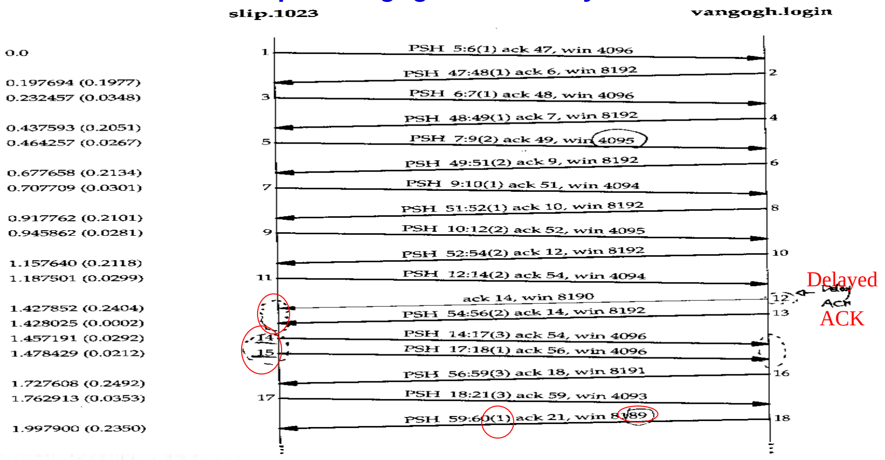
- **Small packets called *tinygram*: 41-bytes packets:**
  - 20 : IP header
  - 20: TCP header
  - 1: data
- **The effects of tinygrams:**
  - normally not a problem on LANs
  - add to congestion on wide area networks

Nagle Algorithm (Cont.)

- **A simple and elegant solution of tinygrams: Nagle algorithm:**
  - When a TCP connection has outstanding data that has not yet been acknowledged, small segments cannot be sent until the outstanding data is acknowledged. (i.e., window size = 1)
- **The beauty of this algorithm: self-clocking**
  - the faster the ACKs come back, the faster the data is sent

Nagle Algorithm (Cont.)

- Examination from slip to vangogh.cs.berkeley:


slip.1023
vangogh.login
0.0
0.197694 (0.1977)
0.232457 (0.0348)
0.437593 (0.2051)
0.464257 (0.0267)
0.677658 (0.2134)
0.707709 (0.0301)
0.917762 (0.2101)
0.945862 (0.0281)
1.157640 (0.2118)
1.187501 (0.0299)
1.427852 (0.2404)
1.428025 (0.0002)
1.457191 (0.0292)
1.478429 (0.0212)
1.727608 (0.2492)
1.762913 (0.0353)
1.997900 (0.2350)
1 PSH 5:6(1) ack 47, win 4096
2 PSH 47:48(1) ack 6, win 8192
3 PSH 6:7(1) ack 48, win 4096
4 PSH 48:49(1) ack 7, win 8192
5 PSH 7:9(2) ack 49, win 4095
6 PSH 49:51(2) ack 9, win 8192
7 PSH 9:10(1) ack 51, win 4094
8 PSH 51:52(1) ack 10, win 8192
9 PSH 10:12(2) ack 52, win 4095
10 PSH 52:54(2) ack 12, win 8192
11 PSH 12:14(2) ack 54, win 4094
12 ack 14, win 8190
13 PSH 54:56(2) ack 14, win 8192
14 PSH 14:17(3) ack 54, win 4096
15 PSH 17:18(1) ack 56, win 4096
16 PSH 56:59(3) ack 18, win 8191
17 PSH 18:21(3) ack 59, win 4093
18 PSH 59:60(1) ack 21, win 8189
Delayed ACK

Nagle Algorithm (Cont.)

