Internet Programming & Protocols Lecture 5

UDP internals

UDP applications

DNS DHCP ntp

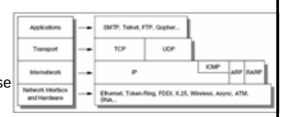
Internet design

- Internet Engineering Task Force (IETF)
  - International community of network designers, operators, vendors, and researchers (started 1986)
  - Concerned with evolution of the Internet architecture
  - Concerned with smooth operation of the Internet
  - Composed of many working groups (ietf.org)
  - See RFC 1360
- Request for Comment (RFC)
  - Technical and organizational notes about the Internet (since 1969)
  - Describe protocols, procedures, programs, and concepts
  - Official specs of the Internet as approved by IETF
    - MAY, MUST, SHOULD and NOT
    - RFC 1149 (CPIP -- carrier pigeon internet protocol)

A pigeon leg with a packet of network data attached.

"The IETF already has more than enough RFCs that codify the obvious, make stupidity illegal, support truth, justice, and the IETF way, and generally demonstrate the author is a brilliant and valuable Contributor to The Standards Process"

User Datagram Protocol (UDP)



- Defined in RFC 768
- connectionless (datagram)
- Lightweight – good for query/response
- 16-bit port (service number)
  - echo(7), DNS(53), bootp(68),ntp(123), snmp(160), NFS, RPC,netbios(137)
  - Streaming applications (audio video), losing a few packets OK
- unreliable (lost, damaged, duplicated, delayed, out of sequence) ☹
  - Same reliability as IP
  - If you want reliable UDP, application (YOU) must provide it!
- Can do broadcast and multicast with UDP

Many of the original well-known ports are odd numbers. NCP, TCP/IP predecessor, required two port numbers for a service.

Socket syntax and semantics

- Syntax: struct's, casts, pointers, call by reference, call by value
- Semantics: order, error returns, blocking, completion, effect

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0)
err = bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))
lth = sendto(sockfd, sendline, n, 0, pserv_addr, servlen)
lth = recvfrom(sockfd, recvline, MAXLINE, 0, &from, &fromlen);
err = getsockname(sockfd, (struct sockaddr *) &cli_addr, &clilen);
```

A "connection" (flow or socket) is a full-duplex 5-tuple

{source address, source port, destination address, destination port, protocol}

UDP header

```
0       7 8     15 16    23 24    31
+--------+--------+--------+--------+
|   Source Port   | Destination Port|
+--------+--------+--------+--------+
|    Length       |    Checksum     |
+--------+--------+--------+--------+
| data octets . . .
```

```
struct udphdr {
        __u16   source;
        __u16   dest;
        __u16   len;
        __u16   check;
};
```

Ethernet	IP	TCP/UDP	Application

8-byte header

- 16-bit port number
- Length is number of bytes of UDP data and header
- Simple 16-bit checksum over pseudo header and data (optional!)
- 0 or more bytes of data (max is controlled by SO_SNDBUF)
  - How many bytes to send a bit?
- If length is bigger than MTU, IP will fragment
  - NFS likes to use BIG datagrams

UDP checksum

source address		
destination address		
0	**proto**	**UDP lth**

- 16-bit checksum (RFC 1071)
  - End-to-end data integrity check
    - Calculated at sender
    - Verified at destination
  - 1s complement of sum of 16-bit words of pseudo-header prepended to UDP header and data
    - Pseudo header: IP addresses, IP proto, and UDP length (RFC 768)
  - Any overflow wrapped around
  - Same algorithm for IP and TCP checksum
- Why bother? (NFS often disables it for speed)
  - Link layers have CRC's etc.
  - BUT, could be errors in router's memory, or one of the links may not have adequate error detection
- If checksum fails, packet is usually dropped (silently)
  - Kernel may keep a counter

UDP sockets

- OS allocates send and receive buffer for each UDP socket
- Default buffer sizes vary by OS/release
- Receive buffer will hold incoming packets til application reads them
  - Silently dropped if buffer is full
  - Though OS may count overflows (netstat –s)
- Send buffer controls max UDP datagram and can buffer outgoing datagrams (watch out for bogus timings on transmit side)
- Program can retrieve current buffer sizes with getsockopt()
- program can set buffer sizes with setsockopt()
  - SO_RCVBUF   SO_SNDBUF
  - To send lots of data with UDP, application needs to send() lots of datagrams
- setsockopt() SO_NO_CHECK can disable UDP checksums
- If UDP packet arrives for an "inactive" port, OS sends back ICMP "port unreachable". If sender is using connect(), next recvfrom() will "fail"

Socket options

- modify socket characteristics, maybe improve performance!
- setsockopt(), getsockopt()
- must refer to open sockfd, issue before connect/bind

