Today

- UDP
- TCP

IP options

Stevens traceroute with LSRR

```
./traceroute.lsrr -g www.utk.edu deneb.cs.utk.edu
LSRR to 80a9350f
LSRR to 80a95e4a
LSRR size 11 bytes
traceroute to deneb.cs.utk.edu (128.169.94.74), 30 hops max
 1  R5HM01V277.NS.UTK.EDU (128.169.92.1)  3 ms  2 ms  2 ms
 2  192.168.101.1 (192.168.101.1)  3 ms  3 ms  3 ms
 3  CHEKOV.IT.UTK.EDU (128.169.53.15)  6 ms *  4 ms
 4  128.169.53.1 (128.169.53.1)  4 ms  4 ms  4 ms
 5  192.168.101.40 (192.168.101.40)  5 ms  5 ms  5 ms
 6  DENEB.CS.UTK.EDU (128.169.94.74)  5 ms  5 ms  6 ms

19:57:50.067592 DENEB.52907 > CHEKOV.33435: udp 12 [ttl 1]
          4700 0030 c3ad 0000 0111 e47d 80a9 5e4a
          80a9 350f 8307 0480 a95e 4a00 ceab 829b
          0014 0000 0101 0000 37f7 ed7e 0001 0476
19:57:50.068384 R5HM01V277 > DENEB: icmp: time exceeded in-transit [
          45c0 0040 685e 0000 ff01 9700 80a9 5c01
          80a9 5e4a 0b00 a3a4 0000 0000 4700 0030
          c3ad 0000 0111 e47d 80a9 5e4a 80a9 350f
```


fragmentation

Fragmentation is good:

- hides sizing from upper protos
- sending large chunks simpler for app.
- use biggest MTU over as many hops as possible

Fragmenation is bad:

- causes inefficient use of resources (bandwidth, re-assembly)
- fragment loss is costly (entire datagram resent)
- re-assembly is hard (buffer, timer, out of order)

avoid fragmentation if you can

IP spec: host must accept at least 576-byte datagram

IP header

/usr/include/netinet/ip.h

```
struct ip {
#ifdef _BIT_FIELDS_LTOH
        u_char  ip_hl:4,                /* header length */
                ip_v:4;                 /* version */
#else
        u_char  ip_v:4,                 /* version */
                ip_hl:4;                /* header length */
#endif
        u_char  ip_tos;                 /* type of service */
        short   ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
        short   ip_off;                 /* fragment offset field */
#define IP_DF 0x4000                    /* dont fragment flag */
#define IP_MF 0x2000                    /* more fragments flag */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src, ip_dst; /* source and dest address *
};
```

netstat -s

```
IP      ipForwarding       =     1     ipDefaultTTL       =   255
        ipInReceives       =113432464  ipInHdrErrors      =     0
        ipInAddrErrors     =     0     ipInCksumErrs      =     0
        ipForwDatagrams    =     0     ipForwProhibits    =     0
        ipInUnknownProtos  =    78     ipInDiscards       =    26
        ipInDelivers       =113539915  ipOutRequests      =124842324
        ipOutDiscards      =    52     ipOutNoRoutes      =     0
        ipReasmTimeout     =    60     ipReasmReqds       =484486
        ipReasmOKs         =484282     ipReasmFails       =   204
        ipReasmDuplicates  =     4     ipReasmPartDups    =     0
        ipFragOKs          =2391489    ipFragFails        =     0
        ipFragCreates      =13920765   ipRoutingDiscards  =     0
        tcpInErrs          =  2030     udpNoPorts         =391186
        udpInCksumErrs     =    78     udpInOverflows     =  4262
        rawipInOverflows   =     0
```


netstat -s

```
ICMP    icmpInMsgs          = 32388     icmpInErrors        =     0
        icmpInCksumErrs     =     9     icmpInUnknowns      =     0
        icmpInDestUnreachs  = 17553     icmpInTimeExcds     = 13360
        icmpInParmProbs     =     7     icmpInSrcQuenchs    =   739
        icmpInRedirects     =    42     icmpInBadRedirects  =    42
        icmpInEchos         =   588     icmpInEchoReps      =    66
        icmpInTimestamps    =     0     icmpInTimestampReps =     0
        icmpInAddrMasks     =     0     icmpInAddrMaskReps  =     0
        icmpInFragNeeded    =   165     icmpOutMsgs         = 17193
        icmpOutDrops        =   169     icmpOutErrors       =     0
        icmpOutDestUnreachs = 16581     icmpOutTimeExcds    =    24
        icmpOutParmProbs    =     0     icmpOutSrcQuenchs   =     0
        icmpOutRedirects    =     0     icmpOutEchos        =     0
        icmpOutEchoReps     =   588     icmpOutTimestamps   =     0
        icmpOutTimestampReps=     0     icmpOutAddrMasks    =     0
        icmpOutAddrMaskReps =     0     icmpOutFragNeeded   =     0
        icmpInOverflows     =     0
```