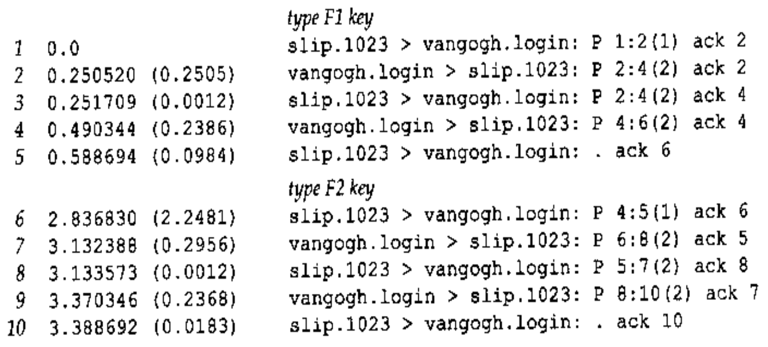
- Things we notice:
  - the lack of delayed ACKs from slip to vangogh: on this example there is always data ready to send before the delayed ACK timer expires
  - the various amounts of data being sent from the left to the right: by the Nagle algorithm
  - the 14th, 15th segments response the 12th, 13th segments, that's also obey the Nagle algorithm
  - the 12th segment is a delayed ACK, for without data
  - the 18th segment echoed only 1 byte: this was done by the TCP module in the kernel, and the data not up to the application of the server yet; the advertised window showed 8189 (not 8191) also indicated the same thing

Disable the Nagle Algorithm

- There are times when the Nagle algorithm needs to be turned off:
  - the X Window System server: small messages (mouse movements) must be delivered without delay to provide real-time feedback for interactive users doing certain operations
  - terminal's special function keys

An Example (with the Nagle Algorithm)

- From Slip to vangogh.cs.berkeley.edu
- type the F1 function key and via versa..



```
                                 type F1 key
 1   0.0                         slip.1023 > vangogh.login: P 1:2(1) ack 2
 2   0.250520 (0.2505)           vangogh.login > slip.1023: P 2:4(2) ack 2
 3   0.251709 (0.0012)           slip.1023 > vangogh.login: P 2:4(2) ack 4
 4   0.490344 (0.2386)           vangogh.login > slip.1023: P 4:6(2) ack 4
 5   0.588694 (0.0984)           slip.1023 > vangogh.login: . ack 6
                                 type F2 key
 6   2.836830 (2.2481)           slip.1023 > vangogh.login: P 4:5(1) ack 6
 7   3.132388 (0.2956)           vangogh.login > slip.1023: P 6:8(2) ack 5
 8   3.133573 (0.0012)           slip.1023 > vangogh.login: P 5:7(2) ack 8
 9   3.370346 (0.2368)           vangogh.login > slip.1023: P 8:10(2) ack 7
10   3.388692 (0.0183)           slip.1023 > vangogh.login: . ack 10
```

Figure 19.5 Watching the Nagle algorithm when typing characters that generate multiple bytes of data.

An Example (with the Nagle Algorithm)

- (continued…)

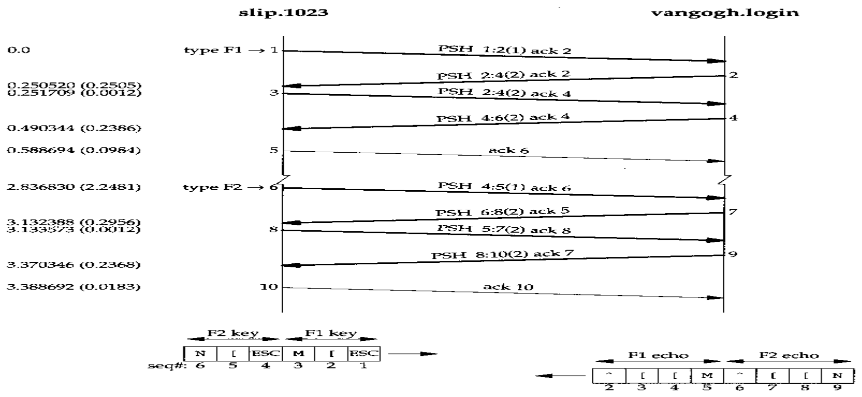
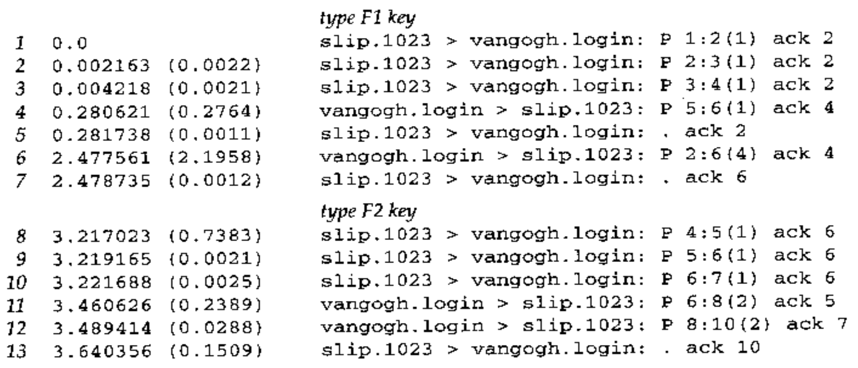


Figure 19.6 Time line for Figure 19.5 (watching the Nagle algorithm).

