Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   - datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
5 routing algorithms
   - link state
   - distance vector
   - hierarchical routing
6 routing in the Internet
   - RIP
   - OSPF
   - BGP
7 broadcast and multicast routing
Figure  Internet as a block box
Figure 3: Internet as a combination of LANs and WANs connected together.
From the previous discussion, it is clear that the passage of a message from a source to a destination involves many decisions. When a message reaches a connecting device, a decision needs to be made to select one of the output ports through which the packet needs to be send out. In other words, the connecting device acts as a switch that connects one port to another port.
Circuit Switching

- A physical circuit (channel) is established between the source and destination before the delivery of the message.
- Never implemented at the network layer; it is mostly used at the physical layer.

In circuit switching, the whole message is sent from the source to the destination without being divided into packets.
Packet Switching

- Datagram approach
- Virtual circuit approach

In packet switching, the message is first divided into manageable packets at the source before being transmitted. The packets are assembled at the destination.
Figure A connectionless packet-switched network
Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in every host, router
- router examines header fields in all IP datagrams passing through it
Two key network-layer functions

- **forwarding**: move packets from router’s input to appropriate router output
- **routing**: determine route taken by packets from source to dest.
  - **routing algorithms**

**analogy:**

- **routing**: process of planning trip from source to dest
- **forwarding**: process of getting through single interchange
Interplay between routing and forwarding

Routing algorithm determines end-end-path through network.
Forwarding table determines local forwarding at this router.

Value in arriving packet’s header:

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

Routing algorithm:

1. Interplay between routing and forwarding
2. Routing algorithm determines end-end-path through network.
3. Forwarding table determines local forwarding at this router.

Value in arriving packet’s header:

1. Interplay between routing and forwarding
2. Routing algorithm determines end-end-path through network.
3. Forwarding table determines local forwarding at this router.

Value in arriving packet’s header:

1. Interplay between routing and forwarding
2. Routing algorithm determines end-end-path through network.
3. Forwarding table determines local forwarding at this router.

Value in arriving packet’s header:
Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   - datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
5 routing algorithms
   - link state
   - distance vector
   - hierarchical routing
6 routing in the Internet
   - RIP
   - OSPF
   - BGP
7 broadcast and multicast routing
Connection, connection-less service

- **datagram** network provides network-layer *connectionless* service

- **virtual-circuit** network provides network-layer *connection* service

- analogous to TCP/UDP connection-oriented / connectionless transport-layer services
A connectionless packet-switched network

Figure
In a connectionless packet-switched network, the forwarding decision is based on the destination address of the packet.
Figure  Delay in a connectionless network

Time

Source

Transmission time

Waiting time

Time

Waiting time

Time

Waiting time

Time

Total delay

Destination
A connection-oriented packet switched network

Legend

- 4 3 2 1: Packets
- Virtual circuit

Network

Sender

R1 - R2 - R5 - R3 - R4 - Receiver

A connection-oriented packet-switched network
In a connection-oriented packet switched network, the forwarding decision is based on the VC number of the packet.
Figure Sending request packet in a virtual-circuit network

Network

Legend

A to B Request packet
Virtual circuit

A to B

<table>
<thead>
<tr>
<th>Incoming</th>
<th>Outgoing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Label</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Request packet

A to B

<table>
<thead>
<tr>
<th>Incoming</th>
<th>Outgoing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Label</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
</tr>
</tbody>
</table>

A to B

<table>
<thead>
<tr>
<th>Incoming</th>
<th>Outgoing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Label</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

A to B

Label Port

Port Port

Outgoing Incoming

A to B

Label

Outgoing Incoming

A to B

Label

Outgoing Incoming

A to B
Figure Setup acknowledgement in a virtual-circuit network
Flow of one packet in an established virtual circuit

**Legend**
- Datagram
- Virtual circuit

**Table 1**

<table>
<thead>
<tr>
<th>Port</th>
<th>Label</th>
<th>Port</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>3</td>
<td>66</td>
</tr>
</tbody>
</table>

**Table 2**

<table>
<thead>
<tr>
<th>Port</th>
<th>Label</th>
<th>Port</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66</td>
<td>3</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Label</th>
<th>Port</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22</td>
<td>4</td>
<td>77</td>
</tr>
</tbody>
</table>
**Figure**  
*Delay in a connection-oriented network*
Virtual circuits

“source-to-dest path behaves much like telephone circuit”
- performance-wise
- network actions along source-to-dest path

- call setup, teardown for each call before data can flow
- each packet carries VC identifier (not destination host address)
- every router on source-dest path maintains “state” for each passing connection
- link, router resources (bandwidth, buffers) may be allocated to VC (dedicated resources = predictable service)
VC implementation

a VC consists of:

1. *path* from source to destination
2. *VC numbers*, one number for each link along path
3. *entries in forwarding tables* in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - new VC number comes from forwarding table
VC forwarding table

Forwarding table in northwest router:

<table>
<thead>
<tr>
<th>Incoming interface</th>
<th>Incoming VC #</th>
<th>Outgoing interface</th>
<th>Outgoing VC #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

VC routers maintain connection state information!
Virtual circuits: signaling protocols

- used to setup, maintain, teardown VC
- used in ATM, frame-relay, X.25
- not used in today’s Internet

1. initiate call
2. incoming call
3. accept call
4. call connected
5. data flow begins
6. receive data
Datagram networks: Connectionless

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of “connection”
- packets forwarded using destination host address
Datagram forwarding table

<table>
<thead>
<tr>
<th>dest address</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>address-range 1</td>
<td>3</td>
</tr>
<tr>
<td>address-range 2</td>
<td>2</td>
</tr>
<tr>
<td>address-range 3</td>
<td>2</td>
</tr>
<tr>
<td>address-range 4</td>
<td>1</td>
</tr>
</tbody>
</table>

Routing algorithm

4 billion IP addresses, so rather than list individual destination address list range of addresses (aggregate table entries)

IP destination address in arriving packet’s header
### Datagram forwarding table

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

Q: but what happens if ranges don’t divide up so nicely?
**Longest prefix matching**

*longest prefix matching*

When looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010*** *******</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 *******</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011*** *******</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

Examples:

DA: 11001000 00010111 00010110 10100001  
*which interface?*

DA: 11001000 00010111 00011000 10101010  
*which interface?*
Datagram or VC network: why?

**Internet (datagram)**
- data exchange among computers
  - “elastic” service, no strict timing req.
- many link types
  - different characteristics
  - uniform service difficult
- “smart” end systems (computers)
  - can adapt, perform control, error recovery
  - *simple inside network, complexity at “edge”*

**ATM (VC)**
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- “dumb” end systems
  - telephones
  - *complexity inside network*
Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   - datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
5 routing algorithms
   - link state
   - distance vector
   - hierarchical routing
6 routing in the Internet
   - RIP
   - OSPF
   - BGP
7 broadcast and multicast routing
Router architecture overview

two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link
Input port functions

- **Physical layer:** bit-level reception
- **Data link layer:** e.g., Ethernet

**Decentralized switching:**
- Given datagram dest., lookup output port using forwarding table in input port memory ("match plus action")
- Goal: complete input port processing at 'line speed'
- Queuing: if datagrams arrive faster than forwarding rate into switch fabric
Switching fabrics

- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics

![Diagram of memory, bus, and crossbar switching fabrics](image)
Switching via memory

*first generation routers:*

- traditional computers with switching under direct control of CPU
- packet copied to system’s memory
- speed limited by memory bandwidth (2 bus crossings per datagram)
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- **bus contention**: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers
Switching via interconnection network

- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network
Output ports

- **buffering** required when datagrams arrive from fabric faster than the transmission rate.
- **Scheduling** discipline chooses among queued datagrams for transmission.

Datagram (packets) can be lost due to congestion, lack of buffers.

