CNT 4007 Computer Networks
- Chapter 4 : Network Layer

Jonathan C.L. Liu, Ph.D.
Department of Computer, Information Science and Engineering (CISE), University of Florida
# IP Datagram Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version</td>
<td>32 bits</td>
</tr>
<tr>
<td>number</td>
<td></td>
</tr>
<tr>
<td>header length</td>
<td>32 bits</td>
</tr>
<tr>
<td>(bytes)</td>
<td></td>
</tr>
<tr>
<td>&quot;type&quot; of data</td>
<td></td>
</tr>
<tr>
<td>max number</td>
<td>16-bit identifier</td>
</tr>
<tr>
<td>remaining hops</td>
<td>flgs fragment</td>
</tr>
<tr>
<td>(decremented at each router)</td>
<td>offset</td>
</tr>
<tr>
<td>time to live</td>
<td>upper layer</td>
</tr>
<tr>
<td>header checksum</td>
<td>length total datagram length (bytes)</td>
</tr>
<tr>
<td>Options (if any)</td>
<td>for fragmentation/reassembly</td>
</tr>
<tr>
<td>upper layer protocol</td>
<td>32 bit source IP address</td>
</tr>
<tr>
<td>to deliver payload to</td>
<td>32 bit destination IP address</td>
</tr>
<tr>
<td></td>
<td>Options (if any)</td>
</tr>
<tr>
<td></td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>(variable length, typically a TCP or UDP segment)</td>
</tr>
</tbody>
</table>

**How much overhead with TCP?**
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer

E.g. timestamp, record route taken, specify list of routers to visit.
network links have MTU (max.transfer size) - largest possible link-level frame. different link types, different MTUs. large IP datagram divided ("fragmented") within net one datagram becomes several datagrams "reassembled" only at final destination IP header bits used to identify, order related fragments
Example

- 4000 byte datagram
- MTU = 1500 bytes

1480 bytes in data field

offset = 1480/8

One large datagram becomes several smaller datagrams
• **IP address**: 32-bit identifier for host, router *interface*

• **interface**: connection between host/router and physical link

router’s typically have multiple interfaces

host typically has one interface

IP addresses associated with each interface

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1 1
Subnets

- **IP address:** subnet part (high order bits) host part (low order bits)

- **What's a subnet?**
  - device interfaces with same subnet part of IP address can physically reach each other without intervening router
**Recipe**

To determine the subnets, detach each interface from its host or router, creating islands of isolated networks. Each isolated network is called a subnet.
Subnets

How many?
**CIDR**: Classless InterDomain Routing

subnet portion of address of arbitrary length

address format: **a.b.c.d/x**, where x is # bits in subnet portion of address

```
  11001000  00010111  00010000  00000000

200.23.16.0/23
```
Q: How does a host get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- **DHCP**: Dynamic Host Configuration Protocol: dynamically get address from as server
  “plug-and-play”
Dynamic Host Configuration Protocol

DHCP overview:
host broadcasts “DHCP discover” msg
DHCP server responds with “DHCP offer” msg
host requests IP address: “DHCP request” msg
DHCP server sends address: “DHCP ack” msg
DHCP client-server scenario

DHCP server: 223.1.2.5

DHCP discover
src: 0.0.0.0, 68
dest: 255.255.255.255, 67
yiaddr: 0.0.0.0
transaction ID: 654

DHCP offer
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs

DHCP request
src: 0.0.0.0, 68
dest: 255.255.255.255, 67
yiaddr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

DHCP ACK
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs
**Q:** How does *network* get subnet part of IP addr?

**A:** gets allocated portion of its provider ISP’s address space

<table>
<thead>
<tr>
<th>ISP's block</th>
<th>Organization 0</th>
<th>Organization 1</th>
<th>Organization 2</th>
<th>...</th>
<th>Organization 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000</td>
<td>11001000 00010111 00010000 00000000</td>
<td>11001000 00010111 00010010 00000000</td>
<td>11001000 00010111 00010100 00000000</td>
<td>...</td>
<td>11001000 00010111 00011110 00000000</td>
</tr>
<tr>
<td>200.23.16.0/20</td>
<td>200.23.16.0/23</td>
<td>200.23.18.0/23</td>
<td>200.23.20.0/23</td>
<td>...</td>
<td>200.23.30.0/23</td>
</tr>
</tbody>
</table>
Hierarchical addressing allows efficient advertisement of routing information:

Organization 0
200.23.16.0/23
Organization 1
200.23.18.0/23
Organization 2
200.23.20.0/23
Organization 7
200.23.30.0/23

"Send me anything with addresses beginning 200.23.16.0/20"

"Send me anything with addresses beginning 199.31.0.0/16"
Hierarchical addressing: more specific routes

ISP R Us has a more specific route to Organization 1

Send me anything with addresses beginning 200.23.16.0/20

Send me anything with addresses beginning 200.23.18.0/23
**IP addressing: the last word...**

**Q:** How does an ISP get block of addresses?

**A:** ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes
All datagrams leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)
Motivation: local network uses just one IP address as far as outside world is concerned: range of addresses not needed from ISP: just one IP address for all devices can change addresses of devices in local network without notifying outside world can change ISP without changing addresses of devices in local network devices inside local net not explicitly addressable, visible by outside world (a security plus).
NAT: Network Address Translation

Implementation: NAT router must:

- **outgoing datagrams: replace** (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  
  ... remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- **remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair

- **incoming datagrams: replace** (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table
**NAT: Network Address Translation**

1: Host 10.0.0.1 sends datagram to 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

3: Reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

**NAT translation table**

<table>
<thead>
<tr>
<th>WAN side addr</th>
<th>LAN side addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>138.76.29.7, 5001</td>
<td>10.0.0.1, 3345</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

**Diagram Notes:**
- **WAN side addr** refers to the address seen from the internet.
- **LAN side addr** refers to the address seen from the local network.
- The NAT router handles the translation between these two addresses.
16-bit port-number field:
60,000 simultaneous connections with a single LAN-side address!

NAT is controversial:
- routers should only process up to layer 3
- violates end-to-end argument
  - NAT possibility must be taken into account by app designers, eg, P2P applications

address shortage should instead be solved by IPv6