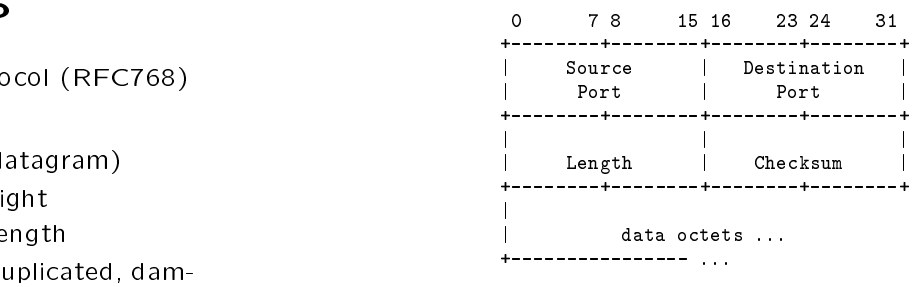

UDP

User Datagram Protocol (RFC768)

- connectionless (datagram)
- stateless, lightweight
- 16-bit port and length
- unreliable (lost, duplicated, damaged, delayed, out of sequence)
- optional checksum
- used by tftp,ntp,snmp,dns,nfs,rpc, audio/video
- needed for broadcast/multicast
- like its carrier IP, but simpler (IP has to do fragmenation using buffers/timers)

UDP header

IP protocol 17.



/usr/include/netinet/udp.h

```
struct udphdr {
        u_short uh_sport;               /* source port */
        u_short uh_dport;               /* destination port */
        short   uh_ulen;                /* udp length */
        u_short uh_sum;                 /* udp checksum */
};
```

- 16-bit source and destination port
- 16 bit length (includes 8-bytes of UDP header plus data)

- checksum over data and pseudo-header or 0
- if bigger than the MTU, IP will fragment
- NFS sends 8K UDP datagrams (older NFS, no checksum)

UDP echo

assignment 5

```
 on cetus1a:  asnmt5 altair 7   (and then type stuff)
 on deneb: asctcpdump.pl -N -x port 7
20:17:35.545727 CETUS1A.34875 > ALTAIR.echo: udp 8 (DF)
 4500 0024 92c2 4000 ff11 2c67 80a9 5e15      E..$..@...,g..^.


 80a9 5d37 883b 0007 0010 029a 7465 7374      ..]7.;......test


 696e 670a                                    ing.

20:17:35.558512 ALTAIR.echo > CETUS1A.34875: udp 8
 4500 0024 3569 0000 3c11 8cc1 80a9 5d37      E..$5i..<.....]7
 80a9 5e15 0007 883b 0010 0000 7465 7374      ..^....;....test
 696e 670a                                    ing.
```


UDP/ICMP tcpdump

- UDP sendto to non-listening port results in ICMP port unreachable
- connected UDP socket will get error on next read/recv

```
   whisper (ttcp) sends UDP to port 5002 on altair

WHISPER.1343 > ALTAIR.5002: udp 1024
          4500 041c 220a 0000 4011 9875 80a9 5dc8
          80a9 5d37 053f 138a 0408 1c98 2021 2223
          2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
          3435 3637 3839 3a3b 3c3d 3e3f 4041 4243
ALTAIR > WHISPER: icmp: ALTAIR udp port 5002 unreachable
          4500 0038 6f2c 0000 ff01 9046 80a9 5d37
          80a9 5dc8 0303 c393 0000 0000 4500 041c
          220a 0000 4011 9875 80a9 5dc8 80a9 5d37
          053f 138a 0408 1c98
```


UDP buffers

- a UDP socket has both a send and receive buffer
- buffer size can be set with setsockopt() SO_RCVBUF and SO_SNDBUF
- for SunOS rcvbuf: 18032 sndbuf: 9000
- if no room in sndbuf (output q), ENOBUFS from sendto()
- if datagram bigger than sndbuf, EMSGSIZE from sendto()
- if no room in rcvbuf, silently dropped
- bad checksum, receiver silently drops datagram (though kernel counter may be bumped, netstat -s)
- if checksum field is 0, receiving kernel does not validate. good luck

netstat -s

SunOS

```
udp:
      0 incomplete headers
      0 bad data length fields
      0 bad checksums
      200847380 socket overflows
```


TCP

Transmission Control Protocol (RFC793)