Output port queueing

- Buffering when arrival rate via switch exceeds output line speed
- Queueing (delay) and loss due to output port buffer overflow!
How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gbps link: 2.5 Gbit buffer

- recent recommendation: with N flows, buffering equal to

\[
\sqrt[4]{N} \times \frac{RTT \cdot C}{N}
\]
Input port queuing

- fabric slower than input ports combined -> queueing may occur at input queues
  - queueing delay and loss due to input buffer overflow!
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

output port contention: only one red datagram can be transferred. lower red packet is blocked

one packet time later: green packet experiences HOL blocking
Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   - datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
5 routing algorithms
   - link state
   - distance vector
   - hierarchical routing
6 routing in the Internet
   - RIP
   - OSPF
   - BGP
7 broadcast and multicast routing
The Internet network layer

host, router network layer functions:

- **Routing protocols**
  - path selection
  - RIP, OSPF, BGP

- **IP protocol**
  - addressing conventions
  - datagram format
  - packet handling conventions

- **ICMP protocol**
  - error reporting
  - router "signaling"

transport layer: TCP, UDP
Figure 7.2  **IP datagram**

![Diagram of IP datagram](image)

**a. IP datagram**

**b. Header format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VER</td>
<td>4</td>
</tr>
<tr>
<td>HLEN</td>
<td>4</td>
</tr>
<tr>
<td>Service type</td>
<td>8</td>
</tr>
<tr>
<td>Total length</td>
<td>16</td>
</tr>
<tr>
<td>Identification</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>3</td>
</tr>
<tr>
<td>Fragmentation offset</td>
<td>13</td>
</tr>
<tr>
<td>Time to live</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>8</td>
</tr>
<tr>
<td>Header checksum</td>
<td>16</td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
</tr>
<tr>
<td>Options + padding</td>
<td>(0 to 40 bytes)</td>
</tr>
</tbody>
</table>
**IP datagram format**

- **IP protocol version number**
- **header length** (in 4-byte words)
- **“type” of data**
- **max number remaining hops** (decremented at each router)
- **upper layer protocol** to deliver payload to

How much overhead?
- 20 - 60 bytes of IP

Length of data?
- Total length – header length

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>version (ver)</td>
<td>4 bits</td>
</tr>
<tr>
<td>header length (len)</td>
<td>4 bits</td>
</tr>
<tr>
<td>type of service</td>
<td>8 bits</td>
</tr>
<tr>
<td>fragment offset</td>
<td>16 bits</td>
</tr>
<tr>
<td>fragment length</td>
<td>16 bits</td>
</tr>
<tr>
<td>time to live (TTL)</td>
<td>8 bits</td>
</tr>
<tr>
<td>upper layer</td>
<td>12 bits</td>
</tr>
<tr>
<td>source IP</td>
<td>32 bits</td>
</tr>
<tr>
<td>destination IP</td>
<td>32 bits</td>
</tr>
<tr>
<td>options</td>
<td>variable length</td>
</tr>
<tr>
<td>data</td>
<td>variable length</td>
</tr>
</tbody>
</table>

Total datagram length (bytes) for fragmentation/reassembly

0-40 bytes e.g. timestamp, record route taken, specify list of routers to visit.
Service Type

Values for codepoints

<table>
<thead>
<tr>
<th>Category</th>
<th>Codepoint</th>
<th>Assigning Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XXXXX0</td>
<td>Internet</td>
</tr>
<tr>
<td>2</td>
<td>XXXX11</td>
<td>Local</td>
</tr>
<tr>
<td>3</td>
<td>XXXX01</td>
<td>Temporary or experimental</td>
</tr>
</tbody>
</table>
• Time to live: control the maximum number of hops (routers) visited
  • If the value is zero, the router discards the datagram

• Protocol: the higher-level protocol that uses the services of the IP layer

<table>
<thead>
<tr>
<th>Value</th>
<th>Protocol</th>
<th>Value</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ICMP</td>
<td>17</td>
<td>UDP</td>
</tr>
<tr>
<td>2</td>
<td>IGMP</td>
<td>89</td>
<td>OSPF</td>
</tr>
<tr>
<td>6</td>
<td>TCP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 1

An IP packet has arrived with the first 8 bits as shown:

01000010

The receiver discards the packet. Why?

Solution

There is an error in this packet. The 4 left-most bits (0100) show the version, which is correct. The next 4 bits (0010) show the wrong header length (2 \(\times\) 4 = 8). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.
Example 2

In an IP packet, the value of HLEN (header length) is 1000 in binary. How many bytes of options are being carried by this packet?

**Solution**

The HLEN value is 8, which means the total number of bytes in the header is $8 \times 4$ or 32 bytes. The first 20 bytes are the base header, the next 12 bytes are the options.
Example 3

In an IP packet, the value of HLEN is $5_{16}$ and the value of the total length field is $0028_{16}$. How many bytes of data are being carried by this packet?

**Solution**

The HLEN value is 5, which means the total number of bytes in the header is $5 \times 4$ or 20 bytes (no options). The total length is 40 bytes, which means the packet is carrying 20 bytes of data ($40 - 20$).
**Example 4**

An IP packet has arrived with the first few hexadecimal digits as shown below:

```
45000028000100000102...
```

How many hops can this packet travel before being dropped? The data belong to what upper layer protocol?

**Solution**

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP.
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
**IP datagram format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version</td>
<td>Protocol number</td>
</tr>
<tr>
<td>header length</td>
<td>Length of header in 4-byte words</td>
</tr>
<tr>
<td>16-bit identifier</td>
<td>Max number of remaining hops (decremented at each router)</td>
</tr>
<tr>
<td>flag</td>
<td>For fragmentation/reassembly</td>
</tr>
<tr>
<td>offset</td>
<td>0-40 bytes of IP</td>
</tr>
<tr>
<td>checksum</td>
<td>For header check sum</td>
</tr>
<tr>
<td>source IP address</td>
<td>32 bit address</td>
</tr>
<tr>
<td>destination IP address</td>
<td>32 bit address</td>
</tr>
<tr>
<td>options (if any)</td>
<td>Additional fields</td>
</tr>
<tr>
<td>data</td>
<td>Variable length, typically a TCP or UDP segment</td>
</tr>
</tbody>
</table>

**How much overhead?**
- 20 - 60 bytes of IP

**Length of data?**
- Total length – header length

**Example fields:**
- Time to live
- Upper layer protocol
- Options (if any)
- Data
**Flags field**

**D**: Do not fragment

- D = 1 cannot fragment the datagram
- D = 0 can be fragmented if necessary

**M**: More fragments

- M = 1 the datagram is not the last fragment
- M = 0 this is the last or only fragment
example:

- 4000 byte datagram
- MTU = 1400 bytes
Detailed fragmentation example

Bytes 0000–3999

Original datagram

Bytes 0000–1399

Fragment 1

Bytes 0000–1399

Fragment 1

Bytes 1400–2199

Fragment 2.1

Bytes 1400–2799

Fragment 2

Bytes 2800–3999

Fragment 3

Bytes 2200–2799

Fragment 2.2
A packet has arrived with an M bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

**Solution**

If the M bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.
Example 6

A packet has arrived with an M bit value of 1. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

Solution

If the M bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don’t know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.
Example 7

A packet has arrived with an M bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

**Solution**

Because the M bit is 1, it is either the first fragment or a middle one. Because the offset value is 0, it is the first fragment.
A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

**Solution**

To find the number of the first byte, we multiply the offset value by 8. This means that the first byte number is 800. We cannot determine the number of the last byte unless we know the length of the data.
Example 9

A packet has arrived in which the offset value is 100, the value of HLEN (header length) is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

Solution

The first byte number is $100 \times 8 = 800$. The total length is 100 bytes and the header length is 20 bytes ($5 \times 4$), which means that there are 80 bytes in this datagram. If the first byte number is 800, the last byte number must be 879.
Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   ▪ datagram format
   ▪ IPv4 addressing
   ▪ ICMP
   ▪ IPv6
5 routing algorithms
   ▪ link state
   ▪ distance vector
   ▪ hierarchical routing
6 routing in the Internet
   ▪ RIP
   ▪ OSPF
   ▪ BGP
7 broadcast and multicast routing
IP addressing: introduction

- **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 or two interfaces (e.g., wired Ethernet, wireless 802.11)
- **IP addresses associated with each interface**
  
  - 223.1.1.1 = 11011111 00000001 00000001 00000001
  
  | 1 | 1 | 1 | 1 | 1 |
Example 1

Find the error, if any, in the following IPv4 addresses:

a. 111.56.045.78
b. 221.34.7.8.20
c. 75.45.301.14
d. 11100010.23.14.67

Solution

a. There should be no leading zeroes (045).
b. We may not have more than 4 bytes in an IPv4 address.
c. Each byte should be less than or equal to 255.
d. A mixture of binary notation and dotted-decimal notation.
Example 2

Find the number of addresses in a range if the first address is 146.102.29.0 and the last address is 146.102.32.255.

