

# TCP/IP 通訊協定及應用

Spring 2002

中央大學 吳曉光博士

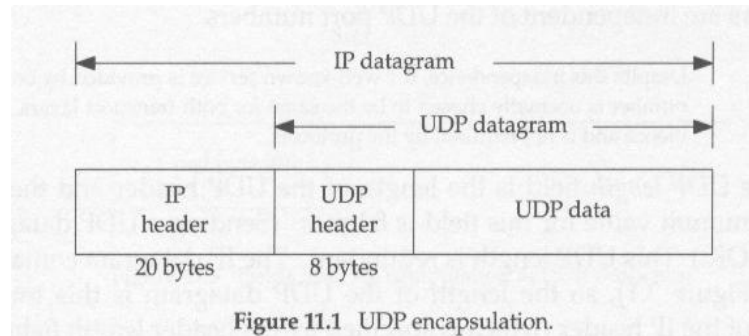
<http://wmlab.csie.ncu.edu.tw/course/tcp>

*We  
provide  
Wireless  
Wireless Network & Multimedia Laboratory  
Solution*

# Chapter 11: UDP: User Datagram Protocol

# Introduction

## ◆ UDP encapsulation:

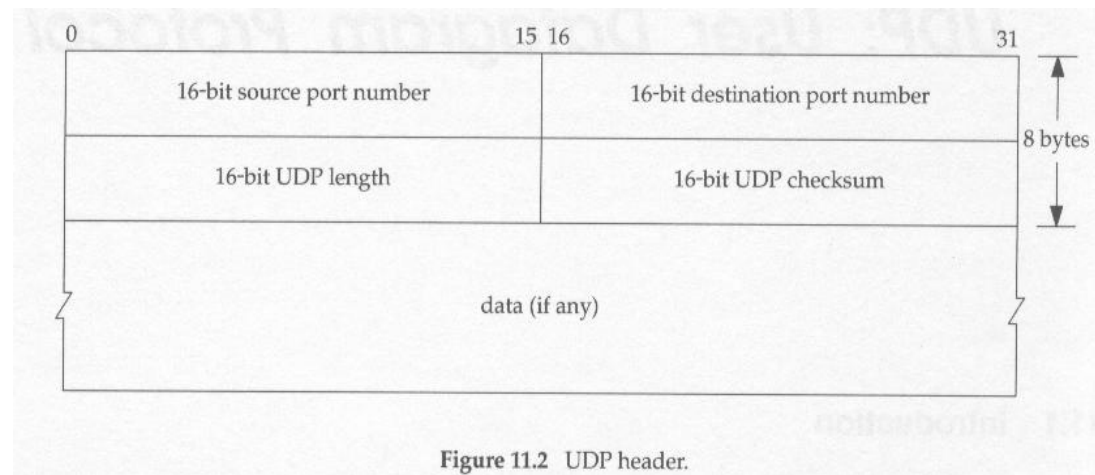


- UDP is a simple, datagram-oriented, transport layer protocol
- UDP provides no reliability

## ◆ UDP Header

- The port numbers identify the sending process and the receiving process
- UDP length field is the length of the UDP header and the UDP data in bytes. The minimum value is 8 bytes