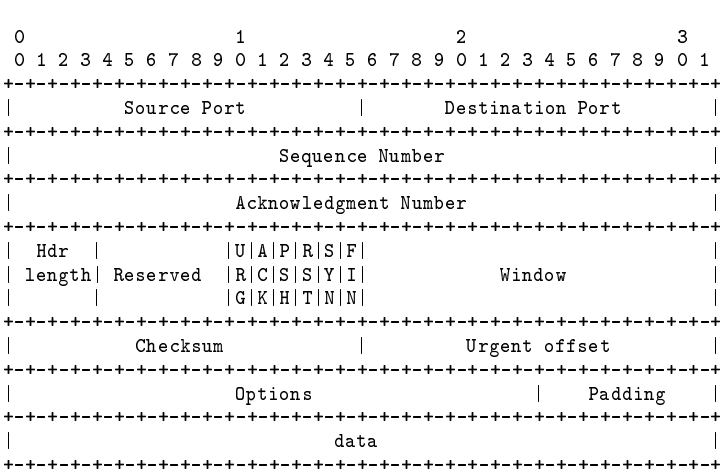
- connection-oriented, stream of bytes
- 16-bit ports
- reliable, stateful
- timers, checksums, sequence numbers, buffers, ACKs
- flow control (end-to-end), 16-bit window
- urgent data
- segmentation based on MSS
- smtp,telnet,ftp,X, ssh, http, rlogin, nntp

```
4.4BSD lite  (Stevens IP illustrated v2)
UDP: 9 functions, 800 lines of C code
TCP: 28 functions, 4500 lines of C code
```

How to provide a reliable stream over unreliable datagrams.

TCP header

IP proto = 6



TCP header fields

- 16-bit source and destination **ports**
- 32-bit **sequence number**, byte number of first data byte in packet
- 32-bit **ACK number**, byte number +1 of last data byte received
- 4-bit **Hdr length** number of 4-byte words in header
- **flags**

```
URG   urgent byte (MSG_OOB) coming, urgent offset valid
ACK   ack of data up to ack number
PSH   all datain buffer sent, push data up to application (meaningle
RST   reset, no port or service aborted
SYN   synchronize, start connection  (connect())
FIN   finish, end connection       (close())
```

- **window**, space available in my recieve buffer (initially, SO_RCVBUF)
- **checksum** over TCP pseudo header and data
- 16-bit **urgent offset** of byte of urgent data within stream (added to seq. number)

TCP header

/usr/include/netinet/tcp.h

```
struct tcphdr {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
#ifdef _BIT_FIELDS_LTOH
        u_int   th_x2:4,                /* (unused) */
                th_off:4;               /* data offset */
#else
        u_int   th_off:4,               /* data offset */
                th_x2:4;                /* (unused) */
#endif
        u_char  th_flags;
#define TH_FIN  0x01
#define TH_SYN  0x02
#define TH_RST  0x04
#define TH_PUSH 0x08
#define TH_ACK  0x10
#define TH_URG  0x20
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};
```


TCP logic

- formal connection established
- full duplex
- segments are numbered for ordering and acknowledgement
- duplicates discarded
- checksum for integrity
- timers on connect, ack return, close, 0-window, and delayed ack
- timeout: retransmit, exponential backoff
- flow control: receiver advertises space available (window)
- congestion control: calculates RTT, avoid small packets
- state machine: transistions based on packets or socket calls (connect/bind/listen/accept/close)

TCP states

netstat -a

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp        0      0 *:8660       *:*          LISTEN
tcp        0      0 whisper:4624 DENEB:echo   TIME_WAIT
tcp        0      0 *:6012       *:*          LISTEN
tcp        0      0 whisper:ssh  mist:1020    ESTABLISHED
tcp        0      0 *:6011       *:*          LISTEN
tcp        0      0 whisper:ssh  thistle:1020 ESTABLISHED
tcp        1      0 whisper:4604 KENNER:www   CLOSE_WAIT
```

also SO_DEBUG and trpt (SunOS)
print state info

TCP session

```
on deneb: ttcp -t -s -p9 -n1 -l1 achernar
on altair:  tcpdump -t -N port 9

DENEB.1054 > ACHERNAR.discard: S 689216000:689216000(0)
  win 8192 <mss 1460>
ACHERNAR.discard > DENEB.1054: S 684928000:684928000(0)
  ack 689216001 win 8192 <mss 1460>
DENEB.1054 > ACHERNAR.discard: . ack 1 win 8192
DENEB.1054 > ACHERNAR.discard: P 1:2(1) ack 1 win 8192
ACHERNAR.discard > DENEB.1054: . ack 2 win 8191
DENEB.1054 > ACHERNAR.discard: F 2:2(0) ack 1 win 8192
ACHERNAR.discard > DENEB.1054: . ack 3 win 8191
ACHERNAR.discard > DENEB.1054: F 1:1(0) ack 3 win 8192
DENEB.1054 > ACHERNAR.discard: . ack 2 win 8192
```

How many packets/bytes does it take to send 1 byte of data with UDP? with TCP?

TCP header dump

send a byte, ACK the byte

```
DENEB.1055 > ACHERNAR.discard: P 1:2(1) ack 1 win 8192
          4500 0029 c4f4 0000 3c06 fbfe 80a9 5e4a


          80a9 5e3f 041f 0009 2baa b601 2b69 4801



          5018 2000 58b1 0000 2000 6002 2000

ACHERNAR.discard > DENEB.1055: . ack 2 win 8191
          4500 0028 f743 0000 3c06 c9b0 80a9 5e3f
          80a9 5e4a 0009 041f 2b69 4801 2baa b602
          5010 1fff 78ba 0000 80a9 1804 bba2
```


next time

more TCP