Solution

We can subtract the first address from the last address in base 256 (see Appendix B). The result is 0.0.3.255 in this base. To find the number of addresses in the range (in decimal), we convert this number to base 10 and add 1 to the result.

Number of addresses = \((0 \times 256^3 + 0 \times 256^2 + 3 \times 256^1 + 255 \times 256^0) + 1 = 1024\)
Example 3

The first address in a range of addresses is 14.11.45.96. If the number of addresses in the range is 32, what is the last address?

Solution

We convert the number of addresses minus 1 to base 256, which is 0.0.0.31. We then add it to the first address to get the last address. Addition is in base 256.

\[
\text{Last address} = (14.11.45.96 + 0.0.0.31)_{256} = 14.11.45.127
\]
Figure  Bitwise NOT operation

- **Input**: 32 bits
- **Output**: 32 bits

**NOT operation**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Operation for each bit
Example 4

The following shows how we can apply the NOT operation on a 32-bit number in binary.

<table>
<thead>
<tr>
<th>Original number:</th>
<th>00010001</th>
<th>01111001</th>
<th>00001110</th>
<th>00100011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement:</td>
<td>11101110</td>
<td>10000110</td>
<td>11110001</td>
<td>11011100</td>
</tr>
</tbody>
</table>

We can use the same operation using the dotted-decimal representation and the short cut.

<table>
<thead>
<tr>
<th>Original number:</th>
<th>17</th>
<th>121</th>
<th>14</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement:</td>
<td>238</td>
<td>134</td>
<td>241</td>
<td>220</td>
</tr>
</tbody>
</table>
Figure  
**Bitwise AND operation**

- **Input 1** 32 bits
- **Input 2** 32 bits
- **Output** 32 bits

**AND**

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation for each bit
Example 5

<table>
<thead>
<tr>
<th>First number:</th>
<th>00010001</th>
<th>01111001</th>
<th>00001110</th>
<th>00100011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second number:</td>
<td>11111111</td>
<td>11111111</td>
<td>10001100</td>
<td>00000000</td>
</tr>
<tr>
<td>Result</td>
<td>00010001</td>
<td>01111001</td>
<td>00001110</td>
<td>00000000</td>
</tr>
</tbody>
</table>

We can use the same operation using the dotted-decimal representation and the short cut.

<table>
<thead>
<tr>
<th>First number:</th>
<th>17</th>
<th>.</th>
<th>121</th>
<th>.</th>
<th>14</th>
<th>.</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second number:</td>
<td>255</td>
<td>.</td>
<td>255</td>
<td>.</td>
<td>140</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>Result:</td>
<td>17</td>
<td>.</td>
<td>121</td>
<td>.</td>
<td>12</td>
<td>.</td>
<td>0</td>
</tr>
</tbody>
</table>

We have applied the first short cut on the first, second, and the fourth byte; we have applied the second short cut on the third byte. We have written 14 and 140 as the sum of terms and selected the smaller term in each pair as shown below.

<table>
<thead>
<tr>
<th>Powers</th>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte (14)</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>8</td>
<td>+</td>
</tr>
<tr>
<td>Byte (140)</td>
<td>128</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>8</td>
<td>+</td>
</tr>
<tr>
<td>Result (12)</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>8</td>
<td>+</td>
</tr>
</tbody>
</table>
Figure: Bitwise OR operation

- Input 1: 32 bits
- Input 2: 32 bits
- Output: 32 bits

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation for each bit
Example 6

The following shows how we can apply the OR operation on two 32-bit numbers in binary.

<table>
<thead>
<tr>
<th>First number:</th>
<th>00010001</th>
<th>01111001</th>
<th>00001110</th>
<th>00100011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second number:</td>
<td>11111111</td>
<td>11111111</td>
<td>10001100</td>
<td>00000000</td>
</tr>
<tr>
<td>Result</td>
<td>11111111</td>
<td>11111111</td>
<td>10001110</td>
<td>00100011</td>
</tr>
</tbody>
</table>

We can use the same operation using the dotted-decimal representation and the short cut.

<table>
<thead>
<tr>
<th>First number:</th>
<th>17</th>
<th>.</th>
<th>121</th>
<th>.</th>
<th>14</th>
<th>.</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second number:</td>
<td>255</td>
<td>.</td>
<td>255</td>
<td>.</td>
<td>140</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>Result:</td>
<td>255</td>
<td>.</td>
<td>255</td>
<td>.</td>
<td>142</td>
<td>.</td>
<td>35</td>
</tr>
</tbody>
</table>

We have used the first short cut for the first and second bytes and the second short cut for the third byte.

<table>
<thead>
<tr>
<th>Powers</th>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte (14)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Byte (140)</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Result (12)</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
When IP addressing was first started, it used a concept called “classful addressing”. A newer concept called “classless addressing” is slowly replacing it though.

Regarding “classful addressing”, the address space is divided into five classes: A, B, C, D and E.

<table>
<thead>
<tr>
<th>Class</th>
<th># of addresses</th>
<th>Percent of the Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>$2^{31} = 2,147,483,648$ addresses, 50%</td>
<td></td>
</tr>
<tr>
<td>Class B</td>
<td>$2^{30} = 1,073,741,824$ addresses, 25%</td>
<td></td>
</tr>
<tr>
<td>Class C</td>
<td>$2^{29} = 536,870,912$ addresses, 12.5%</td>
<td></td>
</tr>
<tr>
<td>Class D</td>
<td>$2^{28} = 268,435,456$ addresses, 6.25%</td>
<td></td>
</tr>
<tr>
<td>Class E</td>
<td>$2^{28} = 268,435,456$ addresses, 6.25%</td>
<td></td>
</tr>
</tbody>
</table>
Figure Finding the class of address

Legend
- Check next bit
- Address class

Start → 1 → 1 → 1 → 1

Class: A
Class: B
Class: C
Class: D
Class: E

Octet 1 Octet 2 Octet 3 Octet 4
Class A 0........
Class B 10.....
Class C 110.....
Class D 1110....
Class E 1111....

