EL5373

INTERNET ARCHITECTURE AND PROTOCOLS

Teng Zhang

0536650

tz624@students.poly.edu

Workstation: FENCHI

MAC: 0:14:6c:2e:66:8

LAB REPORT 5

[3 Pages]

Exercise 1

ANS1:
Sock *host* echo :send a packet to the port 7 of remote host and request to send back a same packet.
Sock –s 5555 : operate as a server using port 5555.
Sock –I –n3 –w2048 host 5555 :send a packet with the size of 2048 divided into three fragments to the port 5555 of the host.

Exercise 3

ANS:
Netstat output:

```
Kernel Interface table
Iface      MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500   0   46588      0      0      0    7608      0      0      0 BMRU
lo       16436   0  198153      0      0      0  198153      0      0      0 LRU
```

Tcpdump output:

```
20:32:34.396562 arp reply fenchi is-at 0:14:6c:2e:66:8
20:32:34.397883 fenchi.echo > 128.238.66.104.32769: udp 500 (DF)
20:32:39.388651 arp who-has 128.238.66.104 tell fenchi
20:32:49.318985 fenchi.32772 > 128.238.66.104.echo: udp 500 (DF)
```

Sock command:

```
[guest@fenchi TengZhang_0536650]$ sock -u -i -n1 -w1472 128.238.66.106 echo

20:43:59.761312 fenchi.32772 > 128.238.66.106.echo: udp 1472 (DF)
20:43:59.771669 fenchi > 128.238.66.106: icmp: fenchi udp port 32772
unreachable [tos 0xc0]

[guest@fenchi TengZhang_0536650]$ sock -u -i -n1 -w1473 128.238.66.106 echo

20:45:37.624045 fenchi.32772 > 128.238.66.106.echo: udp 1473 (frag
44494:1480@0+)
20:45:37.627440 fenchi > 128.238.66.106: icmp: fenchi udp port 32772
unreachable [tos 0xc0]
```

As above, the Netstat output show that the MTU met is 1500, but if the value of size is 1473, the UDP datagram is sent with fragmentation. If the value is 1472, there is no fragmentation. So actually the maximum value of size which the UDP datagram can be sent without fragmentation is 1472. Exactly, there are 28 bytes used for IP header(20 bytes) and UDP header(8 bytes).

Exercise 4

ANS:

Tcpdump output:

```
20:39:20.782969 fenchi > 128.238.66.104: udp (frag 12187:1128@8880)
20:39:20.782997 fenchi > 128.238.66.104: udp (frag 12187:1480@7400+)
20:39:20.783005 fenchi > 128.238.66.104: udp (frag 12187:1480@5920+)
20:39:20.783012 fenchi > 128.238.66.104: udp (frag 12187:1480@4440+)
20:39:20.783020 fenchi > 128.238.66.104: udp (frag 12187:1480@2960+)
20:39:20.783027 fenchi > 128.238.66.104: udp (frag 12187:1480@1480+)
20:39:20.783034 fenchi.32772 > 128.238.66.104.echo: udp 10000 (frag
12187:1480@0+)
20:39:20.793605 fenchi > 128.238.66.104: icmp: fenchi udp port 32772
unreachable [tos 0xc0]
20:39:25.778649 arp who-has 128.238.66.104 tell fenchi
```

The total size of received package is 1480*6+1128=10008. The extra 8 bytes is the UDP header which in the first package. It is impossible that every package has a UDP header because the total size will be 10056 instead of 10008 if it is true.

Exercise 5

ANS:

