- IPv4 Addressing and Subnetting -

Hardware Addressing

A **hardware address** is used to uniquely identify a host *within* a local network. Hardware addressing is a function of the **Data-Link** layer of the OSI model (Layer-2).

Ethernet utilizes the 48-bit **MAC address** as its hardware address. The MAC address is often hardcoded on physical network interfaces, though some interfaces support changing the MAC address using special utilities. In virtualization environments, dynamically assigning MAC addresses is very common.

A MAC address is most often represented in **hexadecimal,** using one of two accepted formats:

00:43:AB:F2:32:13
0043.ABF2.3213

The first six hexadecimal digits of a MAC address identify the *manufacturer* of the physical network interface. This is referred to as the **OUI (Organizational Unique Identifier).** The last six digits uniquely identify the host itself, and are referred to as the **host ID**.

The MAC address has one shortcoming – it contains no *hierarchy.* MAC addresses provide no mechanism to create **boundaries** between networks. There is no method to distinguish one network from another.

This lack of hierarchy poses *significant* difficulties to network scalability. If *only* Layer-2 hardware addressing existed, all hosts would technically exist on the *same* network. Internetworks like the Internet could not exist, as it would be impossible to separate *my* network from *your* network.

Imagine if the entire Internet existed purely as a single Layer-2 switched network. Switches, as a rule, will forward a broadcast out *every* port. With billions of hosts on the Internet, the resulting broadcast storms would be devastating. The Internet would simply collapse.

The scalability limitations of Layer-2 hardware addresses are mitigated using **logical addresses,** covered in great detail in this guide.

Logical Addressing

Logical addressing is a function of the **Network** layer of the OSI Model (Layer-3), and provides a hierarchical structure to separate networks. Logical addresses are never hardcoded on physical network interfaces, and can be dynamically assigned and changed freely.

A logical address contains two components:

- **Network ID** – identifies which network a host belongs to.
- **Host ID** – uniquely identifies the host on that network.

Examples of logical addressing protocols include **Internetwork Packet Exchange (IPX)** and **Internet Protocol (IP)**. IPX was predominantly used on Novell networks, but is now almost entirely deprecated. **IP** is the most widely-used logical address, and is the backbone protocol of the Internet.

Internet Protocol (IP)

In the 1970's, the Department of Defense developed the **Transmission Control Protocol (TCP)**, to provide both Network and Transport layer functions. When this proved to be an inflexible solution, those functions were separated - with the **Internet Protocol (IP)** providing Network layer services, and TCP providing Transport layer services.

Together, TCP and IP provide the core functionality for the **TCP/IP** or **Internet protocol suite**.

IP provides two fundamental Network layer services:

- **Logical addressing** – provides a unique address that identifies both the *host*, and the *network* that host exists on.
- **Routing** – determines the *best path* to a particular destination network, and then *routes* data accordingly.

IP was originally defined in RFC 760, and has been revised several times. IP Version 4 (**IPv4**) was the first version to experience widespread deployment, and is defined in RFC 791. IPv4 will be the focus of this guide.

IPv4 employs a **32-bit address**, which limits the number of possible addresses to 4,294,967,296. IPv4 will eventually be replaced by IP Version 6 (**IPv6**), due to a shortage of available IPv4 addresses. IPv6 is covered in great detail in another guide.

IPv4 Addressing

A core function of IP is to provide logical addressing for hosts. An **IP address** provides a hierarchical structure to both uniquely identify a *host*, and what *network* that host exists on.

An IP address is most often represented in **decimal,** in the following format:

158.80.164.3

An IP address is comprised of four **octets,** separated by periods:

First Octet	Second Octet	Third Octet	Fourth Octet
158	80	164	3

Each octet is an **8-bit** number, resulting in a **32-bit IP address**. The smallest possible value of an octet is *0,* or *00000000* in binary. The largest possible value of an octet is *255*, or *11111111* in binary.

The above IP address represented in binary would look as follows:

First Octet	Second Octet	Third Octet	Fourth Octet
10011110	01010000	10100100	00000011

Decimal to Binary Conversion

The simplest method of converting between decimal and binary is to remember the following table:

128	64	32	16	8	4	2	1

To convert a decimal number of *172* to binary, start with the leftmost column. Since *172* is greater than *128*, that binary bit will be set to *1*. Next, add the value of the next column (*128 + 64 = 192)*. Since *172* is less than *192,* that binary bit will be set to *0.*

Again, add the value of the next column (*128 + 32 = 160)*. Since *172* is greater than *160*, that binary bit will be set to *1.* Continue this process until the columns with binary bits set to *1* add up to 172:

Decimal	128	64	32	16	8	4	2	1
Binary	1	0	1	0	1	1	0	0

Binary to Decimal Conversion

Converting from binary back to decimal is even simpler. Apply the binary number to the conversion table, and then add up any columns with binary bits set to 1.