Binary notation

Byte 1 Byte 2 Byte 3 Byte 4
Class A 0–127
Class B 128–191
Class C 192–223
Class D 224–239
Class E 240–255

Dotted-decimal notation
Example 7

Find the class of each address:

a. 00000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111
c. 10100111 11011011 10001011 01101111
d. 11110011 10011011 11111011 00001111

Solution

a. The first bit is 0. This is a class A address.
b. The first 2 bits are 1; the third bit is 0. This is a class C address.
c. The first bit is 1; the second bit is 0. This is a class B address.
d. The first 4 bits are 1s. This is a class E address.
Example 8

Find the class of each address:

a. 227.12.14.87
b. 193.14.56.22
c. 14.23.120.8
d. 252.5.15.111

Solution

a. The first byte is 227 (between 224 and 239); the class is D.
b. The first byte is 193 (between 192 and 223); the class is C.
c. The first byte is 14 (between 0 and 127); the class is A.
d. The first byte is 252 (between 240 and 255); the class is E.
Figure Netid and hostid

Byte 1 | Byte 2 | Byte 3 | Byte 4

Class A: Netid | Hostid

Class B: Netid | Hostid

Class C: Netid | Hostid

Class D: Multicast address

Class E: Reserved for future use
• Class A has 128 blocks or network ids
  • First byte is the same (netid), the remaining 3 bytes can change (hostids)
  • Network id 0 (first), Net id 127 (last) and Net id 10 are reserved – leaving 125 ids to be assigned to organizations/companies
  • Each block contains 16,777,216 addresses – this block should be used by large organizations.
  • The first address in the block is called the “network address” – defines the network of the organization

Example
• Netid 73 is assigned
• Last address is reserved
• Recall: routers have addressees
• Class B is divided into 16,384 blocks (65,536 addresses each)
• 16 blocks are reserved
• First 2 bytes are the same (netid), the remaining 2 bytes can change (hostids)
• For example, Network id 128.0 covers addresses 128.0.0.0 to 128.0.255.255
• Network id 191.225 is the last netid for this block

Example
• Netid 180.8 is assigned
• Last address is reserved
• Recall: routers have addresses
• Class C is divided into 2,097,152 blocks (256 addresses each)
• 256 blocks are reserved
• First 3 bytes are the same (netid), the remaining 1 byte can change (hostids)
• For example, Network id 192.0.0 covers addresses 192.0.0.0 to 192.0.0.255
Class D addresses are made of one block, used for multicasting.
The only block of class E addresses was reserved for future purposes.
The range of addresses allocated to an organization in classful addressing was a block of addresses in Class A, B, or C.
Figure  Two-level addressing in classful addressing

Class A: $n = 8$
Class B: $n = 16$
Class C: $n = 24$
The network address is the first address.

The network address defines the network to the rest of the Internet.

Given the network address, we can find the class of the address, the block, and the range of the addresses in the block.
Example 9

An address in a block is given as 73.22.17.25. Find the number of addresses in the block, the first address, and the last address.

Solution

1. The number of addresses in this block is \( N = 2^{32-n} = 2^{24} \times 16,777,216. \)
2. To find the first address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 0s. The first address is 73.0.0.0/8, in which 8 is the value of \( n. \)
3. To find the last address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 1s. The last address is 73.255.255.255/8.
**Netid 73:** common in all addresses

Network address: **73.0.0.0/8**

73.0.0.1, 73.0.0.8, 73.22.17.25, 73.255.255.254

Block

73.0.0.0

⋯

73.255.255.255

73.255.255.255 (Special)
Example 10

An address in a block is given as 180.8.17.9. Find the number of addresses in the block, the first address, and the last address.

Solution

1. The number of addresses in this block is \( N = 2^{32-n} = 2^{16} = 65,536 \).

2. To find the first address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 0s. The first address is 18.8.0.0/16, in which 16 is the value of \( n \).

3. To find the last address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 1s. The last address is 18.8.255.255/16.
Netid 180.8: common in all addresses

180.8.0.1  180.8.0.9  180.8.17.9  180.8.255.254

Switch

180.8.0.0/16 Network address  180.8.255.255 (Special)
An address in a block is given as 200.11.8.45. Find the number of addresses in the block, the first address, and the last address.

**Solution**

1. The number of addresses in this block is \( N = 2^{32-n} = 2^8 = 256. \)
2. To find the first address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 0s. The first address is 200.11.8.0/24, in which 24 is the value of \( n. \)
3. To find the last address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 1s. The last address is 200.11.8.255/24.
**Figure**  Solution to Example 11

Netid **200.11.8:** common in all addresses

Network address: **200.11.8.0/24**

200.11.8.1  
200.11.8.7  
200.11.8.45  
200.11.8.254

Block

200.11.8.0  
***  
200.11.8.255

200.11.8.255 (Special)
Figure  Sample Internet

LAN: **220.3.6.0/24**

- 220.3.6.1
- 220.3.6.12
- 220.3.6.23
- 220.3.6.26

LAN: **134.18.0.0/16**

- 134.18.10.88
- 134.18.12.32
- 134.18.14.121

Switched WAN **200.78.6.0/24**

**Rest of the Internet**

- **200.78.6.14**
- **200.78.6.146**
- **200.78.6.92**

**R1**

- 200.78.6.14

**R2**

- 200.78.6.92

**R3**

- 200.78.6.146
**Figure 93** Network addresses

![Diagram of network addresses]

**Routing Process**

1. **Destination address**
2. Find **Network address**
3. Look up in **Routing Table**

**Routing Table**

<table>
<thead>
<tr>
<th>Network address</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_1 \cdot c_1 \cdot d_1 \cdot e_1 )</td>
<td>1</td>
</tr>
<tr>
<td>( b_2 \cdot c_2 \cdot d_2 \cdot e_2 )</td>
<td>2</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( b_m \cdot c_m \cdot d_m \cdot e_m )</td>
<td>( m )</td>
</tr>
</tbody>
</table>

**Interface number**

(source: Network addresses)
Given the network address, we can easily determine the block and range of addresses.

Suppose given the IP address, can we determine the network address (beginning of the block)?

To route packets to the correct network, a router must extract the network address from the destination IP address.

How would we extract the network address from the IP address? We would use a mask.

A mask is a 32-bit binary number that gives the first address in the block (the network address) when bitwise ANDed with an address in the block.
Figure Network mask

Mask for class A

<table>
<thead>
<tr>
<th>11111111</th>
<th>00000000</th>
<th>00000000</th>
<th>00000000</th>
</tr>
</thead>
</table>

255.0.0.0

Mask for class B

<table>
<thead>
<tr>
<th>11111111</th>
<th>11111111</th>
<th>00000000</th>
<th>00000000</th>
</tr>
</thead>
</table>

255.255.0.0

Mask for class C

<table>
<thead>
<tr>
<th>11111111</th>
<th>11111111</th>
<th>11111111</th>
<th>00000000</th>
</tr>
</thead>
</table>

255.255.255.0
• If bit is ANDed with 1, it’s preserved
• If bit is ANDed with 0, it’s changed to a 0.

A simple way to determine the netid for un-subnetted cases: (1) if mask byte is 255, retain corresponding byte of the address, (2) if mask byte is 0, set corresponding address byte to 0.
A router receives a packet with the destination address 181.24.67.32. Show how the router finds the network address of the packet.

**Solution**

Since the class of the address is B, we assume that the router applies the default mask for class B, 255.255.0.0 to find the network address.

```
Destination address  →  181
   Default mask     →  255
   ➔ Network address ➔  181
```
### Recall IP Addresses: Classful Addressing

<table>
<thead>
<tr>
<th>Class</th>
<th># of addresses</th>
<th>Percent of the Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$2^{31} = 2147483648$</td>
<td>50%</td>
</tr>
<tr>
<td>B</td>
<td>$2^{30} = 1073741824$</td>
<td>25%</td>
</tr>
<tr>
<td>C</td>
<td>$2^{29} = 536870912$</td>
<td>12.5%</td>
</tr>
<tr>
<td>D</td>
<td>$2^{28} = 268435456$</td>
<td>6.25%</td>
</tr>
<tr>
<td>E</td>
<td>$2^{28} = 268435456$</td>
<td>6.25%</td>
</tr>
</tbody>
</table>

#### Byte 1 | Byte 2 | Byte 3 | Byte 4
---|---|---|---
Class A | **Netid** | | **Hostid**
Class B | **Netid** | | **Hostid**
Class C | | **Netid** | **Hostid**
Class D | | | **Multicast address**
Class E | | | **Reserved for future use**
No Netid case

