Internet Programming & Protocols Lecture 3

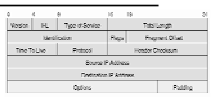
Routing

tcpdump/ethereal

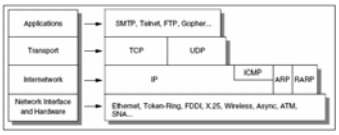
ICMP

traceroute

IP

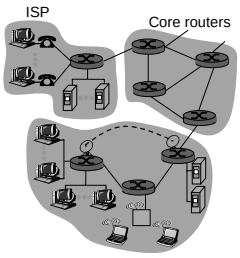


- Internet Protocol (IP)
- Defined by RFC 791 (IP version 4)
- Network layer
  - Datagrams (more "survivable" than circuit based – DARPA)
  - Deliver datagrams from sender to receiver
  - Unreliable (best effort)
- End nodes distinguished by unique 32-bit address
- Routing of datagrams based on destination address



Internet routing

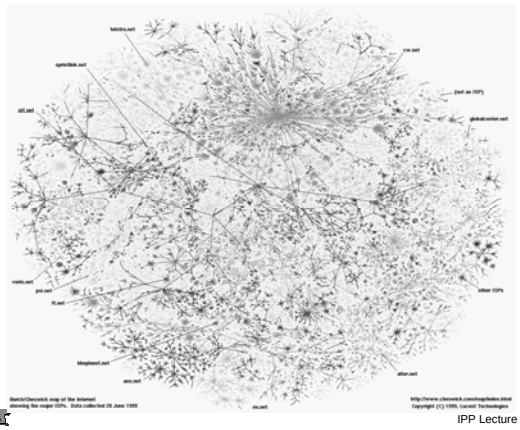
- Network edge:
  - ISP's, enterprise, end hosts
  - Your default router routes to enterprise router
  - Enterprise router forwards packets into core of Internet
- network core:
  - routers
  - network of networks
- Every interface has an IP address
  - Router looks up destination net to determine "next hop", which interface to send packet out
- Routers exchange route info
  - Enterprise advertises its nets
  - Reachable nets and "costs"
  - Routing protocols (BGP, OSPF, RIP)



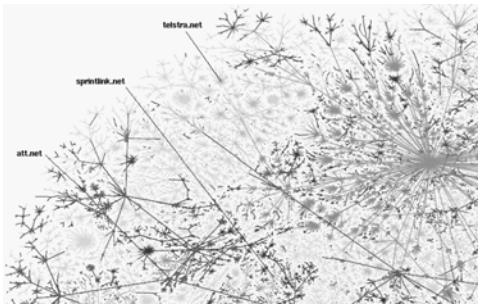
enterprise/campus

Routing tables

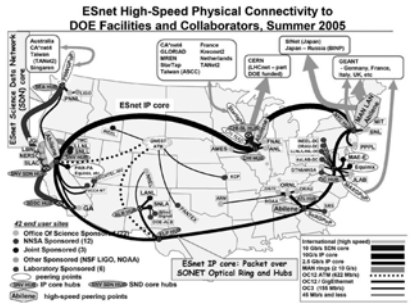
- Routing exchanges are periodic and often – affect performance?
  - Volume of "control packets" on the net
  - Routers "busy" during route updates
- Class A, B, C nets – how many routing tables entries?
  - Size of routing tables affects size/number of routing messages
  - Amount of searching routers have to do to select route
  - As internet grows, more routing table entries
- Classless Inter-Domain Routing (CIDR) (RFC 1519)
  - Pre-CIDR: Network ID ended on 8-, 16, 24- bit boundary
  - CIDR: Network ID can end at any bit boundary
    - IP Address : 12.4.0.0 IP Mask: 255.254.0.0 12.4.0.0/15
  - A sequence of class C address assigned to an enterprise can be represented by a single "supernet" (network/mask), requiring only one routing table entry