For example, consider the binary number of *11110001:*

Decimal	128	64	32	16	8	4	2	1
Binary	1	1	1	1	0	0	0	1

By adding *128 + 64 + 32 + 16+ 1,* it can be determined that *11110001* equals *241.*

The Subnet Mask

Part of an IP address identifies the *network.* The other part of the address identifies the *host.* A **subnet mask** is required to provide this distinction**:**

158.80.164.3 255.255.0.0

The above IP address has a subnet mask of *255.255.0.0*. The subnet mask follows two rules:

- If a binary bit is set to a **1** (or *on*) in a subnet mask, the corresponding bit in the address identifies the **network**.
- If a binary bit is set to a **0** (or *off*) in a subnet mask, the corresponding bit in the address identifies the **host**.

Looking at the above address and subnet mask in binary:

```
IP Address:       10011110.01010000.10100100.00000011
Subnet Mask:      11111111.11111111.00000000.00000000
```

The first 16 bits of the subnet mask are set to *1*. Thus, the first 16 bits of the address (*158.80*) identify the *network*. The last 16 bits of the subnet mask are set to *0*. Thus, the last 16 bits of the address (*164.3*) identify the unique *host* on that network.

The network portion of the subnet mask must be **contiguous**. For example, a subnet mask of *255.0.0.255* is not valid.

The Subnet Mask (continued)

Hosts on the same logical network will have *identical* network addresses, and can communicate freely. For example, the following two hosts are on the same network:

Host A:	158.80.164.100 255.255.0.0
Host B:	158.80.164.101 255.255.0.0

Both share the same network address (*158.80*), which is determined by the *255.255.0.0* subnet mask. Hosts that are on *different* networks cannot communicate without an intermediating device. For example:

Host A:	158.80.164.100 255.255.0.0
Host B:	158.85.164.101 255.255.0.0

The subnet mask has remained the same, but the network addresses are now different (*158.80* and *158.85* respectively). Thus, the two hosts are *not* on the same network, and cannot communicate without a **router** between them. **Routing** is the process of forwarding packets from one network to another.

Consider the following, trickier example:

Host A:	158.80.1.1 255.248.0.0
Host B:	158.79.1.1 255.248.0.0

The specified subnet mask is now *255.248.0.0*, which doesn't fall cleanly on an octet boundary. To determine if these hosts are on separate networks, first convert everything to binary:

Host A Address:	10011110.01010000.00000001.00000001
Host B Address:	10011110.01001111.00000001.00000001
Subnet Mask:	11111111.11111000.00000000.00000000

Remember, the **1** (or **on)** bits in the subnet mask identify the *network* portion of the address. In this example, the first *13 bits* (the 8 bits of the first octet, and the first 5 bits of the second octet) identify the network. Looking at only the first 13 bits of each address:

Host A Address:	10011110.01010
Host B Address:	10011110.01001

Clearly, the network addresses are *not* identical. Thus, these two hosts are on separate networks, and require a router to communicate.

IP Address Classes

The IPv4 address space has been structured into several **classes**. The value of the **first octet** of an address determines the class of the network:

Class	*First Octet Range*	*Default Subnet Mask*
Class A	1 - 127	255.0.0.0
Class B	128 - 191	255.255.0.0
Class C	192 - 223	255.255.255.0
Class D	224 - 239	-

Class A networks range from **1** to **127.** The *default* subnet mask is 255.0.0.0. Thus, by *default,* the first octet defines the network, and the last three octets define the host. This results in a maximum of **127** Class A networks, with **16,777,214** hosts per network!

Example of a Class A address:

Address:	64.32.254.100
Subnet Mask:	255.0.0.0

Class B networks range from **128** to **191**. The *default* subnet mask is 255.255.0.0. Thus, by *default,* the first two octets define the network, and the last two octets define the host. This results in a maximum of **16,384** Class B networks, with **65,534** hosts per network.

Example of a Class B address:

Address:	152.41.12.195
Subnet Mask:	255.255.0.0

Class C networks range from **192** to **223.** The *default* subnet mask is 255.255.255.0. Thus, by *default,* the first three octets define the network, and the last octet defines the host. This results in a maximum of **2,097,152** Class C networks, with **254** hosts per network.

Example of a Class C address:

Address:	207.79.233.6
Subnet Mask:	255.255.255.0

Class D networks are reserved for **multicast** traffic. Class D addresses do not use a subnet mask.