32 addresses/block

Number of blocks: 1

Address range per block: 0 to 31

Netids: N/A

Network Addresses: 00000

Broadcast Addresses: 11111
1-bit Netid case

16 addresses/block

Number of blocks: 2

Address range per block: 0 to 15

Netids: 0, 1

Network Addresses: 00000, 10000

Broadcast Addresses: 01111, 11111
### 5-bit Address Space Illustration

#### 2-bit Netid Case

8 addresses/block

Number of blocks: 4

Address range per block: 0 to 7

Netids: 00, 01, 10, 11

Network Addresses: 00000, 01000, 10000, 11000

Broadcast Addresses: 00111, 01111, 10111, 11111

<table>
<thead>
<tr>
<th>Netid</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>01</td>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>10</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>11</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>1 1 0 0</td>
<td>1 1 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>01000</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>10000</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>11000</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>00111</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>01111</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>10111</td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>11111</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
</tr>
</tbody>
</table>
### 5-bit Address Space Illustration

#### 3-bit Netid Case

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### 3-bit Netid Case

- **4 addresses/block**
- **Number of blocks: 8**
- **Address range per block: 0 to 3**
- **Netids: 000, 001, 010, 011, 100, 101, 110, 111**
- **Network Addresses**: 00000, 00100, 01000, 01100, 10000, 10100, 11000, 11100
- **Broadcast Addresses**: 00011, 00111, 01011, 01111, 10011, 10111, 11011, 11111
Mixing 3-bit & 2-bit Cases

4 addresses/block and 8 addresses/block

Number of blocks: 6

Address range per block: 0 to 3 and 0 to 7

Netids: 000, 001, 010, 011, 10, 11

Network Addresses: 00000, 00100, 01000, 01100, 10000, 11000

Broadcast Addresses: 00011, 00111, 01011, 01111, 10111, 11111
✓ When we talked about CLASSFUL addressing – we realized the problem of wasted host addresses and depleting available network addresses.

✓ In subnetting, a network is divided into several smaller networks called subnetworks or subnets – each subnet will have it’s own address

✓ Typically, there are 2 steps in reaching a destination: first we must reach the network (netid) and then we reach the destination (hostid)
A network with two levels of hierarchy (not subnetting)

- The 2 level approach is not enough sometimes – you can only have 1 physical network – in example, all host are at the same level – no grouping

Network: 141.14.0.0/16
A network with three levels of hierarchy (subnetted)

Network: 141.14.0.0/16
Addresses in a network with and without subnetting

With subnetting, there are 3 levels (versus 2 levels). Partition the hostid space into subnetid and hostid.

(1\textsuperscript{st}) network, (2\textsuperscript{nd}) subnetwork and (3\textsuperscript{rd}) host
Figure  Network mask and subnetwork mask

Network mask

Subnetwork mask

\[ n \text{ bits} \quad \text{Change} \quad 32 - n \text{ bits} \]

\[ n_i \text{ bits} \quad 32 - n_i \text{ bits} \]

- netid
- hostid
- subnetid
- hostid
Similar to Hierarchy concept in a telephone number

(408) 864 – 8902
Area code  Exchange  Connection
Default mask and subnet mask

Default Mask
255.255.0.0

141.14.72.24  AND  141.14.0.0
IP address                Network address

a. Without subnetting

Subnet Mask
255.255.192.0

141.14.72.24  AND  141.14.64.0
IP address                Network address

b. With subnetting
A class B network can be divided into four subnetworks. The value of \( n = 16 \) and the value of

\[
    n_1 = n_2 = n_3 = n_4 = 16 + \log_2 4 = 18.
\]

This means that the subnet mask has **eighteen** 1s and **fourteen** 0s. In other words, the subnet mask is 255.255.192.0 which is different from the network mask for class B (255.255.0.0).
Example 14

A class B network is divided into four subnets. Since one of the addresses in subnet 2 is 141.14.120.77, we can find the subnet address as:

<table>
<thead>
<tr>
<th>Address</th>
<th>→</th>
<th>141</th>
<th>.</th>
<th>14</th>
<th>.</th>
<th>120</th>
<th>.</th>
<th>77</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask</td>
<td>→</td>
<td>255</td>
<td>.</td>
<td>255</td>
<td>.</td>
<td>192</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>Subnet Address</td>
<td>→</td>
<td>141</td>
<td>.</td>
<td>14</td>
<td>.</td>
<td>64</td>
<td>.</td>
<td>0</td>
</tr>
</tbody>
</table>

The values of the first, second, and fourth bytes are calculated using the first short cut for AND operation. The value of the third byte is calculated using the second short cut for the AND operation.

<table>
<thead>
<tr>
<th>Address (120)</th>
<th>0 + 64 + 32 + 16 + 8 + 0 + 0 + 0 + 0 + 0 + 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask (192)</td>
<td>128 + 64 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0</td>
</tr>
<tr>
<td>Result (64)</td>
<td>0 + 64 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0</td>
</tr>
</tbody>
</table>
Example 15

What is the subnetwork address if the destination address is 200.45.34.56 and the subnet mask is 255.255.240.0?

Solution

Address: 11001000 00101101 00100010 001111000

Mask: 11111111 11111111 11110000 00000000

AND: 11001000 00101101 00100000 00000000

The subnetwork address is 200.45.32.0.
### Recall - 5-bit Address Space Illustration

#### 1-bit Netid case (no subnets)

16 addresses/block  

Number of blocks: 2  

Address range per block: 0 to 15  

Netids: 0, 1  

Network Addresses: 00000, 10000  

Broadcast Addresses: 01111, 11111
5-bit Address Space Illustration

1-bit Subnet case

Number of blocks/networks: 2

Number subnets per block: 2

8 addresses/subnet

Address range per subnet: 0 to 7

Subnet ids: 0, 1

Network Addresses: 00000, 01000, 10000, 11000

Broadcast Addresses: 00111, 01111, 10111, 11111
5-bit Address Space Illustration

2-bit Subnet case

Number of blocks/networks: 2

Number subnets per block: 4

4 addresses/subnet

Address range per subnet: 0 to 3

Subnet ids: 00, 01, 10, 11

Network Addresses: 00000, 00100, 01000, 01100
10000, 10100, 11000, 11100

Broadcast Addresses: 00011, 00111, 01011, 01111
10011, 10111, 11011, 11111
Illustrating the mask concept (1 of 3)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

What is the mask?
Illustrating the mask concept (2 of 3)

What is the mask (subnet mask)?
**Illustrating the mask concept (3 of 3)**

**What is the mask (subnet mask) ?**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The number of subnets must be a power of 2.

Determine the number of subnets added by looking at the number of 1s added to the default mask and performing $2$ raised to that number.

For example, $2^3 = 8$ subnets
A company is granted the site address 201.70.64.0 (class C). The company needs six subnets. Design the subnets.

**Solution**

The number of 1s in the default mask is 24 (class C).

The company needs six subnets. This number 6 is not a power of 2. The next number that is a power of 2 is 8 \((2^3)\). We need 3 more 1s in the subnet mask. The total number of 1s in the subnet mask is 27 \((=24 + 3)\).

The total number of 0s is 5 \((=32 – 27)\). The mask would be
The number of subnets is 8.
The number of addresses in each subnet is $2^5$ (5 is the number of 0s) or 32.
Example 16 (Solution Continued)

Start here

201.70.64.0  Add 31  201.70.64.31
1st subnet
Add 1

201.70.64.32  Add 31  201.70.64.63
2nd subnet
Add 1

201.70.64.224  201.70.64.255
8th subnet

Finish here
A company is granted the site address 181.56.0.0 (class B). The company needs 1000 subnets. Design the subnets.

**Solution**

The number of 1s in the default mask is 16 (class B).