```
#include <sys/socket.h>
getsockopt(fd, level, optname, void *val, int *len)
setsockopt(fd, level, optname, void *val, int len)
```

- level specifies SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, IPPROTO_IPV6
- *val* can be varying type depending on option, hence *len* field needed too
- IP options: IP_HDRINCL, IP_OPTIONS, IP_TOS, IP_TTL and options for managing multicast
- UDP: SO_BROADCAST, SO_RCVBUF, SO_SNDBUF, SO_NO_CHECK
- TCP: later …

sockopt.c

```
#include <sys/types.h>
#include <sys/socket.h>    /* for SOL_SOCKET and SO_xx values */
main()
{
     int   sockfd, maxseg, sendbuff, optlen;

     if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        perror("can't create socket");
     if (getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,&sendbuff, &optlen) < 0)
        perror("SO_SNDBUF getsockopt error");
     printf("send buffer size = %d\n", sendbuff);
     sendbuff = 16384; /* just some number for example purposes */
     if (setsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,&sendbuff, sizeof(sendbuff)) < 0)
        perror("SO_SNDBUF setsockopt error");
     optlen = sizeof(sendbuff);
     if (getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,&sendbuff, &optlen) < 0)
        perror("SO_SNDBUF getsockopt error");
     printf("send buffer size = %d\n", sendbuff);
}
```


UDP on the net

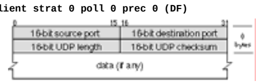
- UDP applications are lightweight (compared to TCP)
  - Query/response
  - May not matter if no response (don't care if packets are lost)
  - Can implement on small devices (toaster) or little OS (booting)
  - ntp, syslog, snmp, rpc
- Streaming (video audio), a few dropped packets don't matter
- Take advantage of local broadcast or multicast (DHCP)

Ethernet	IP	TCP/UDP	Application

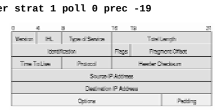
Network Time Protocol (NTP)

- UDP protocol to request time from another host (RFC 1305)
- Requester measures roundtrip delay (hopes route is symmetric)
- NTP adjusts your machine's time AND clock frequency
- Accuracies can be within a few milliseconds!
- Protocol specifies format of time request/reply, encoding of time etc.

```
160.36.58.221.32889 > 160.91.192.246.123: v1 client strat 0 poll 0 prec 0 (DF)
    4500 004c 9880 4000 4011 65cd a024 3add
    a05b c0f6 8079 007b 0038 1749 0b00 0000
    0000 0000 0000 0000 0000 0000 0000 0000
    0000 0000 0000
```