CIDR (Classless Inter-Domain Routing)

Classless Inter-Domain Routing (CIDR) is a simplified method of representing a subnet mask. CIDR identifies the number of binary bits set to a **1** (or *on*) in a subnet mask, preceded by a slash.

For example, a subnet mask of *255.255.255.240* would be represented as follows in binary:

11111111.11111111.11111111.11110000

The first 28 bits of the above subnet mask are set to *1*. The CIDR notation for this subnet mask would thus be ***/28***.

The CIDR mask is often appended to the IP address. For example, an IP address of *192.168.1.1* and a subnet mask of *255.255.255.0* would be represented as follows using CIDR notation:

192.168.1.1 /24

Address Classes vs. Subnet Mask

Remember the following three rules:

- The **first octet** on an address dictates the *class* of that address.
- The **subnet mask** determines what part of an address identifies the *network*, and what part identifies the *host*.
- Each class has a ***default*** subnet mask. A network using its default subnet mask is referred to as a **classful network**.

For example, *10.1.1.1* is a Class A address, and its default subnet mask is *255.0.0.0* (*/8* in CIDR).

It is entirely possible to use subnet masks *other* than the default. For example, a Class B subnet mask can be applied to a Class A address:

10.1.1.1 /16

However, **this does not change the class of the above address**. It remains a Class A *address*, which has been subnetted using a Class B *mask*.

Remember, the ***only*** thing that determines the class of an IP address is the first octet of that address. Likewise, the subnet mask is the ***only*** thing that determines what part of an address identifies the network, and what part identifies the host.

Subnet and Broadcast Addresses

On *each* IP network, two host addresses are reserved for special use:

- The **subnet** (or **network**) address
- The **broadcast** address

Neither of these addresses can be assigned to an actual host.

The **subnet** address is used to identify **the network itself**. A routing table contains a list of known networks, and each network is identified by its subnet address. Subnet addresses contain **all 0 bits in the host portion** of the address.

For example, *192.168.1.0/24* is a subnet address. This can be determined by looking at the address and subnet mask in binary:

IP Address:	11000000.10101000.00000001.00000000
Subnet Mask:	11111111.11111111.11111111.00000000

Note that all host bits in the address are set to *0*.

The **broadcast** address identifies *all* hosts on a particular network. A packet sent to the broadcast address will be received and processed by every host on that network. Broadcast addresses contain **all 1 bits in the host portion** of the address.

For example, *192.168.1.255/24* is a broadcast address. Note that all host bits are set to *1:*

IP Address:	11000000.10101000.00000001.11111111
Subnet Mask:	11111111.11111111.11111111.00000000

Broadcasts are one of three types of IP packets:

- **Unicasts** are packets sent from one host to one other host
- **Multicasts** are packets sent from one host to a *group* of hosts
- **Broadcasts** are packets sent from one host to all other hosts on the local network

A router, by default, will **never forward** a multicast or broadcast packet from one interface to another.

A switch, by default, will forward a multicast or broadcast packet **out every port**, except for the port that originated the multicast or broadcast.

Subnetting

Subnetting is the process of creating new networks (or *subnets)* by **stealing bits** from the host portion of a subnet mask. There is one caveat: stealing bits from hosts creates **more** networks but **fewer** hosts per network.

Consider the following Class C network:

192.168.254.0

The default subnet mask for this network is 255.255.255.0. This single network can be segmented, or *subnetted*, into multiple networks. For example, assume a minimum of *10* new networks are required. Resolving this is possible using the following magical formula:

$$2^n$$

The exponent '**n'** identifies the number of bits to steal from the host portion of the subnet mask. The default Class C mask (255.255.255.0) looks as follows in binary:

11111111.1111111.1111111.00000000

There are a total of 24 bits set to *1*, which are used to identify the network. There are a total of 8 bits set to *0*, which are used to identify the host, and these host bits can be *stolen*.

Stealing bits essentially involves changing host bits (set to *0* or *off*) in the subnet mask to network bits (set to *1* or *on*). Remember, network bits in a subnet mask **must always be contiguous** - skipping bits is not allowed.

Consider the result if three bits are stolen. Using the above formula:

2^n = 2^3 = 8 = **8 new networks created**

However, a total of 8 new networks *does not* meet the original requirement of at least 10 networks. Consider the result if four bits are stolen:

2^n = 2^4 = 16 = **16 new networks created**

A total of 16 new networks *does* meet the original requirement. Stealing four host bits results in the following *new* subnet mask:

11111111.11111111.11111111.11110000 = 255.255.255.240

Subnetting (continued)

In the previous example, a Class C network was subnetted to create *16* new networks, using a subnet mask of *255.255.255.240* (or */28* in CIDR). Four bits were stolen in the subnet mask, leaving only four bits for hosts.