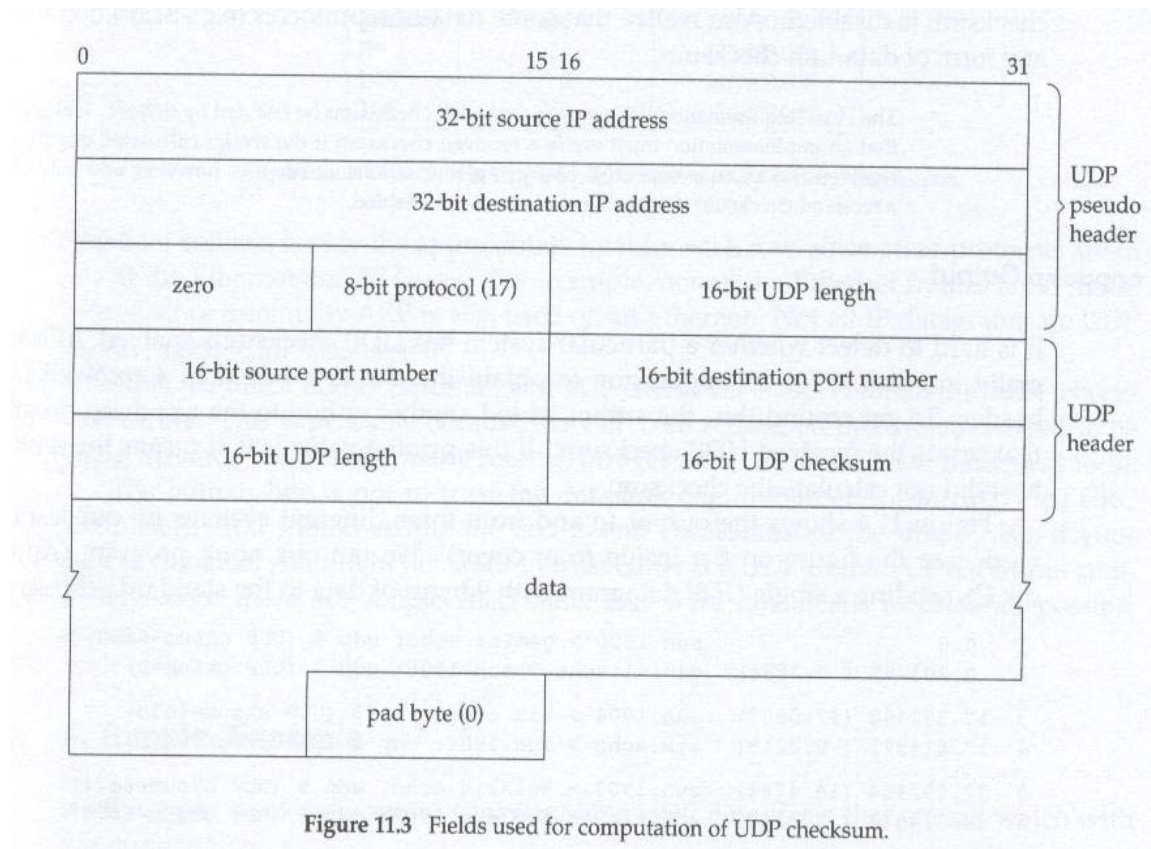
# UDP Header



- ◆ Compare TCP Checksum and UDP Checksum
  - TCP checksum is mandatory
  - UDP checksum is optional
- ◆ Differences between UDP checksum and IP checksum
  - UDP datagram can be an odd number of bytes
  - Both UDP and TCP include a 12-byte pseudo-header purpose to let UDP double-check the data has arrived at the correct destination

# UDP Checksum

- UDP checksum is an end-to-end checksum. If the receiver detects a checksum error, UDP datagram is discarded



# UDP Checksum

- value 0 means the sending host did not calculate the checksum
- The UDP checksums cannot detect an error that swaps to of the 16-bit values

```

1  0.0      sun.1900 > gemini.echo: udp 9 (UDP cksum=6e90)
2  0.303755 ( 0.3038) gemini.echo > sun.1900: udp 9 (UDP cksum=0)
3  17.392480 (17.0887) sun.1904 > aix.echo: udp 9 (UDP cksum=6e3b)
4  17.614371 ( 0.2219) aix.echo > sun.1904: udp 9 (UDP cksum=6e3b)
5  32.092454 (14.4781) sun.1907 > solaris.echo: udp 9 (UDP cksum=6e74)
6  32.314378 ( 0.2219) solaris.echo > sun.1907: udp 9 (UDP cksum=6e74)

```

Figure 11.4 tcpdump output to see whether other hosts enable UDP checksum.

## ◆ Some Statistics

Layer	Number of checksum errors	Approximate total number of packets
Ethernet	446	170,000,000
IP	14	170,000,000
UDP	5	140,000,000
TCP	350	30,000,000

Figure 11.5 Counts of corrupted packets detected by various checksums.