Internet map

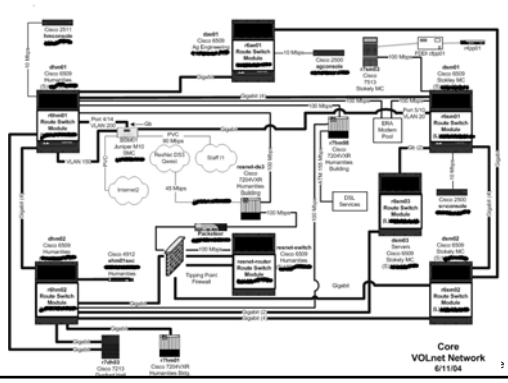


ESnet



Internet2 (OC192)

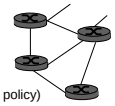
UT campus net



UTK net

- 'bout 25,000 registered hosts
- Redundant paths between backbone routers
- Circuit costs
  - OC-12 (622 mbs) $14k/month (internet2 and commodity traffic (155 mbs))
  - DS3 (45 mbs) $10k/mo (dorms)
  - Future: OC192 (lambda), dorms to 100 mbs ($9k/mo)
- Network Staff: 20 folks (5 just for wireless)

Routers



- Custom OS (Cisco, Juniper)
  - Manages routing protocol
  - Accepts packets
  - Determines outgoing interface ("best" route: distance, cost, policy)
  - Queues packet to outgoing interface (delay, drops ?)
- Mutliple interfaces (NICs)
- Memory for routing tables and buffering/queuing
- Switching fabric
  - Move packets from input port to output port
- Note
  - Each packet could take a different route (see traceroute)
  - Return route may differ
  - Packets could get out of order
  - Packets could loop (TTL will drop em, ICMP sent to sender)
  - Packets could be dropped/lost due to buffer exhaustion (silently) ***ANIMATION***

traceroute

```
traceroute www.utk.edu
traceroute to oscar.ws.utk.edu (160.36.178.162), 30 hops max, 38 byte packets
 1  r6hm01v150.ns.utk.edu (160.36.56.1)  8.937 ms  1.775 ms  0.288 ms
 2  r6hm02v13.ns.utk.edu (160.36.2.26)  0.285 ms  0.244 ms  0.236 ms
 3  r6sm03v17.ns.utk.edu (160.36.2.58)  0.343 ms  0.269 ms  0.261 ms
 4  oscar.ws.utk.edu (160.36.178.162)  0.300 ms  0.244 ms  0.357 ms

traceroute pcgiga.cern.ch
traceroute to pcgiga.cern.ch (192.91.245.29), 30 hops max, 38 byte packets
 1  r6hm01v150.ns.utk.edu (160.36.56.1)  0.338 ms  0.244 ms  0.230 ms
 2  bsm01v200.ns.utk.edu (160.36.1.104)  0.457 ms  0.853 ms  0.400 ms
 3  sox12.ns.utk.edu (160.36.128.150)  13.477 ms  5.839 ms  5.844 ms
 4  atla.abilene.sox.net (199.77.193.10)  6.184 ms  6.103 ms  6.080 ms
 5  iplsng-atlang.abilene.ucaid.edu (198.32.8.79)  17.113 ms  17.120 ms  17.095 ms
 6  chinng-iplsng.abilene.ucaid.edu (198.32.8.76)  21.054 ms  28.553 ms  20.991 ms
 7  ar5-chicago-abilene.cern.ch (192.91.246.126)  21.109 ms  21.089 ms  21.056 ms
 8  cernh5-so-100.cern.ch (192.65.184.54)  137.374 ms  137.525 ms  137.391 ms
 9  cernh4-vlan2.cern.ch (192.65.192.4)  137.757 ms  276.071 ms  137.545 ms
10  pcgiga.cern.ch (192.91.245.29)  137.460 ms  137.366 ms  137.388 ms
```

IP version 6

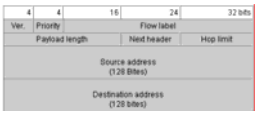
- Not really an issue for this class
- Fix IPv4 problems
  - Limited address space
  - Better performance
  - Security
- IETF call for proposals in 1990
  - Debate on TTL, address size, checksums, mobile hosts, security
  - Must interoperate with IPv4
- Some IPv6 nets operational (ORNL has its IPv6 net addresses)
  - Class C aggregation has helped routing/addressing issues of IPv4
  - Private IPv4 addresses have reduced need for IPv4 addresses
  - Security extensions to IPv4