To determine the number of hosts this results in, for each of the new 16 networks, a slightly modified formula is required:

$$2^n - 2$$

Consider the result if four bits are available for hosts:

$2^n - 2 \quad = \quad 2^4 - 2 \quad = \quad 16 - 2 \quad =$ **14 usable hosts per network**

Thus, subnetting a Class C network with a /28 mask creates 16 new networks, with 14 usable hosts per network.

Why is the formula for calculating usable hosts $2^n - 2$? Because it is **never possible** to assign a host an address with all *0* or all *1* bits in the *host* portion of the address**.** These are reserved for the subnet and broadcast addresses, respectively. Thus, every time a network is subnetted, useable host addresses are lost.

The 2^n-2 Rule and Subnetted Networks

To avoid confusion, it was historically unacceptable to use the first and last new *networks* created when subnetting, as it is possible for a classful network to have the same subnet and broadcast address as its subnetted networks. This required the $2^n - 2$ formula to also be used when calculating the number of new *networks* created while subnetting.

However, this is **no longer a restriction** for modern equipment and routing protocols. Specifically, on Cisco IOS devices, the following command is now enabled by default:

```
Router(config)#  ip subnet-zero
```

The *ip subnet-zero* commands allows for the use of networks with all ***0*** or all ***1* bits** in the *stolen* network portion of the address. Thus, the formula for calculating the number of new networks created is simply $\mathbf{2^n}$.

Remember though, the formula for calculating usable ***hosts* is always $\mathbf{2^n - 2}$**.

(Reference: http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080093f18.shtml)

Determining the Range of Subnetted Networks

Determining the *range* of the newly created networks can be accomplished using several methods. The *long* method involves some binary magic.

Consider the example *192.168.254.0* network again, which was subnetted using a *255.255.255.240* mask:

192.168.254.0:	11000000.10101000.11111110.00000000
255.255.255.240:	11111111.11111111.11111111.11110000

Subnetting stole four bits in the fourth octet, creating a total of *16* new networks. Looking at *only* the fourth octet, the first newly created network is *0000*. The second new network is *0001*. Calculating all possible permutations of the four stolen bits:

Binary	***Decimal***	***Binary***	***Decimal***	***Binary***	***Decimal***
.0000 *xxxx*	.0	.0110 *xxxx*	.96	.1100 *xxxx*	.192
.0001 *xxxx*	.16	.0111 *xxxx*	.112	.1101 *xxxx*	.208
.0010 *xxxx*	.32	.1000 *xxxx*	.128	.1110 *xxxx*	.224
.0011 *xxxx*	.48	.1001 *xxxx*	.144	.1111 *xxxx*	.240
.0100 *xxxx*	.64	.1010 *xxxx*	.160		
.0101 *xxxx*	.80	.1011 *xxxx*	.176		

Note that this equates to exactly *16* new networks. The decimal value represents the first (or the *subnet*) address of each newly created network. To determine the range for the hosts of the *first* new network:

Binary	***Decimal***	***Binary***	***Decimal***	***Binary***	***Decimal***
.0000 0000	.0	.0000 0110	.6	.0000 1100	.12
.0000 0001	.1	.0000 0111	.7	.0000 1101	.13
.0000 0010	.2	.0000 1000	.8	.0000 1110	.14
.0000 0011	.3	.0000 1001	.9	.0000 1111	.15
.0000 0100	.4	.0000 1010	.10		
.0000 0101	.5	.0000 1011	.11		

The binary value has been split to emphasize the separation of the stolen *network* bits from the *host* bits. The first address has all *0* bits in the host portion (*0000*), and is the **subnet address** for this network. The last address has all *1* bits in the host portion, and thus is the **broadcast address** for this network. Note that there are exactly **14 usable addresses** to assign to hosts.

Determining the Range of Subnetted Networks (continued)

Calculating the ranges of subnetted networks can quickly become tedious when using the long binary method. The *shortcut* method involves taking the subnet mask (*255.255.255.240* from the previous example), and subtracting the subnetted octet (*240*) from *256*.

