Internet Protocols, Network layer protocols

Outline

- IP details
  - fragmentation
- Helper protocols
  - ARP
  - ICMP
- NAT
- IPv6

IP functions

- route packets
  - routing: process of determining path for data
  - ip routes packets when they come from
    - transport layer (down stack)
    - link layer (up stack) - we are router and forward pkts
- fragmentation acc. to link-layer MTU
- handle ip options
- send/recv ICMP error and control messages

IP Header

0		15	16	31
vers:4	hlen:4	TOS:8	total length:16	
ip datagram ID:16			flags:3	fragment offset:13
TTL:8		proto type:8	ip header checksum:16	
ip **source address**:32				
ip **destination address**:32				
ip options (if any) 32 bit aligned				

IP Header

- ip version == 4
- header length in 32-bit words, h == 5 with no options (20 bytes)
- type of service and precedence
  - not used much in past but starting to be used
  - bits 0-2, precedence
  - bits 3-5, TOS, hint to routing about how to queue
    - D (bit 3) - low delay (telnet),
    - T (4) - high throughput (FTP),
    - R (5) - reliability

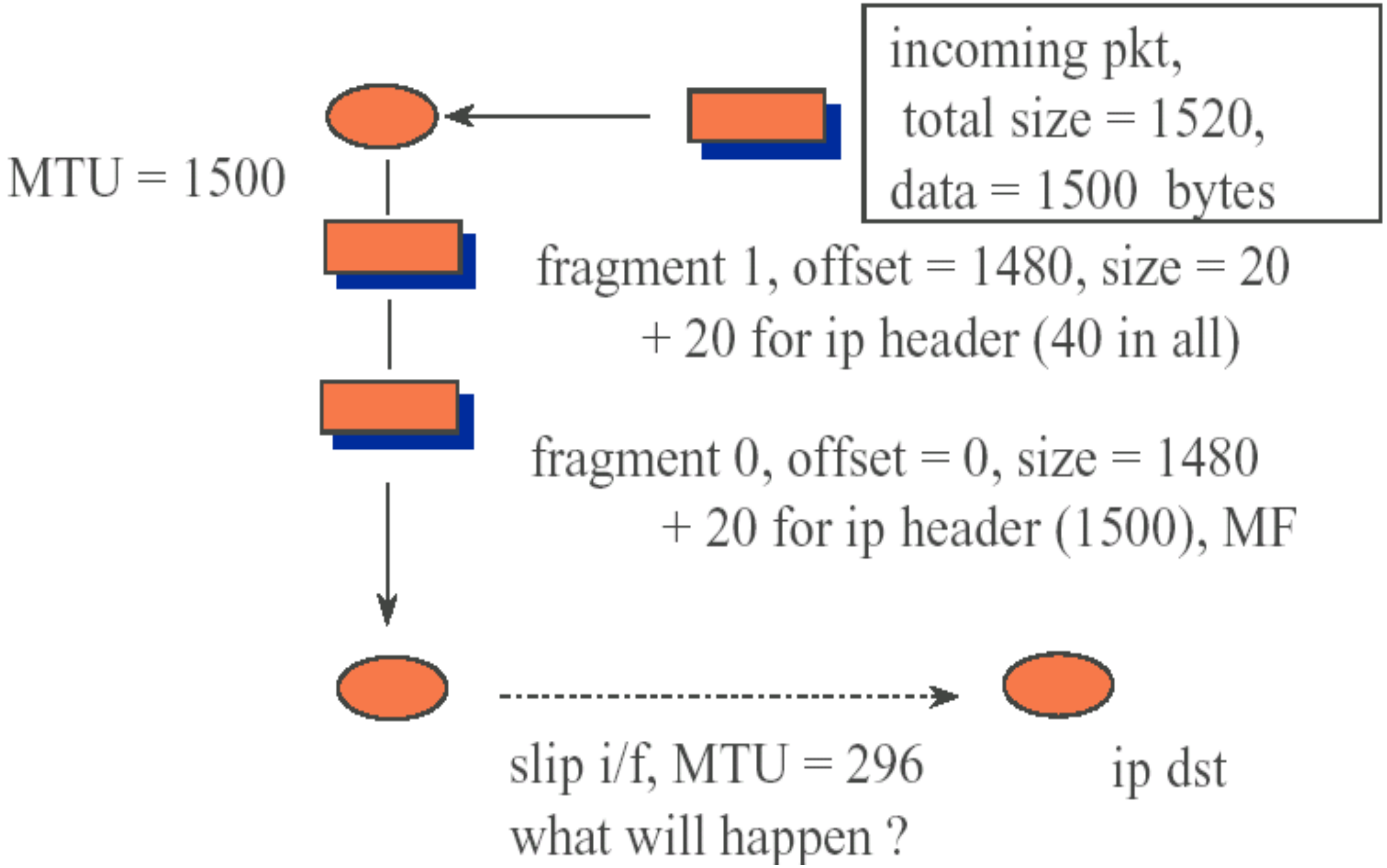
IP Header

- total length - max ip datagram is 64k
- fragmentation
  – fragment ip_id stays the same for all fragments
  – flags (DONT_FRAGMENT, MORE_FRAGMENTS)
  – fragment offset from 0 start of packet, e.g.,
  – 0, 0x400, 0x800
  – ip length is length of fragment, not total datagram

How it works

- ip fragments because outgoing packet is too big for MTU of i/f
- fragments must be reassembled at final ip destination and can be fragmented again on way
- if any fragment lost, all of datagram must be resent (not by IP)
- IP uses best effort even to allocate internal buffers
- TCP tries to avoid, UDP not smart enough
- IP fragmentation not a STRONG mechanism

IP Fragmentation



ip_id, ip_src retained in all (new) fragments

More fragmentation

- reassembly done at ultimate destination
  - pros:
    - simplicity - fragments can be routed independently
    - simplicity - intermediate routers don’t have to store
  - cons:
