8.13 Lack of Flow Control with UDP

We now examine the effect of UDP not having any flow control. First, we modify our `dg_cli` function to send a fixed number of datagrams. It no longer reads from standard input. Figure 8.19 shows the new version. This function writes 2,000 1,400-byte UDP datagrams to the server.

We next modify the server to receive datagrams and count the number received. This server no longer echoes datagrams back to the client. Figure 8.20 shows the new `dg_echo` function. When we terminate the server with our terminal interrupt key (`SIGINT`), it prints the number of received datagrams and terminates.

Figure 8.19 `dg_cli` function that writes a fixed number of datagrams to the server.

udpcliserv/dgcliloop1.c

```
#include     "unp.h"

#define NDG      2000          /*  datagrams to send */
#define DGLEN    1400          /*  length of each datagram */

void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
    int     i;
    char    sendline[DGLEN];

    for (i = 0; i < NDG; i++) {
        Sendto(sockfd, sendline, DGLEN, 0, pservaddr, servlen);
    }
}
```

Figure 8.20 `dg_echo` function that counts received datagrams.

udpcliserv/dgecholoop1.c

```
#include     "unp.h"

static void recvfrom_int(int);
static int count;

void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
    socklen_t len;
    char    mesg[MAXLINE];

    Signal(SIGINT, recvfrom_int);

    for ( ; ; ) {
        len = clilen;
        Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

        count++;
    }
}

static void
recvfrom_int(int signo)
{
    printf("\nreceived %d datagrams\n", count);
    exit(0);
}
```

We now run the server on the host `freebsd`, a slow SPARCStation. We run the client on the RS/6000 system `aix`, connected directly with 100Mbps Ethernet. Additionally, we run `netstat -s` on the server, both before and after, as the statistics that are output tell us how many datagrams were lost. Figure 8.21 shows the output on the server.

Figure 8.21 Output on server host.

```
freebsd % netstat -s -p udp
udp:
        71208 datagrams received
        0 with incomplete header
        0 with bad data length field
        0 with bad checksum
        0 with no checksum
        832 dropped due to no socket
        16 broadcast/multicast datagrams dropped due to no socket
        1971 dropped due to full socket buffers
        0 not for hashed pcb
        68389 delivered
        137685 datagrams output
freebsd % udpserv06             start our server
                                                                         we run the client here
     ^C                              we type our interrupt key after the client is finished
received 30 datagrams
freebsd % netstat -s -p udp
udp:
        73208 datagrams received
        0 with incomplete header
        0 with bad data length field
        0 with bad checksum
        0 with no checksum
        832 dropped due to no socket
        16 broadcast/multicast datagrams dropped due to no socket
        3941 dropped due to full socket buffers
        0 not for hashed pcb
        68419 delivered
        137685 datagrams output
```

The client sent 2,000 datagrams, but the server application received only 30 of these, for a 98% loss rate. There is *no* indication whatsoever to the server application or to the client application that these datagrams were lost. As we have said, UDP has no flow control and it is unreliable. It is trivial, as we have shown, for a UDP sender to overrun the receiver.

If we look at the `netstat` output, the total number of datagrams received by the server host (not the server application) is 2,000 (73,208 - 71,208). The counter "dropped due to full socket buffers" indicates how many datagrams were received by UDP but were discarded because the receiving socket's receive queue was full (p. 775 of TCPv2). This value is 1,970 (3,491 - 1,971), which when added to the counter output by the application (30), equals the 2,000 datagrams received by the host. Unfortunately, the `netstat` counter of the number dropped due to a full socket buffer is systemwide. There is no way to determine which applications (e.g., which UDP ports) are affected.

The number of datagrams received by the server in this example is not predictable. It depends on many factors, such as the network load, the processing load on the client host, and the processing load on the server host.

If we run the same client and server, but this time with the client on the slow Sun and the server on the faster RS/6000, no datagrams are lost.

```
aix % udpserv06

^?                                                                     we type our interrupt key after the client is finished

received 2000 datagrams
```

UDP Socket Receive Buffer

The number of UDP datagrams that are queued by UDP for a given socket is limited by the size of that socket's receive buffer. We can change this with the `SO_RCVBUF` socket option, as we described in Section 7.5. The default size of the UDP socket receive buffer under FreeBSD is 42,080 bytes, which allows room for only 30 of our 1,400-byte datagrams. If we increase the size of the socket receive buffer, we expect the server to receive additional datagrams. Figure 8.22 shows a modification to the `dg_echo` function from Figure 8.20 that sets the socket receive buffer to 240 KB.

Figure 8.22 `dg_echo` function that increases the size of the socket receive queue.

udpcliserv/dgecholoop2.c

```
#include    "unp.h"

static void recvfrom_int(int);
static int count;

void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
    int     n;
    socklen_t len;
    char    mesg[MAXLINE];

    Signal(SIGINT, recvfrom_int);
    n = 220 * 1024;
    Setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &n, sizeof(n));

    for ( ; ; ) {
        len = clilen;
        Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

        count++;
    }
}

static void
recvfrom_int(int signo)
{
    printf("\nreceived %d datagrams\n", count);
    exit(0);
}
```

If we run this server on the Sun and the client on the RS/6000, the count of received datagrams is now 103. While this is slightly better than the earlier example with the default socket receive buffer, it is no panacea.

> Why do we set the receive socket buffer size to 220 x 1,024 in Figure 8.22? The maximum size of a socket receive buffer in FreeBSD 5.1 defaults to 262,144 bytes (256 x 1,024), but due to the buffer allocation policy (described in Chapter 2 of TCPv2), the actual limit is 233,016 bytes. Many earlier systems based on 4.3BSD restricted the size of a socket buffer to around 52,000 bytes.

[Team LiB] PREVIOUS NEXT