IPv6

- 128-bit addresses
  - IPv4 addresses are one subset
  - 10^{23} addresses/sq meter
- Fixed header with optional subheaders
- Routing based on prefixes rather than address classes
- Routers don't fragment (sending host responsible, based on ICMP size exceeded message)
- Things that need to change
  - DNS, routers
  - OS/kernel handle IPv6 packets
  - Network utilities (ping, arp, ifconfig, netstat)
  - Network applications (be able to talk to both IPv4 and IPv6 hosts)
- OS implementations (linux/solaris/windows) available …
- No changes to transport layers (TCP or UDP) … not our problem

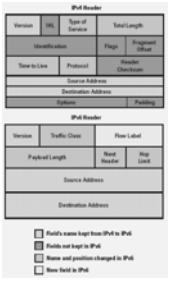
IPv6 header

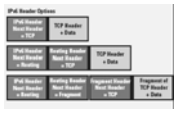
- Version (6)
- Delivery priority
- Flow label – special router handling
- Payload length (bytes)
- Type of next header
- Hop limit (TTL)
- Source/destination addresses



- Additional headers for routing, fragmentation, security, destination (e.g. transport headers like UDP or TCP)

IPv6 headers





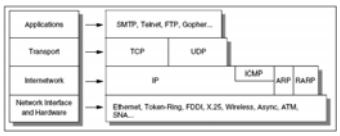
Streamlined
- Fragmentation fields moved out of base header
- IP options moved out of base header
- Header checksum eliminated
- Header length field eliminated
- Length field excludes IPv6 header
- Alignment changed from 32 to 64 bits

Revised
- Time to live -> hop limit
- Protocol -> next header
- Precedence and TOS -> traffic class
- Addresses increased 32 bits -> 128 bits

Extended
- Flow label field added

IP summary

- IP is the network layer for Internet protocols
- IP defines address limits
- IP tries to deliver packets from host A to host B
  - Based on datagrams
  - Best effort (packets may be lost, delayed, duplicated, mangled)
  - Delays and losses will affect performance … what this class is about!
- IP routes packets
  - Complicated routing protocols
  - packets from A to B may not follow same path



Network tools

- Passive tools
  - netstat, strace, ifconfig
  - network analyzers (tcpdump, ethereal)
    - Traffic characterization
    - Protocol analysis
    - Flow diagnosis
    - Intrusion detectors
    - Sniffers
- Active tools
  - ping, traceroute
  - ttcp, iperf, netperf

Packet watching tools

- tcpdump and ethereal
- Tools for capturing and analyzing packets on the wire
- Handy for diagnosing protocol or performance problems
- Reads raw packets off the network – promiscuous
  - Sees all the packets to/from a machine (or non-switched subnet)
  - on UNIX you need root privilege to watch the wire
  - Basis of hacker sniffing tools, so a "controlled substance"
- Command options for filtering flood of packets into what interests you
- Can also read/analyze previously captured data (disk file)
  - Privilege not needed
- Recall packets are encapsulations of various protocols
- Tools help parse/interpret the packet plus give you the raw hex

```
16       20      20/8                    4
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ..... -+-+-+-+
| mac   |    IP   |TCP/UDP|  App/Data ..... | CRC |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- ..... -+-+-+-+
```


tcpdump

- UNIX command line packet analyzer (based on libpcap)
- Command line: series of options and filter expressions
- Options

