

---

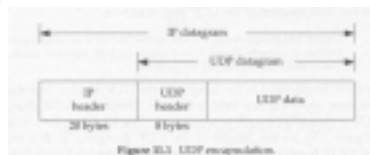
# Chapter 11:

## UDP: User Datagram Protocol

### Introduction

---

#### □ UDP encapsulation:

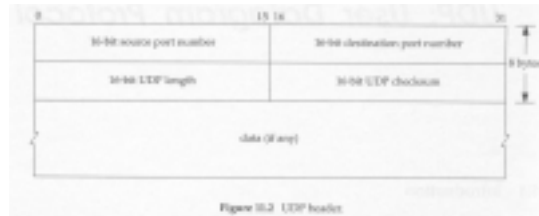


- ❖ UDP is a simple, *datagram-oriented* (different from *stream-oriented*), transport layer protocol
- ❖ UDP provides no reliability

#### □ 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

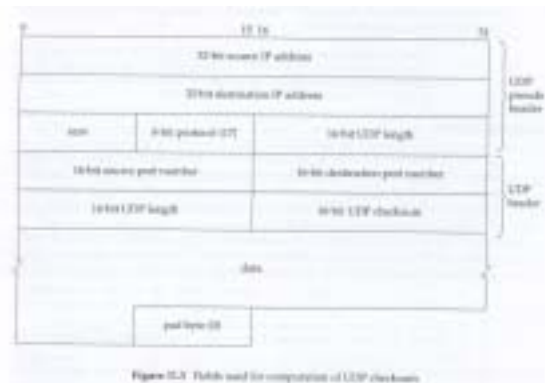
## UDP Header



- ❑ **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 simply discarded.



## UDP Checksum (Cont.)

- ❑ 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

[illegible]

- ## ❑ Some Statistics

Layer	Number of checksum errors	Approximate total number of packets
Internet	446	172,000,000
IP	34	172,000,000
UDP	5	180,000,000
TCP	206	30,000,000

Figure 11.5 Counts of corrupted packets detected by various checksums.

## A Simple Example

```

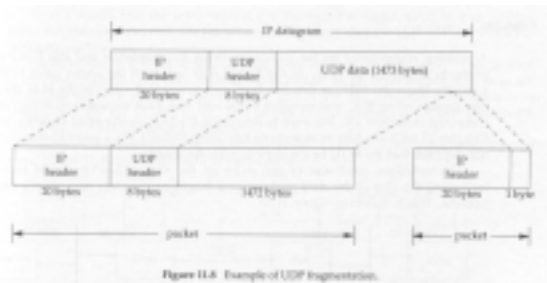
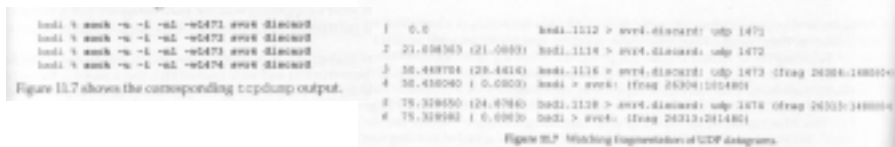
root@kali:~# cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 64 | xargs -n 1 sh -c 'dd if=/dev/urandom of=/dev/null bs=1M count=1000000'
root@kali:~# cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 64 | xargs -n 1 sh -c 'dd if=/dev/urandom of=/dev/null bs=1M count=1000000'
root@kali:~# cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 64 | xargs -n 1 sh -c 'dd if=/dev/urandom of=/dev/null bs=1M count=1000000'

```

- ❖ 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:



## IP Fragmentation (Cont.)

- The physical network layer limits the size of the frame.
- Fragmentation can take place either at the original sending host or at an intermediate router.
- The IP layer at the destination performs the reassembly.
- All IP fragments have the same ID number.
- **Fragment offset:** the offset from the beginning of the original datagram
- One fragment lost: entire datagram must be retransmitted.
- Data portion of a fragment: multiple of 8 bytes (other than the final one)
- The port numbers only occurs in the first fragment.
- Any transport layer header appears only in the first fragment.

## ICMP Unreachable Error (Fragmentation Required)

### ❑ When occurs the ICMP unreachable error

- ❖ a datagram that requires fragmentation, but the DF (don't fragment) flag is turned on



Figure 9.16 ICMPv4 unreachable error when fragmentation required but don't fragment bit set.

