# IPv4 Addressing and IPv6

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

November 29, 2017

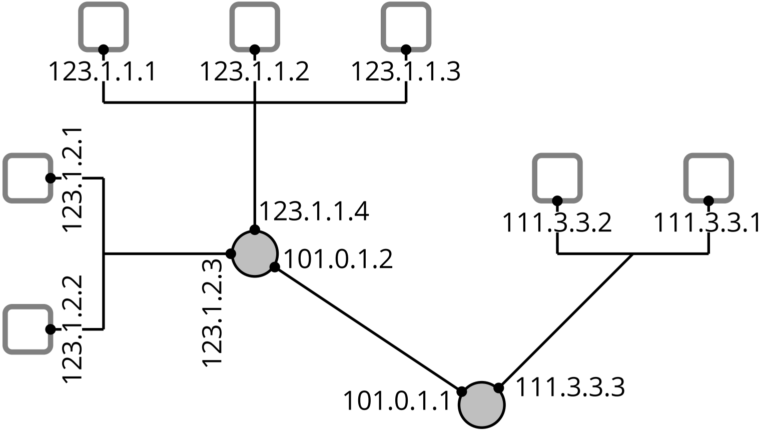
# Outline

- IPv4 Addressing
  - network addresses
  - classless interdomain routing
  - address allocation and routing
  - longest-prefix matching

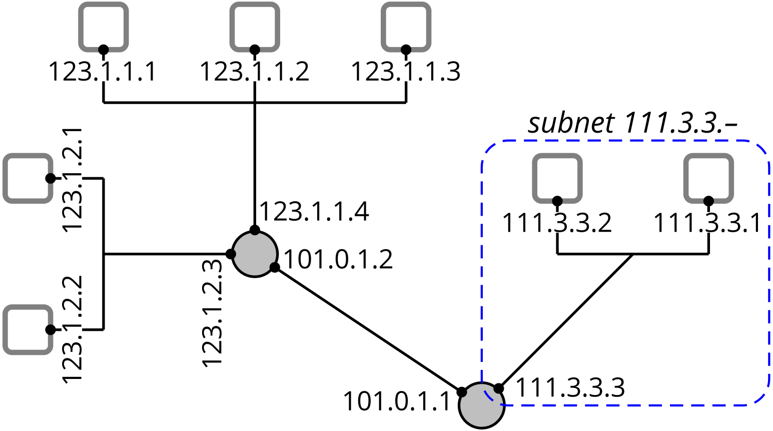
# Outline

- IPv4 Addressing
    - network addresses
    - classless interdomain routing
    - address allocation and routing
    - longest-prefix matching
- IPv6
    - motivations and design goals
    - datagram format
    - comparison with IPv4
    - extensions

# *Inter*connection of *Net*works



# *Inter*connection of *Net*works



# IPv4 Addressing

- 32-bit *addresses*

# IPv4 Addressing

- 32-bit *addresses*

- An IP address is associated with an ***interface***, not a host
  - a host with more than one interface may have more than one IP address

# IPv4 Addressing

- 32-bit *addresses*
- An IP address is associated with an ***interface***, not a host
  - a host with more than one interface may have more than one IP address
- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding

# IPv4 Addressing

- 32-bit *addresses*

- An IP address is associated with an ***interface***, not a host
  - a host with more than one interface may have more than one IP address

- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding

- The key idea is to assign addresses with the ***same prefix*** to interfaces that are on the ***same subnet***

# Classless Interdomain Routing

# Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
  - e.g., in the previous example we have
    123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—

# Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
  - e.g., in the previous example we have
    123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—

- Network addresses prefix-length notation: ***address/prefix-length***

# Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
  - e.g., in the previous example we have
    123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***
  - e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24

# Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
  - e.g., in the previous example we have
    123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***
  - e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24
  - 123.1.1.0/24 means that all the addresses share the same leftmost 24 bits with address 123.1.1.0

# Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
  - e.g., in the previous example we have
    123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—

- Network addresses prefix-length notation: ***address/prefix-length***
  - e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24
  - 123.1.1.0/24 means that all the addresses share the same leftmost 24 bits with address 123.1.1.0

- This addressing scheme is not limited to entire bytes. For example, a network address might be 128.138.207.160/27

# Examples

- Network address 128.138.207.160/27

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 101}^{\text{subnet}}\ 00000_{\text{two}}$$

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\ 00000_{\text{two}}$$

128.138.207.185?