  -x hex dump -n no DNS -e include ethernet header

  -s NN capture NN bytes of each packet -v verbose

  -N no domain appendage -t no time -tt sec.fraction

  -w file.dmp write data to file -r file.dmp read data from file

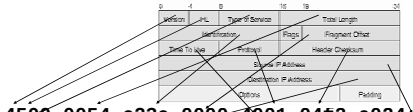
  -i interface -X include ascii with -x
- Filter expression
  - udp and port 7
  - host whisper and not port 22
  - icmp
  - host alice and host bob
  - `ip[0] & 0xf != 5' capture packets with IP options
  - "tcp[13]=2" capture TCP SYN packets

tcpdump of a ping

tcpdump –x –N icmp

```
14:14:02.202831 whisper > cetus1: icmp: echo request (DF)
                        4500 0054 0000 4000 4001 8730 a024 3add
                        a024 3853 0800 5f7e ca30 3a00 6af2 f842
                        4318 0300 0809 0a0b 0c0d 0e0f 1011 1213
                        1415 1617 1819
14:14:02.202939 cetus1 > whisper: icmp: echo reply
                        4500 0054 c23e 0000 4001 04f2 a024 3853
                        a024 3add 0000 677e ca30 3a00 6af2 f842
                        4318 0300 0809 0a0b 0c0d 0e0f 1011 1213
                        1415 1617 1819
```


The IP header in hex



4500 0054 c23e 0000 4001 04f2 a024 3853

160 .36 . 56 .83

a024 3add 0000 677e ca30 3a00 6af2 f842

160 36 58 221 the ICMP payload …….

Version 4, IHL 5 words (20 bytes), TOS 0, length 84. bytes

ID c23e flags/offset 0, TTL 64., protocol 1 == ICMP, checksum 04f2

tcpdump example

- 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.
```

IP header UDP header UDP data

tcpdump IP fragmentation

192.168.1.4 sends a 4000 byte UDP datagram to 192.168.1.3 over Ethernet (MTU 1500)

tcpdump –n –x –t host 192.168.1.3

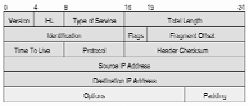
```
192.168.1.4.32845 > 192.168.1.3.2000: udp 4000 (frag 10734:1480@0+)
 4500 05dc 29ee 2000 4011 a7cb c0a8 0104
 c0a8 0103 804d 07d0 0fa8 e363 0000 0000

192.168.1.4 > 192.168.1.3: udp (frag 10734:1480@1480+)
  4500 05dc 29ee 20b9 4011 a712 c0a8 0104
  c0a8 0103 8463 0f7e 538b 4b2b 3a41 e372

192.168.1.4 > 192.168.1.3: udp (frag 10734:1048@2960)
  4500 042c 29ee 0172 4011 c809 c0a8 0104
  c0a8 0103 ed7c 5333 de4a 5127 1316 0757
```

in 2nd word of IP header

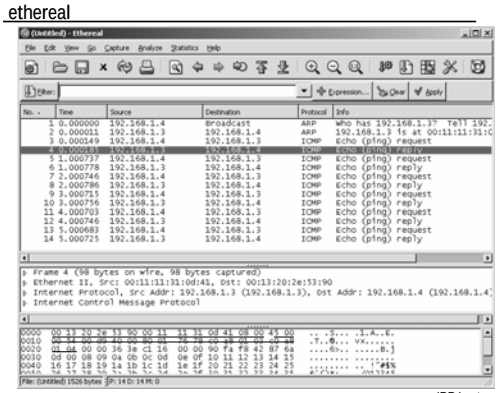
ID 29ee

Flags 2 (more) 0 (last)

Offset 0, b9, 172



ethereal



ethereal capture setup

Make sure you select the correct interface

ethereal capturing ...

ARP

- Not IP, ethernet type 0806
- Host broadcasts ARP request, responder sends ARP reply

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Hardware                    |          protocol             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  HLEN         |    PLEN       |         operation             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            sender hardware address (0-3)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| sender HA (4-5)               |  sender internet addr (0-1)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  sender IA (2-3)              |  target HA (0-1)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   target hardware addr (2-5)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   target internet addr (0-3)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


ARP tcpdump

tcpdump –e –x –n –t –s 128 arp

```
0:13:20:2e:53:90 Broadcast arp 42: arp who-has 192.168.1.3 tell 192.168.1.4
    0001 0800 0604 0001 0013 202e 5390 c0a8
    0104 0000 0000 0000 c0a8 0103