# A Simple Example

tcpdump.

```
bsdi % sock -v -u -i -n4 svr4 discard
connected on 140.252.13.35.1108 to 140.252.13.34.9

bsdi % sock -v -u -i -n4 -w0 svr4 discard
connected on 140.252.13.35.1110 to 140.252.13.34.9
```

to be written. Figure 11.6 shows the tcpdump output for both commands.

```
1 0.0          bsdi.1108 > svr4.discard: udp 1024
2 0.002424 ( 0.0024) bsdi.1108 > svr4.discard: udp 1024
3 0.006210 ( 0.0038) bsdi.1108 > svr4.discard: udp 1024
4 0.010276 ( 0.0041) bsdi.1108 > svr4.discard: udp 1024

5 41.720114 (41.7098) bsdi.1110 > svr4.discard: udp 0
6 41.721072 ( 0.0010) bsdi.1110 > svr4.discard: udp 0
7 41.722094 ( 0.0010) bsdi.1110 > svr4.discard: udp 0
8 41.723070 ( 0.0010) bsdi.1110 > svr4.discard: udp 0
```

Figure 11.6 tcpdump output when UDP datagrams are sent in one direction.

- There is no communication between the sender and receiver before the first datagram is sent
- There are no acknowledgments by the receiver when the data is received



# IP Fragmentation

## ◆ Example:

```
bsdi % sock -u -i -n1 -w1471 svr4 discard
bsdi % sock -u -i -n1 -w1472 svr4 discard
bsdi % sock -u -i -n1 -w1473 svr4 discard
bsdi % sock -u -i -n1 -w1474 svr4 discard
```

Figure 11.7 shows the corresponding tcpdump output.

```
1  0.0          bsd1.1112 > svr4.discard: udp 1471
2  21.008303 (21.0083) bsd1.1114 > svr4.discard: udp 1472
3  50.449704 (29.4414) bsd1.1116 > svr4.discard: udp 1473 (frag 26304:1480@0+)
4  50.450040 ( 0.0003) bsd1 > svr4: (frag 26304:1@1480)
5  75.328650 (24.8786) bsd1.1118 > svr4.discard: udp 1474 (frag 26313:1480@0+)
6  75.328982 ( 0.0003) bsd1 > svr4: (frag 26313:2@1480)
```

Figure 11.7 Watching fragmentation of UDP datagrams.

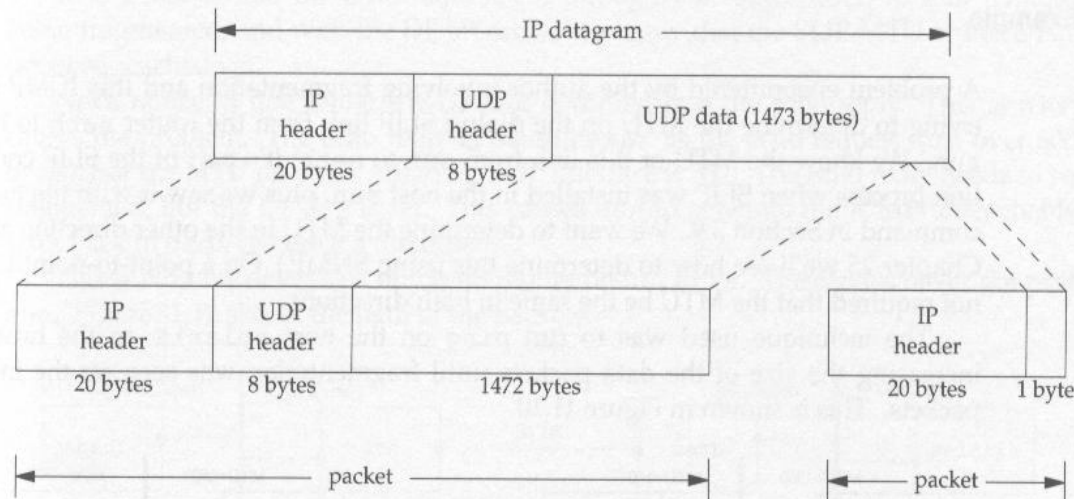


Figure 11.8 Example of UDP fragmentation.