An Example (without the Nagle algorithm)

- The same command typing as previous one (but disable the Nagle algorithm):

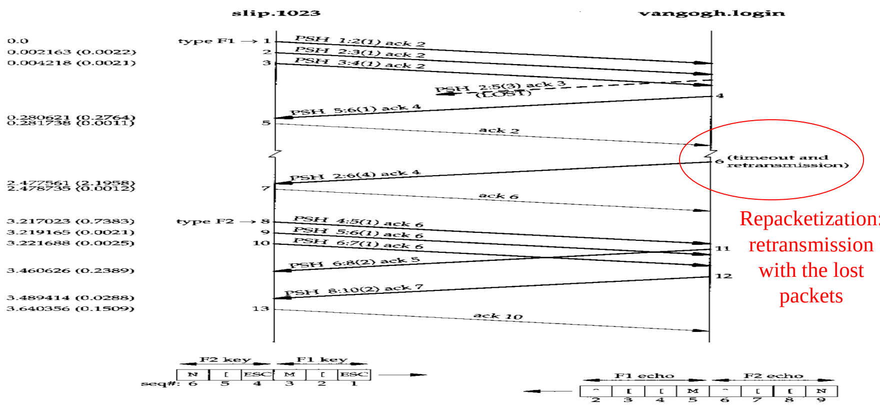


```
                                  type F1 key
 1  0.0                           slip.1023 > vangogh.login: P 1:2(1) ack 2
 2  0.002163  (0.0022)            slip.1023 > vangogh.login: P 2:3(1) ack 2
 3  0.004218  (0.0021)            slip.1023 > vangogh.login: P 3:4(1) ack 2
 4  0.280621  (0.2764)            vangogh.login > slip.1023: P 5:6(1) ack 4
 5  0.281738  (0.0011)            slip.1023 > vangogh.login: . ack 2
 6  2.477561  (2.1958)            vangogh.login > slip.1023: P 2:6(4) ack 4
 7  2.478735  (0.0012)            slip.1023 > vangogh.login: . ack 6
                                  type F2 key
 8  3.217023  (0.7383)            slip.1023 > vangogh.login: P 4:5(1) ack 6
 9  3.219165  (0.0021)            slip.1023 > vangogh.login: P 5:6(1) ack 6
10  3.221688  (0.0025)            slip.1023 > vangogh.login: P 6:7(1) ack 6
11  3.460626  (0.2389)            vangogh.login > slip.1023: P 6:8(2) ack 5
12  3.489414  (0.0288)            vangogh.login > slip.1023: P 8:10(2) ack 7
13  3.640356  (0.1509)            slip.1023 > vangogh.login: . ack 10
```

Figure 19.7 Disabling the Nagle algorithm during an Rlogin session.

An Example (without the Nagle algorithm)

- (continued..)



Repacketization: retransmission with the lost packets

Figure 19.8 Time line for Figure 19.7 (Nagle algorithm disabled).

Window Size Advertisements

- **In figure 19.4:**
  - slip advertises a window of 4096
  - vangogh advertises a windows of 8192
  - Segment 5 advertises a window of 4095 bytes => still 1 byte in the TCP buffer for the application (the Rlogin client) to read
  - The server normally advertises a window of 8192 bytes, because the server’s TCP has nothing to send until the Rlogin server reads the received data and echoes it. The data from the server is sent after the Rlogin server has read its input from the client.
  - The client TCP, on the other hand, often has data to send when the ACK arrives, since it’s buffering the received characters just waiting for the ACK. When the client TCP sends the buffered data, the Rlogin client has not had a chance to read the data received from the server, so the client’s advertised window is less than 4096.

Summary

- Interactive data
- Tinygrams
- Delayed acknowledge (piggyback)
- Nagle algorithm