The company needs 1000 subnets. This number is not a power of 2. The next number that is a power of 2 is 1024 ($2^{10}$). We need 10 more 1s in the subnet mask.

The total number of 1s in the subnet mask is 26 ($=16 + 10$).

The total number of 0s is 6 ($=32 - 26$).

The mask is

```
11111111 11111111 11111111 11000000
```

or

```
255.255.255.192.
```

The number of subnets is 1024.

The number of addresses in each subnet is $2^6$ (6 is the number of 0s) or 64.
Example 17 (Solution Continued)

Subtract 63 from 255 to get 192
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*

Network consisting of 3 subnets:
• Although class A and B addresses are dwindling – there are plenty of class C addresses

• The problem with C addresses is, they only have 256 hostids – not enough for any midsize to large size organization – especially if you plan to give every computer, printer, scanner, etc. multiple IP addresses

• Supernetting allows an organization the ability to combine several class C blocks in creating a larger range of addresses

• Note: breaking up a network = subnetting

• Note: combining Class-C networks = supernetting
When assigning class C block, the choices of blocks need to follow a set of rules:

- #1 – the # of blocks must be a power of 2
- #2 – blocks must be contiguous (no gaps between blocks)
- #3 – the 3rd byte of the first address in the superblock must be evenly divisible by the number of blocks – ie. if the # of blocks is $N$, the 3rd byte must be divisible by $N$
A company needs 600 addresses. Which of the following set of class C blocks can be used to form a supernet for this company?

a. 198.47.32.0 198.47.33.0 198.47.34.0
b. 198.47.32.0 198.47.42.0 198.47.52.0 198.47.62.0
c. 198.47.31.0 198.47.32.0 198.47.33.0 198.47.34.0
d. 198.47.32.0 198.47.33.0 198.47.34.0 198.47.35.0

Solution

a: No, there are only three blocks. Must be a power of 2
b: No, the blocks are not contiguous.
c: No, 31 in the first block is not divisible by 4.
d: Yes, all three requirements are fulfilled. (1. Power of 2, 2. Contiguous and 3. 3rd byte of 1st address is divisible by 4: 32/4=8)
A supernet has a first address of 205.16.32.0 and a supernet mask of 255.255.248.0. How many blocks are in this supernet and what is the range of addresses?

**Solution**

- The default mask has 24 1s because 205.16.32.0 is a class C.
- Because the supernet mask is 255.255.248.0, the supernet has 21 1s.
- Since the difference between the default and supernet masks is 3, there are $2^3$ or 8 blocks in this supernet.
- Because the blocks start with 205.16.32.0 and must be contiguous, the blocks are 205.16.32.0, 205.16.33.0, 205.16.34.0………. 205.16.39.0.
- The first address is 205.16.32.0. The last address is 205.16.39.255.
- The total number of addresses is 8 x 256 = 2048
Explain Supernetting Conceptually

| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Back out this bit from netid into host id

Causes these 2 blocks to combine as a single block
Figure Comparison of subnet, default, and supernet mask.

Divide 1 class C block into 8 subblocks

Subnet mask: 11111111 11111111 11111111 111 00000

$n_{\text{sub}} = 24 + 3 = 27$

Default mask: 11111111 11111111 11111111 00000000

$n = 24$

Supernet mask: 11111111 11111111 111111 00000000000

$n_{\text{super}} = 24 - 3 = 21$

Combine 8 class C blocks into 1 superblock.
Classless Addressing

✓ Variable-length blocks are used that belong to no classes.
the number of addresses in a block; it must be a power of 2 (2^0, 2^1, 2^2, 2^3, . . .).
In classless addressing, the prefix defines the network and the suffix defines the host.
What is the prefix length and suffix length if the whole Internet is considered as one single block with 4,294,967,296 addresses?

**Solution**

In this case, the prefix length is 0 and the suffix length is 32. All 32 bits vary to define $2^{32} = 4,294,967,296$ hosts in this single block.
Example 21

What is the prefix length and suffix length if the Internet is divided into 4,294,967,296 blocks and each block has one single address?

Solution

In this case, the prefix length for each block is 32 and the suffix length is 0. All 32 bits are needed to define $2^{32} = 4,294,967,296$ blocks. The only address in each block is defined by the block itself.
The number of addresses in a block is inversely related to the value of the prefix length, $n$. A small $n$ means a larger block; a large $n$ means a small block.
The following addresses are defined using slash notations.

a. In the address 12.23.24.78/8, the network mask is 255.0.0.0. The mask has eight 1s and twenty-four 0s. The prefix length is 8; the suffix length is 24.

b. In the address 130.11.232.156/16, the network mask is 255.255.0.0. The mask has sixteen 1s and sixteen 0s. The prefix length is 16; the suffix length is 16.

c. In the address 167.199.170.82/27, the network mask is 255.255.255.224. The mask has twenty-seven 1s and five 0s. The prefix length is 27; the suffix length is 5.
Extracting Block Information

Given an addresses using slash notations:

a. The number of addresses in the block:
   \[ N = 2^{32} - n \]

b. First address:
   \[(\text{the given address}) \ AND \ (\text{network mask})\]

c. Last address:
   \[(\text{the given address}) \ OR \ [\text{NOT (network mask)}]\]
One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

Solution
The value of n is 27. The network mask has twenty-seven 1s and five 0s. It is 255.255.255.240.

a. The number of addresses in the network is $2^{32-n} = 32$.

b. We use the AND operation to find the first address (network address). The first address is 167.199.170.64/27.

<table>
<thead>
<tr>
<th>Address in binary:</th>
<th>10100111 11000111 10101010 01010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network mask:</td>
<td>11111111 11111111 11111111 11100000</td>
</tr>
<tr>
<td>First address:</td>
<td>10100111 11000111 10101010 01000000</td>
</tr>
</tbody>
</table>
To find the last address, we first find the complement of the network mask and then OR it with the given address: The last address is 167.199.170.95/27.

<table>
<thead>
<tr>
<th>Address in binary:</th>
<th>10100111 11000111 10101010 01010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement of network mask:</td>
<td>00000000 00000000 00000000 00011111</td>
</tr>
<tr>
<td>Last address:</td>
<td>10100111 11000111 10101010 01011111</td>
</tr>
</tbody>
</table>
Example 24

One of the addresses in a block is 17.63.110.114/24. Find the number of addresses, the first address, and the last address in the block.

Solution

The network mask is 255.255.255.0.

a. The number of addresses in the network is $2^{32-24} = 256$.

b. To find the first address, we use the short cut methods discussed early in the chapter. The first address is 17.63.110.0/24.

<table>
<thead>
<tr>
<th>Address:</th>
<th>17</th>
<th>63</th>
<th>110</th>
<th>114</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network mask:</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>First address (AND):</td>
<td>17</td>
<td>63</td>
<td>110</td>
<td>0</td>
</tr>
</tbody>
</table>
To find the last address, we use the complement of the network mask and the first short cut method we discussed before. The last address is 17.63.110.255/24.

<table>
<thead>
<tr>
<th>Address in binary:</th>
<th>10100111 11000111 10101010 01010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement of network mask:</td>
<td>00000000 00000000 00000000 00011111</td>
</tr>
<tr>
<td>Last address:</td>
<td>10100111 11000111 10101010 01011111</td>
</tr>
</tbody>
</table>
One of the addresses in a block is 110.23.120.14/20. Find the number of addresses, the first address, and the last address in the block.

Solution
The network mask is 255.255.240.0.

a. The number of addresses in the network is $2^{32} - 20 = 4096$.

b. To find the first address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The first address is 110.23.112.0/20.
c. To find the last address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The OR operation is applied to the complement of the mask. The last address is 110.23.127.255/20.