```
160.91.192.246.123 > 160.36.58.221.32889: v1 server strat 1 poll 0 prec -19
   4500 004c 7e4d 0000 f911 0700 a05b c0f6
   a024 3add 007b 8079 0038 e346 0c01 00ed
   0000 0000 0000 0000 4750 5300 c6b5 89d8
   0000 0000 c6b5
```



NTP protocol header and timestamp formats

NTP Protocol Header Format (32 bits)

LI	VN	Mode	Strat	Poll	Prec
Root Delay					
Root Dispersion					
Reference Identifier					
Reference Timestamp (64)					
Originate Timestamp (64)					
Receive Timestamp (64)					
Transmit Timestamp (64)					
Extension Field 1 (optional)					
Extension Field 2… (optional)					
Key/Algorithm Identifier					
Message Hash (64 or 128)					

Cryptosum

Authenticator (Optional)

LI	**leap warning indicator**
VN	**version number (4)**
Strat	**stratum (0-15)**
Poll	**poll interval (log2)**
Prec	**precision (log2)**

NTP Timestamp Format (64 bits)

Seconds (32)	Fraction (32)

Value is in seconds and fraction since 0h 1 January 1900

NTPv4 Extension Field

Field Length	Field Type
Extension Field (padded to 32-bit boundary)	

Last field padded to 64-bit boundary

NTP v3 and v4
NTP v4 only
authentication only

UDP syslog

syslog request from a C program (results in a UDP packet to port 514)

C code

openlog("tomtest",LOG_PID,LOG_MAIL);

syslog(LOG_AUTH|LOG_NOTICE,"sys log test auth/notice");

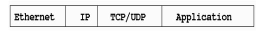
tcpdump -x -s 256 port 514

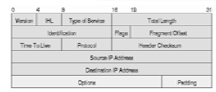
```
08:00:02.557018 thistle.syslog > thdsun.syslog: udp 44
4500 0048 341d 0000 4011 1d74 86a7 0f0c      E..H4...@..t....
86a7 0cba 0202 0202 0034 6db4 3c33 373e      .........4m.<37>
746f 6d74 6573 745b 3937 3833 5d3a 2073      tomtest[9783]: s
7973 206c 6f67 2074 6573 7420 6175 7468      ys log test auth
2f6e 6f74 6963 650a                          /notice.
```

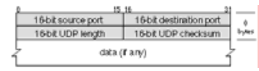
Sometimes payload is human readable

Our UDP client /server

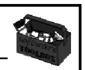
```
160.91.212.75.32834 > 160.36.58.221.7654: UDP, length 20
 0x0000: 0009 1228 e9ca 0006 5bdc 88a7 0800 4500 ...(....[.....E.
 0x0010: 0030 0005 4000 4011 eb0f a05b d44b a024 .0..@.@....[.K.$
 0x0020: 3add 8042 1de6 001c 4fd6 6361 6e20 796f :..B....O.can.yo
 0x0030: 7520 6865 6172 206d 6520 6e6f 770a      u.hear.me.now.

160.36.58.221.7654 > 160.91.212.75.32834: UDP, length 20
 0x0000: 0006 5bdc 88a7 0009 1228 e9ca 0800 4500 ..[......(....E.
 0x0010: 0030 0000 4000 3a11 f114 a024 3add a05b .0..@.:....$:..[
 0x0020: d44b 1de6 8042 001c 1cf4 6361 6e20 796f .K...B....can.yo
 0x0030: 7520 6865 6172 206d 6520 6e6f 770a      u.hear.me.now.
```







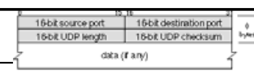
strace



- strace traces the system calls that a process makes
  - ***strace udpserver***
  - Can also start strace on a running process to see what it's doing ***strace -ppid***
- strace of our UDP client/server

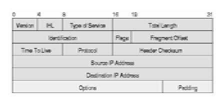
```
Client
strace -p 19690
Process 19690 attached - interrupt to quit
read(0, "can you hear me now\n", 1024) = 20
sendto(3, "can you hear me now\n", 20, 0, {sa_family=AF_INET, sin_port=htons(7654),
    sin_addr=inet_addr("160.36.58.221")}, 16) = 20
recvfrom(3, "can you hear me now\n", 512, 0, {sa_family=AF_INET, sin_port=htons(7654),
    sin_addr=inet_addr("160.36.58.221")}, [16]) = 20
write(1, "can you hear me now\n", 20) = 20
read(0,

Server
recvfrom(3, "can you hear me now\n", 8192, 0, {sin_family=AF_INET,
    sin_port=htons(32834), sin_addr=inet_addr("160.91.212.75")}}, [16]) = 20
sendto(3, "can you hear me now\n", 20, 0, {sin_family=AF_INET, sin_port=htons(32834),
    sin_addr=inet_addr("160.91.212.75")}}, 16) = 20
recvfrom(3,
```


NFS



•Read a remote file, big (28808) NFS datagrams (frags)