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

128.138.207.185?

$$10000000 \quad 10001010 \quad 11001111 \quad 10111001_{\text{two}}$$

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 101}^{\text{subnet}}\ 00000_{\text{two}}$$

128.138.207.185?

$$10000000 \quad 10001010 \quad 11001111 \quad 10111001_{\text{two}}$$

128.138.207.98?

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

128.138.207.185?

$$10000000 \quad 10001010 \quad 11001111 \quad 10111001_{\text{two}}$$

128.138.207.98?

$$10000000 \quad 10001010 \quad 11001111 \quad 01100010_{\text{two}}$$

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

128.138.207.185?

$$10000000\quad 10001010\quad 11001111\quad 10111001_{\text{two}}$$

128.138.207.98?

$$10000000\quad 10001010\quad 11001111\quad 01100010_{\text{two}}$$

128.138.207.194?

# Examples

- Network address 128.138.207.160/27

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

128.138.207.185?

$$10000000\quad 10001010\quad 11001111\quad 10111001_{\text{two}}$$

128.138.207.98?

$$10000000\quad 10001010\quad 11001111\quad 01100010_{\text{two}}$$

128.138.207.194?

$$10000000\quad 10001010\quad 11001111\quad 11000010_{\text{two}}$$

# Ranges

- What is the range of addresses in 128.138.207.160/27?

# Ranges

- What is the range of addresses in 128.138.207.160/27?

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\ 00000_{\text{two}}$$

# Ranges

- What is the range of addresses in 128.138.207.160/27?

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

$$\begin{array}{llll}
10000000 & 10001010 & 11001111 & 10100000_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100001_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100010_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100011_{\text{two}} \\
 & & \vdots & \\
10000000 & 10001010 & 11001111 & 10111111_{\text{two}}
\end{array}$$

# Ranges

- What is the range of addresses in 128.138.207.160/27?

$$\overbrace{10000000\quad 10001010\quad 11001111\quad 101}^{\text{subnet}}\,00000_{\text{two}}$$

$$\begin{array}{llll}
10000000 & 10001010 & 11001111 & 10100000_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100001_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100010_{\text{two}} \\
10000000 & 10001010 & 11001111 & 10100011_{\text{two}} \\
 & & \vdots & \\
10000000 & 10001010 & 11001111 & 10111111_{\text{two}}
\end{array}$$

128.138.207.160–128.138.207.191

# Net Mask

- Network addresses, *mask* notation: ***address/mask***

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\ \text{times}}\ \overbrace{00\cdots0}^{32-p\ \text{times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\ \text{times}}\ \overbrace{00\cdots0}^{32-p\ \text{times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=?

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\text{ times}}\ \overbrace{00\cdots0}^{32-p\text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\text{ times}}\ \overbrace{00\cdots0}^{32-p\text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=?

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\text{ times}}\ \overbrace{00\cdots0}^{32-p\text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=192.168.0.3/255.255.255.0

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length *p* corresponds to a mask

$$M = \overbrace{11\cdots1}^{p \text{ times}}\ \overbrace{00\cdots0}^{32-p \text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=192.168.0.3/255.255.255.0
  - 195.176.181.11/32=?

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length *p* corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\text{ times}}\ \overbrace{00\cdots0}^{32-p\text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=192.168.0.3/255.255.255.0
  - 195.176.181.11/32=195.176.181.11/255.255.255.255

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length *p* corresponds to a mask

$$M = \overbrace{11\cdots1}^{p\text{ times}}\ \overbrace{00\cdots0}^{32-p\text{ times}}{}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=192.168.0.3/255.255.255.0
  - 195.176.181.11/32=195.176.181.11/255.255.255.255

- In Java:

# Net Mask

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length $p$ corresponds to a mask

$$M = \overbrace{11\cdots1}^{p \text{ times}}\ \overbrace{00\cdots0}^{32-p \text{ times}}\ {}_{\text{two}}$$

  - e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
  - 127.0.0.1/8=127.0.0.1/255.0.0.0
  - 192.168.0.3/24=192.168.0.3/255.255.255.0
  - 195.176.181.11/32=195.176.181.11/255.255.255.255

- In Java:

```
int match(int address, int network, int mask) {
    return (address & mask) == (network & mask);
}
```

# Classless Interdomain Routing

# Classless Interdomain Routing

- This *any-length prefix* scheme is also called ***classless interdomain routing*** (CIDR)
  - as opposed to the original scheme which divided the address space in “classes”

| *address class* | *prefix length* |
|---|---|
| A | 8 |
| B | 16 |
| C | 24 |

# Classless Interdomain Routing

- This *any-length prefix* scheme is also called ***classless interdomain routing*** (CIDR)
  - as opposed to the original scheme which divided the address space in “classes”

| *address class* | *prefix length* |
|---|---|
| A | 8 |
| B | 16 |
| C | 24 |

- Why is the idea of the common prefix so important?

