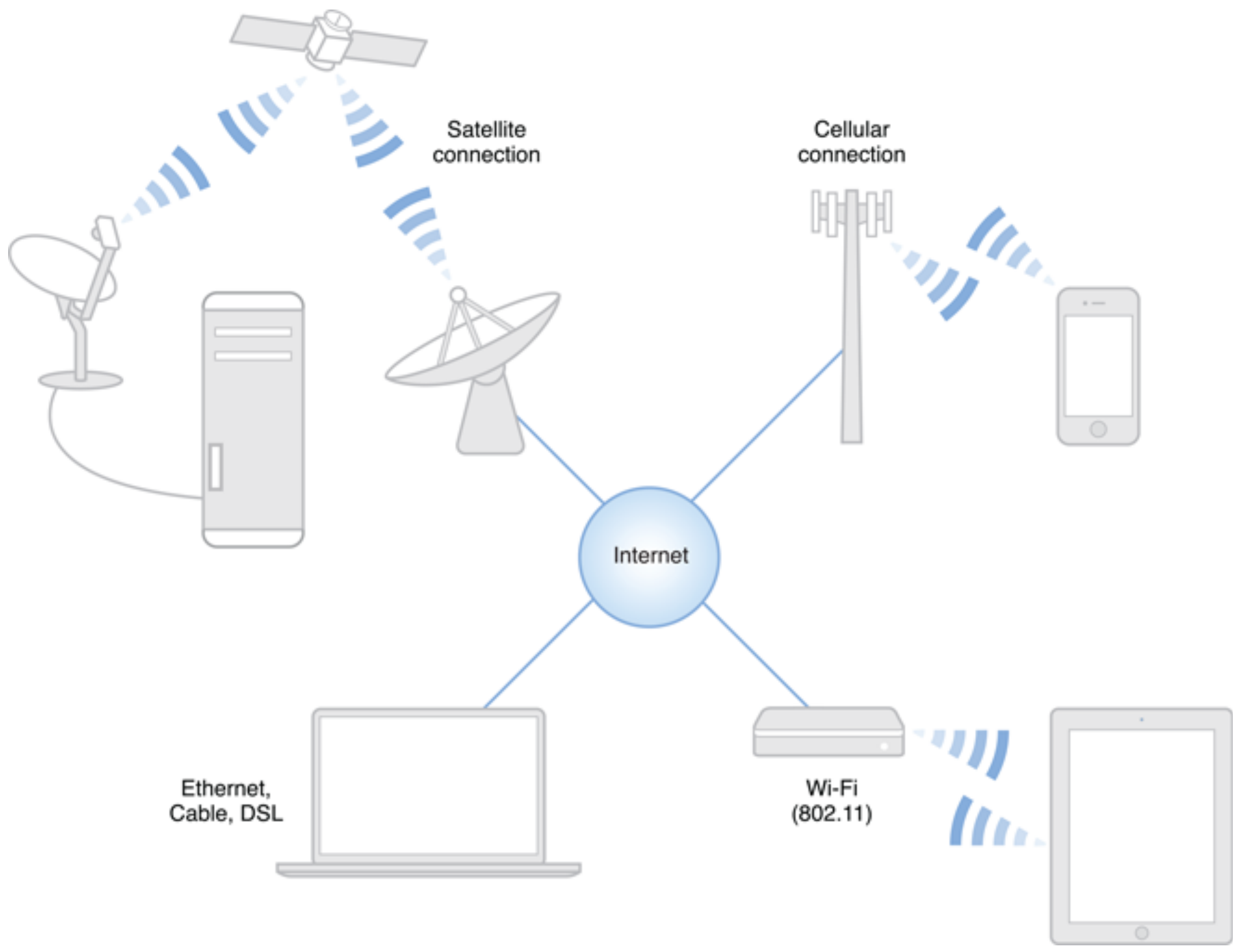
# Introduction

**Important:** This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you adopt the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be vetted against final documentation. For information about updates to this document, go to the Apple Developer website. In the relevant reference library, enter the document's title in the Documents text field that appears.

Because most networking in OS X and iOS is based on the TCP/IP communication protocol, you should have at least a basic understanding of the fundamentals of the TCP/IP networking model itself before you begin writing code. Although you can perform some high-level networking tasks without learning about the underlying protocols, knowing how things work at a low level will help you better understand why things go wrong and how to handle them when they do.