256 – 240 = 16

Assuming *ip subnet-zero* is enabled, the first network will begin at *0.* Then, simply continue adding *16* to identify the first address of each new network:

0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240

Knowing the *first* address of each new network makes it simple to determine the *last* address of each network:

First address of network	0	16	32	48	64	80	96	112	128	144
Last address of network	15	31	47	63	79	95	111	127	143	159

Only the first 10 networks were calculated, for brevity. The first address of each network becomes the **subnet address** for that network**.** The last address of each network becomes the **broadcast address** for that network**.**

Once the first and last address of each network is known, determining the usable range for hosts is straightforward:

Subnet address	0	16	32	48	64	80	96	112	128	144
	1	17	33	49	65	81	97	113	129	145
Usable Range	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
	14	30	46	62	78	94	110	126	142	158
Broadcast address	15	31	47	63	79	95	111	127	143	159

Hosts on the same network (such as *192.168.254.2* and *192.168.254.14*) can communicate freely.

Hosts on different networks (such as *192.168.254.61* and *192.168.254.66*) require a router to communicate.

Class A Subnetting Example

Consider the following subnetted Class A network: 10.0.0.0 255.255.248.0

Now consider the following questions:

- How many new networks were created?
- How many usable hosts are there per network?
- What is the full range of the first three networks?

By default, the *10.0.0.0* network has a subnet mask of *255.0.0.0*. To determine the number of bits stolen:

```
255.0.0.0:          11111111.00000000.00000000.00000000
255.255.248.0:      11111111.11111111.11111000.00000000
```

Clearly, **13 bits** have been stolen to create the new subnet mask. To calculate the total number of new networks:

$2^n \quad = \quad 2^{13} \quad =$ **8192 new networks created**

There are clearly **11 bits** remaining in the host portion of the mask:

$2^n - 2 \quad = \quad 2^{11} - 2 \quad = \quad 2048 - 2 \quad =$ **2046 usable hosts per network**

Calculating the ranges is a bit tricky. Using the shortcut method, subtract the third octet *(248)* of the subnet mask (*255.255.248.0*) from *256.*

256 – 248 = 8

The first network will begin at *0*, again. **However**, the ranges are spread across multiple octets. The ranges of the first three networks look as follows:

Subnet address	10.0.0.0	10.0.8.0	10.0.16.0
	10.0.0.1	10.0.8.1	10.0.16.1
Usable Range	↕	↕	↕
	10.0.7.254	10.0.15.254	10.0.23.254
Broadcast address	10.0.7.255	10.0.15.255	10.0.23.255

Private vs. Public IPv4 Addresses

The rapid growth of the Internet resulted in a shortage of available IPv4 addresses. In response, a specific subset of the IPv4 address space was designated as *private,* to temporarily alleviate this problem.

A **public address** can be routed on the Internet. Thus, hosts that must be Internet-accessible must be configured with (or *reachable* by) public addresses. Allocation of public addresses is governed by the Internet Assigned Numbers Authority (IANA).

A **private address** is intended for internal use within a home or organization, and can be freely used by anyone. However, private addresses can *never be routed* on the Internet. In fact, Internet routers are configured to immediately drop traffic with private addresses.

Three private address ranges were defined in RFC 1918, one for each IPv4 class:

- Class A - **10.x.x.x /8**
- Class B - **172.16.x.x /12**
- Class C - **192.168.x.x /24**

It is possible to *translate* between private and public addresses, using **Network Address Translation (NAT).** NAT allows a host configured with a private address to be *stamped* with a public address, thus allowing that host to communicate across the Internet. It is also possible to translate multiple privately-addressed hosts to a single public address, which conserves the public address space.

NAT provides an additional benefit – hiding the specific addresses and addressing structure of the internal (or *private*) network.

Note: NAT is *not* restricted to private-to-public address translation, though that is the most common application. NAT can also perform public-to-public address translation, as well as private-to-private address translation.

NAT is only a temporarily solution to the address shortage problem. IPv4 will eventually be replaced with IPv6, which supports a vast address space.

Both NAT and IPv6 are covered extensively in other guides.

Reserved IPv4 Addresses

In addition to the three private IPv4 ranges, several other addresses and ranges are reserved for specific purposes:

- The **0.0.0.0 /0** network is used to identify **all networks,** and is referred to as the **default route.** If a default route exists in a routing table, it will be used only if there is *not* a more *specific* route to a particular destination. Routing and default routes are covered extensively in another guide.

- The **0.0.0.0 /8** range is used to identify hosts on the *local* network. Addresses in this range can only be used as a *source* address. The most commonly used address in this range is **0.0.0.0 /32,** which a host will use when dynamically attempting to learn its IP address via Dynamic Host Configuration Protocol (DHCP). DHCP is covered extensively in another guide.

- The entire **127.x.x.x /8** range is reserved for diagnostic purposes. The most commonly used address in this range is **127.0.0.1**, which identifies the local host, and is referred to as the **loopback** or **localhost** address.