# ICMP Unreachable Error (Fragmentation Required)



- The port numbers only occurs in the first fragment
  - Any transport layer header appears only in the first fragment
- ◆ When occurs the ICMP unreachable error
- a datagram that requires fragmentation, but the DF flag is turned on

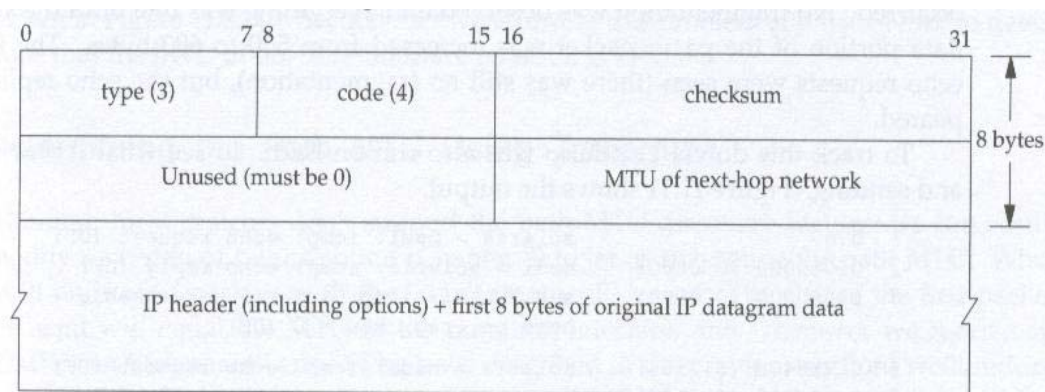
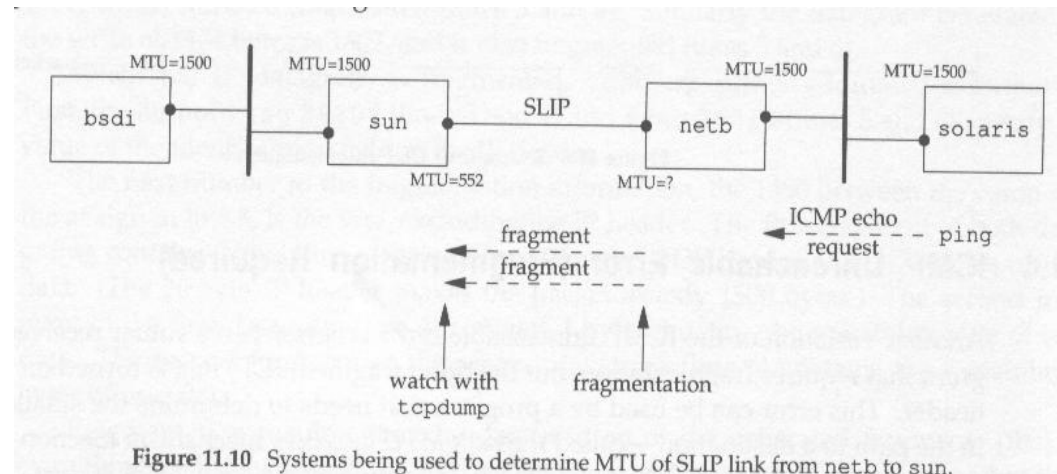


Figure 11.9 ICMP unreachable error when fragmentation required but don't fragment bit set.