## At a Glance

The Internet is a vast network of interconnected computers and other devices that provide communication around the world. When you visit a website, the server that provides that content might be across the room or across an ocean. Your request might pass through a Wi-Fi router, a cable modem, a trunk line, a satellite connection, or ostensibly even be delivered by carrier pigeon (RFC 1149). At a high level, each of these networking media are equivalent (performance characteristics notwithstanding). However, at a lower level, devices communicate with one another in different ways, depending on what type of physical network they are communicating over.



At a low level, when you access a website, your computer breaks up each request into tiny pieces, called packets, and sends each packet individually to a special device called a router, which forwards those packets to another router, which in turn forwards them to yet another router, and so on, until the packets reach the destination server, wherever it might be. The response is sent back in a similar fashion. At each step along the way, those packets can be split into multiple packets, wrapped up in other types of packets, and so on, depending on the hardware involved.

This document explains how the Internet works under the hood, but at a moderately high level intended for programmers. If, after reading this document, you want to learn more about how various Internet technologies work, you can find a number of third party books that describe them in more detail. In addition, most of the lower-level Internet technologies are described by RFCs (short for “Request for Comment”) that describe the protocols in great detail.

## See Also

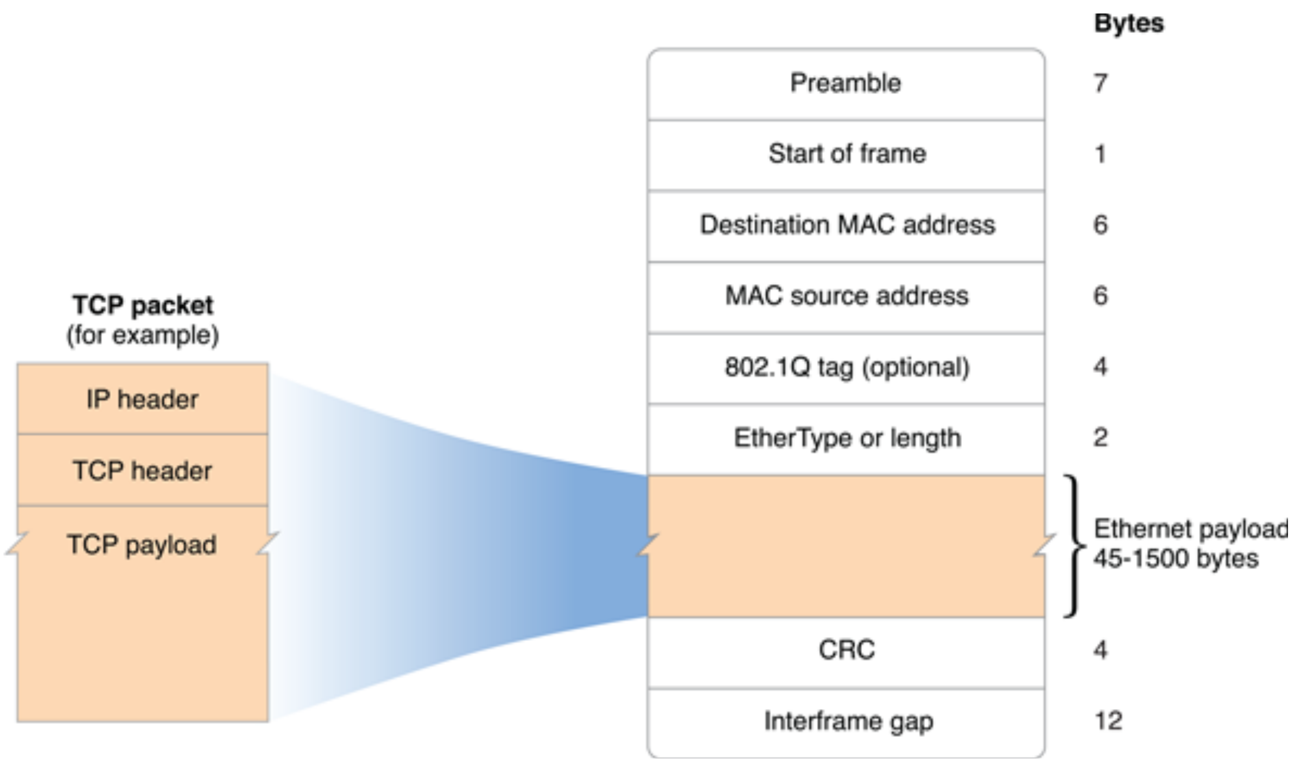
This document is a companion document for Networking Overview, which explains how to write good networking code.