```
160.36.58.221.800 > 160.36.56.227.2049: 116 read [|nfs] (DF)
   4500 0090 0000 4000 4011 8654 a024 3add
   a024 38e3 0320 0801 007c 551d deaa 7034
160.36.56.227.2049 > 160.36.58.221.800: reply ok 1472 read [|nfs] (frag 50426:1480@0+)
    4500 05dc c4fa 2000 4011 dc0d a024 38e3
    a024 3add 0801 0320 7088 2b02 deaa 7034
160.36.56.227 > 160.36.58.221: (frag 50426:1480@1480+)
    4500 05dc c4fa 20b9 4011 db54 a024 38e3
    a024 3add 5e30 dbcc c3c9 0416 38a0 4c09
160.36.56.227 > 160.36.58.221: (frag 50426:1480@2960+)
    4500 05dc c4fa 2172 4011 da9b a024 38e3
    a024 3add 58f1 b810 547b 785d c351 1504
 ...
160.36.56.227 > 160.36.58.221: (frag 50426:1480@26640+)
    4500 05dc c4fa 2d02 4011 cf0b a024 38e3
    a024 3add d352 5900 e408 0070 12a1 0d1f
160.36.56.227 > 160.36.58.221: (frag 50426:688@28120)
    4500 02c4 c4fa 0dbb 4011 f16a a024 38e3
    a024 3add 38d4 a987 8602 357b 408f 18c7
```



fragmentation

- Fragmentation is good
  - Hides sizing from upper layers
  - Sending large chunks simpler (faster?) for application
  - Use biggest MTU over as many hops as possible
- Fragmentation is bad
  - Causes inefficient use of resources (bandwidth, re-assembly)
  - Fragment loss is costly – original "big" datagram must be re-sent
  - Re-assembly is hard (buffer management, timers, out of order)
- Avoid fragmentation if you can
  - TCP tries hard not to fragment (MSS negotiation) … later
  - MTU discovery can assist (though this is slow and complex too) …later
- Bigger MTU's are better, but tied to engineering of physical/link layer
- IP spec: host must accept at least a 576-byte datagram (min MTU)

fragmentation attack – who would've known

- teardrop ('97)
- Hackers send IP fragments with funky offsets
- Some OS's (who shall not be named) went belly up

Did you write that packet re-assembly code?

Dynamic Host Configuration Protocol (DHCP)

- Rather than statically configuring network info on your PC (IP address, net mask, broadcast address, default router, DNS servers), DHCP can do it automatically
  - Makes it easier to move machines
  - Easier for central management
  - Can assign same address each time, or conserve IP addresses by assigning from a pool of addresses on the local subnet
- Lightweight UDP protocol (ports 67 and 68)
  - When machine boots not much of an OS running, so simple protocol is needed
  - Booting PC broadcasts a DHCP request
  - DHCP server hears the request and replies with config info
- Specs in RFC 2131
  - Format of packets
  - Request/reply semantics
    - DISCOVER, OFFER, REQUEST, ACK, RELEASE

DHCP

```
     tcpdump –e –x
0:11:43:70:35:5b Broadcast ip 342:
    0.0.0.0.bootpc > 255.255.255.255.bootps:
    xid:0x3077b440 [|bootp] [tos 0x10]
 4510 0148 0000 0000 1011 a996 0000 0000
 ffff ffff 0044 0043 0134 1412 0101 0600
 3077 b440 0000 0000 0000 0000 0000 0000
 0000 0000 0000 0000 0011 4370 355b 0000

0:20:78:d9:3a:3f Broadcast arp 60: arp who-
    has pc102 tell linksys
  0001 0800 0604 0001 0020 78d9 3a3f c0a8
  0101 0000 0000 0000 c0a8 0166 0000 0000