Command field:

```
[guest@fenchi TengZhang_0536650]$ sock -u -i -n1 -w65507 128.238.66.106
echo

20:53:56.952085 fenchi > 128.238.66.106: udp (frag 44500:395@65120)
20:53:56.952107 fenchi > 128.238.66.106: udp (frag 44500:1480@63640+)
20:53:56.952113 fenchi > 128.238.66.106: udp (frag 44500:1480@62160+)
20:53:56.952119 fenchi > 128.238.66.106: udp (frag 44500:1480@60680+)
20:53:56.952125 fenchi > 128.238.66.106: udp (frag 44500:1480@59200+)
20:53:56.952132 fenchi > 128.238.66.106: udp (frag 44500:1480@57720+)
20:53:56.952138 fenchi > 128.238.66.106: udp (frag 44500:1480@56240+)
20:53:56.952143 fenchi > 128.238.66.106: udp (frag 44500:1480@54760+)
20:53:56.952148 fenchi > 128.238.66.106: udp (frag 44500:1480@53280+)
20:53:56.956985 fenchi > 128.238.66.106: udp (frag 44500:1480@51800+)
20:53:56.956999 fenchi > 128.238.66.106: udp (frag 44500:1480@50320+)
20:53:56.957002 fenchi > 128.238.66.106: udp (frag 44500:1480@48840+)
20:53:56.957005 fenchi > 128.238.66.106: udp (frag 44500:1480@47360+)
20:53:56.961907 fenchi > 128.238.66.106: udp (frag 44500:1480@45880+)
20:53:56.961910 fenchi > 128.238.66.106: udp (frag 44500:1480@44400+)
```

```
20:53:56.961913 fenchi > 128.238.66.106: udp (frag 44500:1480@42920+)
20:53:56.961916 fenchi > 128.238.66.106: udp (frag 44500:1480@41440+)
20:53:56.966831 fenchi > 128.238.66.106: udp (frag 44500:1480@39960+)
20:53:56.966835 fenchi > 128.238.66.106: udp (frag 44500:1480@38480+)
20:53:56.966837 fenchi > 128.238.66.106: udp (frag 44500:1480@37000+)
20:53:56.966840 fenchi > 128.238.66.106: udp (frag 44500:1480@35520+)
20:53:56.971755 fenchi > 128.238.66.106: udp (frag 44500:1480@34040+)
20:53:56.971759 fenchi > 128.238.66.106: udp (frag 44500:1480@32560+)
20:53:56.971762 fenchi > 128.238.66.106: udp (frag 44500:1480@31080+)
20:53:56.971765 fenchi > 128.238.66.106: udp (frag 44500:1480@29600+)
20:53:56.976680 fenchi > 128.238.66.106: udp (frag 44500:1480@28120+)
20:53:56.976683 fenchi > 128.238.66.106: udp (frag 44500:1480@26640+)
20:53:56.976685 fenchi > 128.238.66.106: udp (frag 44500:1480@25160+)
20:53:56.976688 fenchi > 128.238.66.106: udp (frag 44500:1480@23680+)
20:53:56.981605 fenchi > 128.238.66.106: udp (frag 44500:1480@22200+)
20:53:56.981610 fenchi > 128.238.66.106: udp (frag 44500:1480@20720+)
20:53:56.981612 fenchi > 128.238.66.106: udp (frag 44500:1480@19240+)
20:53:56.981615 fenchi > 128.238.66.106: udp (frag 44500:1480@17760+)
20:53:56.986529 fenchi > 128.238.66.106: udp (frag 44500:1480@16280+)
20:53:56.986532 fenchi > 128.238.66.106: udp (frag 44500:1480@14800+)
20:53:56.986535 fenchi > 128.238.66.106: udp (frag 44500:1480@13320+)
20:53:56.986537 fenchi > 128.238.66.106: udp (frag 44500:1480@11840+)
20:53:56.991457 fenchi > 128.238.66.106: udp (frag 44500:1480@10360+)
20:53:56.991470 fenchi > 128.238.66.106: udp (frag 44500:1480@8880+)
20:53:56.991473 fenchi > 128.238.66.106: udp (frag 44500:1480@7400+)
20:53:57.073086 fenchi > 128.238.66.106: udp (frag 44500:1480@5920+)
20:53:57.073105 fenchi > 128.238.66.106: udp (frag 44500:1480@4440+)
20:53:57.073113 fenchi > 128.238.66.106: udp (frag 44500:1480@2960+)
20:53:57.073121 fenchi > 128.238.66.106: udp (frag 44500:1480@1480+)
20:53:57.073129 fenchi.32772 > 128.238.66.106.echo: udp 65507 (frag
44500:1480@0+)
20:54:01.948653 arp who-has 128.238.66.106 tell fenchi
20:54:09.087447 arp reply fenchi is-at 0:14:6c:2e:66:8
```

```
[guest@fenchi TengZhang_0536650]$ sock -u -i -n1 -w65508 128.238.66.106
echo
write returned -1, expected 65508: Message too long
```

As above, the system can send the UDP size of 65507, but when the value changed to 65508, the system can not to send it. So the size of UDP datagram with fragmentation

allowed sending by system is 65507 which is 2^16-20(IP header)-8(UDP header).