Next

Next
Previous

# Networking Terminology

In networking terminology, a host is any device that is connected to a network and provides an endpoint for networked communication. A host might be a desktop computer, a server, an iOS device, a virtual machine running on a server, or even the VoIP telephone sitting on your desk. It is called a host because it hosts the applications and daemons that run on it.
Similarly, an infrastructure device is any piece of equipment that is responsible for making the network function. The difference between a host and an infrastructure device is that an infrastructure device usually passes on information sent by a host, whereas a host primarily sends or receives information on its own behalf. There are even some infrastructure devices that are completely transparent to TCP/IP networking, such as Ethernet hubs and switches (more on these later).
When one host sends data across a network, it divides the data into small pieces called packets. These packets can be varying lengths, up to the maximum size allowed by the physical network interconnect (again, more on this later).
A packet generally contains three basic parts: a header that tells where the packet should be sent, a payload that contains the actual data, and a trailer that contains checksum information to ensure that the packet was received correctly. Some packet types include this checksum information as part of the header, and thus do not have a trailer. For example, shows the structure of an Ethernet packet.
**Figure 1-1** Structure of an Ethernet packet



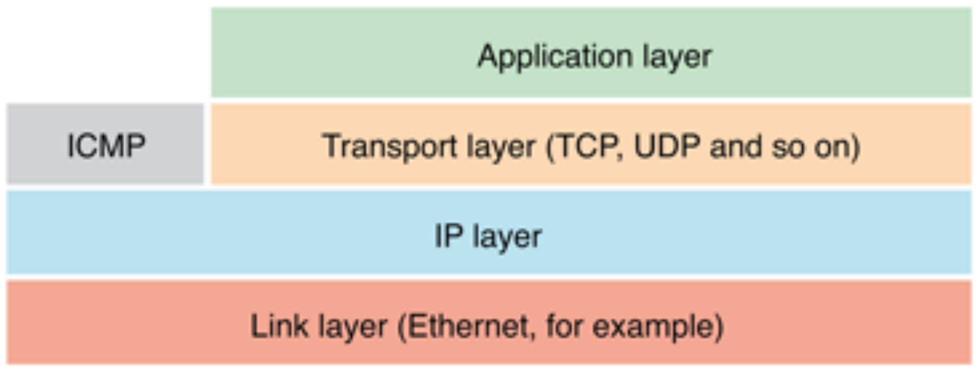
The receiving host then reassembles these packets and provides them to a program as either a stream of bytes or a series of messages, depending on the protocol. That program can respond by sending data, which its host then divides into packets and sends back to the first host.
When one packet contains another packet (generally of a different type), this is called encapsulation. For example, the Ethernet packet shown in Figure 1-1 contains a TCP/IP packet as its payload; the TCP/IP packet is said to be encapsulated within the Ethernet packet.

Next
Previous

Next
Previous

# Networking Layers

The TCP/IP networking model consists of four basic layers: the link layer, the IP layer (short for Internet protocol), the transport layer, and the application layer. These layers are described in the following sections.


Application layer
ICMP
Transport layer (TCP, UDP and so on)
IP layer
Link layer (Ethernet, for example)

## The Link Layer

The bottommost layer is the link layer, or physical layer. This layer of the networking stack involves the actual hardware used to communicate with nearby physically connected hosts. The link layer transports raw packets from one host to another on the same physical network.

Most end users encounter only five link layers directly: Wi-Fi, cellular networking, Ethernet, Bluetooth, and FireWire. However, their data passes through links of many different types on its way to its destination. A user might send data over a cable modem based on DOCSIS or a DSL modem based on VHDSL. From there, the user's data might pass across a SONET/SDH link (used for trunk lines) or a satellite connection using IPoS (IP over satellite) or IPoDVB (IP over digital video broadcast).

A network interface is a piece of hardware that provides a link-layer interconnect. A host can have many network interfaces. For example, you might have one or more (wired) Ethernet interfaces, a Wi-Fi interface, a Bluetooth interface, or even a cellular modem (whether attached by USB to a Mac or built into an iPhone or iPad). In addition, certain pieces of software provide (virtual) network interfaces—VPN software, virtualization environments (like the Classic environment in OS X 10.4 and earlier), and so on.