    - any fragment lost, entire datagram lost
- path MTU is a way around
- note: routers may not see all fragments

IP header

- proto type - TCP, UDP, ICMP
- checksum
  – over header only, useful?
  – same algorithm used by tcp/udp
  – with ip itself, only over header
    - deemed not useful in IPv6
  – routers must redo IP checksum since ttl changes

TTL

- TTL - time to live, actually hop count, not time
- when packet crosses router
  - if ttl == 1
    - discard and send ICMP ttl exceeded to ip src
  - ttl--
- **important guarantee that datagrams will be discarded even if network loops**

IP options

- not much used and possibly not very useable
- variable length encoding mechanism
- options come in multiples of 32 bits
- pro: extensible format
- con: not as easy to parse as fixed format

Options

- end of option list
- loose source routing: specify inexact path
- strict source routing exact path (with ip addresses)
- record route - possibly useful
- gather timestamps

Options bad things

- encoding is not efficient for routers
- length is limited by IP header length – not big enough for size of Inet
- source routing not secure -- someone could stick in an intermediate route and spy on your packets

ARP, The problem

- problem: **how does ip address get mapped to ethernet address?**
- 2 machines on same enet can only communicate if they know MAC/hw addr
- solutions:
  – configure addresses by hand (ouch!)
  – encode in IP address (48 bits in 32?)
  – use broadcast?

Solution, ARP

- rfc 826
- host A, wants to resolve IP addr B,
  - send BROADCAST arp request
- same link only
- ethernet (or MAC) specific, although protocol designed to be extensible
- implemented in driver, not IP

% arp -a (Unix)

 # arp -a
banshee.cs.pdx.edu (131.252.20.128) at 0:0:a7:0:2d:a0
pdx-gwy.cs.pdx.edu (131.252.20.1) at 0:0:c:0:f9:17
longshot.cs.pdx.edu (131.252.20.129) at 8:0:11:1:44:68
walt-suncs.cs.pdx.edu (131.252.21.2) at 8:0:20:e:21:25
walt-cs.cs.pdx.edu (131.252.20.2) at 8:0:20:e:21:25

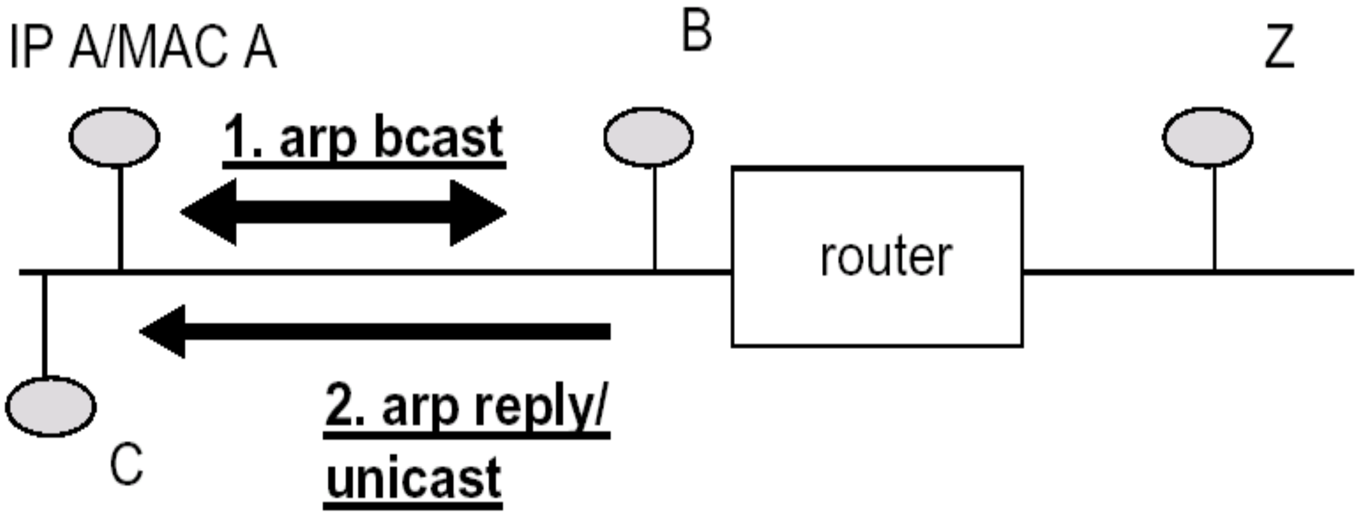
(DNS name,ip address,Ethernet address)