# Classless Interdomain Routing

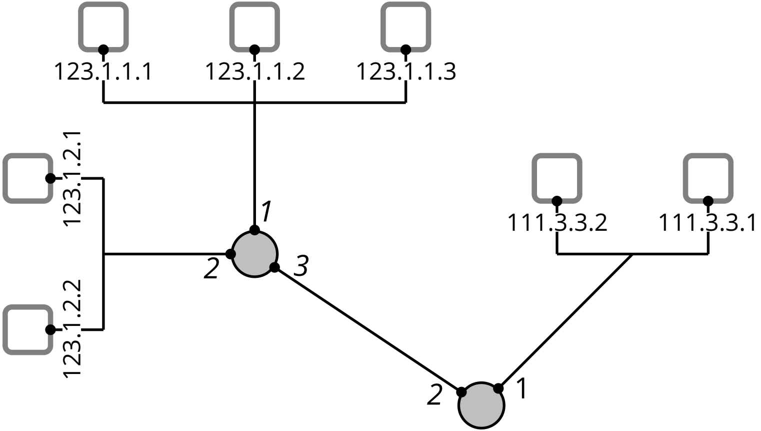
- This *any-length prefix* scheme is also called ***classless interdomain routing*** (CIDR)
  - as opposed to the original scheme which divided the address space in “classes”

| *address class* | *prefix length* |
|---|---|
| A | 8 |
| B | 16 |
| C | 24 |

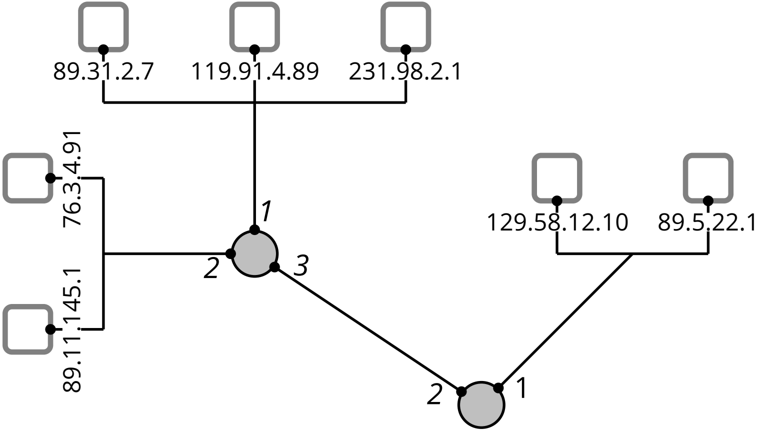
- Why is the idea of the common prefix so important?

- Routers outside a (sub)network can ignore the specifics of each address within the network
  - there might be some 64 thousands hosts in 128.138.0.0/16, but they all appear as one address from the outside