<table>
<thead>
<tr>
<th>Address:</th>
<th>110</th>
<th>23</th>
<th>120</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network mask:</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>255</td>
</tr>
<tr>
<td>Last address (OR):</td>
<td>110</td>
<td>23</td>
<td>127</td>
<td>255</td>
</tr>
</tbody>
</table>
Block Allocation

Three restrictions needed to be applied to the allocated block:

a. The number of requested addresses, \( N \), needs to be a power of 2.

b. The value of prefix length, \( n \), can be found from the number of addresses in the block.
\[
    n = 32 - \log_2 N
\]

c. The beginning address needs to be divisible by the number of addresses of the block.
   • For example, if a block contains 4 addresses, the beginning address must be divisible by 4. If the block has less than 256 addresses, we need to check only the rightmost byte. If it has less than 65,536 addresses, we need to check only the two rightmost bytes, and so on
Which of the following can be the beginning address of a block that contains 16 addresses?

a. 123.45.24.52  
b. 205.16.37.32  
c. 190.16.42.44  
d. 17.17.33.80

**Solution**

The address **205.16.37.32** is eligible because 32 is divisible by 16. The address **17.17.33.80** is eligible because 80 is divisible by 16.
Which of the following can be the beginning address of a block that contains 1024 addresses?

a. 205.16.37.32
b. 190.16.42.0
c. 17.17.32.0
d. 123.45.24.52

Solution

• To be divisible by 1024, the rightmost byte of an address should be 0 because any value in that first byte will be a fraction of 1024 (i.e. 0 to 255).

• To be divisible by 1024, the rightmost byte should be 0 and the second rightmost byte must be divisible by 4 because for every unique number in the second byte position, there exist 256 addresses in the first byte position that maps to it. To get 1024 addresses overall, you will need an increment of 4 in the 2\textsuperscript{nd} byte position.

• Therefore, the 2\textsuperscript{nd} byte needs to be divisible by 4.

• Only the address 17.17.32.0 meets this condition.
# Prefix length for classful addressing

<table>
<thead>
<tr>
<th>Class</th>
<th>Prefix length</th>
<th>Class</th>
<th>Prefix length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>/8</td>
<td>D</td>
<td>/4</td>
</tr>
<tr>
<td>B</td>
<td>/16</td>
<td>E</td>
<td>/4</td>
</tr>
<tr>
<td>C</td>
<td>/24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An organization is granted the block 130.34.12.64/26. The organization needs four subnetworks, each with an equal number of hosts. Design the subnetworks and find the information about each network.

**Solution**

The number of addresses for the whole network can be found as \( N = 2^{32} - 2^6 = 64 \). The first address in the network is 130.34.12.64/26 and the last address is 130.34.12.127/26. We now design the subnetworks:

1. We grant 16 addresses for each subnetwork to meet the first requirement (64/16 is a power of 2).
2. The subnetwork mask for each subnetwork is:

\[
n_1 = n_2 = n_3 = n_4 = n + \log_2 (N/N) = 26 + \log_2 4 = 28
\]
3. We grant 16 addresses to each subnet starting from the first available address. The following Figure shows the subblock for each subnet. Note that the starting address in each subnetwork is divisible by the number of addresses in that subnetwork.
Figure Solution to Example 28

a. Original block

N = 64 addresses

n = 26

130.34.12.64/26
First address

130.34.12.127/26
Last address

b. Subblocks

N = 16 addresses

N = 16 addresses

N = 16 addresses

N = 16 addresses

n = 28

n = 28

n = 28

n = 28

130.34.12.64/28
First address

130.34.12.80/28
First address

130.34.12.96/28
First address

130.34.12.112/28
First address

130.34.12.127/28
Last address
An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets as shown below:

- One subblock of 120 addresses.
- One subblock of 60 addresses.
- One subblock of 10 addresses.

**Solution**

There are $2^{32} - 24 = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24.

a. The number of addresses in the first subblock is not a power of 2. We allocate 128 addresses. The subnet mask is 25. The first address is 14.24.74.0/25; the last address is 14.24.74.127/25.
b. The number of addresses in the second subblock is not a power of 2 either. We allocate 64 addresses. The subnet mask is 26. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

c. The number of addresses in the third subblock is not a power of 2 either. We allocate 16 addresses. The subnet mask is 28. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

d. If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.209. The last address is 14.24.74.255.

e. Figure 5.31 shows the configuration of blocks. We have shown the first address in each block.
Figure  Solution to Example 29

a. Original block

14.24.74.0/24

N = 256 addresses

n = 24

14.24.74.0/25

14.24.74.128/26

14.24.192.0/28

b. Subblocks

14.24.74.255/24
Last address
Assume a company has three offices: Central, East, and West. The Central office is connected to the East and West offices via private, WAN lines. The company is granted a block of 64 addresses with the beginning address 70.12.100.128/26. The management has decided to allocate 32 addresses for the Central office and divides the rest of addresses between the two other offices.

1. The number of addresses are assigned as follows:

   Central office $N_c = 32$  
   East office $N_e = 16$  
   West office $N_w = 16$

2. We can find the prefix length for each subnetwork:

   \[
   n_c = n + \log_2(64/32) = 27 \\
   n_e = n + \log_2(64/16) = 28 \\
   n_w = n + \log_2(64/16) = 28
   \]
3. Figure 5.32 shows the configuration designed by the management. The Central office uses addresses 70.12.100.128/27 to 70.12.100.159/27. The company has used three of these addresses for the routers and has reserved the last address in the subblock. The East office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The West office uses the addresses 70.12.100.176/28 to 70.12.100.191/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The company uses no address for the point-to-point connections in WANs.
Figure  Example 30

Network: 70.12.100.128/26

All addresses from 70.12.100.128 to 70.12.100.191 are delivered to this network.
An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

- The first group has 64 customers; each needs approximately 256 addresses.
- The second group has 128 customers; each needs approximately 128 addresses.
- The third group has 128 customers; each needs approximately 64 addresses.

We design the subblocks and find out how many addresses are still available after these allocations.
Solution

Let us solve the problem in two steps. In the first step, we allocate a subblock of addresses to each group. The total number of addresses allocated to each group and the prefix length for each subblock can be found as:

<table>
<thead>
<tr>
<th>Group</th>
<th>Subblock Size</th>
<th>Prefix Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>$64 \times 256 = 16,384$</td>
<td>$n_1 = 16 + \log_2 (65536/16384) = 18$</td>
</tr>
<tr>
<td>Group 2</td>
<td>$128 \times 128 = 16,384$</td>
<td>$n_2 = 16 + \log_2 (65536/16384) = 18$</td>
</tr>
<tr>
<td>Group 3</td>
<td>$128 \times 64 = 8192$</td>
<td>$n_3 = 16 + \log_2 (65536/8192) = 19$</td>
</tr>
</tbody>
</table>

Figure 5.33 shows the design for the first hierarchical level. Figure 5.34 shows the second level of the hierarchy. Note that we have used the first address for each customer as the subnet address and have reserved the last address as a special address.
Figure  Solution to Example 31: first step

ISP

190.100.0.0/16

190.100.0.0/16 to 190.100.255.255/16 /16

Unassigned: 190.100.160.0 to 190.100.255.255 (24567 addresses)

190.100.0.0/18 ● ● ● 190.100.63.255/18

/18

190.100.64.0/18 ● ● ● 190.100.127.255/18

/18

190.100.128.0/19 ● ● ● 190.100.159.255/19

/19

Group 1

Group 2

Group 3
Solution to Example 31: second step

Group 1

- Group: \( n = 18 \)
- Subnet: \( n = 18 + \log_2 (16385/256) = 24 \)

Group 2

- Group: \( n = 18 \)
- Subnet: \( n = 18 + \log_2 (16385/128) = 25 \)

Group 3

- Group: \( n = 19 \)
- Subnet: \( n = 19 + \log_2 (8192/64) = 26 \)
Special Addresses

In classful addressing some addresses were reserved for special purposes. The classless addressing scheme inherits some of these special addresses from classful addressing.
Special Blocks

- All-Zeros Address
- **All-Ones Address: Limited Broadcast Address**
- Loopback Addresses
- Private Addresses
- Multicast Addresses
An address of all 0’s is used during bootstrap time if the host doesn’t know it’s IP address. The un-named host sends an all 0 source address and limited broadcast (all 1’s) destination address to the bootstrap server (DHCP server).
An host want to send a message to every other host

- The broadcast are confined to the local network
Example of this host on this address

A host that does not know its IP address uses the IP address 0.0.0.0 as the source address and 255.255.255.255 as the destination address to send a message to a bootstrap server.