0:20:78:d9:3a:3f Broadcast ip 590:
    linksys.bootps > 255.255.255.255.bootpc:
    xid:0x3077b440 Y:pc102 ether
    0:11:43:70:35:5b [|bootp]
  4500 0240 000c 0000 4011 b6f8 c0a8 0101
  ffff ffff 0043 0044 022c d615 0201 0600
  3077 b440 0000 0000 0000 0000 c0a8 0166
  0000 0000 0000 0000 0011 4370 355b 0000
```

Why is reply broadcast?

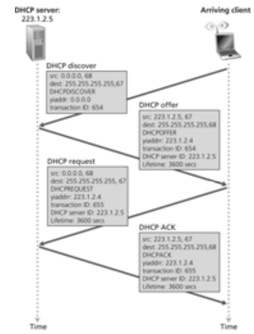


Figure 5.21 ♦ DHCP client-server interaction

Broadcast nasties

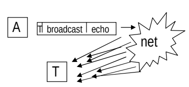
- Luckily you can't broadcast to ALL internet hosts (255.255.255.255) ☺
- BUT you can broacast (UDP or ICMP) to all hosts on a local subnet (128.214.255.255)
  - Handy for broadcasting time
  - Or DHCP
  - But can be abused (UDP echo, port 7) or ICMP ping (smurf attack)

SMURF attack

Hacker on his slow dial up connection, sends ICMP echo with broadcast destination (preferably of a net with high speed link).
Source address is spoofed and is the target of the flood of ICMP replies from the destination net.
If the target net has a slow link, then whole target subnet may be slowed.
Hackers like these high-leverage attacks: they send one packet and generate lots of nasty traffic.

Hackers also use broadcast ICMP echo (with a legit source address) to try and map active hosts on a destination net. (ping)

-routers can (should) block inbound broadcasts



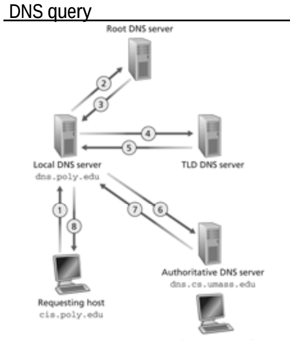
Domain Name Service (DNS)

- Host tables (/etc/hosts) too big 1984
- RFC 1034 defines distributed domain name system
  - Defines message formats
  - Request/reply semantics
  - UDP (mostly) port 53
- You can buy a domain name and have it registered
- Domain name server hierarchy consists of
  - Root servers (IP addresses hardwired into your local servers)
  - Top Level Domain Servers (TLD), e.g. for .com and for .edu
  - Authoritative servers
  - Local servers (maybe on your own machine or dept. engine, UNIX ***named***)
    - during net config, you tell your machine where local servers are
    - On UNIX IP addresses of local name servers in /etc/resolv.conf
      - Or indirect through Sun yp
      - Windows network config info (or provided by DHCP)
    - API gethostbyname() gethostbyaddr() ➔ DNS packets are sent out

Root servers

Figure 2.19 ♦ DNS root servers in 2004 (name, organization, location)

DNS query



- This can be slow!
  - Propagation delay for each path
  - Busy-server delays
  - Secondary-servers failsafe-ish
    - Timeout try another
    - If your server list is bad, net appears "down"
- Local servers can cache replies
  - Cache entries timeout so they can be refreshed
- Use UNIX dig command to explore

Figure 2.20 ♦ Interaction of the various DNS servers

dig

```
dig pcgiga.cern.ch
; <<>> DiG 9.2.0 <<>> pcgiga.cern.ch
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57289
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 6, ADDITIONAL: 0
;; QUESTION SECTION:
;pcgiga.cern.ch. IN A
;; ANSWER SECTION:
pcgiga.cern.ch. 10800 IN A 192.91.245.29
;; AUTHORITY SECTION:
cern.ch. 10800 IN NS ns2.cern.ch.
cern.ch. 10800 IN NS dxmon.cern.ch.
cern.ch. 10800 IN NS sunic.sunet.se.
cern.ch. 10800 IN NS ccpnvx.in2p3.fr.
cern.ch. 10800 IN NS ns-sec.ripe.net.
cern.ch. 10800 IN NS scsnms.switch.ch.
;; Query time: 279 msec
;; SERVER: 160.36.56.73#53(160.36.56.73)
;; WHEN: Tue Aug 23 16:04:17 2005
;; MSG SIZE rcvd: 200
```


DNS packets

