## IP address format:

<table>
<thead>
<tr>
<th>Class</th>
<th>7</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Network ID</td>
<td>Host ID</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>Network ID</td>
<td>Host ID</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Network ID</td>
<td>Host ID</td>
</tr>
<tr>
<td>D</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multicast Address</td>
<td></td>
</tr>
</tbody>
</table>

Dotted decimal notation: 10000000 00001011 00000011 00011111 ↔ 128.11.3.31

Symbolic name to IP address translation: domain name server (DNS).
Notice hierarchical organization ("2-level").

Each interface (NIU) has an IP address; single host can have multiple IP addresses.

Running out of addresses...
Potential problem: Waste of address space. Giving each network own network ID is inefficient.

Solution: *Subnetting*; group several physical networks into one.

To determine subnet ID:

- Perform ANDing of IP address and subnet mask
- 3-level hierarchy
Forwarding and address resolution:

\[\rightarrow \text{ mechanics of routing}\]

\[\rightarrow \text{ routing table is given}\]

<table>
<thead>
<tr>
<th>Subnet ID</th>
<th>Subnet Mask</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.10.2.0</td>
<td>255.255.255.0</td>
<td>Interface 0</td>
</tr>
<tr>
<td>128.10.3.0</td>
<td>255.255.255.0</td>
<td>Interface 1</td>
</tr>
<tr>
<td>128.10.4.0</td>
<td>255.255.255.0</td>
<td>128.10.4.250</td>
</tr>
</tbody>
</table>

Either destination host is directly connected on the same LAN or not.
Table look-up I:

- For each entry, compute \( \text{DestSubnetID} = \text{DestAddr} \text{ AND SubnetMask} \).

- Compare \( \text{DestSubnetID} \) with \( \text{SubnetID} \) and take forwarding action.

One more task left: Translate destination (or next hop IP address into LAN address.

\[\rightarrow\] required to hand-off packet to LAN layer

\[\rightarrow\] address resolution protocol (ARP)
Table look-up II:

- If ARP table contains entry, using LAN address send to destination.
- If ARP table does not contain entry, broadcast ARP Request packet with destination IP address.
  - e.g., Ethernet broadcast address: all 1’s
- Encapsulate ARP packet into LAN frame.
- Update ARP table upon receipt of reply.

Dynamically maintain ARP table: use timer for each entry (15 min) to invalidate entries.

→ aging
Other approaches to solve address depletion problem:

- IPv6
  - → 128 bits

- classless IP addressing
  - → $a.b.c.d/x$
  - → variable length subnetting
  - → interdomain routing
  - → CIDR (classless interdomain routing)

- dynamically assigned IP addresses
  - → reusable
  - → e.g., DHCP (dynamic host configuration protocol)
IETF standards: RFC (Request for Comments)

- RFC 791 (IP)
- RFC 826 (ARP)
- RFC 903 (RARP)
- RFC 894 (Ethernet)
- RFC 793 (TCP)
- RFC 768 (UDP)
- etc.
Transport Protocols: TCP/UDP Structure

- end-to-end mechanism
- runs on top of link-based mechanism
- treat network layer as black box

Three-level encapsulation:

```
[ MAC  |  IP  |  TCP/UDP  |  Payload (TCP/UDP)  |  MAC Trailer  ]
```

```
| Payload (IP)  |
```

```
| Payload (MAC)  |
```
Network layer assumptions:

- unreliable
- out-of-order delivery (not frequent)
- absence of QoS guarantees (delay, throughput, etc.)
- insecure (IPv4)
  \[ \rightarrow \text{IPsec} \]

Additional (informal) performance properties:

- Works “fine” under low load conditions
- Can break down under high load conditions
- Behavioral range wide: to some extent predictable
Goal of UDP (user datagram protocol):

→ process identification

→ port number as demux key

→ minimal support beyond IP
UDP packet format:

```
<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>
```

Checksum calculation (pseudo header):

```
| Source Address |
| Source Address |
| Destination Address |
| 00 · · · 0  | Protocol | UDP Length |
```

→ pseudo header, UDP header and payload
UDP usage:

- Multimedia streaming
  → at minimum requires process identification
  → congestion control carried out above UDP

- Stateless client/server applications
  → persistent state a hinderance
  → lightweight
Goals of TCP (transmission control protocol):

- process identification
- reliable communication: ARQ
- speedy communication: congestion control
- segmentation

→ connection-oriented, i.e., stateful
→ complex mixture of functionalities
Segmentation task: Provide “stream” interface to higher level protocols

→ view: contiguous stream of bytes

• segment stream of bytes into blocks or segments of fixed size

• segment size determined by TCP MTU (Maximum Transmission Unit)

• use also for reliability mechanism
TCP packet format:

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sequence Number

Acknowledgement Number

<table>
<thead>
<tr>
<th>Header Length</th>
<th>U R C S T P H F</th>
<th>Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Checksum

Urgent Pointer

Options (if any)

DATA (if any)
- Sequence Number: position of first byte of payload
- Acknowledgement: next byte of data expected (receiver)
- Header Length (4 bits): 4 B units
- URG: urgent pointer flag
- ACK: ACK packet flag
- PSH: override TCP buffering
- RST: reset connection
- SYN: establish connection
- FIN: close connection
- Window Size: receiver’s advertised window size
- Checksum: prepend pseudo-header
- Urgent Pointer: byte offset in current payload where urgent data begins
- Options: MTU; take min of sender & receiver (default 556 B)
Checksum calculation (pseudo header):

\[
\begin{array}{|c|c|c|}
\hline
\text{Source Address} & \text{Destination Address} & \text{Protocol} & \text{TCP Segment Length} \\
\hline
00 \cdots 0 & & \\
\hline
\end{array}
\]

\[\rightarrow\] pseudo header, TCP header and payload