Refinements

- o.s. will cache arp replies in **arp cache (ip , MAC, 20 minute timeout)**
  - don't need to do arp on every packet
- machine may store all arp broadcast to get sender ip/mac mapping
- recv. machines can update their cache

ARP protocol

1. A to B, arp request/broadcast on link
2. B to A, arp reply/unicast



ARP header

<table>
<tr><th colspan="2">0</th><th>16 … 31</th></tr>
<tr><td colspan="2">Hardware Type (1 byte)</td><td>Protocol Type (1 byte)</td></tr>
<tr><td>HLEN</td><td>PLEN</td><td>ARP Operation Code</td></tr>
<tr><td colspan="3">Sender HA (MAC) (bytes 0-3)</td></tr>
<tr><td colspan="2">Sender HA (bytes 4-5)</td><td>Sender IP Addr (0-1)</td></tr>
<tr><td colspan="2">Sender IP (2-3)</td><td>Target HA (0-1)</td></tr>
<tr><td colspan="3">Target HA (MAC) (bytes 2-5)</td></tr>
<tr><td colspan="3">Target IP Address (4 bytes)</td></tr>
</table>

Header details

- header format is not fixed, somewhat dynamic (not used though)
- hw type, ethernet == 1
- protocol type, ip = 0x800
- hwlen, 6 (MAC), plen 4 (ip)
- operation: (used by rarp too)
  - 1: arp request, 2: arp reply
  - 3: rarp request, 4: rarp reply

More Details

- sender hw addr, 6 bytes
  - the answer, if reply
- sender ip: 4 bytes
- target hw address: 6 bytes
  - 0 in request
- target ip: 4 bytes

Proxy ARP

- basic idea: machine A answers requests for machine B (that can’t arp for some reason), forwards packets to B somehow
  – machine A might have 2 IP addresses associated with one interface

Proxy ARP pros, cons

- pros
  - same network numbers
  - can aid dumb host that can’t arp
  - remote serial host appears on same ethernet courtesy of terminal emulator/router
- cons
  - can drive you nuts -- debugging
  - not simple and not secure

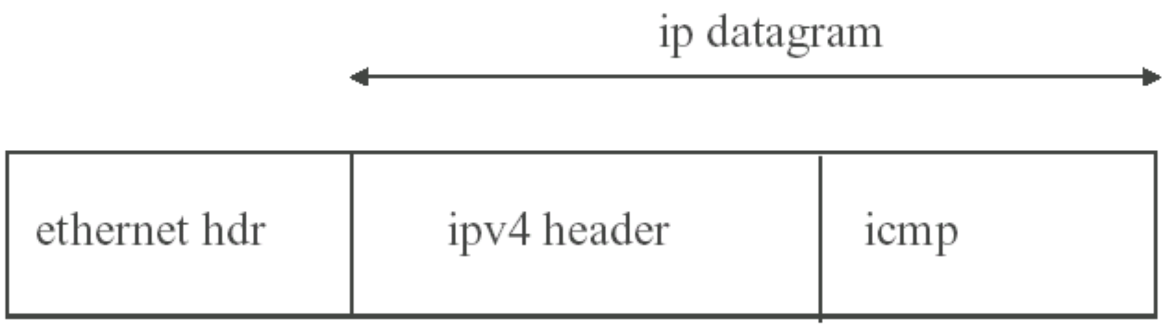
gratuitous/promiscuous arp

- **grat arp** - at boot or change of ip address, issue broadcast arp request for YOURSELF
  - unix ifconfig does this
  - detect other boxes with same IP address
  - allow recv boxes to cache your MAC addr
- **promiscuous arp** - issue bcast arp to change other's ideas of ip/mac mapping
  - **problem: no one guaranteed to be listening**

