I. More ARP

- after resolving a hardware address, why not store it?

- ARP assumes that there will most likely be more than 1 communication between two nodes, so it stores the hardware address in a table

- table referred to as the **ARP cache**

- ARP cache is small on most machines

  - if full, and another ARP request occurs, oldest entry replaced (FIFO)
  - entries are also timed-out, removed after a period of time, could be as short as 2 minutes
  - entries can also be manually added/deleted
  - on Linux, ARP data stored in file: /proc/net/arp

- why not keep cache between boot-up? Things change:

  - NICs go bad
  - IP addrs change
  - machines taken off network

- ARP Lab

  useful commands:

  - ipconfig /all (Win)
  - hostname (linux)
  - ifconfig (linux)

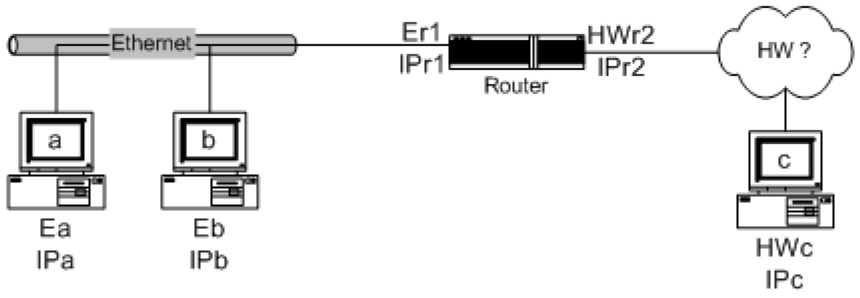
    MTU – maximum transmission unit
    TX
    RX
    lo

  - arp –a (linux & Win)
  - arp –d
  - wireshark &

    in one window, sleep 10 ; ping –c1 hostname
    in wireshark, filter: arp host hostname

II. Even More ARP

We want to send from station a to station c in the following diagram:



1. what 2 pieces of info does A need? a) C’s network addr b) C’s hardware addr

2. assuming we have the network addr for station c (e.g. IPc), what do we do to get the hardware addr? ARP.

3. who do we ARP (assuming an empty ARP cache)? IPc? (no) Why? Because we know IPc is on a different network. How do we know IPc is on a different network? How many parts to an IP address? 2, the network portion and the host portion. Thus we know the network portion of IPa is different than the network portion of IPc.

    - when building the ARP packet, we know we want the HW addr of the router so we provide the IP addr of the router, and send down to layer 2

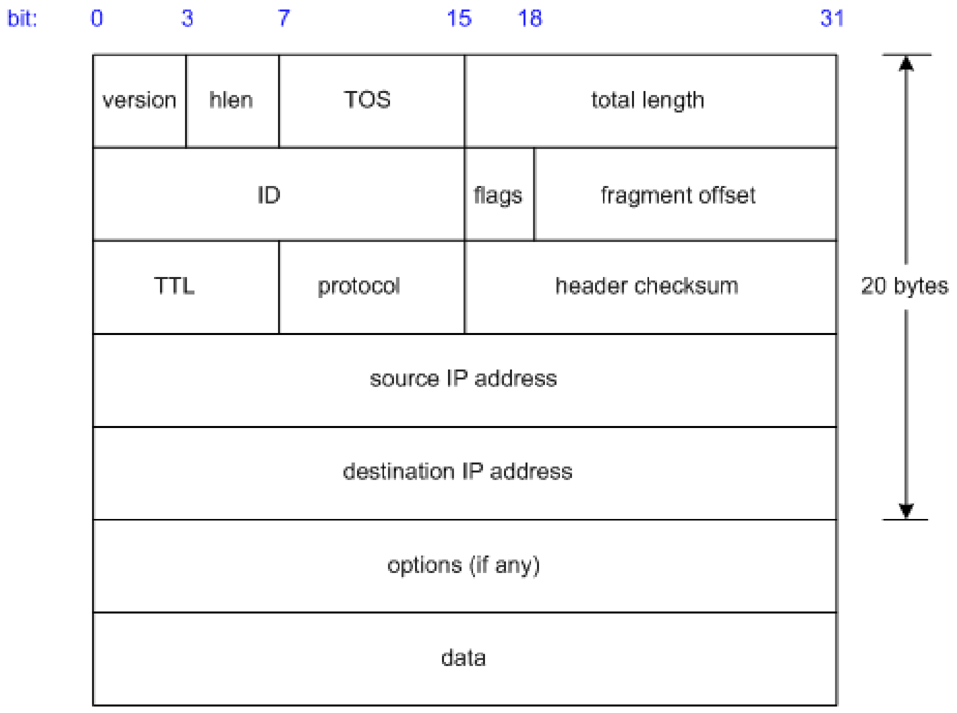
    - then at layer 2, who do we send this to? everybody on our LAN (broadcast)

4. How do we know the IP address for the router? Entered at config time. ARP for HW address the first time, then use cache.

Note that IP layer is only useful between networks, otherwise not (really) necessary.