Exercise 7

ANS1:

The transfer time of ftp is from `21:12:31.067313` to `21:12:32.086062`, the time spent for data transfer is 1.018749s.ftp window showed that:

```
1031850 bytes received in 1.18 secs (8.5e+02 Kbytes/sec)
```

So the value calculated and the value displayed in ftp window are not consistent, because the 1.18s contains the time of establishing the TCP connection.

ANS2:

The transfer time of tftp is from `21:16:26.486830` to `21:16:29.915644`, the time spent for data transfer is 3.428814s.

tftp window showed that:

```
Received 1052514 bytes in 3.5 seconds
```

The two time is almost the same, because TFTP uses UDP to communicate, it does not need the time to build the connection.

FTP is faster, because TFTP uses stop-and-wait protocol. And FTP uses TCP which can send multiple packages at a time.

Exercise 8

```
21:27:52.261317 fenchi.32772 > 128.238.66.106.tftp:  21 RRQ
"small.dum" (DF)
21:27:52.263427 128.238.66.106.32772 > fenchi.32772: udp 516 (DF)
21:27:52.265364 fenchi.32772 > 128.238.66.106.32772: udp 4 (DF)
21:27:52.266260 128.238.66.106.32772 > fenchi.32772: udp 496 (DF)
21:27:52.266500 fenchi.32772 > 128.238.66.106.32772: udp 4 (DF)
```

<table>
<tr><td>Opcode(1=RRQ,2=WRQ)</td><td>filename</td><td>0</td><td>mode</td><td>0</td></tr>
<tr><td></td><td></td><td colspan="3" rowspan="5"></td></tr>
<tr><td>Opcode(3=data)</td><td>Block number</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Opcode(4=ACK)</td><td>Block number</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Opcode(5=error)</td><td>Block number</td><td colspan="2">Error message</td><td>0</td></tr>
</table>

ANS1:

The TFTP uses UDP port 69 for TFTP control messages. That means the RRQ must be sent to the UDP port 69, then the port number changed to an ephemeral port to transfer the data according to the parameters of the networking stack.

ANS2:

TFTP is designed for small and infrequent file transfers, where throughput is not a major concern. TFTP has a security hole that is lack of a login procedure.

ANS3:

TFTP is an application protocol. The whole package will be break into several packages which are less than 512 bytes. Then we can transfer the file that larger than UDP datagram size.

Exercise 9

ANS1:

The host use the ephemeral port number 37341 and 37350 while the remote server use the TCP port 21 and TCP port 20 and ephemeral port number 50815.

ANS2:

The client can know the transfer status or request to add more transfer task to the downloading list without interrupt current transferring if data transfer and controlling are separately used two different connections.

Exercise 10

```
[guest@fenchi TengZhang_0536650]$ ftp -d 128.238.66.104
Connected to 128.238.66.104 (128.238.66.104).
220 (vsFTPd 1.1.3)
Name (128.238.66.104:guest): guest
---> USER guest                          //username on server
331 Please specify the password.         //username ok, password required
Password:
---> PASS XXXX                           //password on server
230 Login successful. Have fun.          //password ok
---> SYST                                 //ender the system
215 UNIX Type: L8
Remote system type is UNIX.
Using binary mode to transfer files.

ftp> dir /home/LAB/small.dum
ftp: setsockopt (ignored): Permission denied    //socket option
---> PASV                                       //passive mode
227 Entering Passive Mode (128,238,66,104,73,5)//remote host
---> LIST /home/LAB/small.dum                   //list the file
150 Here comes the directory listing.           //connection is established
-rw-r--r--    1 0        0             990 Mar 23  2004 small.dum //file information
226 Directory send OK.
ftp> quit
---> QUIT
221 Goodbye.
```

ANS1: To open a data connection, the client first chooses an ephemeral port number(73,5) and then sends the port number to the server using the PORT command via the control

connection. Then the server issues an active open to that port on the client host. File transfer begins after the data connection is set up.
ANS2: the server send the response on the control connection.