# Example: Good Address Allocation

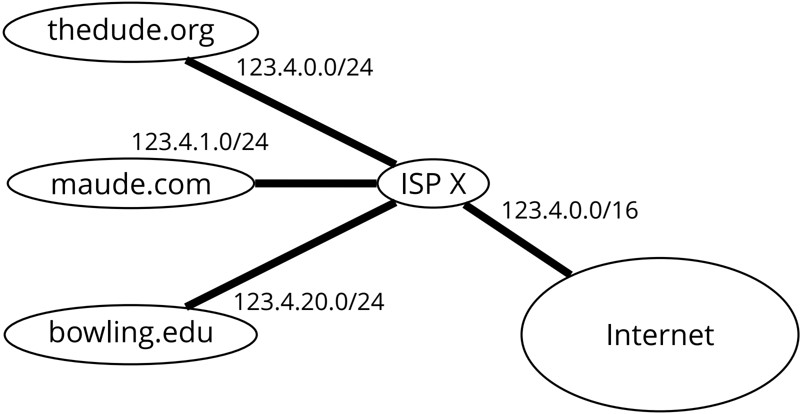


# Example: Bad Address Allocation

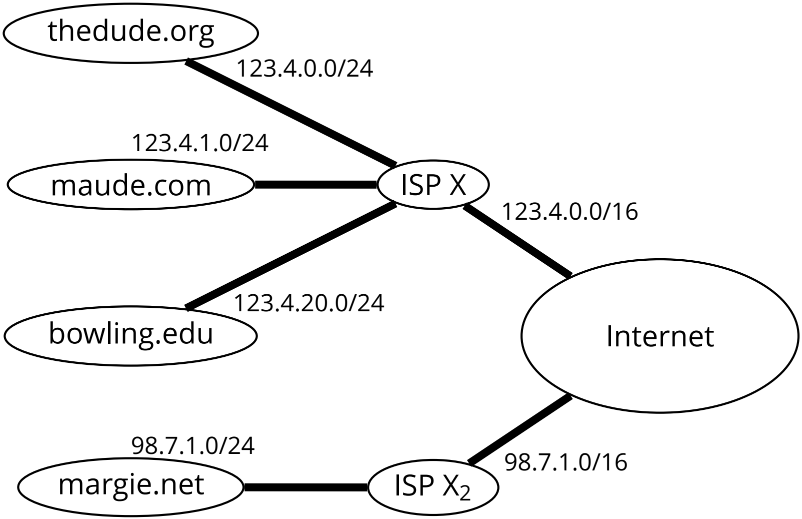


# Allocation of Address Blocks

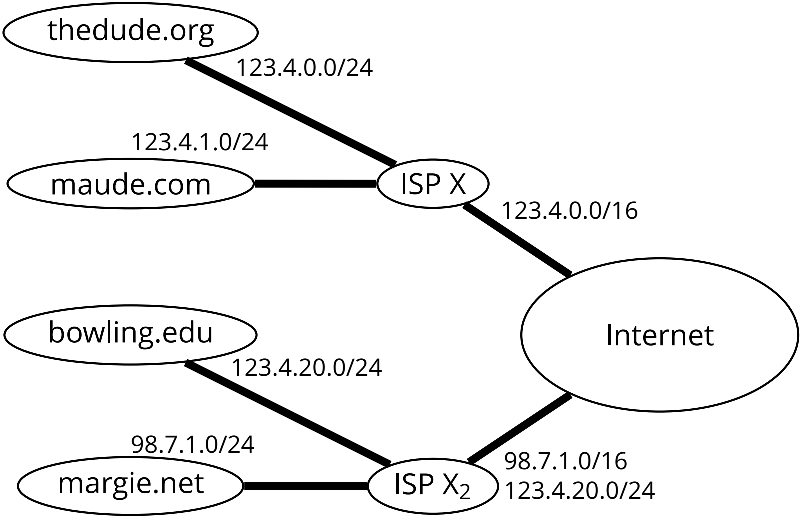
# Allocation of Address Blocks



# Allocation of Address Blocks



# Allocation of Address Blocks



# Longest-Prefix Matching

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
E.g.,

  - 123.4.1.69→1
  - 68.142.226.44→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,

  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,

  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,

  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,

  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→4

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→4
  - 123.4.20.11→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→4
  - 123.4.20.11→2

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→4
  - 123.4.20.11→2
  - 123.4.21.10→?

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix
  E.g.,
  - 123.4.1.69→1
  - 68.142.226.44→4
  - 98.7.2.71→2
  - 200.100.2.1→3
  - 128.138.207.167→4
  - 123.4.20.11→2
  - 123.4.21.10→1

| *forwarding table* | |
|---|---|
| *network* | *port* |
| 123.4.0.0/16 | 1 |
| 98.7.1.0/16 | 2 |
| 123.4.20.0/24 | 2 |
| 128.0.0.0/1 | 3 |
| 66.249.0.0/16 | 3 |
| 0.0.0.0/1 | 4 |
| 128.138.0.0/16 | 4 |

# Special Addresses

IPv4 defines a number of special addresses or address blocks

# Special Addresses

IPv4 defines a number of special addresses or address blocks

- “Private,” non-routable address blocks
  10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16

# Special Addresses

IPv4 defines a number of special addresses or address blocks

- “Private,” non-routable address blocks
  10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16
- Default route
  0.0.0.0/0

# Special Addresses

IPv4 defines a number of special addresses or address blocks

- “Private,” non-routable address blocks
  10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16
- Default route
  0.0.0.0/0
- Loopback (a.k.a., localhost)
  127.0.0.0/8

# Special Addresses

IPv4 defines a number of special addresses or address blocks

- "Private," non-routable address blocks
  10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16
- Default route
  0.0.0.0/0
- Loopback (a.k.a., localhost)
  127.0.0.0/8
- IP Multicast
  224.0.0.0/4

# Special Addresses

IPv4 defines a number of special addresses or address blocks

- “Private,” non-routable address blocks
  10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16
- Default route
  0.0.0.0/0
- Loopback (a.k.a., localhost)
  127.0.0.0/8
- IP Multicast
  224.0.0.0/4
- Broadcast
  255.255.255.255/32

# IPv6

- “New-generation IP”

# IPv6

- “New-generation IP”
- Why?