- The **169.254.x.x /16** range is reserved for Automatic Private IP Addressing (APIPA). A host assigns itself an address in this range, if it cannot dynamically obtain an address from a DHCP server.

- The **224.x.x.x – 239.x.x.x** ranges are reserved for **multicast**, and are referred to as **Class D** addresses.

- The **240.x.x.x – 255.x.x.x** ranges are reserved for future and experimental use, and were formerly referred to as **Class E** addresses.

- The **255.255.255.255** address can be used as a broadcast address for the local network.

The IPv4 Header

The IPv4 header is comprised of **12 required fields** and **1 optional field.** The minimum length of the header is **160 bits (20 bytes).**

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Decremented by each router traversed*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

The 4-bit **Version field** is set to a value of *4* for IPv4.

The 4-bit **Internet Header Length field** identifies the length of the IPv4 header, measured in 32-bit *words*. The minimum of length of an IPv4 header is 160 bits, or 5 words (32 x 5 = 160).

The 8-bit **Differentiated Service Code Point (DSCP) field** is used to classify traffic for Quality of Service (QoS) purposes. QoS is covered in great detail in other guides. This field was previously referred to as the Type of Service (ToS) field.

The 16-bit **Total Length field** identifies the total packet size, measured in *bytes,* including both the IPv4 header and the data payload. The minimum size of an IPv4 packet is 20 bytes – essentially a header with no payload. The maximum packet size is **65,535 bytes**.

The IPv4 Header (continued)

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Limits the lifetime of a packet*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

An IPv4 packet that is larger than the **Maximum Transmission Unit (MTU)** size of a link must be **fragmented**. By default, the MTU for Ethernet is **1500 bytes.**

Three fields are used when a packet must be fragmented - the 16-bit **Identification field**, the 3-bit **Flags field**, and the 13-bit **Fragment Offset field**. Each fragment of the packet is marked with the same *Identification* number. The *Fragment Offset* allows the destination host to reassemble the fragments in the proper order.

The *Flags* field dictates two conditions:

- **Don't Fragment (DF)** – indicates the packet cannot be fragmented. If a packet exceeds a link's MTU size and this flag is set, then the packet is dropped. An ICMP error message will then be sent back to the source host.

- **More Fragments (MF)** – all fragments have this bit set to one, *except* for the last fragment, where the bit is set to zero. This allows the destination host to know when it has received all fragments.

The IPv4 Header (continued)

Field	*Length*	*Description*
Version	4 bits	*Version of IP (in this case, IPv4)*
Internet Header Length	4 bits	*Specifies the length of the IP header (minimum 160 bits)*
DSCP	8 bits	*Classifies traffic for QoS*
Total Length	16 bits	*Specifies the length of both the header and data payload*
Identification	16 bits	*Uniquely identifies fragments of a packet*
Flags	3 bits	*Flags for fragmentation*
Fragment Offset	13 bits	*Identifies the fragment relative to the start of the packet*
Time to Live	8 bits	*Limits the lifetime of a packet*
Protocol	8 bits	*Specifies the next upper layer protocol*
Header Checksum	16 bits	*Checksum for error checking*
Source Address	32 bits	*Source IPv4 address*
Destination Address	32 bits	*Destination IPv4 address*
Options	Variable	*Optional field for various parameters*

The 8-bit **Time to Live (TTL) field** limits the lifetime of the packet, preventing it from being endlessly forwarded. When a router forwards a packet, it will decrement the TTL value by one. Once the TTL value reaches zero, the packet is dropped.

The 8-bit **Protocol field** identifies the next upper-layer header, and is covered in the next section.

The 16-bit **Header Checksum field** is used to error-check the IPv4 header. The receiving host will discard the packet if it fails the checksum calculation.

The 32-bit **Source Address field** identifies the *sending* host. The 32-bit **Destination Address field** identifies the *receiving* host. The value of both of these fields can be changed as the packet is forwarded, using NAT.

The variable-length **Options field** provides additional optional IPv4 parameters, outside the scope of this guide.

IPv4 Protocol Numbers

The 8-bit **Protocol field** specifies the next upper-layer header within the data payload of the packet. These upper-layer protocols are identified using **IP Protocol Numbers**.