```
0.621362 manitou.32769 >
    ns01.knoxville.tn.knox.comcast.net.domain: 6257+ A?
    pcgiga.cern.ch. (32) (DF)
     4500 003c 3373 4000 4011 615d c0a8 0104
     442f a005 8001 0035 0028 a61a 1871 0100
     0001 0000 0000 0000 0670 6367 6967 6104   …pcgiga.
     6365 726e 0263 6800 0001 0001             cern.ch
0.739407 ns01.knoxville.tn.knox.comcast.net.domain >
    manitou.32769: 6257* 1/6/9 A pcgiga.cern.ch (382)
    (DF)
     4500 019a cef9 4000 fc11 0878 442f a005
     c0a8 0104 0035 8001 0186 5238 1871 8580
     0001 0001 0006 0009 0670 6367 6967 6104
     6365 726e 0263 6800 0001 0001 0670 6367
     6967 6104 6365 726e 0263 6800 0001 0001
     0000 2a30 0004 c05b f51d c027 0002 0001
     0000 7b85 0011 0663 6370 6e76 7805 696e
     3270
```

Note: 100+ ms delay

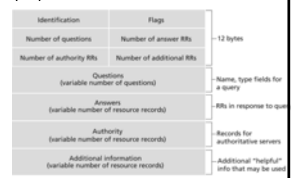
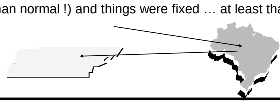


Figure 2.22 • DNS message format

Hackers and DNS

- DNS servers and core routers are critical infrastructure of Internet
  - Denial of service attacks (packet flooding)
  - Breaking in to server to re-route traffic to/through bad guy's site
- Routers usually have custom OS (e.g., Cisco IOS)
- DNS servers are typically UNIX boxes
- UT incident 199?
  - Hackers exploited buffer overflow in UT DNS server (Solaris), got root
  - Modified DNS addresses returned for utk.edu to IP addresses in Brazil
  - So when your client asked for IP address of koosh.cs.utk.edu, you got some funky address in Brazil?!?
  - In Brazil, the packets were eventually forwarded to proper UT IP address, but packets could have been sniffed/altered etc. ☹
  - Eventually it was noticed that packets to UT were "slow" (RTT much larger than normal !) and things were fixed … at least that Solaris bug was fixed.



UDP and the kernel – sending a UDP datagram

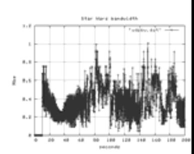
- sendto() is a system call, passes address of user message to kernel
- Kernel verifies sendto parameters are OK
- If enough room in socket's SNDBUF, copies user message to kernel space, if not enough room, return error (ENOBUFS) … in any case your application sendto() is "complete"
- Kernel constructs IP packet and calculates checksums (IP and UDP)
- IP layer looks up destination address in routing table, and may need to issue ARP request (asynchronous event)
- With Ether address of destination, construct Ether packet and queue to ether driver (packet could be dropped if TXQUE is full)
- Ether driver checks TXQUE and sends out the next packet
  - NIC handles CSMA/CD, CRC
  - NIC issues interrupt when transfer complete

UDP and the kernel – incoming UDP datagram

- NIC receives datagram with its NIC address in destination address field, issues an interrupt
- Interrupt handler requests Ether driver to read the datagram
- Datagram copied into kernel address space
- Driver inspects Ether type field (IP, ARP) and queues packet to appropriate kernel handler
- (assuming not fragmented), kernel IP handler verifies IP checksum and other IP fields, inspects IP proto field and queues packet payload to UDP handler
- UDP handler verifies checksum and UDP length (discards if fail), checks to see if there is process listening on the destination UDP port (if not, requests ICMP handler to send PORT_UNREACHABLE).
- UDP handler adds packet to socket's RCVBUF, if room (if not, drops)
- If associated process is blocked on recvfrom(), process is moved to "ready queue".
- when process runs, datagram copied into user's buffer

Things that slow us down …

- Transport layer (UDP)
  - Some UDP applications (streaming) do not backoff under heavy network load, hurting the other transport protocol (TCP) – not "TCP-friendly"
    - RealPlayer audio: 10 pkts/sec (rate-based) 70 kbs
      - 100 users, 7 mbs → 70% of 10mbs ethernet
    - Star Wars mpeg streaming video 400 kbs
  - DNS lookups can slow a network application
  - Hackers use UDP to flood the network (denial of service)



- Sending a packet to a remote host
  1. ARP for local DNS server (IP address in /etc/resolv.conf)
  2. Send DNS query to local DNS (this could take a while)
  3. ARP for subnet router
  4. Send one or more packets to remote via subnet router and then out into the Internet …

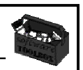
UDP vs TCP

- Must use UDP for multicast/broadcast
  - Would need to send N copies with TCP
- UDP for simple request-reply apps or light-weight
- UDP where some packet loss can be tolerated
  - Audio/video streaming
  - Rate-based (TCP's startup unacceptable)
  - Jitter (RTT variations) is a problem (app buffers incoming pkts, RealAudio)
  - Delay for retransmissions unacceptable (jitter)
- UDP if app needs to talk to 100's of different hosts
  - Don't need "connections", just modify socket address struct
- Some people are using UDP to get "better" performance than TCP
  - Is it fair? Is it TCP-friendly? (RUDP, DDCP)
  - Hackers like it for denial of service attack (no flow control, just blast)
- TCP for reliability
  - Often a UDP application ends up adding a lot of TCP baggage like flow control, timers, retransmit, buffer management

Reliable UDP

- Various "standards" for providing reliable UDP
  - application "header" for reliability
  - Timeouts and retransmission
  - Some reference implementations
- RTP (real time protocol) RFC 1889
  - Protocol for real-time streams (audio/video)
- RUDP (RFC 1151)
  - Reliable UDP for telephony signalling over the Internet
- DCCP (internet draft) datagram control (and congestion) protocol
  - TCP-friendly UDP
- Several UDP-based file transfer protocols
  - Tsunami, FOBS, SABUL, RBUDP, UDT
  - Better than TCP? (more later)
- ORNL's atou (Almost TCP over UDP) – research toy
- Lots of open research issues in reliable multicast !

Network tools



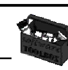
- strace
- dig
- netstat –a

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 *:32768                 *:*
udp        0      0 *:syslog                *:*
udp        0      0 *:32902                 *:*
udp        0      0 *:8037                  *:*
udp        0      0 *:875                   *:*
udp        0      0 *:sunrpc                *:*
```

