## Special Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>absence of address</td>
</tr>
<tr>
<td>127.0.0/24</td>
<td>this host (loopback address) for example 127.0.0.1</td>
</tr>
<tr>
<td>10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16</td>
<td>private networks (e.g. in IEW) cannot be used on the public Internet</td>
</tr>
<tr>
<td>169.254.0.0/16</td>
<td>link local address (can be used only between systems on same LAN)</td>
</tr>
<tr>
<td>224/4</td>
<td>multicast</td>
</tr>
<tr>
<td>240/5</td>
<td>reserved</td>
</tr>
<tr>
<td>255.255.255.255/32</td>
<td>link local broadcast</td>
</tr>
</tbody>
</table>
## Examples of Special Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>::/128</td>
<td>absence of address</td>
</tr>
<tr>
<td>::1/128</td>
<td>this host (loopback address)</td>
</tr>
<tr>
<td>fc00::/7</td>
<td>Unique local addresses (i.e. fcxx: and fdxx:)</td>
</tr>
<tr>
<td></td>
<td>= private networks (e.g. in IEW)</td>
</tr>
<tr>
<td></td>
<td>cannot be used on the public Internet</td>
</tr>
<tr>
<td>fd24:ec43:12ca:1a6:a00::20ff:fe78:30f9</td>
<td>For example</td>
</tr>
<tr>
<td>fc00::/7</td>
<td>EPFL Private</td>
</tr>
<tr>
<td>fe80::/10</td>
<td>link local address (can be used only between systems on same LAN)</td>
</tr>
<tr>
<td>ff00::/8</td>
<td>multicast</td>
</tr>
<tr>
<td>ff02::1:ff00:0/104</td>
<td>Solicited node multicast</td>
</tr>
<tr>
<td>ff02::1/128</td>
<td>link local broadcast</td>
</tr>
<tr>
<td>ff02::2/128</td>
<td>all link local routers</td>
</tr>
</tbody>
</table>
IPv4 Packet Format

We will see the functions of the fields other than the addresses in a following module.
IPv6 Packet Format

We will see the functions of the fields other than the addresses in a following module.

Higher layer protocol
1 = ICMP, 6 = TCP, 17 = UDP

e.g.
Ethernet Frame format

Ethernet frame = Ethernet PDU
An Ethernet frame typically transports an IP packet, sometimes also other

Type of protocol contained in the Ethernet packet (hexa):
0800: IPv4
0806: ARP (used by IPv4)
86DD: IPv6
8847: MPLS unicast
88F7: Precision Time Protocol

Ethernet V.2 frame
- preamble
- SFD
- DA
- SA
- Type
- MAC payload
  - e.g. IPv4 packet
- FCS

DA = destination address
SA = source address

bits used to detect start of frame
MAC header
MAC payload
MAC trailer
Multicast MAC Addresses

IP multicast address is algorithmically mapped to a multicast MAC address.

Last 23 bits of IPv4 multicast address are used in MAC address

Last 32 bits of IPv6 multicast address are used in MAC address

<table>
<thead>
<tr>
<th>MAC multicast addr.</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-00-5e-XX-XX-XX</td>
<td>IPv4 multicast</td>
</tr>
<tr>
<td>33-33-XX-XX-XX-XX</td>
<td>IPv6 multicast</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP dest address</th>
<th>229.130.54.207</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP dest address (hexa)</td>
<td>e5-82-36-cf</td>
</tr>
<tr>
<td>IP dest address (bin)</td>
<td>...-10000010-...</td>
</tr>
<tr>
<td>Keep last 23 bits (bin)</td>
<td>...-00000010-...</td>
</tr>
<tr>
<td>Keep last 23 bits (hexa)</td>
<td>02-36-cf</td>
</tr>
<tr>
<td>MAC address</td>
<td>01-00-5e-03-36-cf</td>
</tr>
</tbody>
</table>
SLAAC Step 2: Duplicate Test

A sends a Neighbour Solicitation (NS) message to check for address duplication, sent to the Solicited Node Multicast Address.