# ICMP Unreachable Error (Fragmentation Required)

## ◆ Example:



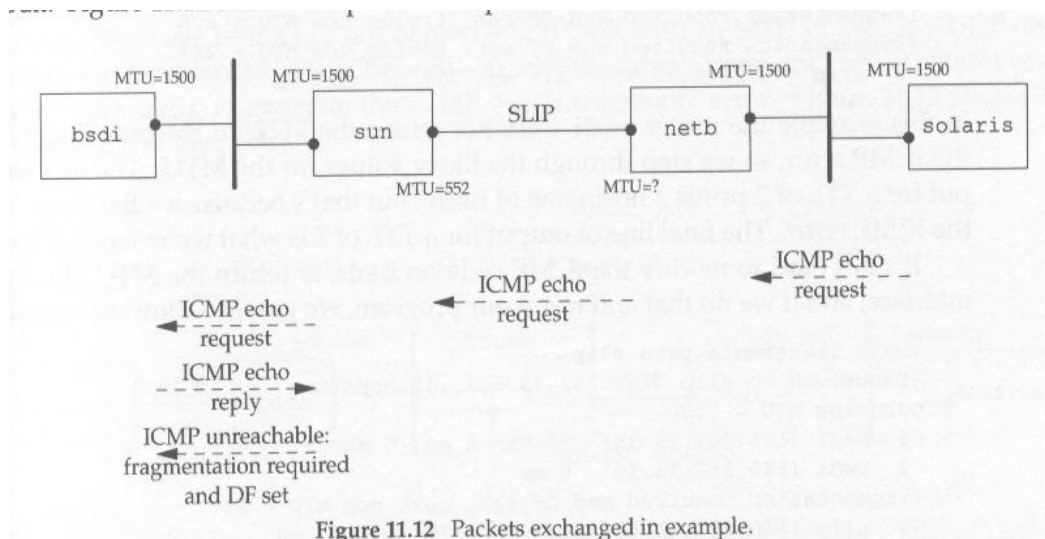
```
1 0.0          solaris > bsd1: icmp: echo request (DF)
2 0.000000 (0.0000) bsd1 > solaris: icmp: echo reply (DF)
3 0.000000 (0.0000) sun > bsd1: icmp: solaris unreachable -
                    need to frag, mtu = 0 (DF)

4 0.738400 (0.7384) solaris > bsd1: icmp: echo request (DF)
5 0.748800 (0.0104) bsd1 > solaris: icmp: echo reply (DF)
6 0.748800 (0.0000) sun > bsd1: icmp: solaris unreachable -
                    need to frag, mtu = 0 (DF)
```

Figure 11.11 tcpdump output for ping of bsd1 from solaris with 600-byte IP datagram.

# ICMP Unreachable Error (Fragmentation Required)

- DF flag is set causes sun to generate the ICMP unreachable error back to bsd1 (where it's discarded)



- ◆ Determining the path MTU using traceroute
  - Whenever we receive an ICMP “can’t fragment” error, we’ll reduce the size of the packet

# Determining the Path MTU Using Traceroute



- ◆ The router bsdi does not return the MTU

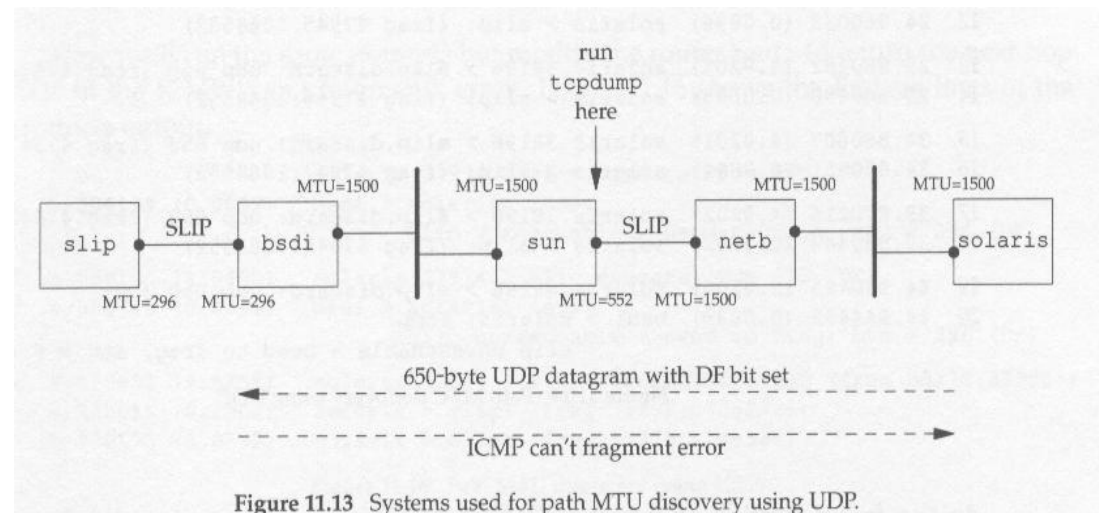
```
sun % traceroute.pmtu slip
traceroute to slip (140.252.13.65), 30 hops max
outgoing MTU = 1500
 1 bsdi (140.252.13.35) 15 ms 6 ms 6 ms
 2 bsdi (140.252.13.35) 6 ms
fragmentation required and DF set, trying new MTU = 1492
fragmentation required and DF set, trying new MTU = 1006
fragmentation required and DF set, trying new MTU = 576
fragmentation required and DF set, trying new MTU = 552
fragmentation required and DF set, trying new MTU = 544
fragmentation required and DF set, trying new MTU = 512
fragmentation required and DF set, trying new MTU = 508
fragmentation required and DF set, trying new MTU = 296
 2 slip (140.252.13.65) 377 ms 377 ms 377 ms
```