## ICMP Unreachable Error (Cont.)

### ❑ Example:

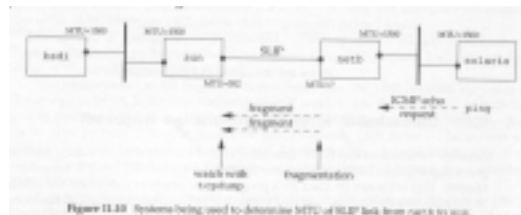


Figure 11.10 Systems being used to determine MTU of SLIP link from host1 to host2.

```

3 0.0          host1 > host2: icmp: echo request (DF)
2 0.000000 (0.0000) host1 > host2: icmp: echo reply (DF)
3 0.000000 (0.0000) host1 > host2: icmp: host2 unreachable -
    send to frag, mtu = 0 (DF)

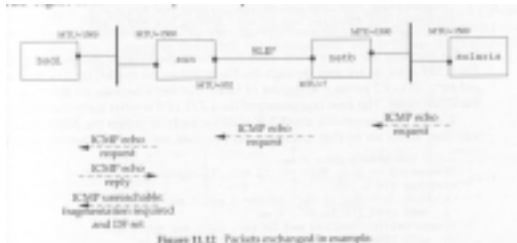
4 0.128400 (0.1284) host2 > host1: icmp: echo request (DF)
5 0.148800 (0.1488) host2 > host1: icmp: echo reply (DF)
6 0.148800 (0.1488) host1 > host2: icmp: host2 unreachable -
    send to frag, mtu = 0 (DF)

```

Figure 11.11 tcpdump output for ping of host1 from host2 with 65536-byte IP datagram.

## ICMP Unreachable Error (Cont.)

- ❑ DF flag is set and causes *sun* to generate the ICMP unreachable error back to *bsdi* (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

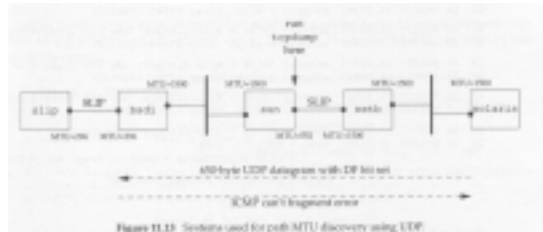
```
sun % traceroute -mto sllp
traceroute to sllp (140.252.13.65), 30 hops max
  outgoing MTU = 2500
  1 bsdi (140.252.13.35) 15 ms 6 ms 6 ms
  2 sllp (140.252.13.65) 6 ms
    Fragmentation required and DF set, trying new MTU = 1492
    Fragmentation required and DF set, trying new MTU = 1808
    Fragmentation required and DF set, trying new MTU = 576
    Fragmentation required and DF set, trying new MTU = 512
    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 = 568
    Fragmentation required and DF set, trying new MTU = 296
  3 sllp (140.252.13.65) 377 ms 377 ms 377 ms
```

- ❑ The ICMP code on *bsdi* to return the MTU

```
sun % traceroute -mto sllp
traceroute to sllp (140.252.13.65), 30 hops max
  outgoing MTU = 2500
  1 bsdi (140.252.13.35) 53 ms 6 ms 6 ms
  2 sllp (140.252.13.65) 6 ms
    Fragmentation required and DF set, next hop MTU = 296
  3 sllp (140.252.13.65) 377 ms 376 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

```

3 0.0 solaris.38196 > slip_discard: udp 650 (IP)
4 0.004216 (0.0042) host > solaris: loop!
    slip unreachable - need to frag, mss = 0 (IP)
5 4.980328 (6.9783) solaris.38196 > slip_discard: udp 650 (IP)
6 4.984323 (0.0040) host > solaris: loop!
    slip unreachable - need to frag, mss = 0 (IP)
7 9.879487 (6.8655) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
8 9.880556 (0.0069) solaris > slip: (frag 47948:55200)
9 14.948338 (6.8623) solaris.38196 > slip_discard: udp 650 (IP)
10 14.944466 (0.0042) host > solaris: loop!
    slip unreachable - need to frag, mss = 0 (IP)
11 19.880555 (6.9455) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
12 19.880482 (0.0041) solaris > slip: (frag 47948:55200)
13 24.876481 (6.8189) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
14 24.948338 (0.0084) solaris > slip: (frag 47948:55200)
15 29.880182 (6.8221) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
16 29.948338 (0.0062) solaris > slip: (frag 47948:55200)
17 34.846687 (6.8201) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
18 34.950051 (0.0084) solaris > slip: (frag 47948:55200)
19 39.870218 (6.9225) solaris.38196 > slip_discard: udp 650 (frag 47948:55200)
20 39.930443 (0.0062) solaris > slip: (frag 47948:55200)
21 44.948338 (0.0100) solaris.38196 > slip_discard: udp 650 (IP)
22 44.944466 (0.0039) host > solaris: loop!
    slip unreachable - need to frag, mss = 0 (IP)

```

Figure 11.13: Path MTU discovery using UDP

## Path MTU Discovery with UDP

### ❑ Four fragments generated by the router bsd1