Any host that would have to same link local address listens to this multicast address.
UDP Uses Port Numbers

<table>
<thead>
<tr>
<th>IP header</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP Source Port</strong></td>
<td><strong>UDP Dest Port</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UDP Message Length</strong></td>
<td><strong>UDP Checksum</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **IP SA=A DA=B prot=UDP**
- **source port=1267**
- **destination port=53**
- ...data...
### TCP Header

- **src port**: Source port number
- **dest port**: Destination port number
- **sequence number**: Sequence number
- **ack number**: Acknowledgment number
- **hlen**: Header length
- **rsvd**: Reserved field
- **flags**: Flags
- **window**: Window size
- **checksum**: Checksum
- **urgent pointer**: URGENT pointer
- **options (SACK, ...)**: Options
- **padding**: Padding
- **segment data (if any)**: Data

### Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>used for explicit congestion notification</td>
</tr>
<tr>
<td>CWR</td>
<td>used for explicit congestion notification</td>
</tr>
<tr>
<td>ECN</td>
<td>used for explicit congestion notification</td>
</tr>
<tr>
<td>urg</td>
<td>urgent ptr is valid</td>
</tr>
<tr>
<td>ack</td>
<td>ack field is valid</td>
</tr>
<tr>
<td>psh</td>
<td>this seg requests a push</td>
</tr>
<tr>
<td>rst</td>
<td>reset the connection</td>
</tr>
<tr>
<td>syn</td>
<td>connection setup</td>
</tr>
<tr>
<td>fin</td>
<td>sender has reached end of byte stream</td>
</tr>
</tbody>
</table>

### IP Header

- **src port**: Source port number
- **dest port**: Destination port number
- **sequence number**: Sequence number
- **ack number**: Acknowledgment number
- **hlen**: Header length
- **rsvd**: Reserved field
- **flags**: Flags
- **window**: Window size
- **checksum**: Checksum
- **urgent pointer**: URGENT pointer
- **options (SACK, ...)**: Options
- **padding**: Padding
- **segment data (if any)**: Data

**TCP header**

- (20 Bytes + options)

**<= MSS bytes**
Dijkstra’s Shortest Path Algorithm

The nodes are 0...\(N\); the algorithm computes shortest paths from node 0.

\(c(i,j)\): cost of link \((i,j)\).

\(V\): set of nodes visited so far.

\(\text{pred}(i)\): estimated set of predecessors of node \(i\) along a shortest path (multiple shortest paths are possible).

\(m(j)\): estimated distance from node 0 to node \(j\).

At completion, \(m(i)\) is the true distance from 0 to \(i\).

\[
\begin{align*}
m(0) &= 0; \ m(i) = \infty \ \forall \ i \neq 0; \ V = \emptyset \ ; \ \text{pred}(i) = \emptyset \ \forall i; \\
\text{for } k = 0: N \text{ do} \\
&\quad \text{find } i \notin V \text{ that minimizes } m(i) \\
&\quad \text{if } m(i) \text{ is finite} \\
&\quad \quad \text{add } i \text{ to } V \\
&\quad \quad \text{for all neighbours } j \notin V \text{ of } i \\
&\quad \quad \quad \text{if } m(i) + c(i,j) < m(j) \\
&\quad \quad \quad \quad m(j) = m(i) + c(i,j) \\
&\quad \quad \quad \quad \text{pred}(j) = \{i\} \\
&\quad \quad \text{else if } m(i) + c(i,j) = m(j) \\
&\quad \quad \quad m(j) = m(i) + c(i,j) \\
&\quad \quad \quad \text{pred}(j) = \text{pred}(j) \cup \{i\}
\end{align*}
\]
Practical Aspects

OSPF packet are sent directly over IP (OSPF=protocol 89 (0x59)). Reliable transmission is managed by OSPF with OSPF Acks and timers.