0:11:11:31:d:41 0:13:20:2e:53:90 arp 60: arp reply 192.168.1.3 is-at 0:11:11:31:d:41
    0001 0800 0604 0002 0011 1131 0d41 c0a8
    0103 0013 202e 5390 c0a8 0104 0000 0000
```

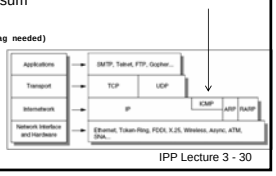
ARP Packet	hardware address space	2 bytes
	protocol address space	2 bytes
	hardware address byte length (n) / protocol address byte length (m)	2 bytes
	operation code	2 bytes
	hardware address of sender	n bytes
	protocol address of sender	m bytes
	hardware address of target	n bytes
	protocol address of target	m bytes

1 request 2 reply

ICMP

- Internet Control Message Protocol RFC 792
- rides on IP, not really transport protocol part of IP (proto=1)
- provides feedback (port unreachable)
- error types carry offending IP header
- no ICMP messages sent about ICMP messages
- generated by kernel/routers (and ping (need su))
- Unreliable (could be lost)
- ICMP packet includes type and checksum

```
Type    meaning
0       echo reply
3       destination unreachable (net,host,port,frag needed)
4       source quench (shutup)
5       redirect
8       echo request
11      time exceeded (TTL went to 0)
12      parameter problem (IP options etc.)
13      timestamp   (14 reply)

see #include <netinet/ip_icmp.h>
```



ICMP ping – Packet InterNet Groper

- ping is a good low level test of host reachability (no application/daemon needs to be running) uses ICMP ECHO request(8)/reply(0)
- Provides some performance feedback (round trip time, loss)
- Privilege required for program to send/receive ICMP packets

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Identifier           |       Sequence Number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
CETUS1A > ALTAIR: icmp: echo request (DF)
4500 0054 2d37 4000 ff01 2c36 80a9 5e15
80a9 5d37 0800 fa7f 1dd0 0003 37f0 0fdc
000d acd0 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233

ALTAIR > CETUS1A: icmp: echo reply (DF)
4500 0054 36d3 4000 fe01 8836 80a9 5d37
80a9 5e15 0000 0280 1dd0 0003 37f0 0fdc
000d acd0 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
```

```
16       20      16+                 4
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-  -+-+-+-+
|ether  |   IP   | ICMP header and data| CRC |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-  -+-+-+-+
```

DF Dont' Fragment bit set

ICMP unreachable packet

```
Destination unreachable: type=3
code: 0=net, 1=host, 2=protocol, 3=port, 4=frag need
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           unused                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
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
```

WHISPER sends UDP packet to port 5002 on ALTAIR, but no process is listening on that port.

ICMP timestamp

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Identifier           |       Sequence Number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Originate Timestamp                                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Receive Timestamp                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Transmit Timestamp                                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
160.91.212.75 > 160.91.1.57: icmp 28: time stamp query id 41777 seq 256
  4500 0030 0000 4000 4001 2492 a05b d44b
  a05b 0139 0d00 01ab a331 0100 e68a b602
  0000 0000 0000 0000 36f9 fdf6 90a6 eafe
160.91.1.57 > 160.91.212.75: icmp 28: time stamp reply id 41777 seq 256 : org
    0xe68ab602 recv 0x2b68ada xmit 0x2b68ada
  4500 0030 8e07 4000 fe01 d889 a05b 0139
  a05b d44b 0e00 e589 a331 0100 e68a b602
  02b6 8ada 02b6 8ada 36f9 fdf6 90a6 eafe
```


ICMP ping with Record Route IP option

ping -R www.utk.edu

