Today

- UDP reliability
- tftp
- broadcast
- multicast

UDP reliability

- bad news: lost, duplicated, out of order, delayed, damaged
- kernel: enable UDP checksum
- UDP in PVM: timers, sequence numbers, ACKs, roundtrip estimates, retransmit with backoff
- performance: big datagrams, more than one in transit

tftp

trivial file transfer protocol

- simple file transfer
- easy to implement, tiny
- often used by bootstrap codes
- RFC 1350
- no authentication
- no directory listing
- supports binary or netascii

tftp protocol

- request/ACK
- operations: read,write,data,ack,error
- UDP port 69
- pseudo-connection oriented

  inetd receives on 69

  inetd fork/exec's tftpd and disables 69

  tftpd binds new port and fork's

  parent exit's, re-enabling inetd port 69

  tftpd child does work with client

- 512-byte blocks, slow
- sequence numbers, ACKs, and timeout
- timers by data sender only
- stateful

TFTP protocol

RFC 783

```
       2 bytes    string   1 byte     string   1 byte
       -----------------------------------------------
RRQ/  | 01/02 |  Filename  |   0  |    Mode    |   0  |
WRQ    -----------------------------------------------
                                netascii or octet

       2 bytes    2 bytes       n bytes
       ---------------------------------
DATA  | 03    |   Block #  |    Data    |
       ---------------------------------

       2 bytes    2 bytes
       -------------------
ACK   | 04    |   Block #  |
       --------------------

       2 bytes  2 bytes        string    1 byte
       ----------------------------------------
ERROR | 05    |  ErrorCode |   ErrMsg   |   0  |
       ----------------------------------------
```


inetd and tftp

inetd.conf

```
tftp    dgram   udp     wait    root    /usr/etc/in.tftpd
   in.tftpd -s /tftpboot
```

inetd fork

```
... pid = fork();
       if (pid && sep->se_wait) {
         sep->se_wait = pid;
         FD_CLR(sep->se_fd, &allsock);
         nsock--;
      }
```

reapchild()

```
      pid = wait3(&status, WNOHANG, (struct rusage *)0);
         if (sep->se_wait == pid) {
            FD_SET(sep->se_fd, &allsock);
            nsock++;
            sep->se_wait = 1;
         }
```


tftpd

- started by **inetd**
- fd 0 is socket
- with a datagram ready to read
- server needs to get new socket fd, and fork/exit to release inetd
- chroot to /tftpboot

client

- RFC doesn't define user interface
- client parses command line
- or sets up command interpreter
- evaluates command and sends packets to server and writing/reading to local file
- many routines are common to both server and client
- client must use port number from server

example

```
on altair:
tftp> connect whisper
tftp> get small

ALTAIR.1086 > WHISPER.tftp: 17 RRQ "small"
  4500 002d 3fd0 0000 3c11 829e 80a9 5d37      E..-?...<.....]7
  80a9 5dc8 043e 0045 0019 0000 0001 736d      ..]..>.E......sm
  616c 6c00 6e65 7461 7363 6969 0000           all.netascii..
WHISPER.1697 > ALTAIR.1086: udp 26
  4500 0036 5156 0000 4011 6d0f 80a9 5dc8      E..6QV..@.m...].
  80a9 5d37 06a1 043e 0022 451c 0003 0001      ..]7...>."E.....
  7468 6973 2069 7320 6120 736d 616c 6c20      this is a small
  6669 6c65 0d0a                               file..
ALTAIR.1086 > WHISPER.1697: udp 4
  4500 0020 3fd1 0000 3c11 82aa 80a9 5d37      E.. ?...<.....]7
  80a9 5dc8 043e 06a1 000c 0000 0004 0001      ..]..>..........
```