Each network interface is generally connected to one or more additional interfaces; the connection between them is called a link. Although these interfaces are not always physically connected to one another, a link appears to be equivalent to a physical wire as far as the operating system and software are concerned. For example, two computers connected by an Ethernet hub or switch have a link between them (because the hub and switch are transparent), but two computers separated by a network router do not (because the operating system has to know to send data to the router instead of sending it directly to that host—more on routing later).

When writing software, unless your software interacts with the link layer directly (by sending and receiving raw packets, configuring a network interface, reading the hardware ID of a network interface, and so on), the operating system largely hides the differences between link layers from you (although it cannot fully isolate you from differences in bandwidth, latency, or reliability).

## The IP Layer

Sitting on top of the link layer is the IP layer. The IP layer provides packet transport from one host to another in such a way that the packets can pass across multiple physical networks.

The IP layer adds the notion of routing to the picture, which lets you send a packet to a distant destination. Briefly put, the packets from a host travel to a nearby router across some physical link, which routes the packet to another router, which routes it to another router, and so on, until it reaches its destination. Reply packets travel back in much the same way. The path your packets take is called a route, and each link that the packets follow from one router to another along the route is called a hop.

The IP layer also helps abstract away differences in the link layer. In particular, different link layers support packets with different maximum sizes—referred to as the maximum transmission unit (MTU). To hide this difference, the IP layer splits packets into multiple pieces—a process known as fragmentation—and reassembles them at the other end. (Note that fragmentation applies only to taking a packet and splitting it; in the case of TCP/IP, the sending host packetizes TCP, but because it did not start out as discrete packets, this is not considered fragmentation.)

On most networks, the amount of time a network takes to deliver a single packet is independent of the packet's size up to the MTU. For example, on an Ethernet network, it takes the same time slice whether you send a packet that is 100 bytes or 1500 bytes in length. For this reason, repeatedly fragmenting a packet results in a hefty overhead.

For example, if you send a 1500-byte packet through a network with a 1400-byte MTU on its way to a third network with an MTU of 1300 (these numbers are arbitrary), your 1500-byte packet would first get broken up into a 1400-byte packet and a 100-byte packet. Later, the 1400 byte packet would get broken up into a 1300-byte packet and a second 100-byte packet, for a total of three packets (1300 + 100 + 100). If you instead sent a 1300-byte packet and a 200-byte packet, your data would take one-third less bandwidth on the final network.

Fragmentation also has another cost. If a fragment of a packet is lost, the entire packet is lost. This means that on a network with high packet loss, fragmentation can increase the rate of retransmission required (which, in turn, can make the packet loss worse).

To avoid this problem, when performing TCP communication, most modern operating systems use a technique called path MTU discovery to determine the largest packet that can pass from one host to another without fragmentation, thus maximizing the use of available bandwidth. If you are writing code that sends and receives raw packets, you should consider taking similar steps. Where possible, however, you should use TCP/IP, which does this for you.

## Transport Layer

On top of the IP layer, you'll find several transport layers. When writing networking code, you will almost invariably be working with one of these transport layers or with a higher-level layer built on top of them.
The two most common protocols at this layer are the transmission control protocol (TCP) and the user datagram protocol (UDP).
Both TCP and UDP provide basic data transport from one host to another, much like IP, but add the notion of port numbers. These port numbers let you run multiple services on a single host by providing a way to tell the receiving host which service should receive that particular message.
Like the layers below it, UDP provides no guarantee that the data will ever reach its destination. If your program uses UDP, you must handle any retransmission yourself, if necessary. UDP may be a good choice for situations where low latency is required, such as real-time game communication, because if the client state update isn't available within a few milliseconds, the data is no longer useful anyway. However, as a rule, you should generally avoid UDP unless you have to support an existing protocol that uses it.
**Note:** Because UDP packets do not block waiting for confirmation of previous packets, you can cause serious network congestion if you use UDP improperly. If you need to use UDP, be sure to stop sending packets if you lose communication with the other endpoint for a period of time. If you are transmitting high-bandwidth data (such as a real-time audio or video stream), be sure to design your protocol so that each endpoint can determine the number of packets that it failed to receive. Also, be sure that the two endpoints scale down their transmission rate automatically when congestion occurs, and implement path MTU discovery where possible.

