TCP/IP: Internet Protocol

IP Addressing

• universal numeric identification method for effective software implementation; a sending host
  must know the IP address of the destination
  - cannot rely on physical addresses (MAC, etc) due to differing technologies
• each IP connected node is assigned a unique 32-bit address, divided into a network id and host
  id pair.
• classful IP addresses are categorized into 5 classes based on most significant bits of network id;
  creatively designed for different number of networks with varying numbers of hosts
• 3 primary classes assigned for general use, 2 reserved:

<table>
<thead>
<tr>
<th>Class</th>
<th>Address Range</th>
<th>NetID bits</th>
<th>HostID bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0nnn nnnn</td>
<td>1+7</td>
<td>24</td>
</tr>
<tr>
<td>B</td>
<td>10nn nnnn nnnn nnnn</td>
<td>2+14</td>
<td>16</td>
</tr>
<tr>
<td>C</td>
<td>110n nnnn nnnn nnnn nnnn nnnn</td>
<td>3+21</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>1110 xxxx xxxx xxxx xxxx xxxx xxxx xxxx</td>
<td>(multicasts)</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1111 xxxx xxxx xxxx xxxx xxxx xxxx xxxx</td>
<td>(reserved)</td>
<td></td>
</tr>
</tbody>
</table>

• IP network numbers for global Internet are uniquely assigned by the Internet Assigned Number
  Authority (IANA) – www.iana.org - host numbers are assigned “locally”
• Dotted quad notation: each octet is written as decimal separated by periods
  ex: ryker.aet.cup.edu = 158.83.201.201
• IP addressing weaknesses:
  a) if a host is moved from one network to another, its IP address must be changed
  b) classful allocation is wasteful
  c) IP addressing range becoming a limitation – NOTE: IPv6 designed to solve this

• subnet / classless addressing: local network/host partitioning of IP addresses on any bit boundary
  - subnet mask: a bit pattern used in routing to extract just network address from a given
    destination IP address, i.e.: N = (D & M)
  - transparent to the outside
  - CIDR notation: abbreviated form of a network address plus network mask
    ex: CalU = 158.83.0.0/16 (no subnetting), 158.83.200.0/23 (with 7 subnet bits)

• special/reserved IP addresses:

<table>
<thead>
<tr>
<th>Address</th>
<th>NetID</th>
<th>HostID</th>
<th>Meaning / Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>network</td>
<td>net address</td>
<td>all 0s</td>
<td>“this net”, used in routing configuration</td>
</tr>
<tr>
<td>directed broadcast</td>
<td>net address</td>
<td>all 1s</td>
<td>“all hosts” on specified net</td>
</tr>
<tr>
<td>local broadcast</td>
<td>all 1s</td>
<td>all 1s</td>
<td>“all hosts” on direct network</td>
</tr>
<tr>
<td>loopback</td>
<td>127</td>
<td>x.x.x</td>
<td>“IP stack only”, for software testing, no actual network traffic</td>
</tr>
</tbody>
</table>

• IP routing:
  - IP addresses (& netmasks) are also assigned to each port of routers as well as each network
    connection of every host on the network
  - both routers & hosts use this info to decide where to send packets (more later)

• multi-home hosts: host with multiple network interfaces
  - increases reliability and/or performance
Address Resolution

- Recall: in order to send a network frame to a destination, the sender must specify the destination by its underlying hardware address according to physical technology used
  - i.e. Ethernet’s MAC address
- thus: IP addresses are virtual and are not understood by the network hardware
  - must be mapped to physical addresses before sending a frame
- address resolution: process of mapping a protocol address (IP) to its respective hardware address
  - must be performed by each sending node
- address resolution algorithms:
  1. lookup table – maps IP address to a hardware address, one entry for each node
     pro: easy to implement; con: compute-intensive, useable only for very small networks
  2. direct computation – direct association between protocol address & hardware address
     pro: very fast via Boolean operations; con: not always possible
  3. message exchange – use the network! to handle resolution request/reply packets
     pro: most flexible, more dynamic; con: more complex to implement

ARP: Address Resolution Protocol (RFC 826, ‘82)

- provided by the TCP/IP suite to handle IP → hardware address mappings
- is run by all nodes using IP
- includes 2 message types: broadcasted requests & directed replies
- ARP message format on Ethernet consists of 28 octets

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>HATYPE</td>
<td></td>
<td>PATYPE</td>
<td></td>
<td>4</td>
<td>HALEN</td>
<td>PALEN</td>
<td></td>
<td>OPERATION</td>
<td></td>
<td>8</td>
<td></td>
<td>SHADDR (first 4)</td>
<td></td>
<td>12</td>
<td>SHADDR (last 2)</td>
<td>SPADDR (first 2)</td>
<td></td>
<td>16</td>
<td>SPADDR (last 2)</td>
<td>THADDR (first 2)</td>
<td></td>
<td>20</td>
<td></td>
<td>THADDR (last 4)</td>
</tr>
<tr>
<td>28 octets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARP Message: Internal Detail

- ARP message fields:
  - HATYPE - hardware address type (1 = Ethernet)
  - PATYPE - protocol address type (0x800 = IP)
  - HALEN, PALEN - number of octets in hardware, protocol address fields
  - OPERATION - i.e. “opcode”; 1 = ARP request, 2 = ARP reply
  - SHADDR, THADDR - hardware address of sender, target; 48-bits ea.
  - SPADDR, TPADDR - protocol address of sender, target; 32-bits ea.