- ◆ The ICMP code on bsdi to return the MTU

```
sun % traceroute.pmtu slip
traceroute to slip (140.252.13.65), 30 hops max
outgoing MTU = 1500
 1 bsdi (140.252.13.35) 53 ms 6 ms 6 ms
 2 bsdi (140.252.13.35) 6 ms
fragmentation required and DF set, next hop MTU = 296
 2 slip (140.252.13.65) 377 ms 378 ms 377 ms
```

# Path MTU Discovery with UDP

## ◆ Example:



- The following command generates ten 650-byte UDP datagrams, with a 5-second pause between each datagram:  
`solaris % sock -u -i -n10 -w650 -p5 slip discard`



# Path MTU Discovery with UDP

```

1  0.0          solaris.38196 > slip.discard: udp 650 (DF)
2  0.004218 (0.0042) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 0 (DF)

3  4.980528 (4.9763) solaris.38196 > slip.discard: udp 650 (DF)
4  4.984503 (0.0040) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 0 (DF)

5  9.870407 (4.8859) solaris.38196 > slip.discard: udp 650 (frag 47942:552@0+)
6  9.960056 (0.0896) solaris > slip: (frag 47942:106@552)

7  14.940338 (4.9803) solaris.38196 > slip.discard: udp 650 (DF)
8  14.944466 (0.0041) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 0 (DF)

9  19.890015 (4.9455) solaris.38196 > slip.discard: udp 650 (frag 47944:552@0+)
10 19.950463 (0.0604) solaris > slip: (frag 47944:106@552)

11 24.870401 (4.9199) solaris.38196 > slip.discard: udp 650 (frag 47945:552@0+)
12 24.960038 (0.0896) solaris > slip: (frag 47945:106@552)

13 29.880182 (4.9201) solaris.38196 > slip.discard: udp 650 (frag 47946:552@0+)
14 29.940498 (0.0603) solaris > slip: (frag 47946:106@552)

15 34.860607 (4.9201) solaris.38196 > slip.discard: udp 650 (frag 47947:552@0+)
16 34.950051 (0.0894) solaris > slip: (frag 47947:106@552)

17 39.870216 (4.9202) solaris.38196 > slip.discard: udp 650 (frag 47948:552@0+)
18 39.930443 (0.0602) solaris > slip: (frag 47948:106@552)

19 44.940485 (5.0100) solaris.38196 > slip.discard: udp 650 (DF)
20 44.944432 (0.0039) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 0 (DF)

```

Figure 11.14 Path MTU discovery using UDP.

# Path MTU Discovery with UDP

- ◆ Four fragments generated by the router bsdi

```

arrives (lines 3 and 4 from Figure 11.14).
1 0.0          solaris.38196 > slip.discard: udp 650 (frag 47942:272@0+)
2 0.304513 (0.3045) solaris > slip: (frag 47942:272@272+)
3 0.334651 (0.0301) solaris > slip: (frag 47942:8@544+)
4 0.466642 (0.1320) solaris > slip: (frag 47942:106@552)

```

Figure 11.15 First datagram arriving at host slip from solaris.