- lsof –i

```
COMMAND     PID    USER   FD   TYPE DEVICE SIZE NODE NAME
aessrv     1350 dunigan    3u  IPv4   2986       UDP *:8654
dhaessrv   1351 dunigan    3u  IPv4   2988       UDP *:8037
sslsrv     1352 dunigan    3u  IPv4   3026       TCP *:7779 (LISTEN)
web100srv  1356 dunigan    3u  IPv4   2991       TCP *:3005 (LISTEN)
udpack    28815 dunigan    3u  IPv4 440082       UDP *:32902
bsrv      28822 dunigan    3u  IPv4 441364       TCP *:8622 (LISTEN)
```


Network tools



netstat –s

```
Ip:
    4862122 total packets received
    0 forwarded
    241 incoming packets discarded
    3703012 incoming packets delivered
    5124445 requests sent out
    64 outgoing packets dropped
    876367 reassemblies required
    74069 packets reassembled ok
    28404 fragments created
Icmp:
    2797 ICMP messages received
    1352 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 1324
        timeout in transit: 51
        echo requests: 72
    1411 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 1339
        echo replies: 72
```

```
Udp:
    287729 packets received
    55 packets to unknown port received.
    241 packet receive errors
    380417 packets sent
```

Note: these stats are for the system as a whole, it may be difficult to relate stats to particular flow.

Next time …

- TCP socket programming
- Assignments 2 and 3