III. Internet Protocol (IP)

- IP is responsible for datagram delivery across network boundaries
- IP is a connectionless/unreliable protocol
- IP datagram (see RFC 791)



- **version** (4 bits) – identifies the version of IP, either 4 (IPv4) or 6 (IPv6)
- **header length** (4 bits) – length of header in “32 bit words”, i.e. 4 octet chunks
  - usually 5, 5 * 32 bits = 5 x 4 octets = 20 octets
  - maximum header length: since we have 4 bits, 2^4 -1 * 4 = 60 octets
- **type of service (TOS)** (1 octets) – specifies how an upper-layer protocol would like a current datagram to be handled, and assigns datagrams various levels of importance
- **total packet length** (2 octets) - specifies the length, in octets, of the entire IP packet, including the data and header
  - min size: 21 octets
  - max size: 2^16 approx 65K octets
  - max size Ethernet frame? 1500, is this a problem?

- **identifier** (2 octets) **-** contains an integer that identifies the current datagram used to help piece together datagram fragments

- **fragment flags** (3 bits) - consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation

  - left-most bit is reserved (and must be 0, MBZ)

  - middle bit specifies whether the packet can be fragmented (0 = may fragment, 1 = do not fragment).

  - right-most bit specifies whether the packet is the last fragment in a series of fragmented packets (0 = last fragment, 1 = more fragments coming).

- **fragmentation offset** (13 bits) - indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram

- **time-to-live (TTL)** (1 octet) - maintains a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.

- **protocol** (1 octet) – indicates which upper-layer protocol receives incoming packets after IP processing is complete

1: ICMP	2: IGMP	6: TCP	17: UDP

- **header checksum** (2 octets) – error check which helps insure IP header integrity

- **source address** (4 octets) – specifies the sending node

- **destination address** (4 octets) – specifies the receiving node, destination never changes along the way

- **options** (size varies) – allows IP to support various options, such as security

- **data** (size varies) – upper layer data

➢ See ARP/Ping example. Ping non-existent hosts
➢ See ARP, ping, IP, ICMP packet document

IV. More on IP Addressing

- how big are IP (IPv4) addresses? 32 bits, how many total addresses is this? 2^32 = approx 4.3 billion

- we are currently running out of IPv4 addresses, why?

  - inefficient allocation
  - did not anticipate the need

- recall, IP addresses have how many parts? 2, the NW part and the host part

Classful IP Addressing: (see pg. 411…)

Class	Leftmost bits	Network/Host division	Dotted decimal format
A	0	8 bits NW, 24 bits host	NW.H.H.H
B	10	16 bits NW, 16 bits host	NW.NW.H.H
C	110	24 bits NW, 8 bits host	NW.NW.NW.H
D	1110	Multicast	
E	1111	Experimental	

- also denoted by /*n* notation, where *n* determines the # NW bits, e.g. A: /8, B: /16, C: /24

Class A Networks:

- max # total Class A **networks**? 2^7 = 128

- 2 **NW** addresses reserved: 0.0.0.0 & 127.0.0.0, thus max # useable networks: 128 – 2 = 126

- range: 1.xxx.xxx.xxx – 126.xxx.xxx.xxx

- max # **hosts** (per network): 2^24 = approx 16 million

- 2 host addresses reserved: all 0's & all 1's, so 2^24 – 2

- also referred to as /8 addresses, e.g. 126.0.0.0/8

- total IPv4 address space = 2^32 total Class A address space = 2^31

  2^31/2^32 = .5 thus Class A addresses make up 50% of IPv4 addresses

Class B Networks:

- max # total Class B **networks**? 2^14 (why 14, why not 16?)
- range: 128.0.xxx.xxx – 191.255.xxx.xxx
- max # hosts: 2^16 – 2 (why – 2?)

- total IPv4 address space = 2^32 total Class B address space = 2^30

  2^30/2^32 = .25 thus Class B addresses make up 25% of IPv4 addresses

Class C Networks:

- max # total Class C **networks**? 2^21 (why 21, why not 24?)
- range: 192.0.0.xxx – 223.255.255.xxx
- max # hosts: 2^8 – 2 (why – 2?) = 254

- total IPv4 address space = 2^32 total Class C address space = 2^29

  2^29/2^32 = .125 thus Class C addresses make up 12.5% of IPv4 addresses

V. Network Masks

- RFC 950 circa 1985

- recall classful addressing

Class A	N.H.H.H	1.xxx.xxx.xxx – 126.xxx.xxx.xxx	/8
Class B	N.N.H.H	128.0.xxx.xxx – 191.255.xxx.xxx	/16
Class C	N.N.N.H	192.0.0.xxx – 223.255.255.xxx	/24

- a network mask identifies **the network portion** of the IP address

- **natural** network masks

  Class A: 255.0.0.0
  Class B: 255.255.0.0
  Class C: 255.255.255.0

- network masks are and-ed with the network address to identify the network portion

- Thus given IP addr: 198.22.16.239 (class C) and the corresponding mask, we get

```
11000110  00010110  00010000  11101111            original address
11111111  11111111  11111111  00000000            Class C natural mask
______________________________________
11000110  00010110  00010000   00000000           network portion of the addr
```

- on Windows from cmd prompt: **netsh interface ipv4 show config**