```

2 0.0 solaria.37974 > slisp-discard: udp 655 (frag 47942:272624)
2 0.334553 19.38431 solaria > slisp: (frag 47942:272624)
3 0.334553 19.33001 solaria > slisp: (frag 47942:895464)
4 0.459642 19.13281 solaria > slisp: (frag 47942:1060582)

```

Figure 13.15: First datagrams arriving at host slisp from solaria.

### ❑ Three fragments generated by the router bsd1 return the next-hop MTU in the ICMP “can’t fragment” error

```

3 0.0 solaria.37974 > slisp-discard: udp 655 (frag 47942:272624)
3 0.004189 19.00421 host > solaria: icmp:
  slisp unreachable - need to frag, mtu = 296 (frag 47942:272624)
3 4.858193 19.34401 solaria.37974 > slisp-discard: udp 655 (frag 47942:272624)
4 4.858193 19.00421 host > solaria: icmp:
  slisp unreachable - need to frag, mtu = 296 (frag 47942:272624)
5 6.778855 19.00421 solaria.37974 > slisp-discard: udp 655 (frag 47942:272624)
6 6.813458 19.34401 solaria > slisp: (frag 47942:272624)
7 6.813458 19.00421 solaria > slisp: (frag 47942:1060582)

```

Figure 13.16: Path-MTU discovery using UDP.

## Interaction Between UDP and ARP

### ❑ Example:

```

2 0.0 arp who-has vrr4 tell host1
2 0.00234 19.00021 arp who-has vrr4 tell host1
3 0.000941 19.00071 arp who-has vrr4 tell host1
4 0.002775 19.00081 arp who-has vrr4 tell host1
5 0.003485 19.00071 arp who-has vrr4 tell host1
6 0.004509 19.00081 arp who-has vrr4 tell host1
7 0.004773 19.00081 arp reply vrr4 src= 0:0:0:0:0:0:0:24
8 0.009901 19.00131 arp reply vrr4 src= 0:0:0:0:0:0:0:24
9 0.011127 19.00121 host1 > vrr4: (frag 18463:8967480)
10 0.011255 19.00021 arp reply vrr4 src= 0:0:0:0:0:0:0:24
11 0.012542 19.00131 arp reply vrr4 src= 0:0:0:0:0:0:0:24
12 0.013458 19.00094 arp reply vrr4 src= 0:0:0:0:0:0:0:24
13 0.014526 19.00131 arp reply vrr4 src= 0:0:0:0:0:0:0:24
14 0.015583 19.00131 arp reply vrr4 src= 0:0:0:0:0:0:0:24

```

Figure 16.67: Packet exchange when an IP-to-IP UDP datagram is sent over 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 (Cont.)

### ❑ 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

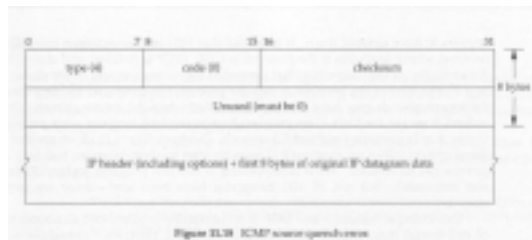
- ❖ Just over 8192 bytes for the maximum size of a UDP datagram that can be read or written
- ❖ Limit the size of an IP datagram to less than 65535 bytes

### ❑ 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



```
E 0.0.0.0:50000 > 192.168.1.1:50000: udp 1024
20 0.12 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: udp 1024
20 0.12 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: icmp: source quench
20 0.12 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: udp 1024
20 0.12 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: icmp: source quench
175 0.70 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: udp 1024
175 0.70 00.0001 192.168.1.1:50000 > 192.168.1.1:50000: icmp: source quench
```

Figure 11.19 ICMP source quench from the router was.

## 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

```

1 0.0          src=1252 > 149.252.13.65:4444: udp 31
2 2.409184 12.40901  src=1042 > 1491.4444: udp 14
3 4.909184 12.40901  src=1252 > 149.252.13.65:4444: udp 39
4 7.409184 12.40901  src=1042 > 1491.4444: udp 14
5 10.909184 12.40901  src=1252 > 149.252.13.65:4444: udp 32
6 12.409184 12.40901  src=1042 > 1491.4444: udp 9

```

Figure 11.20: Logging for UDP datagrams sent by two clients.

## UDP Server Design (Cont.)

- ❖ 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

```

1 0.0          dest=1123 > src=7777: udp 35
2 0.004022 (0.0040)  src > 1491: icmp: src udp port 7777 unreachable

```

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

### ❑ Restricting Foreign IP Address

### ❑ Multiple Recipients per Port

Local Address	Foreign Address	Description
localhost	foreignIP	restricted to one client
localhost	*	restricted to datagrams arriving on one local interface (null)
*	*	receives all datagrams sent to port

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