```
PING oscar.ws.utk.edu (160.36.178.162) from 160.36.58.221 : 56(124) bytes of data.
64 bytes from oscar.ws.utk.edu (160.36.178.162): icmp_seq=1 ttl=252 time=1.83 ms
NOP
RR:     whisper.cs.utk.edu (160.36.58.221)
        r6hm01v12.ns.utk.edu (160.36.2.18)
        r6sm01v16.ns.utk.edu (160.36.2.49)
        r6sm03v667.ns.utk.edu (160.36.178.1)
        oscar.ws.utk.edu (160.36.178.162)
        r6sm03v16.ns.utk.edu (160.36.2.50)
        r6sm01v12.ns.utk.edu (160.36.2.17)
        r6hm01v150.ns.utk.edu (160.36.56.1)
        whisper.cs.utk.edu (160.36.58.221)
```

```
160.36.58.221 > 160.36.178.162: icmp:
echo request (DF)
4f00 007c 0000 4000 4001 ffa7 a024 3add
a024 b2a2 0107 2708 a024 3add 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0800 d3d6
362b 0100 9587 f842 6530 0f00 0809 0a0b

160.36.178.162 > 160.36.58.221: icmp:
echo reply (DF)
4f00 007c ae31 4000 fc01 8f0d a024 b2a2
a024 3add 0107 2724 a024 3add a024 0219
a024 0239 a024 b201 a024 b2a2 a024 023a
a024 021a a024 3801 0000 0000 0000 dbd6
362b 0100 9587 f842 6530 0f00 0809 0a0b
```

IP options: 1 NOP

Record Route : type (7), length (27 bytes), offset(8), address1,…

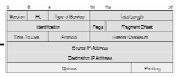
traceroute

- Trouble with ping –R
  - Not supported by all routers
  - Only room for 9 addresses
- traceroute better for recording route(one-way)
  - send a UDP packet (or ICMP-echo) with TTL of 1
  - first router, decrements to 0, sends ICMP time-exceeded with routers IP address as source
  - send UDP packet with TTL of 2, makes it 2nd router, which sends ICMP time-exceeded back
  - and so on
  - final destination receives, usually sends back ICMP port-unreachable
  - need raw socket and socket option IP_HDRINCL to set TTL or IP_TTL option (need privilege)
  - win* version: **tracert** or graphical with **PingPlotter**
  - slow printing due to DNS (try -n)
  - one-way, return route may differ (traceroute server)

traceroute with tcpdump

```
traceroute to thdsun.epm.ornl.gov (134.167.12.186), 30 hops max, 40 byte packets
1  R5HM01V277.NS.UTK.EDU (128.169.92.1)  3 ms  2 ms  2 ms
2  192.168.101.3 (192.168.101.3)  3 ms  3 ms  3 ms
3  mmesgwya32.ctd.ornl.gov (192.31.96.17)  5 ms  6 ms  5 ms
4  orgwyf1.ctd.ornl.gov (192.31.96.65)  4 ms  4 ms  5 ms
5  atmgwy.ens.ornl.gov (198.124.42.8)  5 ms  5 ms  4 ms
6  swge4500n.ens.ornl.gov (160.91.0.2)  7 ms  7 ms  7 ms
7  swge6010.ens.ornl.gov (160.91.0.10)  6 ms  6 ms  7 ms
8  thdsun.epm.ornl.gov (134.167.12.186)  5 ms  5 ms  5 ms