# IPv6

- “New-generation IP”
- Why?
  - the IPv4 address space is too small

# IPv6

- “New-generation IP”
- Why?
  - the IPv4 address space is too small
- Given the obvious difficulty of replacing IPv4, the short-term benefits of IPv6 are debatable

# IPv6

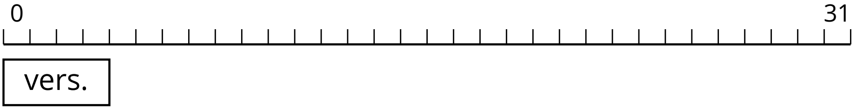
- “New-generation IP”
- Why?
  - the IPv4 address space is too small
- Given the obvious difficulty of replacing IPv4, the short-term benefits of IPv6 are debatable
- Nobody questions the long-term vision

# IPv6

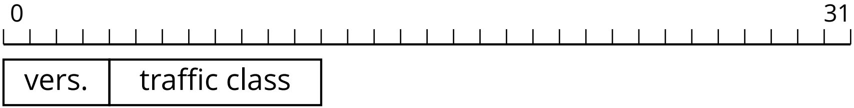
- “New-generation IP”
- Why?
  - the IPv4 address space is too small
- Given the obvious difficulty of replacing IPv4, the short-term benefits of IPv6 are debatable
- Nobody questions the long-term vision
- Also, IPv6 improves various design aspects of IPv4

# IPv6 Datagram Format

# IPv6 Datagram Format



# IPv6 Datagram Format



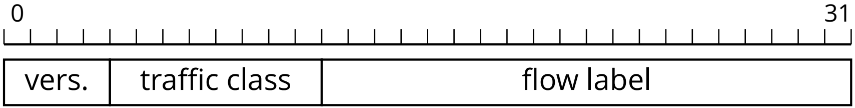
0

31

vers.

traffic class

# IPv6 Datagram Format



0

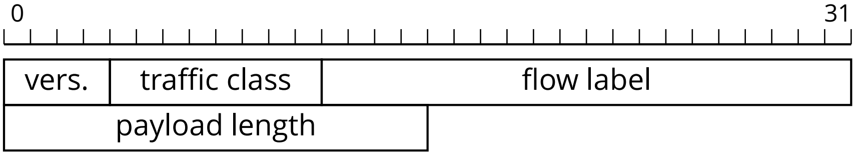
31

vers.

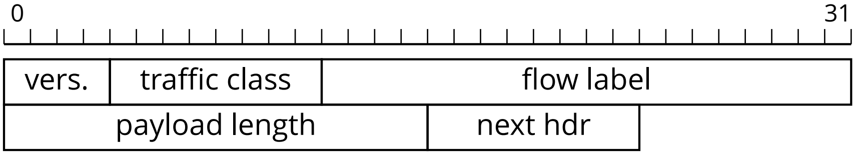
traffic class

flow label

# IPv6 Datagram Format



# IPv6 Datagram Format



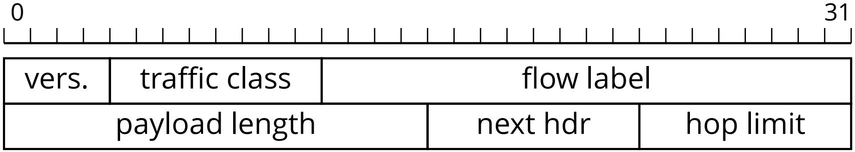
0

31

vers. | traffic class | flow label

payload length | next hdr

# IPv6 Datagram Format



0

31

vers.