ICMP

- IP control (management plane)

ICMP Encapsulation



ICMP transmitted within IP datagram so that it is routeable
(unlike arp)

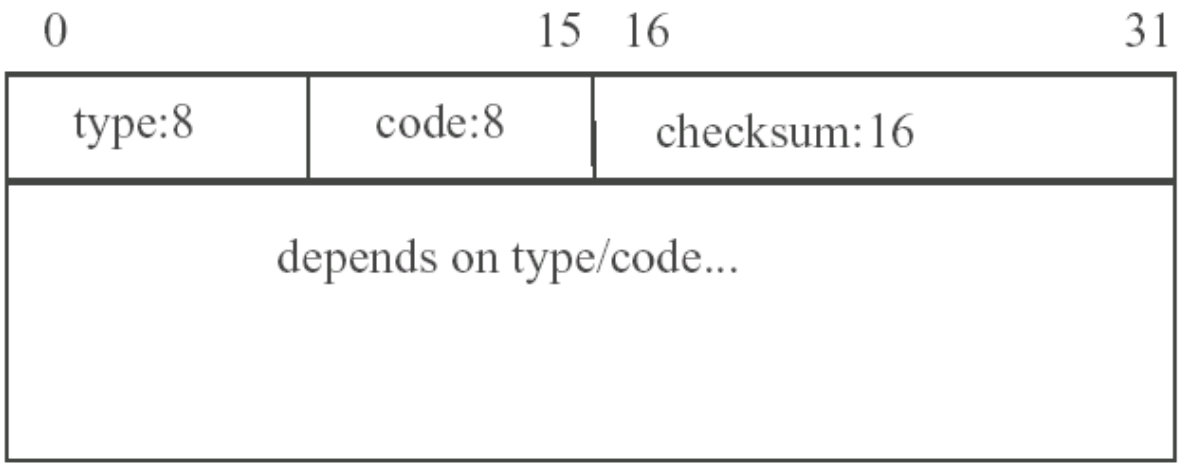
ICMP Ideas

- Considered part of IP (mandatory implementation)
- Functionality includes:
  - error messages (ttl exceeded, destination unreachable, router is congested, parameter problem)
  - network management (echo request/reply)
  - end host configuration (router advert, netmask)
- Error messages go from router/end host to original ip src
  - not understood by intermediate hops

ICMP Ideas

- Error messages typically sent at IP layer, received by source IP/UDP/TCP, which may forward to an application
- ICMP error messages never generated due to:
  - ICMP error message (loop)
  - broadcasts/multicasts
- Why? prevent **broadcast storms**
- **Error messages contain offending IP header + 1st 8 bytes of IP data (eg tcp/udp ports)**

Header



checksum covers icmp header/data, not ip header

ICMP messages (not all)

type	code	purpose	error?
0	0	echo reply (ping)	no
3	1	host unreachable	yes
3	3	port unreachable	yes
3	4	DF and must fragment	yes
4	0	source quench	yep
5	0	redirect - network	kinda
8	0	echo request (ping)	no

continued

type	code	purpose	error?
9/10	0	router advert/solicit	no
11	0	time exceeded, ttl = 0	yes
11	1	timeout during reassembly	yes
12	0/1	parameter problems	yes
13/14	0	timestamp request/reply	no
17/18	0	netmask request/reply	no

Time Exceeded

- If TTL value 0, discard packet and issue ICMP time exceeded, code 0, to IP source
- If fragments not received within a certain time limit at destination, discard fragments and issue ICMP time exceeded, code 1
- Prevents infinite packet loops

Destination unreachable

- Host or router cannot deliver a datagram
- Return IP header first 8 bytes
- Codes
  - 0 Network unreachable
  - 1 Host unreachable
  - 2 Protocol unreachable
  - 3 Port unreachable
  - 4 DF set but must fragment on next hop
- Detects forwarding errors (but not all)

Source Quench

- No flow control in IP (data rate)
- Source quench alerts sender:
  - A packet was discarded
  - Slow down transmission rate
- Returned is IP header plus 8 bytes of data
- But rarely acted upon (other congestion control mechanisms are used)

Parameter problem

- If the IP header format wrong
  - Discard datagram
  - Issue ICMP parameter problem
    - Code 0 faulty header field, pointer field in ICMP addresses start byte of problem in IP header
    - Code 1 required part of option is missing