no such file

```
ALTAIR.1086 > WHISPER.tftp: 19 RRQ "badnews"
  4500 002f 3feb 0000 3c11 8281 80a9 5d37      E../?...<.....]7
  80a9 5dc8 043e 0045 001b 0000 0001 6261      ..]..>.E......ba
  646e 6577 7300 6e65 7461 7363 6969 00        dnews.netascii.
WHISPER.1697 > ALTAIR.1086: udp 19
  4500 002f 5165 0000 4011 6d07 80a9 5dc8      E../Qe..@.m...].
  80a9 5d37 06a1 043e 001b f78f 0005 0001      ..]7...>........
  4669 6c65 206e 6f74 2066 6f75 6e64 00        File not found.
ALTAIR.1086 > WHISPER.1697: udp 4
  4500 0020 3fed 0000 3c11 828e 80a9 5d37      E.. ?...<.....]7
  80a9 5dc8 043e 06a1 000c 0000 0004 0001      ..]..>..........
```


big file

```
ALTAIR.1086 > WHISPER.tftp: 15 RRQ "big"
  4500 002b 4026 0000 3c11 824a 80a9 5d37      E..+@&..<..J..]7
  80a9 5dc8 043e 0045 0017 0000 0001 6269      ..]..>.E......bi
  6700 6e65 7461 7363 6969 0018 2000           g.netascii.. .
WHISPER.1698 > ALTAIR.1086: udp 516
  4500 0220 518f 0000 4011 6aec 80a9 5dc8      E.. Q...@.j...].
  80a9 5d37 06a2 043e 020c 73ef 0003 0001      ..]7...>..s.....
  7468 6973 2069 7320 6120 6269 6720 6669      this is a big fi
ALTAIR.1086 > WHISPER.1698: udp 4
  4500 0020 4027 0000 3c11 8254 80a9 5d37      E.. @'..<..T..]7
  80a9 5dc8 043e 06a2 000c 0000 0004 0001      ..]..>..........
WHISPER.1698 > ALTAIR.1086: udp 516
  4500 0220 5190 0000 4011 6aeb 80a9 5dc8      E.. Q...@.j...].
  80a9 5d37 06a2 043e 020c 5cd3 0003 0002      ..]7...>..\.....
  6c65 7320 6578 6973 742c 2073 7570 706f      les exist, suppo
ALTAIR.1086 > WHISPER.1698: udp 4
  4500 0020 4028 0000 3c11 8253 80a9 5d37      E.. @(..<..S..]7
  80a9 5dc8 043e 06a2 000c 0000 0004 0002      ..]..>..........
...
WHISPER.1698 > ALTAIR.1086: udp 479
  4500 01fb 519f 0000 4011 6b01 80a9 5dc8      E...Q...@.k...].
  80a9 5d37 06a2 043e 01e7 f760 0003 000f      ..]7...>...`....
  6d6d 656e 7473 2e0d 0a23 0d0a 2320 496e      mments...#..# In
ALTAIR.1086 > WHISPER.1698: udp 4
  4500 0020 4035 0000 3c11 8246 80a9 5d37      E.. @5..<..F..]7
  80a9 5dc8 043e 06a2 000c 0000 0004 000f      ..]..>..........
```


broken example

```
Sequent client
tftp> connect sun2
tftp> get tom.tmp
Transfer timed out.

msb8k.8657 > max.tftp: 19 RRQ "tom.tmp"
max.4627 > msb8k.8657: udp 516
msb8k.8657 > max.tftp: 4 ACK
max.4627 > msb8k.8657: udp 516
msb8k.8657 > max.tftp: 4 ACK
max.4627 > msb8k.8657: udp 516
msb8k.8657 > max.tftp: 4 ACK
max.4627 > msb8k.8657: udp 516
msb8k.8657 > max.tftp: 4 ACK
max.4627 > msb8k.8657: udp 516
msb8k.8657 > max.tftp: 4 ACK
msb8k.8657 > max.tftp: 4 ACK
msb8k.8657 > max.tftp: 4 ACK
msb8k.8657 > max.tftp: 4 ACK
msb8k.8657 > max.tftp: 4 ACK
```

multicast

layer 2

- LAN's ability (protocol, controllers) to send to ALL stations (broadcast) or a group of stations (multicast)
- Ethernet, token ring, FDDI
- passes thru bridges
- usually special MAC address (all ones, 1 in low bit of first byte)
- efficient: sender sends only one packet

broadcast

- use of broadcast is anti-social, every station must handle it to some level
- UDP supports broadcast, local net IP address with all 1's, e.g., 128.219.255.255
- must inform kernel first: SO_BROADCAST socket option
- UDP/IP broadcast used by NTP, DHCP, BOOTP, ARP, RARP, RPC
- does not cross routers
- abused by whod, time, ICMP, others (ARP storms)
- used by hackers
- IP multicast in 90's (RFC1112)

multicast

- friendlier than broadcast
- only goes to machines that have "joined"
- not limited to LAN
- applications: video/audio, white board, videoteleconferncing
- MBONE – multicast backbone
  tools: sdr, vic, vat, nv, wb
- multicast programming
- multicast routing

IP multicast mods

To compile/run IP multicast programs your kernel must support IP multicast.