OSPFv2 supports IPv4 only
OSPFv3 supports IPv6 and dual-stack networks

OSPF routers are identified by a 32 bit number
OSPF areas are identified by a 32 bit number
Algorithm BF-C
input: a directed graph with links costs $A(i,j)$; assume $A(i, j) > 0$ and $A(i, j) = \infty$ when nodes $i$ and $j$ are not connected.
output: vector $p$ s.t. $p(i) =$ cost of best path from node $i$ to node 1

$p^0(1) = 0$, $p^0(i) = \infty$ for $i \neq 1$

for $k = 1, 2, \ldots$ do
  $p^k(i) = \min_{j \neq i} [A(i, j) + p^{k-1}(j)]$ for $i \neq 1$
  $p^k(1) = 0$
until $p^k = p^{k-1}$
return($p^k$)
Distributed Bellman-Ford

Requires only to remember distance from self to destination + the best neighbor (nextHop(i))
and works for all initial conditions

**Distributed Bellman-Ford Algorithm, BF-D**

node $i$ maintains an estimate $q(i)$ of the distance $p(i)$ to node 1;
node $i$ remembers the best neighbor nextHop(i)

initial conditions are arbitrary but $q(1) = 0$ at all steps;

from time to time, $i$ sends its value $q(i)$ to all neighbors

when $i$ receives an updated value $q(j)$ from $j$, node $i$ recomputes $q(i)$:

eq (2) \quad \text{if } j == \text{nextHop}(i)
\quad \text{then } q(i) \leftarrow A(i, j) + q(j)
\quad \text{else } q(i) \leftarrow \min(A(i, j) + q(j), q(i))

if eq(2) causes $q(i)$ to be modified, nextHop(i) $\leftarrow j$
The Decision Process

The decision process decides which route is selected; At most one best route to exactly the same prefix is chosen
    Only one route to 2.2/16 can be chosen
    But there can be different routes to 2.2.2/24 and 2.2/16
A route can be selected only if its next-hop is reachable
Routes are compared against each other using a sequence of criteria, until only one route remains. A common sequence is
0. Highest weight (Cisco proprietary)
1. Highest LOCAL-PREF
2. Shortest AS-PATH
3. Lowest MED, if taken seriously by this network
4. E-BGP > I-BGP
5. Shortest path to NEXT-HOP, according to IGP
6. Lowest BGP identifier (router-id of the BGP peer from whom route is received)
(The Cisco and FRR implementation of BGP, used in lab, have a few additional cases, not shown here)
Fairness of TCP Reno

For long lived flows, the rates obtained with TCP are as if they were distributed according to utility fairness, with utility of flow $i$ given by

$$U(x_i) = \frac{\sqrt{2}}{\tau_i} \arctan \frac{x_i \tau_i}{\sqrt{2}}$$

with $x_i = \text{rate} = \frac{W}{\tau_i}, \quad \tau_i = \text{RTT}$

For sources that have same RTT, the fairness of TCP is between maxmin fairness and proportional fairness, closer to proportional fairness.

rescaled utility functions;
RTT = 100 ms
maxmin approx. is $U(x) = 1 - x^{-5}$
TCP Reno

Loss - Throughput Formula

Consider a *large* TCP connection (many bytes to transmit)

Assume we observe that, in average, a fraction $q$ of packets is lost (or marked with ECN)

The throughput should be close to $\theta = \frac{MSS \times 1.22}{RTT \times \sqrt{q}}$

Formula assumes: transmission time negligible compared to RTT, losses are rare, time spent in Slow Start and Fast Recovery negligible, losses occur periodically
Cubic’s Other Bells and Whistles

Cubic’s Loss throughput formula
\[ \theta \approx \max\left( \frac{1.054}{\text{RTT}^{0.25}q^{0.75}}, \frac{1.22}{\text{RTT} \sqrt{q}} \right) \]
in MSS per second.

Cubic’s formula is same as Reno for small RTTs and small BW-delay products.

Other Cubic details

\[ W_{max} \] computation uses a more complex mechanism called “fast convergence”
see Latest IETF Cubic RFC / Internet Draft