Echo request/reply

- Host or router sends echo request to destination IP
- Destination returns echo reply
- Used by ‘ping’ (below)

Router solicitation

- Host wants to learn about network topology issues ICMP RS message
- Routers reply with a router advertisement
- Little used (DHCP is more common now)

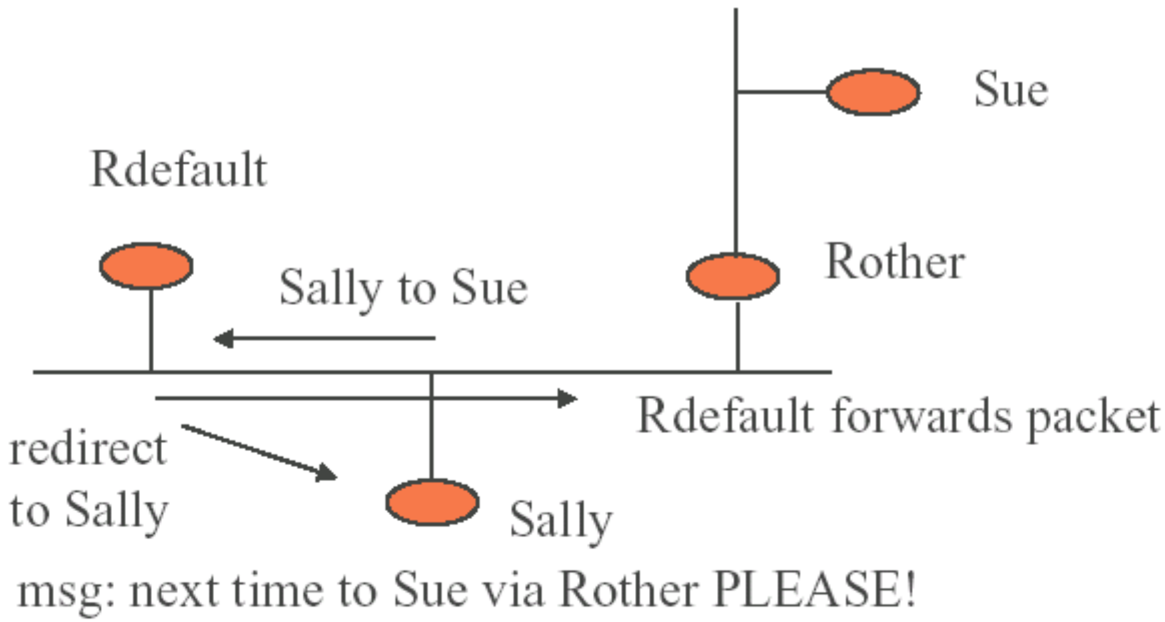
ICMP redirect

- Limited dynamic routing table update technique
- Only done on same link/network
- Situation:
  - 1. assume dumb host with 1 default routing table entry
  - 2. two routers on same link, one is default, one is route to net X
  - 3. dumb host sends pkt to net X via default router
  - 4. default router sends ICMP redirect with correct router address to dumb host

Redirect contd.

- Default router also forwards original packet correctly
- Dumb host changes its routing table to reflect newly learned route to other net
- Means initial configuration can be minimal, and hosts then learn
- Now rarely used

Picture


Sue
Rdefault
Rother
Sally to Sue
Rdefault forwards packet
redirect
to Sally
Sally
msg: next time to Sue via Rother PLEASE!

Address mask

- If host does not know its netmask, issue Address mask request
- Router on network replies with mask
- Can be unicasted or broadcasted
- Can be used at bootstrapping
  - but now little used

ping - ICMP echo request/reply

- ping program useful diagnostic tool, uses ICMP echo request/reply packets
- ping adds identifier/sequence number fields to echo/reply packets
- sequence # allows you to see if packets lost
- ping will also do roundtrip timing

ping example

- *$ ping cse.ogi.edu*

  PING cse.ogi.edu (129.95.20.2): 56 data bytes

  64 bytes from 129.95.20.2 icmp_seq=0 time=8ms

  64 bytes from 129.95.20.2 icmp_seq=1 time=8ms

  64 bytes from 129.95.20.2 icmp_seq=2 time=20ms

  ---cse.ogi.edu PING statistics ---

  3 packets transmitted, 3 packets received, 0% loss

  round-trip (ms) min/avg/max = 8/12/20

(MS cmd = ping)

More ping

- So what do you learn?
  - you can route to destination
  - end system's ip stack is working at least
  - round trip time information
  - are packets being lost (but doesn't tell you why)
- Echo reply sent by end system's ICMP, you don't know if upper layers are working...

traceroute

- *% traceroute north.pole.com*
- traceroute (a command) allows you to determine the routers from one end to another
- Uses ICMP ttl exceeded and (UDP port unreachable *or* ICMP echo reply) messages (2 forms of implementation)
- (MS = tracert)

traceroute example

- % traceroute cse.ogi.edu (from sirius.cs.pdx.edu)

  traceroute to cse.ogi.edu (129.95.20.2), 30 hops max ...