- ARP messages are encapsulated in Ethernet frames with frame type = 0x0806

<table>
<thead>
<tr>
<th>DA</th>
<th>SA</th>
<th>0x0806</th>
<th>ARP Datagram</th>
<th>FCS</th>
</tr>
</thead>
</table>

Ethernet encapsulation of ARP Message

- each IP node uses caching to optimize future ARP requests and to reduce network traffic
- system command: `arp -v`
TCP/IP: Internet Protocol

**RARP:** *Reverse Address Resolution Protocol* (RFC 903, ‘84)

- run by a RARP server to translate from NIC address to IP address
- typically used by diskless workstations or in DHCP settings
- Ethernet frame type = \(0x8035\)

**IPv4 Datagrams:** *Internet Protocol ver. 4* (RFC 791, ‘81)

- are the basic TCP/IP unit of transfer, includes header (20 octets minimum) and data.
- are encapsulated within an Ethernet frame data field; note: *frame* vs. *datagram*
- Ethernet frame TYPE field = \(0x0800\).

### IPv4 Datagram: General Form

```
<table>
<thead>
<tr>
<th>Datagram Header</th>
<th>Datagram Data</th>
</tr>
</thead>
</table>
```

### IP Datagram: General Form

```
<table>
<thead>
<tr>
<th>0</th>
<th>3</th>
<th>4</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>19</th>
<th>23</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERS</td>
<td>HLEN</td>
<td>SERVICE TYPE</td>
<td>TOTAL LENGTH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IDENTIFICATION</td>
<td>FLAGS</td>
<td>FRAGMENT OFFSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TIME TO LIVE</td>
<td>PROTOCOL</td>
<td>HEADER CHECKSUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SOURCE IP ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>DESTINATION IP ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>IP OPTIONS (if any)</td>
<td>PADDING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Data…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### IPv4 Datagram Header Detail

- **VERS** - IP protocol version of datagram creator (4)
- **HLEN** - header length, number of 32-bit words in datagram header (normally 5)
- **SERVICE TYPE** - 5 subfields to specify how the datagram should be handled during transport
- **TOTAL LENGTH** - number of octets in entire datagram
  - note: Data Length = TOTAL LENGTH - HLEN x 4
- **IDENTIFICATION** - unique positive integer (sequence number) to identify datagram (along with IP Source, Destination, and Protocol)
- **FLAGS** - controls fragmentation of datagram during transport
- **FRAGMENT OFFSET** - specifies original datagram offset of data in this datagram fragment (in number of 8 octet units)
- **TIME TO LIVE (TTL)** - specifies how many seconds/gateway hops the datagram is allowed to “exist” during routing (1..255, default = 100)
- **PROTOCOL** - specifies which higher level protocol created message contained in DATA field and therefore the format of its content (analogous to Type Field in Ethernet frame); 1 = ICMP, 6 = TCP, 17 = UDP
- **CHECKSUM** - 1’s complement checksum of datagram header only
- **IP ADDRESSES** - 32-bit IP addresses of datagram’s sender and intended recipient
- **IP OPTIONS** - optional variable-length field used for network testing and debugging
- **PADDING** - zero bits to cause header to end on 32-bit word boundary
- **DATA** - high level protocol data, content and length defined by higher protocol
TCP/IP: Internet Protocol

**IP routing operation**

- recall: each of a router’s ports is assigned an IP addresses & netmasks consistent with the networks to which it attaches
- routers forward IP packets based on the packet’s destination IP plus the IP addresses & netmasks of its ports according to entries in its routing table and a bitmask operation…
  
  i.e., for each table entry \( i \) and a given packet \( P \) with DIP:
  
  \[
  \text{if } ( (\text{DIP} \land \text{Mask}[i]) == \text{Destination}[i] ) \text{ forward } P \text{ to NextHop}[i];
  \]

- hosts also perform “routing” function and use routers to forward IP packets to distant networks
- routing tables typically also contain a *default route* entry to handle case where no other entries apply
- system command: `netstat -r`

**MTUs & Fragmentation**

- depending on the network’s hardware technology, a maximum transfer unit (MTU) size exists for each link, ex: Ethernet’s MTU = 1500
- datagrams larger than the next hop’s MTU must be fragmented into multiple, smaller datagrams
- each fragment uses a modified copy of the original IP header with the MF bit (in the FLAGS field) set and FRAGMENT OFFSET set appropriately; MF = 0 for the last fragment