traffic class

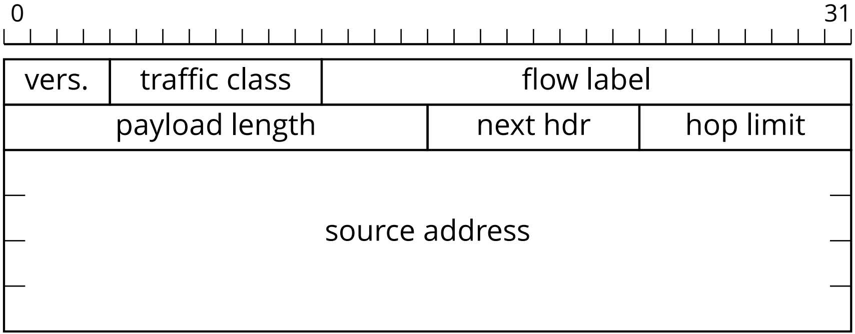
flow label

payload length

next hdr

hop limit

# IPv6 Datagram Format



0

31

vers.

traffic class

flow label

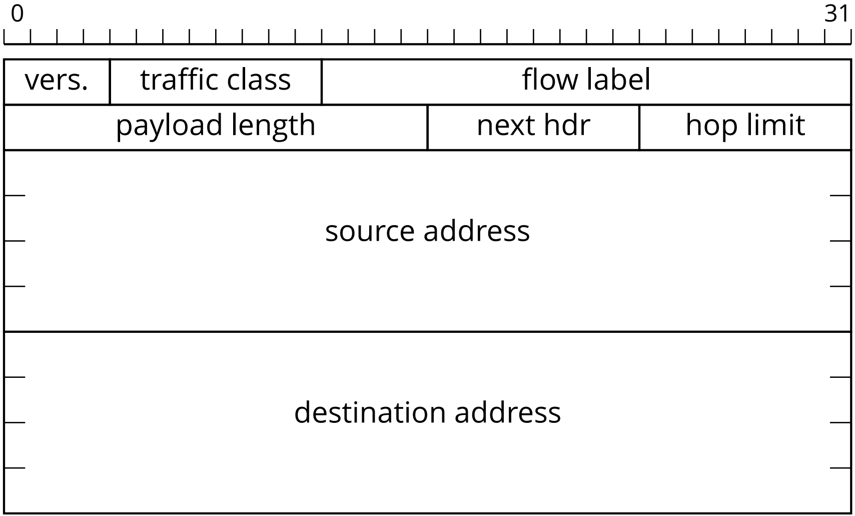
payload length

next hdr

hop limit

source address

# IPv6 Datagram Format



0

31

vers.

traffic class

flow label

payload length

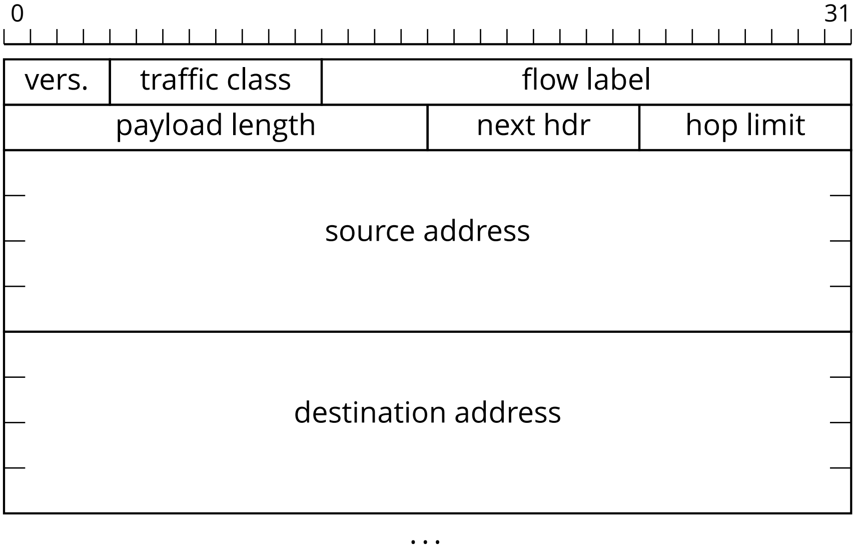
next hdr

hop limit

source address

destination address

# IPv6 Datagram Format



0

31

vers.

traffic class

flow label

payload length

next hdr

hop limit

source address

destination address

. . .