Unlike UDP, TCP adds:

- Delivery guarantees—Data transmitted using TCP is guaranteed to be received in the order in which it was sent and (connection failures notwithstanding) in its entirety. If data cannot be transmitted, the connection is torn down after a long timeout. However, all data received up to that point in time is guaranteed to be in order and without missing fragments in the middle.
- Congestion control—Sending hosts back off the speed of transmission (and retransmission) if data is getting dropped along the way due to an over-utilized link.
- Flow control—When busy, receiving hosts tell sending hosts to wait until they are ready to handle more data.

- Stream-based data flow—Your software sees the data as a series of bytes instead of as a series of discrete records (messages, in UDP parlance). Note that if you want to use TCP to send a series of discrete records, you must encode the record boundaries yourself (using MIME multipart message encoding, for example).
- Path MTU discovery—TCP chooses the largest packet size that avoids fragmentation en route.

By contrast, UDP supports three things that TCP does not:

- Broadcast messages in IPv4—packets sent to a broadcast address are received by every host within its broadcast domain (which is usually its subnet).
- Multicast messages—UDP packets sent to a multicast address are sent out to any host that subscribes to them; those hosts may be on the local area network (LAN) or may be behind a router, but in practice are usually limited to the LAN.
- Preservation of record (packet) boundaries. With UDP, the receiver sees each message individually instead of as a continuous stream of bytes.

TCP and UDP also depend on another protocol, the Internet control message protocol (ICMP) that sits on top of the IP layer. ICMP packets are used for reporting connection failures (destination unreachable, for example) back to the connecting or sending host. Although TCP and UDP sockets can connect successfully without ICMP, the ability to detect connection failures is significantly reduced.

In addition to providing support for TCP and UDP, ICMP echo packets form the basis for the `a target="_self" ping/a` tool, which is commonly used to diagnose network problems. When the destination host (or router) receives an ICMP echo packet, it echoes the packet back to the sender, thus giving you some idea of the rate of packet loss between two hosts.

**Note:** There are a number of other protocols that sit on top of the IP layer (RSVP, IGMP, and IPSEC, for example), but these protocols are more specialized and are thus outside the scope of this overview. Many of these protocols are blocked by firewalls (routers that limit what types of traffic can pass).

On top of these transport layer protocols, you can optionally have encryption layers such as TLS (transport layer security) on top of TCP or DTLS over UDP.

Similarly, various encapsulation schemes are built on top of transport layer protocols—the L2TP and PPTP protocols used by many VPNs, for example. These encapsulation layers appear to your program as a separate network interface (because they provide a virtual link layer). Under the hood, however, traffic over that interface looks like any other TCP/IP stream to a single host (the VPN server).

## The Application Layer

The application layer sits at the top of the protocol stack. This layer includes such protocols as hypertext transfer protocol (HTTP) and file transfer protocol (FTP).
The application layer is so named because the details of this layer are specific to a particular application or class of applications. In other words, this is the layer that is under your program's direct control, whether you are implementing your own application-layer protocols or are merely asking the operating system to do so on your behalf.
Next
Previous

Next
Previous
# Understanding Latency

Latency refers to the round-trip time for a request—specifically, the time for the initial packet to reach its destination, for the destination machine to reply, and for that reply to reach the requestor. Every network has latency. The total amount of latency that you'll get when connecting to a given remote host can vary widely, depending on network conditions.
Assuming your network connection is not overloaded, the bulk of a connection's latency comes from the laws of physics. The minimum latency between two points on the earth can be calculated by dividing the distance by the speed at which light or electricity moves in a particular medium (which is usually a large fraction of the speed of light in a vacuum, c).
For example, consider a packet traveling round trip from New York to San Francisco (about 2,900 miles, or 4,670 km):

- Over copper wire, the data moves somewhere between .66c and c (depending on the type of wire). Thus, a packet takes at minimum 15-24 ms each way, or about 30-48 ms round trip. This delay is barely noticeable.
- Over an optical fiber, the data moves at about 0.65c. Thus, a packet takes at minimum 24 ms each way, or about 48 ms round trip. This delay is also barely noticeable.
- Over a satellite connection, the packet must go to geostationary orbit (at an altitude of 35,786 km) and back down again. For a round trip, it must do this twice. Thus, at approximately c, the minimum round-trip

latency is about 477 milliseconds, or almost half a second. This delay is painfully noticeable.