- ◆ Three fragments generated by the router bsdi return the next-hop MTU in the ICMP “can’t fragment” error

```

tcpdump output.
1 0.0          solaris.37974 > slip.discard: udp 650 (DF)
2 0.004199 (0.0042) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 296 (DF)
3 4.950193 (4.9460) solaris.37974 > slip.discard: udp 650 (DF)
4 4.954325 (0.0041) bsdi > solaris: icmp:
                        slip unreachable - need to frag, mtu = 296 (DF)
5 9.779855 (4.8255) solaris.37974 > slip.discard: udp 650 (frag 35278:272@0+)
6 9.930018 (0.1502) solaris > slip: (frag 35278:272@272+)
7 9.990170 (0.0602) solaris > slip: (frag 35278:114@544)

```

Figure 11.16 Path MTU discovery using UDP.



# Interaction Between UDP and ARP

## ◆ Example:

```

1 0.0          arp who-has svr4 tell bsdi
2 0.001234 (0.0012)  arp who-has svr4 tell bsdi
3 0.001941 (0.0007)  arp who-has svr4 tell bsdi
4 0.002775 (0.0008)  arp who-has svr4 tell bsdi
5 0.003495 (0.0007)  arp who-has svr4 tell bsdi
6 0.004319 (0.0008)  arp who-has svr4 tell bsdi
7 0.008772 (0.0045)  arp reply svr4 is-at 0:0:c0:c2:9b:26
8 0.009911 (0.0011)  arp reply svr4 is-at 0:0:c0:c2:9b:26
9 0.011127 (0.0012)  bsdi > svr4: (frag 10863:800@7400)
10 0.011255 (0.0001) arp reply svr4 is-at 0:0:c0:c2:9b:26
11 0.012562 (0.0013) arp reply svr4 is-at 0:0:c0:c2:9b:26
12 0.013458 (0.0009) arp reply svr4 is-at 0:0:c0:c2:9b:26
13 0.014526 (0.0011) arp reply svr4 is-at 0:0:c0:c2:9b:26
14 0.015583 (0.0011) arp reply svr4 is-at 0:0:c0:c2:9b:26

```

Figure 11.17 Packet exchange when an 8192-byte UDP datagram is sent on an Ethernet.

- Six ARP requests are generated before the first ARP reply is returned
- Only the last fragment is sent, first five fragments have been discarded
- Unexplained anomaly in output seven ARP replies, not six

# Interaction Between UDP and ARP

- ◆ Why we don't see the ICMP message
  - Most Berkeley derived implementations never generate this error
  - The first fragment which containing the UDP header was never received
- ◆ Maximum UDP Datagram Size
  - Just over 8192 bytes for the maximum size of a UDP datagram that can be read or written
  - Limit the size of an IP datagram to less than 65535 bytes
- ◆ How to deal with received datagram exceeds the size
  - The traditional Berkeley => discarding any excess data
  - The sockets API under SVR4 => does not truncate the datagram
  - The TLI API => Instead a flag is returned

# ICMP Source Quench Error

- This is an error that may be generated by a system (router or host) when it receives datagrams at a rate is too fast to be processed