The following is a list of common IP Protocol Numbers, as assigned by the IANA:

Protocol Number	**Upper-Layer Protocol**
1	ICMP
2	IGMP
6	TCP
9	IGRP
17	UDP
46	RSVP
47	GRE
50	IPSEC ESP
51	IPSEC AH
88	EIGRP
89	OSPF

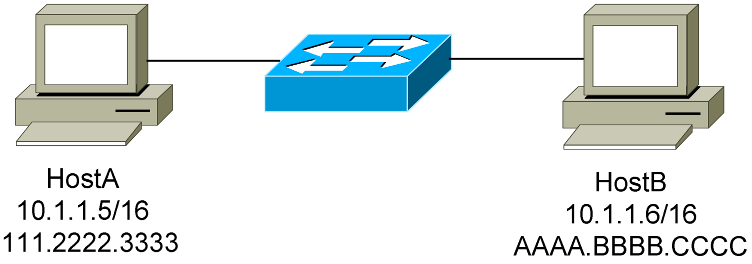
In IPv6, this field is referred to as the **Next Header field.**

(Reference: http://www.iana.org/assignments/protocol-numbers)

Resolving Logical Addresses to Hardware Addresses

A host cannot directly send data to another host's logical address. A destination logical address must be *mapped* to a hardware address, so that the Data-Link layer can package a frame to transmit on the physical medium.

The **Address Resolution Protocol (ARP)** provides this mechanism for IPv4 on Ethernet networks. ARP allows a host to determine the MAC address for a particular destination IP address.



Consider the above diagram. The following demonstrates the steps required for HostA to communicate with HostB:

- First, HostA will determine if the destination IP address of 10.1.1.6 is *itself*. If that address is configured on a local interface, the packet never leaves HostA. In this example, 10.1.1.6 is *not* locally configured on HostA.

- Next, HostA will determine if the 10.1.1.6 address is on the *same network* or *subnet* as itself. HostA consults its **local routing table** to make this determination. In this example, the subnet mask is */16*. Thus, HostA's IP address of 10.1.1.5 and the destination address of 10.1.1.6 are on the same network (*10.1*).

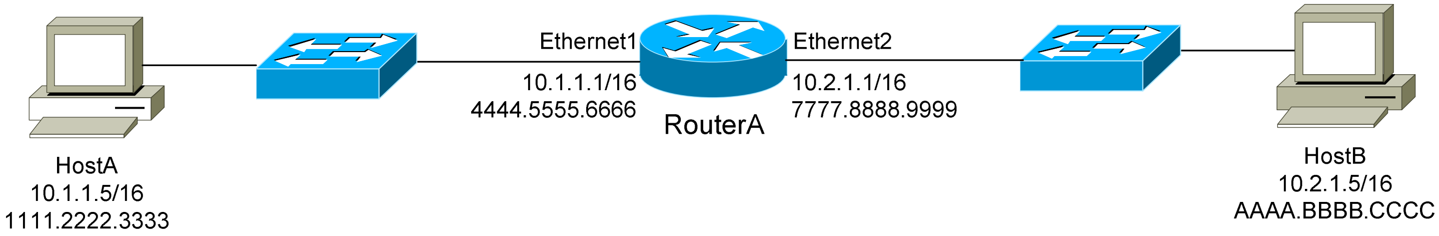
- Because HostA and HostB are on the same network, HostA will then broadcast an **ARP request**, asking for the MAC address of the 10.1.1.6 address.

- HostB responds to the ARP request with an **ARP reply,** containing its MAC address (AAAA.BBBB.CCCC).

- HostA can now construct a Layer-2 frame, with a destination of HostB's MAC address. HostA forwards this frame to the switch, which then forwards the frame to HostB.

Resolving Logical Addresses to Hardware Addresses (continued)

Now consider a slightly modified scenario between HostA and HostB:



- Again, HostA will determine if the destination IP address of 10.2.1.5 is *itself*. In this example, 10.2.1.5 is *not* locally configured on HostA.

- Next, HostA will determine if the 10.2.1.5 address is on the *same network* or *subnet* as itself. In this example, the subnet mask is */16.* Thus, HostA's IP address of 10.1.1.5 and the destination address of 10.2.1.5 are ***not*** on the same network.

- Because HostA and HostB are *not* on the same network, HostA will parse its local routing table for a route to this destination network of 10.2.x.x/16. Hosts are commonly configured with a **default gateway** to reach all other destination networks.

- HostA determines that the 10.1.1.1 address on RouterA is its default gateway. HostA will then broadcast an ARP request, asking for the MAC address of the 10.1.1.1 address.