ALTAIR.47368 > thdsun.33435: udp 12 [ttl 1] (id 32461)
R5HM01V277 > ALTAIR: icmp: time exceeded in-transit [tos 0xc0] (ttl 255, id 6475)
...
ALTAIR.47368 > thdsun.33444: udp 12 (ttl 4, id 32474)
orgwyf1 > ALTAIR: icmp: time exceeded in-transit [tos 0xc0] (ttl 252, id 11097)
...
swge6010 > ALTAIR: icmp: time exceeded in-transit (ttl 58, id 27665)
ALTAIR.47368 > thdsun.33456: udp 12 (ttl 8, id 32490)
thdsun > ALTAIR: icmp: thdsun udp port 33456 unreachable (ttl 248, id 59340)
ALTAIR.47368 > thdsun.33457: udp 12 (ttl 8, id 32492)
```

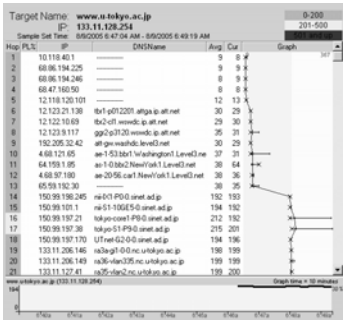
traceroute packet pairs



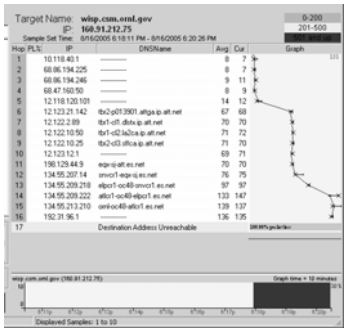
```
   tcpdump -x -n udp or icmp
160.36.58.221.32829 > 192.91.245.29.33452: udp 10
                        4500 0026 a533 0000 0611 7f19 a024 3add
                        c05b f51d 803d 82ac 0012 26f3 1206 7ff7
                        0443 ae31 0100
198.32.8.76 > 160.36.58.221: icmp: time exceeded in-transit
                        4500 0038 0000 0000 f901 1857 c620 084c
                        a024 3add 0b00 cb10 0000 0000 4500 0026
                        a533 0000 0111 8419 a024 3add c05b f51d
                        803d 82ac 0012

160.36.58.221.32829 > 192.91.245.29.33453: udp 10
                        4500 0026 a534 0000 0711 7e18 a024 3add
                        c05b f51d 803d 82ad 0012 f59d 1307 7ff7
                        0443 de84 0100
192.91.246.126 > 160.36.58.221: icmp: time exceeded in-transit
                        4500 0038 0000 0000 f801 30e9 c05b f67e
                        a024 3add 0b00 fc64 0000 0000 4500 0026
                        a534 0000 0111 8418 a024 3add c05b f51d
                        803d 82ad 0012
```


pingplotter

- Windows graphical “traceroute”
- Shows variation
- Shows Packet Loss
- pingplotter.com



pingplotter – oak ridge to ORNL ☺



Program notes

- C struct’s for network headers
- Packet sniffing and tcpdump
- Raw socket programming for ping and traceroute

C headers

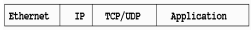
```
#include <net/ethernet.h>

struct ether_header
{
  u_int8_t  ether_dhost[ETH_ALEN];      /* destination eth addr */
  u_int8_t  ether_shost[ETH_ALEN];      /* source ether addr    */
  u_int16_t ether_type;                 /* packet type ID field */
} __attribute__ ((__packed__));




#include <netinet/if_ether.h>

struct  ether_arp {
        struct  arphdr ea_hdr;          /* fixed-size header */
        u_int8_t arp_sha[ETH_ALEN];     /* sender hardware address */
        u_int8_t arp_spa[4];            /* sender protocol address */
        u_int8_t arp_tha[ETH_ALEN];     /* target hardware address */
        u_int8_t arp_tpa[4];            /* target protocol address */
};
```


C headers

```
#include <netinet/ip.h>

struct iphdr
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
  };
```

tcpdump

- tcpdump (and hacker's sniffer software) based on *libpcap*
- Sets the NIC into promiscuous mode (need "root")
- libpcap
  - implementation-independent data-link layer access
  - can read/write network interface (NIC) or file
  - filter language
    "host bob and (port 23 or port 21)"
  - kernel accelerants with Berkeley Packet Filter (BPF)
  - but YOU have to decode the packets
    - You're given a pointer to packet, then you have to overlay the Ether header, the IP header, the TCP header, etc. to decode the packet



libpcap example

Example, we want to print time and seq # for a TCP session

argv[1] will be "tcp and dst host bob and port 2000"

A hacker would want to look for userid's and passwords in the packet!