  1. pdx-gwy (131.252.20.1) 3 ms 4 ms 3 ms
  2. 198.104.197.58 (198.104.197.58) 7 ms 4 ms 8 ms
  3. portland1-gw.nwnet.net (198.104.196.193) 6 ms 5 ms 5 ms
  4. ogi-gw-nwnet.net (198.104.196.129) 8 ms 7 ms 7 ms
  5. cse.ogi.edu (129.95.20.2) 14 ms 7 ms 9 ms

  note: try from usyd to unsw or some other uni in the Sydney area. how many hops? how long?

traceroute algorithm (simplified)

- Set dest IP address, ttl = 1 (to 1st router)

  while we haven't got (UDP port unreachable or ICMP echo reply), repeat 3 times

  send UDP / raw ip packet

  get response

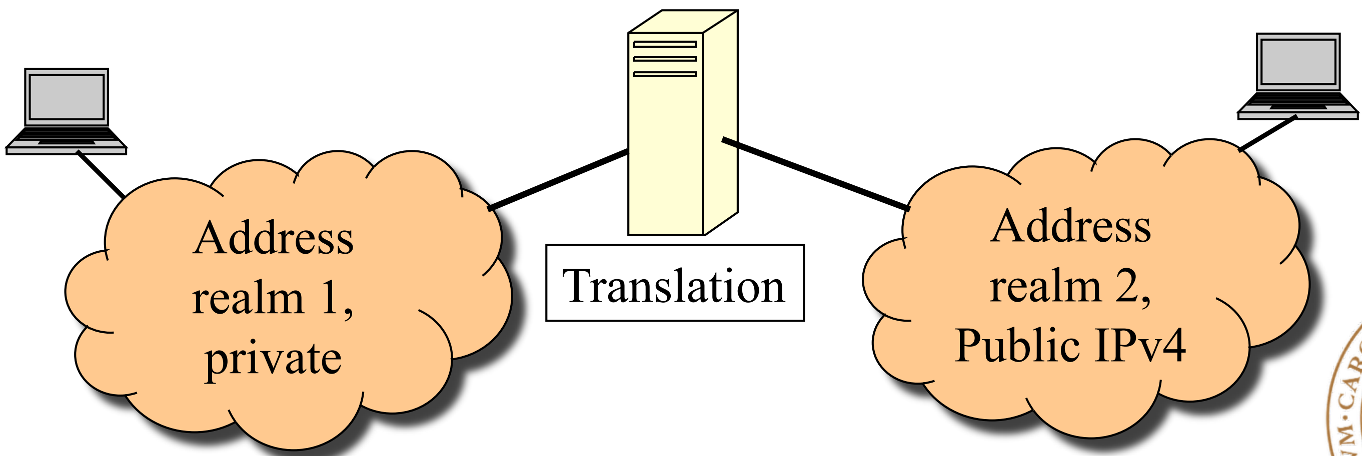
  note roundtrip time

  print output

  ttl++

Network Address Translator, NAT

- CIDR not enough, need way more addresses.
- Currently, patch is called NAT, many hosts share single public IP address



Classical NAT

- NAT has pool of public IPv4 addresses
- One public address assigned to each private node on packet arrival at NAT
- Address held until session closed or timeout

- Problem scalability, still no big gain.

NAPT

- Private hosts share a public IP address
- Each identified flow is assigned a unique sender port number
- Return packet translated to private address and port depending on dst. Port number

- Problem reachability for network initiated communication

ALG

- Another problem:
  - In-band signalling
    - SIP
    - HTML
    - Exchange
    - Netmeeting etc.
  - IP addresses are sent in data using these protocols, mismatch with NAT address

More hacks

- Look at RSA-IP, RSAP-IP, REBEKAH-IP etc. for more elaborate schemes

COMP 5116
Internet Protocols

IPv6 and migration methods

Expected outcomes

- Understanding the background
  - What's wrong with v4
  - How does v6 address this
- What else does v6 introduce
- Knowing about issues with transition from v4 to v6
- Understanding transition Mechanisms