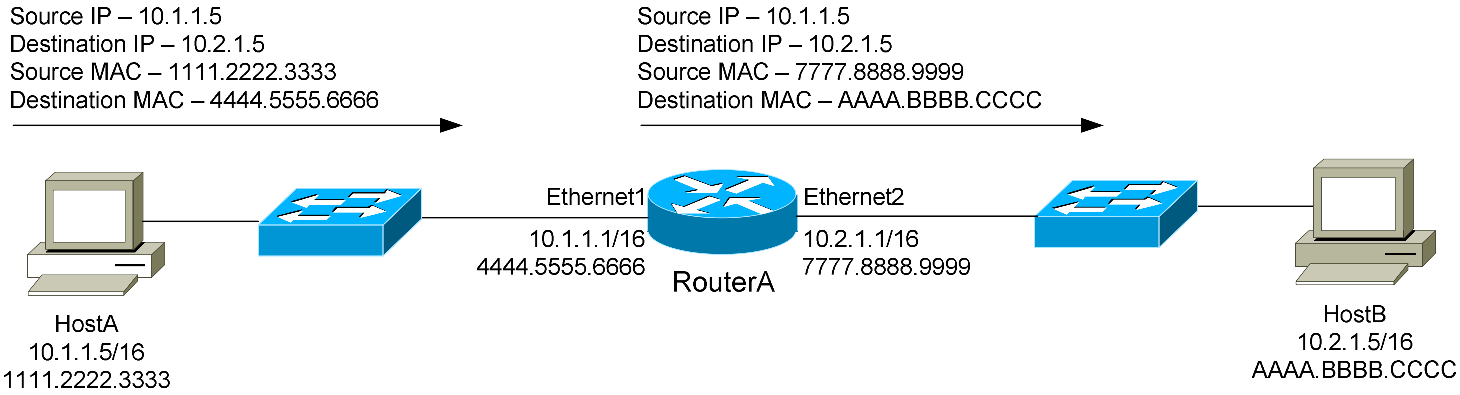
- RouterA responds to the ARP request with an ARP reply containing its MAC address (4444.5555.6666). HostA can now construct a Layer-2 frame, with a destination of RouterA's MAC address.

- Once RouterA receives the frame, it will parse its own routing table for a route to the destination network of 10.2.x.x/16. It determines that this network is directly attached off of its *Ethernet2* interface. RouterA then broadcasts an ARP request for the 10.2.1.5 address.

- HostB responds to the ARP request with an ARP reply containing its MAC address (AAAA.BBBB.CCCC). RouterA can now construct a Layer-2 frame, with a destination of HostB's MAC address.

Resolving Logical Addresses to Hardware Addresses (continued)

Consider the following example again:



Note that as a packet is *routed,* the source and destination IP address remain unchanged. However, both the source and destination MAC address *did* change.

This is because a MAC address contains no network hierarchy, and thus is only significant on the *local* network. In the above scenario, HostA and HostB could not communicate directly using Layer-2 addressing. At every routed *hop,* the source and destination MAC address are adjusted to reflect the source and destination hosts on the *local* network.

The source and destination IP address will *only* be changed if NAT is used.

The ARP Table

A host can build an **ARP table** that contains a list of IP to MAC address translations. The ARP table is only locally significant to that host. There are two methods to populate an ARP table:

- **Statically**
- **Dynamically**

A **static ARP entry** is created manually on a host, and will remain permanently until purposely removed. More commonly, ARP tables are built **dynamically** by caching ARP replies. Cached entries will eventually be aged out of the ARP table. The aging time will vary depending on the operating system, and can range from several seconds to several hours.

Troubleshooting IP using ICMP

The **Internet Control Message Protocol (ICMP)** is used for a multitude of informational and error messaging purposes.

The following is a list of common ICMP types and codes:

Type	*Code*	*Description*
0	**0**	**Echo Reply**
3	**-**	**Destination Unreachable**
	0	*Network Unreachable*
	1	*Host Unreachable*
	2	*Protocol Unreachable*
	3	*Port Unreachable*
	4	*Fragmentation Needed – Don't Fragment Flag Set*
	6	*Destination Network Unknown*
	7	*Destination Host Unknown*
	9	*Destination Network Administratively Prohibited*
	10	*Destination Host Administratively Prohibited*
5		**Redirect**
8		**Echo**
11		**TTL Exceeded**

The two most common troubleshooting tools that utilize ICMP are:

- **Packet Internet Groper (ping)**
- **Traceroute**

Ping is a core connectivity troubleshooting tool, which utilizes the Echo Request and Echo Reply ICMP messages to determine if an IP address is reachable and responding. Ping will additionally provide the **round-trip time** between the source and destination, usually measured in milliseconds.

Traceroute determines the routing path a packet takes to reach its destination. Traceroute will not only identify each router the packet has been forwarded through, but will also measure the delay experienced at each router *hop*.

* * *