- in.h has new define's IN_MULTICAST(i) IP_MULTICAST_TTL IP_MULTICAST_LOOP IP_ADD_MEMBERSHIP
- ether,token, fddi drivers hacked to modify media controller to accept multicast groups
- IP multicast uses 01-00-5E-(3bytes from class D)
- TCP/IP code hacked to support new sockopt's, and to recognize multicast (class D addresses)
- for mcast routing, kernel mods for tunnel support

not Sun OS. Yes: linux, solaris, newer releases

layer 3

multicast programming

- UDP only
- server/clients need to agree on class D address (224.0.0.x are reserved) and port
- select TTL (1 subnet, 32 site, 64 region, 128, continent, 255 world)
- for $TTL > 1$ need **mrouted** and connection to MBONE (or PIM)
- No changes to send to a multicast address (can use existing programs), unless you want to
  1. change TTL (default is 1)
  2. not hear your own packets (LOOP)
- to receive multicast packets requires new code

```
  Receive mods:
build new struct ip_mreq
join group (IP_ADD_MEMBERSHIP)
build 2nd sockaddr_in struct
  one for recv: (INADDR_ANY and port) and bind
  2nd for xmit: multicast address and port
maybe set SO_REUSEADDR IP_MULTICAST_TTL IP_MULTICAST_LOOP
```


multicast details

IP Multicast socket options

```
struct ip_mreq {
        struct in_addr  imr_multiaddr;  /* IP multicast address of group */
        struct in_addr  imr_interface;  /* local IP address of interface */
};

 IP_ADD_MEMBERSHIP      join a multicast group

 IP_DROP_MEMBERSHIP      leave a multicast group

 IP_MULTICAST_TTL       specify TTL for outgoing multicast pkts

 IP_MULTICAST_LOOP      enable/disable loopback of outgoing pkts
hear what you send
```


multicast details

Scope

Scope	TTL	IPv4 class D address
local subnet	1	224.0.0.0 to 224.0.0.255
site	<32	239.255.0.0 to 239.255.255.255
enterprise	<64	239.129.0.0 to 239.195.255.255
global	<255	224.0.1.0 to 238.255.255.255

```
Reserved addresses
224.0.0.1 all-hosts or ALL-SYSTEMS
224.0.0.2 all-routers

Reserved addresses
224.0.0.1 all-hosts or ALL-SYSTEMS
224.0.0.2 all-routers

Reserved addresses
 224.0.0.0-224.0.0.255  link local (reserved for mgt protocols)
224.0.0.1 all-hosts or ALL-SYSTEMS
224.0.0.2 all-routers
224.2.127.254   session announcements, udp 9875 (sdr)
```