# IPv6 Main Design Features

- Expanded addressing

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address

- Header format simplification
  - efficiency: reducing the processing cost for the common case

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification
  - efficiency: reducing the processing cost for the common case
  - bandwidth: reducing overhead due to header bytes

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification
  - efficiency: reducing the processing cost for the common case
  - bandwidth: reducing overhead due to header bytes
- Improved support for extensions and options

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification
  - efficiency: reducing the processing cost for the common case
  - bandwidth: reducing overhead due to header bytes
- Improved support for extensions and options
- Flow labeling

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification
  - efficiency: reducing the processing cost for the common case
  - bandwidth: reducing overhead due to header bytes
- Improved support for extensions and options
- Flow labeling
  - special handling and non-default quality of service

# IPv6 Main Design Features

- Expanded addressing
  - 128-bit addresses
  - *anycast* address
- Header format simplification
  - efficiency: reducing the processing cost for the common case
  - bandwidth: reducing overhead due to header bytes
- Improved support for extensions and options
- Flow labeling
  - special handling and non-default quality of service
  - e.g., video, voice, real-time traffic, etc.

# What is Missing from IPv4?

# What is Missing from IPv4?

- Fragmentation

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency

- Header checksum

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency

- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency

- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?
    - the checksum must be recomputed at every hop, so IPv6 avoids that by getting rid of the checksum altogether

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency
- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?
    - the checksum must be recomputed at every hop, so IPv6 avoids that by getting rid of the checksum altogether
  - avoid redundancy: both link-layer protocols and transport protocols already provide error-detection features

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency
- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?
    - the checksum must be recomputed at every hop, so IPv6 avoids that by getting rid of the checksum altogether
  - avoid redundancy: both link-layer protocols and transport protocols already provide error-detection features
- Options

# What is Missing from IPv4?

- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency
- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?
    - the checksum must be recomputed at every hop, so IPv6 avoids that by getting rid of the checksum altogether
  - avoid redundancy: both link-layer protocols and transport protocols already provide error-detection features
- Options
  - efficiency: a fixed-length header is easier to process

# What is Missing from IPv4?

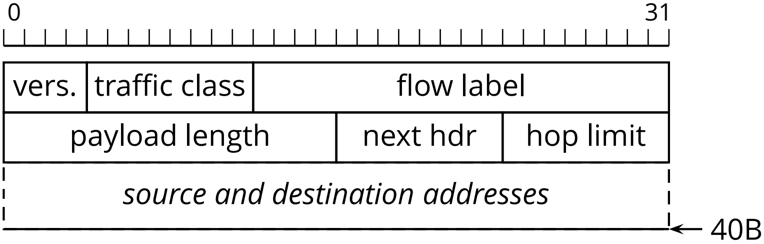
- Fragmentation
  - IPv6 pushes fragmentation onto the end-systems
  - efficiency

- Header checksum
  - efficiency
    - how does the checksum in IPv4 behave with respect to the time-to-live field?
    - the checksum must be recomputed at every hop, so IPv6 avoids that by getting rid of the checksum altogether
  - avoid redundancy: both link-layer protocols and transport protocols already provide error-detection features

- Options
  - efficiency: a fixed-length header is easier to process
  - better modularity for extensions and options