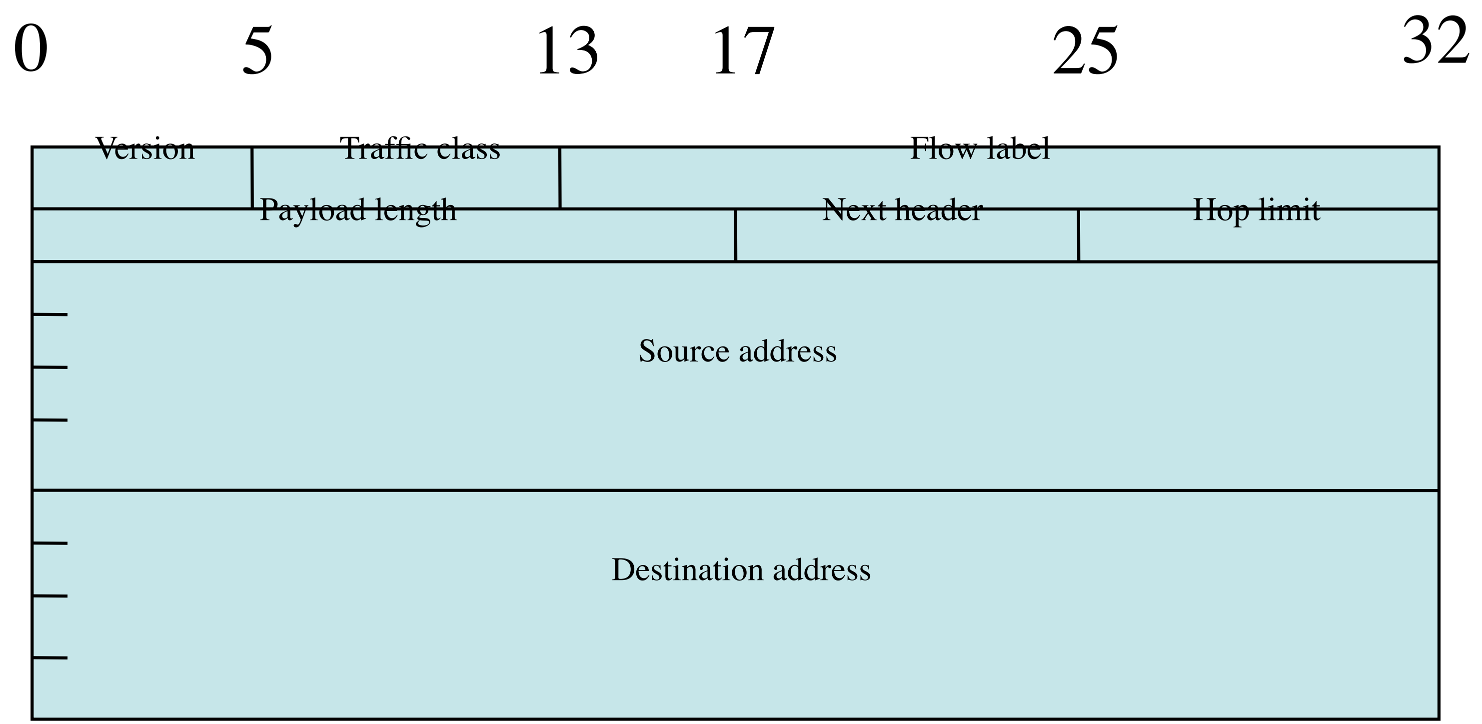
IPv6, Background

- IPv4 address space 2^{32}
  - About half assigned
  - Introduction of 3G, embedded devices etc.
- Clearly, we need a larger address space

IPv6, Background

- IPv6 address space 2^{128}
- Some other improvements over v4
  - Simple fixed 40 byte header (routing)
  - Improved encryption and authentication
  - Address auto-configuration

IPv6 Header


0
5
13
17
25
32
Version
Traffic class
Flow label
Payload length
Next header
Hop limit
Source address
Destination address

IPv6 Extension Headers

- Hop-by-hop Options
  - Information for routers, e.g. jumbogram length
- Routing
  - Source routing list
- Fragment
  - Tells end host how to reassemble packets
- Authentication (for destination host)
- Encapsulating Security Payload
  - For destination host, contains keys etc.
- Destination options (extra options for destination)

IPv6 Addressing

- in theory, 1500 or so addresses per square meter of earth's surface (2 ^128 is big number)
- Notation format FEDC:BA98:7654:3210:0000:0000:0000:0089
- Interoperability with IPv4
  - Use prefix 0000 0000
  - 0000 0000 0000 v4: IPv4 host to IPv6 host
  - 0000 0000 FFFF v4: Tunnel v6 over v4, the v4 address is the tunnel end point.
- Thus, v4 addresses can be embedded in v6 addresses
- However, if a v6 host needs to talk to a v4 host it still needs to occupy a v4 address!!!!!!!!