mding

```
        struct sockaddr_in  him,me;
        struct ip_mreq imr;
        struct passwd   *getpwuid(), *pwd;

        pwd = getpwuid(getuid());
        printf("using %s on port %d ttl %d\n",hostaddr,port,ttl);
        bzero((char *) &him, sizeof(him));
        bzero((char *) &me, sizeof(me));
        him.sin_addr.s_addr = inet_addr(hostaddr);
        me.sin_family = him.sin_family = AF_INET;
        me.sin_port  = him.sin_port = htons(port);
        me.sin_addr.s_addr = htonl(INADDR_ANY);

        if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
                err_sys("client: can't open datagram socket");
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one));
        if (bind(sockfd,&me,sizeof(me)) < 0) err_sys("bind");
        if (setsockopt(sockfd,IPPROTO_IP,IP_MULTICAST_TTL,&ttl, sizeof(ttl)) <0)
            err_sys("setsockttl");
        /* do stuff for mcast */
        imr.imr_interface.s_addr = htonl(INADDR_ANY);
        imr.imr_multiaddr.s_addr = inet_addr(hostaddr);
        if(setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &imr,
                 sizeof(struct ip_mreq)) == -1) err_sys("setsockadd");
#ifdef NOLOOP
        /* if LOOP is set (default) you'll hear your own mcasts */
        if(setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_LOOP, &zero,
             sizeof(zero)) == -1) err_sys("setsockloop");
#endif
```

```
        if ((pid = fork()) == 0) child();
        sprintf(buff,"hello from %s \07\n",pwd->pw_name);
        while(1){
                sendto(sockfd,buff,strlen(buff)+1,0,&him,sizeof(him));
                sleep(5);
        }
}


child()
{
 ....
        for(;;){
                fromlen = sizeof(from);
                lth = recvfrom(sockfd,buff,sizeof(buff),0,&from,&fromlen);
                buff[lth]=0;
                hp = gethostbyaddr(&from.sin_addr,
                 sizeof(struct in_addr), AF_INET);
                name =  hp == 0 ? inet_ntoa(from.sin_addr) : hp->h_name;

                printf("  %s/%s\n",name,buff);
        }
}
```


multicast routing

- IP routers use destination address
- multicast routers use source address
- IGMP (IP proto N) used for local session mgt
- routing protocols: DVMRP, PIM, CBT, MOSPF
  PIM uses existing routers
  DVMRP uses tunnels/MBONE
- dense mode: broadcast and prune back
- sparse: only send to those who ask

mrouted

- route IP multicasts to/from local subnet to rest of the world
- virtual net, on top of IP
- tree topology, no loops
- for now, separate routers (workstations) from Internet routers
- manual configuration with neighboring (MBONE) site (**/etc/mrouted.conf**)
- uses Ip-in-IP (IP protocol 4), tunneling kernel mods
- uses IGMP between routers
- queries local subnet for active services, prunes flow if no reply

tcpdump

```
class D mapped to Ether
0x01 00 5e xx yy zz (low order 23 bits of class D)

8:0:20:10:1:9b 1:0:5e:7:8:9 ip 69: thdsun.7654 > 224.7.8.9.7654:
  udp 27 [ttl 1]
0:e0:52:82:fd:0 1:0:5e:0:0:1 ip 60: 134.167.12.1 > ALL-SYSTEMS:
  igmp query [ttl 1]
8:0:20:10:1:9b 1:0:5e:7:8:9 ip 60: thdsun > 224.7.8.9:
  igmp nreport 224.7.8.9 [ttl 1]

 join and transmit
thdsun > 224.7.8.9: igmp nreport 224.7.8.9 [ttl 1]
          4500 001c c647 0000 0102 7827 86a7 0cba
          e007 0809 1600 01ef e007 0809 6570 6d04
thdsun.7654 > 224.7.8.9.7654: udp 27 [ttl 1]
          4500 0037 c649 0000 0111 77fb 86a7 0cba
          e007 0809 1de6 1de6 0023 0000 6475 6e69
          6761 6e40 7468 6473 756e 2073 6967 6e69
          6e67 204f 4e07

 Router queries every minute
134.167.12.1 > ALL-SYSTEMS: igmp query [ttl 1]
          4500 001c 7a12 0000 0102 cd24 86a7 0c01
          e000 0001 1164 ee9b 0000 0000 1144 4a31
          3630 3043 4d20 0000 0000 0000 0000
thdsun > 224.7.8.9: igmp nreport 224.7.8.9 [ttl 1]
          4500 001c c64e 0000 0102 7820 86a7 0cba
          e007 0809 1600 01ef e007 0809 5045 4f43
          1c01 06ed 0000 0000 0000 0000 4750
```

why not

- not supported on all OS's
- some nets (layer 2) have no multicast (ATM, pt-to-multipt)
- flooding/pruning partially implemented
- PIM on some routers
- based on lossy UDP
- easily abused
- session registration ad hoc
- easy to eavesdrop
- open research: reliable multcast, security, routing, session mgt, ...

many apps don't use multicast, but use reflectors, point-to-multipoint (irc, IM, netmeeting, realaudio)

scale with hierarchy?

next time

BPF, security