**IP Routing Protocols**

- control protocols used by network layer peers to control datagram routing and operations
- classified as either *Interior Gateway Protocols* (IGPs) or *Exterior Gateway Protocols* (EGPs)

**RIP:** *Routing Information Protocol*

**OSPF:** *Open Shortest Path First Protocol*

**BGP:** *Border Gateway Protocol*

**ICMP:** *Internet Control Message Protocol* (RFC 792, ‘81)

- used by routers and host to determine and test routes and addressing
- i.e.: PING (echo request/reply)
TCP/IP: Internet Protocol

**IPv6 Datagram** (RFC 1550, ‘93)

*Design Goal:* achieve a fast, flexible protocol with plenty of address space!

- revision of IPv4 to address problem of limited IP address space with 32 bits.
- quadruples IP address fields to 128 bits (16 octets), will *never* run out again!
  - ex: Ryker becomes 0000:0000:0000:0000:0000:0000:9E53:C9C9
  - or (using abbreviated *colon hex* notation) “::158.83.201.201”
- IPv6 address space is divided into "prefixes", provides increased flexibility
- adds notion of *anycasting* in addition to uncasting and multicasting
  - *unicast* – address specifies a single computer
  - *multicast* – address refers to a set of computers, at multiple locations
  - *anycast* – any one of a group of computers located in the same address prefix, determined by routers; useful in *cluster services*
- reduces header to only 7 fields (but increases 24 octet length to 40):

![IPv6 Datagram Diagram](image)

**IPv6 Datagram Fields:**
- **VERS** - IP protocol version of datagram creator (6)
- **PRIORITY** - provides 8 priority levels in each of 2 categories: congestible but reliable and high demand but lossy
- **FLOW LABEL** - allows for approximation of connection oriented sessions via virtual circuits
- **PAYLOAD LENGTH** - byte length of payload field (excluding header)
- **NEXT HEADER** - link to optional extension header (in payload area) used by higher level protocols
- **HOP LIMIT** - similar to TTL, limits maximum life on datagram
- **IP ADDRESSES** - 128-bit IPv6 addresses of datagram’s sender and intended recipient
- **PAYLOAD** - actual data content of datagram (follows header)

**extension headers** support additional features such as fragmentation, experimental protocols, etc.

**Checksum** field eliminated because:
- networks are now inherently more reliable
- data link and/or physical layers usually provide checksums anyway
Notes 15 Additional Info:

- network standard byte order:
  - *Little Endian* = least significant byte first
  - *Big Endian* = most significant byte first (used by IP)

- NAT: Network Address Translation, for private IP ranges / networks (add’l details later)
  - private networks:
    - 10.0.0.0-10.255.255.255 (/8)
    - 172.16.0.0-172.31.255.255 (/12)
    - 192.168.0.0-192.168.255.255 (/16)

- example of programmatic data abstraction of ARP message (pg. 2):

```c
/*
 * struct sARP - declaration of an ARP message structure
 */

typedef unsigned char  UBYTE;
typedef unsigned short UINT16;

struct sARP {
    UINT16 hatype;      // hardware address type
    UINT16 patype;      // protocol address type
    UBYTE  halen;       // hardware address length in octets
    UBYTE  palen;       // protocol address length in octets
    UINT16 operation;   // ARP operation: 1=request, 2=reply
    UBYTE  shaddr[6];   // sender hardware address
    UBYTE  spaddr[4];   // sender protocol (IP) address
    UBYTE  thaddr[6];   // target hardware address
    UBYTE  tpaddr[4];   // target protocol address
};

struct sARP am;   // ARP message

int main()
{
    printf("sizeof sARP = %d\n", sizeof(am));
}

// output:
// sizeof sARP = 28
```
/ * struct sARP - declaration of an ARP message structure
 */

typedef unsigned char UBYTE;
typedef unsigned short UINT16;

struct sARP {
    UINT16 htype;     // hardware address type
    UINT16 ptype;     // protocol address type
    UBYTE halen;      // hardware address length in octets
    UBYTE palen;      // protocol address length in octets
    UINT16 operation; // ARP operation: 1-request, 2-reply
    UBYTE shaddr[6];  // sender hardware address
    UBYTE spaddr[4];  // sender protocol (IP) address
    UBYTE thaddr[6];  // target hardware address
    UBYTE tpaddr[4];  // target protocol address
};

struct sARP am;

main()
{
    printf("sizeof sARP = %d\n", sizeof(am));
}

// output:
// sizeof sARP = 28