Local Addresses

- link-local used on single link (0xfe) 1111111010 | 0 (54 zeroes total) | if ID (64 bits)
  - auto-address configuration
  - neighbor discovery
  - no routers present
- site-local used within site only 1111111011 | 0 (38) | subnet (16) | if ID
  - routers do not forward outside site
  - intended to replace “intranet” addrs, 10.0.0.0, etc.

address high-level architecture

- FP, format prefix at FRONT is variable length

allocation	reserved	address-space-slice
reserved	00000000	1/256
unicast	001	1/8
link-local unicast	1111 1110 10	1/1024
site-local unicast	1111 1110 11	1/1024
multicast	1111 1111	1/256

IPv6 Hierarchy

- IPv4 address space completely flat (no geographic dependency)
- IPv6 semi-hierarchical (compare telephone numbers)
  - Top level routers have address ranges with regional meaning in routing tables
  - Next level routers have knowledge of ranges to organisations (corporations, ISPs etc.)
  - Site level routers have host and network specific routing tables

IPv6 Autoconfiguration

- Two methods available
  - Dynamic Host Configuration Protocol, DHCP
  - Neighbour Discovery, ND
    - Host issues Router Solicitation message on "all routers multicast address"
    - Router answers with Router Advertisement message
    - Both ICMPv6
    - Advertisement {subnet prefix:hosts 48 bit MAC address}

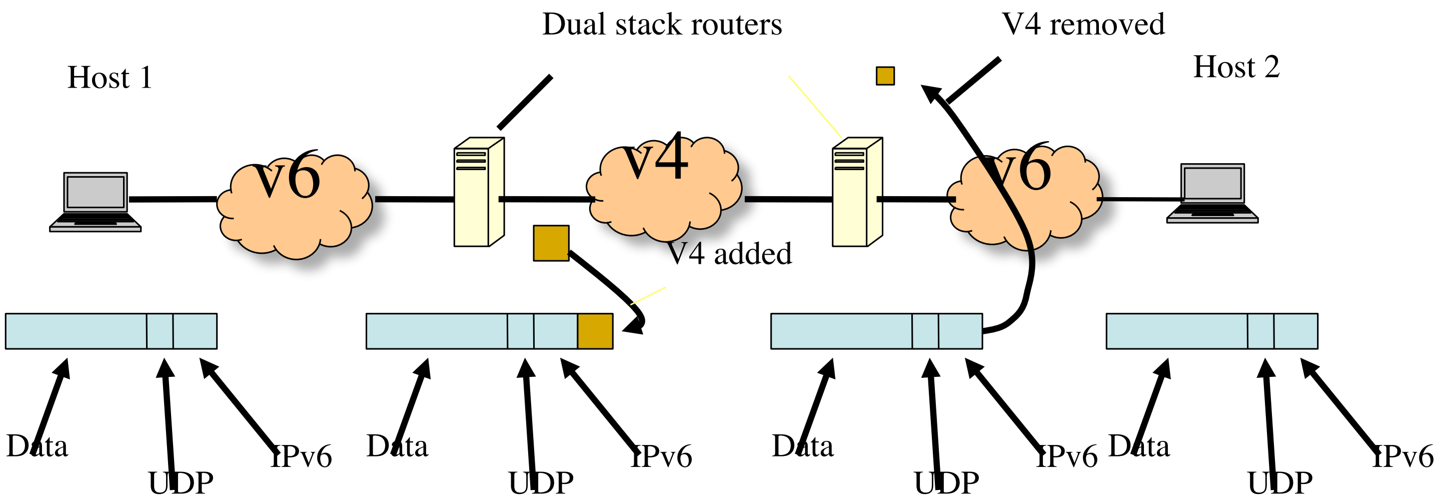
Migration Methods

- **dual-stacks**, IPv6 and IPv4
- Tunnelling
- NAT
  - Traditional NATs
  - RSIP and SIIT
  - REBEKAH-IP
- transition likely to take a very long time

Tunnelling

- tunnels: IPv6 internets can tunnel IPv6 packets over IPv4 networks, “short-term”
- if and when more IPv6, then IPv4 tunnelled over IPv6

Tunnelling


Dual stack routers
V4 removed
Host 1
Host 2
v6
v4
v6
V4 added
Data
UDP
IPv6
Data
UDP
IPv6
Data
UDP
IPv6
Data
UDP
IPv6

Further reading

- RFC 2460 Internet Protocol, Version 6 (IPv6) Specification. S. Deering, R. Hinden. December 1998.