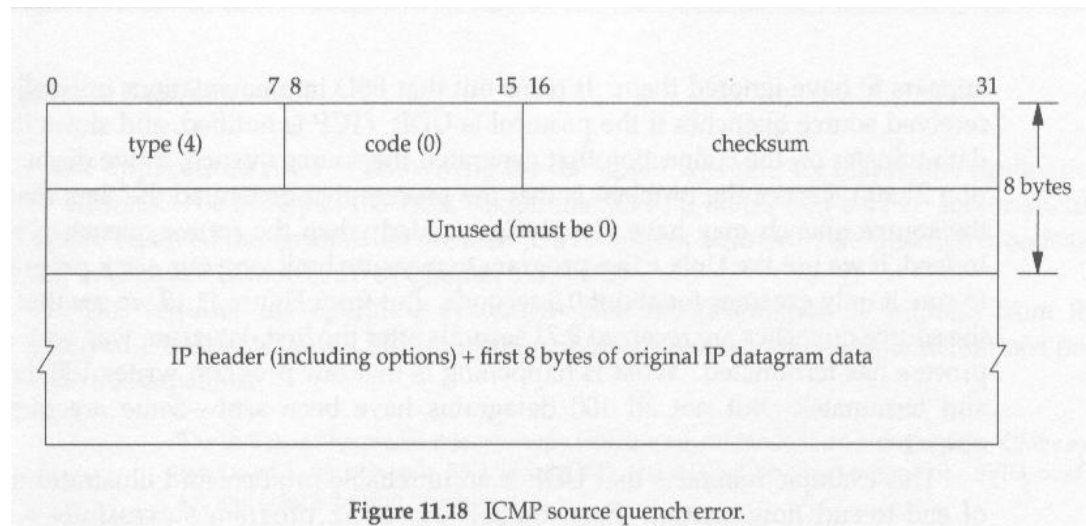


Figure 11.18 ICMP source quench error.

```

1 0.0          bsdi.1403 > solaris.discard: udp 1024
                26 lines that we don't show
27 0.10 (0.00) bsdi.1403 > solaris.discard: udp 1024
28 0.11 (0.01) sun > bsdi: icmp: source quench
29 0.11 (0.00) bsdi.1403 > solaris.discard: udp 1024
30 0.11 (0.00) sun > bsdi: icmp: source quench
                142 lines that we don't show
173 0.71 (0.06) bsdi.1403 > solaris.discard: udp 1024
174 0.71 (0.00) sun > bsdi: icmp: source quench
  
```

Figure 11.19 ICMP source quench from the router sun.

# UDP Server Design

- ◆ Client IP Address and Port Number
  - When an application receives a UDP datagram, it must be told by the operating system who sent the message--the source IP address and port number
- ◆ Destination IP Address
  - Who the datagram was sent to, that is, the destination IP address
- ◆ UDP Input Queue
  - A single server process handles all the client requests on a single UDP port

about 12 seconds, within the 30-second period while the server was sleep.

```

1  0.0          sun.1252 > 140.252.13.63.6666: udp 11
2  2.499184 (2.4992) svr4.1042 > bsdi.6666: udp 14
3  4.959166 (2.4600) sun.1252 > 140.252.13.63.6666: udp 10
4  7.607149 (2.6480) svr4.1042 > bsdi.6666: udp 16
5  10.079059 (2.4719) sun.1252 > 140.252.13.63.6666: udp 12
6  12.415943 (2.3369) svr4.1042 > bsdi.6666: udp 9
  
```

Figure 11.20 tcpdump for UDP datagrams sent by two clients.

# UDP Server Design

- The application is not told when its input queue overflows
- Nothing is sent back to the client to tell it that its datagram was discarded
- UDP input queue is FIFO, ARP input queue was LIFO

## ◆ Restricting Local IP Address

Figure 11.21 shows this scenario.

```

1 0.0          bsdi.1723 > sun.7777: udp 13
2 0.000822 (0.0008)  sun > bsdi: icmp: sun udp port 7777 unreachable
  
```

Figure 11.21 Rejection of UDP datagram caused by server's local address binding.

## ◆ Restricting F

## ◆ Multiple Recipients per Port

Local Address	Foreign Address	Description
<i>localIP.lport</i>	<i>foreignIP.fport</i>	restricted to one client
<i>localIP.lport</i>	<i>*.*</i>	restricted to datagrams arriving on one local interface: <i>localIP</i>
<i>*.lport</i>	<i>*.*</i>	receives all datagrams sent to <i>lport</i>

Figure 11.22 Specification of local and foreign IP addresses and port number for UDP server.

# Summary

- UDP is a simple protocol
- The services it provides to a user process are port numbers and an optional checksum
- Path MTU discovery using Traceroute and UDP
- The ICMP source quench error can be sent by a system that is receiving IP datagrams faster than they can be processed