```
main(argc,argv){
    device = pcap_lookupdev(errbuf);
    pd = pcap_open_live(device, snaplen, 1, 1000, errbuf);
    pcap_lookupnet(device,&net,&netmask,errbuf);
    pcap_compile(pd,&code,argv[1],1,netmask);
    pcap_setfilter(pd,&code);
    pcap_loop(pd, 0, read_pkt, NULL);
```

- **each packet from *libpcap* has the following header**

```
struct pcap_pkthdr {
    struct timeval ts;      /* time stamp */
    bpf_u_int32 caplen;     /* length of portion present */
    bpf_u_int32 len;        /* length this packet (off wire) */
};
```


libpcap cont.

```
read_pkt(u, h, p)
u_char *u;  /* pointer from pcap_loop */
const struct pcap_pkthdr *h;  /* header */
const u_char *p;    /* data-link packet */
{
     pktsec =h->ts.tv_sec;
     ts = h->ts.tv_sec + 1.e-6*h->ts.tv_usec;
     if (ts0 == 0) ts0=ts;
     ts = ts-ts0;
     ...  chop off data-link header (varies by media: pcap_datalink())
     ... align data (p) on IP struct boundary (ipbuf)
     packet_ip(ipbuf);
}
packet_ip(ip)
register struct ip *ip;
{
     struct tcphdr *tp;
     tp = (struct tcphdr *)( (char *)ip + 4*ip->ip_hl);
     if (ip->ip_p == IPPROTO_TCP ){
        seq = ntohl(tp->th_seq);
        if (seq0 == 0) seq0 = seq;
        seq = seq-seq0;
        printf("%f %u\n",ts,seq);
}
```


traceroute logic

get a raw socket for sending "raw" UDP with constructed IP header (TTL)

another raw socket for receiving ICMP's

loop: send a UDP, increasing TTL in IP header each time

receive ICMP's and verify it's an ICMP for this process (carries our IP header)

continue til we get a ICMP "port unreachable"

raw sockets allow you to build your own IP header! (need root)

used by traceroute, ping, and hackers (e.g., bogus IP source address)

traceroute.c

```
if ((s = socket(AF_INET, SOCK_RAW, pe->p_proto)) < 0) {
    perror("traceroute: icmp socket");
    exit(5);
}
if (sndsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
    perror("traceroute: raw socket");
    exit(5);
}
if (setsockopt(sndsock, IPPROTO_IP, IP_HDRINCL, (char *)&on, sizeof(on)) < 0) {
    perror("traceroute: IP_HDRINCL");
    exit(6);
}
```

socket() creates a socket descriptor (like a file descriptor)

setsockopt() set various options for a socket,

like IP header will be provided by user

traceroute.c

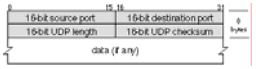
- Send packets with increasing ttl and check ICMP replies

```
for (ttl = 1; ttl <= max_ttl; ++ttl) {
    for (probe = 0; probe < nprobes; ++probe) {
          (void) gettimeofday(&tv, &tz);
          send_probe(++seq, ttl);
          while (cc = wait_for_reply(s, &from)) {
                    if ((i = packet_ok(packet, cc, &from, seq))) {
                              int dt = deltaT(&tv);
                              ... switch on ICMP_type
                              case ICMP_UNREACH_PORT:
                                 …
                              case ICMP_TIMXCEED:
```

traceroute.c (build IP and UDP headers)

```
send_probe(seq, ttl)
{
    struct opacket *op = outpacket;
    struct ip *ip = &op->ip;
    struct udphdr *up = &op->udp;
    int i;

    ip->ip_off = 0;
    ip->ip_p = IPPROTO_UDP;
    ip->ip_len = datalen;
    ip->ip_ttl = ttl;

    up->uh_sport = htons(ident);
    up->uh_dport = htons(port+seq);
    up->uh_ulen = htons((u_short)(datalen - sizeof(struct ip)));
    up->uh_sum = 0;


    i = sendto(sndsock, (char *)outpacket, datalen, 0, &whereto, sizeof(struct sockaddr));
```

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



Network toolbox

- ifconfig
- arp
- netstat
- traceroute/PingPlotter
- ping
- strace
- lsof
- tcpdump/ethereal (assignment 2)

Next time ...

- BSD sockets
- UDP

- Assignments 2 and 3