Example of specific host on this network

An address with a netid of all 0’s is used by a host or router to send another host with in the same network a message.
The IP address with the 1st byte equal to 127 (127.0.0.0) is used for the loop back address.

Loopback address is used to test software on a machine – the packet never leaves the machine – it returns to the protocol software.

Example: a “ping” command can send a packet with a loopback address as the destination address to see if the IP software is capable of receiving and processing a packet.
• Private addresses:
  • They are not recognized globally.

• Multicast addresses:
  • 224.0.0.0/4 is reserved for multicast communication.
Network: **221.45.71.0/24**

- 221.45.71.64/24
- 221.45.71.126/24
- 221.45.71.20/24
- 221.45.71.178/24

**Figure**  Example of a directed broadcast address
Example of direct broadcast address

<table>
<thead>
<tr>
<th>Netid</th>
<th>Hostid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific</td>
<td>All 1s</td>
</tr>
<tr>
<td>221.45.71.64</td>
<td>221.45.71.126</td>
</tr>
</tbody>
</table>

The direct broadcast address is used by a router to send a message to every host on a local network. Every host/router receives and processes the packet with a direct broadcast address.

Router sending to all hosts on a network

If the hostid is all 1’s, it’s called a “broadcast address” and the router use it to send a packet to all host in a specific network. In this case, hosts 20, 64, 126 and etc. will receive the packet from the router.

Example of limited broadcast address

<table>
<thead>
<tr>
<th>Netid and hostid</th>
</tr>
</thead>
<tbody>
<tr>
<td>All 1s</td>
</tr>
</tbody>
</table>

| Destination IP address: | 255.255.255.255 |

A limited broadcast address is used by a host to send a packet to every host on the same network. However, the packet is blocked by routers to confine the packet to the local network.

Host sending to all other hosts on a network

If the hostid and netid are all 1’s, it’s called a “limited broadcast address”. If the host wants to send a packet to all host in a specific network, it would use this address. The router would block this address so that data stays contained within a specific network.
IP addresses: how to get one?

Q: How does a *host* get IP address?
   - The IP address of the computer
   - The subnet mask of the computer
   - The IP address of a router
   - The IP address of a name server

- 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”
DHCP: Dynamic Host Configuration Protocol

goal: allow host to dynamically obtain its IP address from network server when it joins network
  - can renew its lease on address in use
  - allows reuse of addresses (only hold address while connected/“on”)
  - support for mobile users who want to join network (more shortly)

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

arriving DHCP client needs address in this network
DHCP client-server scenario

DHCP server: 223.1.2.5

**DHCP discover**
Broadcast: is there a DHCP server out there?

**DHCP offer**
Broadcast: I’m a DHCP server! Here’s an IP address you can use

**DHCP request**
Broadcast: OK. I’ll take that IP address!

**DHCP ACK**
Broadcast: OK. You’ve got that IP address!
DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

- address of first-hop router for client
- name and IP address of DNS server
- network mask (indicating network versus host portion of address)
connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP

DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet

Ethernet frame broadcast (dest: FFFFFFFF) on LAN, received at router running DHCP server

Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
DHCP: example

- DHCP server formulates DHCP ACK containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server.
- Encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client.
- Client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router.

![Diagram showing DHCP server and client with encapsulation and demultiplexing process.]
IP addressing: the last word...

Q: how does an ISP get block of addresses?
A: ICANN: Internet Corporation for Assigned Names and Numbers (ICANN) http://www.icann.org/
  - allocates addresses
  - manages DNS
  - assigns domain names, resolves disputes
NAT: network address translation

rest of Internet

local network
(e.g., home network)
10.0.0/24

10.0.0.1
10.0.0.2
10.0.0.3

138.76.29.7

10.0.0.4

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

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

NAT: network address translation
NAT: network address translation

- 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, e.g., P2P applications
  - address shortage should instead be solved by IPv6
NAT traversal problem

- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can’t use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7

- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 2500
NAT traversal problem

- **solution 2:** Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

  i.e., automate static NAT port map configuration
solution 3: relaying (used in Skype)

- NATed client establishes connection to relay
- external client connects to relay
- relay bridges packets between to connections

1. connection to relay initiated by NATed host
2. connection to relay initiated by client
3. relaying established

138.76.29.7
10.0.0.1
Network Layer: outline

1 introduction
2 virtual circuit and datagram networks
3 what’s inside a router
4 IP: Internet Protocol
   ▪ datagram format
   ▪ IPv4 addressing
   ▪ ICMP
   ▪ IPv6
5 routing algorithms
   ▪ link state
   ▪ distance vector
   ▪ hierarchical routing
6 routing in the Internet
   ▪ RIP
   ▪ OSPF
   ▪ BGP
7 broadcast and multicast routing
ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)

- network-layer “above” IP:
  - ICMP msgs carried in IP datagrams

- ICMP message: type, code plus first 8 bytes of IP datagram causing error

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>echo reply (ping)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>dest. network unreachable</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dest host unreachable</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>dest protocol unreachable</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>dest port unreachable</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>dest network unknown</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>dest host unknown</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>source quench (congestion control - not used)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>echo request (ping)</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>route advertisement</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>router discovery</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>TTL expired</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>bad IP header</td>
</tr>
</tbody>
</table>
ICMP Encapsulation

Frame header | ICMP message | IP header | IP data | Frame data | Trailer (if any)

8 bits: Type | 8 bits: Code | 8 bits: Checksum | Rest of the header | Data section
Traceroute and ICMP

- source sends series of UDP segments to dest
  - first set has TTL = 1
  - second set has TTL = 2, etc.
  - unlikely port number
- when \( n \)th set of datagrams arrives to nth router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes name of router & IP address
- when ICMP messages arrives, source records RTTs

**stopping criteria:**
- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops
IPv6: motivation

- **initial motivation**: 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

**IPv6 datagram format:**
- fixed-length 40 byte header
- no fragmentation allowed
IPv6 datagram format

**Priority**: identify priority among datagrams in flow

**Flow Label**: identify datagrams in same “flow.”

(concept of “flow” not well defined).

**Next Header**: identify upper layer protocol for data

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td>payload len</td>
<td>next hdr</td>
<td>hop limit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source address</th>
<th>(128 bits)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>destination address</th>
<th>(128 bits)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
</table>

| 32 bits |
Other changes from IPv4

- **checksum**: removed entirely to reduce processing time at each hop
- **options**: allowed, but outside of header, indicated by “Next Header” field
- **ICMPv6**: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?

- **tunneling**: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers
Tunneling

**logical view:**

A (IPv6) —— B (IPv6) —— E (IPv6) —— F (IPv6)

**physical view:**

A (IPv6) —— B (IPv6) —— C (IPv4) —— D (IPv4) —— E (IPv6) —— F (IPv6)

IPv4 tunnel connecting IPv6 routers
Tunneling

logical view:

physical view:

IPv4 tunnel connecting IPv6 routers

A-to-B: IPv6
B-to-C: IPv6 inside IPv4
E-to-F: IPv6
IPv6: adoption

US National Institutes of Standards estimate [2013]:
- ~3% of industry IP routers
- ~11% of US gov’t routers

Long (long!) time for deployment, use
- 20 years and counting!
- think of application-level changes in last 20 years: WWW, Facebook, …