# Higher-Level Protocol and Extensions

# Higher-Level Protocol and Extensions



0 31

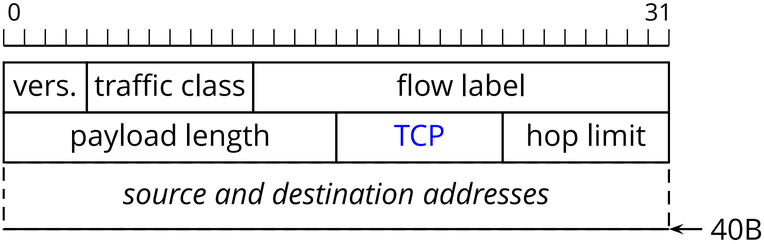
vers. | traffic class | flow label

payload length | next hdr | hop limit

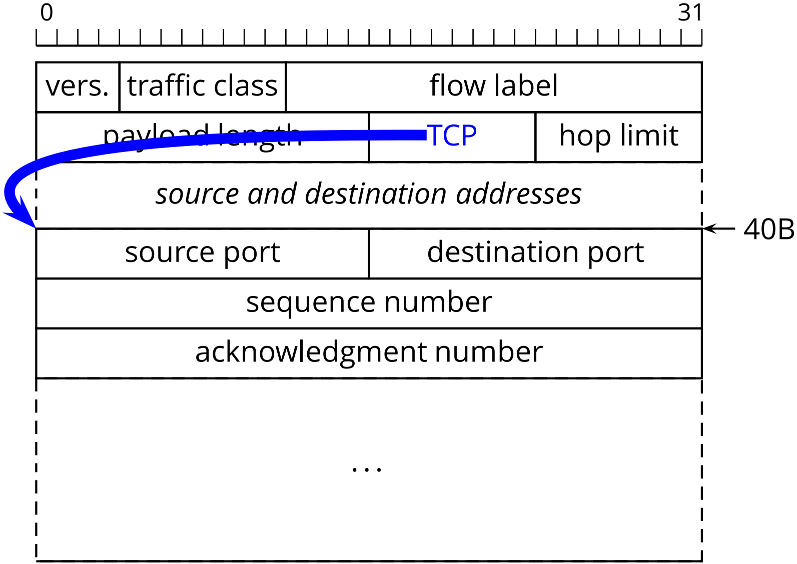
*source and destination addresses*

40B

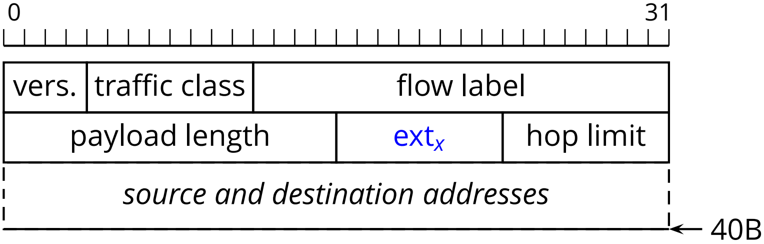
# Higher-Level Protocol and Extensions



# Higher-Level Protocol and Extensions



# Higher-Level Protocol and Extensions



0

31

vers.

traffic class

flow label

payload length

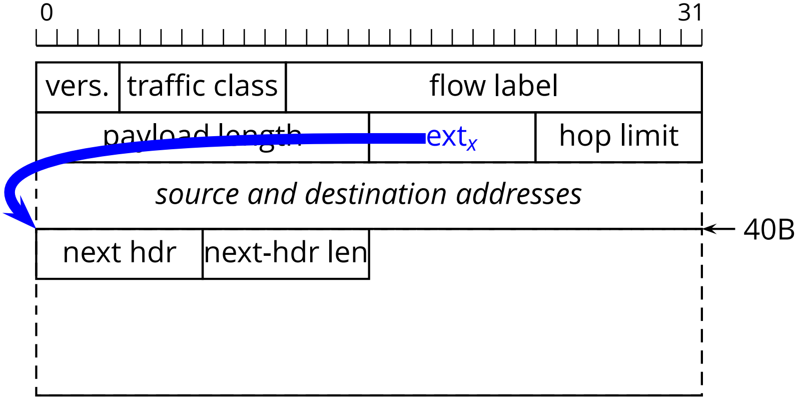
$\text{ext}_x$

hop limit

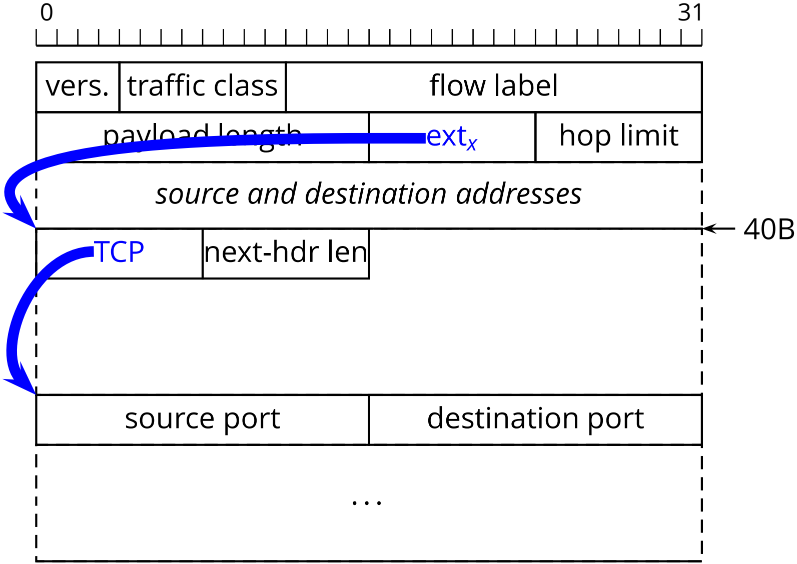
*source and destination addresses*

40B

# Higher-Level Protocol and Extensions



# Higher-Level Protocol and Extensions


0
31
vers.
traffic class
flow label
payload length
ext$_x$
hop limit
*source and destination addresses*
40B
TCP
next-hdr len
source port
destination port
. . .