These calculations represent an absolute lower bound for the latency of a connection through those media. There are several other factors that can add additional latency on top of the link latency:

- Routing delays. A network packet can be further delayed by buffering at routers along its route. Whenever the rate of data exceeds the capacity of a particular network hop, the packets must be delayed until they can be sent. If your packets must travel through a highly congested network hop, this buffer delay can add considerable latency.For example, if a particular network hop is limited to 100 packets per second, a router can send only one packet down that wire every 10 milliseconds. Thus, when a router receives an additional packet to send down that wire, it must buffer the packet until the next free slot. If there are no other packets waiting to be sent, then the packet will be sent in ten milliseconds or less (five, on average). If there are already three packets waiting, then the packet will be delayed by an additional 30 milliseconds. Latency caused by time slots can be particularly noticeable in some types of cellular communications (EDGE, for example).
- Retransmission and exponential backoff. Whenever a host sends a TCP packet, it waits a while for the other end to acknowledge receipt of the packet. If it does not receive that acknowledgement after a period of time, the sender retransmits it. As the number of failures increases, the sender increases the retransmission delay exponentially, under the assumption that the packet loss was probably caused by a saturated link en route to the destination. Thus, a network connection with high packet loss (more than 1–2 percent) can significantly reduce performance.
- Signal propagation delays within hardware that receives, transmits, forwards, or repeats packets. For example, an Ethernet switch must receive an entire packet before it can begin sending the packet to its destination. Although a single switch or repeater adds only a small delay, those delays can add up over long distances. For example, early fiber optic cable systems required repeaters every 10 km (at most). Thus, an older fiber run from New York to San Francisco could easily have nearly 500 repeaters.

Next
Previous

Next
Previous

# Addressing Schemes and Domain Names

At every level of networking, each host is assigned one or more numeric identifiers that uniquely represent it within a particular network (though they are not always globally unique). The exact nature of those identifiers varies depending on whether you are working with networking at a high level (IP, for example) or a low level (raw Ethernet packets, for example).

## Link-Layer Addressing

At the link layer (physical layer), each network interface is usually identified by a globally unique hardware ID:

- Ethernet—MAC address (short for Media Access Control)
- Wi-Fi—MAC address (short for Media Access Control)
- Bluetooth—Bluetooth hardware ID (similar to a MAC address, but it does not share the same namespace)
- GSM Cellular—IMEI (International Mobile Equipment Identity)
- CDMA Cellular—ESN (Electronic Serial Number) or MEID (Mobile Equipment Identifier)

Because the identifier uniquely identifies a single network interface, you should never encounter two physical devices that share the same hardware address on the same physical network. More importantly, if a single host has more than one network interface of the same type, each interface has a different hardware ID.

The hardware ID is used to determine whether a particular device should listen to a packet or ignore it. Some physical networks also limit which packets get sent down a particular wire, based on whether an interface matching that hardware ID was seen on that wire previously. These networks are referred to as switched networks. Networks in which every host sees every packet, by contrast, are called shared networks.

**Note:** Most programs do not need to use link-layer addresses directly.

## IP-Layer Addressing

At the IP layer and above, hosts are identified by an IP address (which is also sometimes called an IP number). An IP address can be in one of two forms: IPv4 or IPv6.

An IPv4 address consists of four bytes, and is usually represented to the user as a series of four numbers separated by decimal points. For example, the loopback address that always sends data back to the sending machine is `127.0.0.1`.

Because the IPv4 address space is limited in size (and rapidly running out), the IPv6 standard was created. This standard greatly expands the number of available addresses.

An IPv6 address is a 128-bit value, and is usually written as eight groups of 16-bit hexadecimal numbers separated by colons. Leading zeros in each group can be omitted as long as there is at least one digit in each group. For example, the IPv6 loopback address, `0000:0000:0000:0000:0000:0000:0000:0001`, can be simplified to `0:0:0:0:0:0:0:1`.

Also, if an IPv6 address contains several groups of zeros in a row, you can omit those groups and replace them with a double colon. You can do this only once per address (because otherwise the address would be ambiguous). Thus, the IPv6 loopback address, `0:0:0:0:0:0:0:1`, can be further simplified to `::1` (by replacing all the leading zero groups with a double colon).

## Domain Name System (DNS)

IPv4 addresses are hard to remember, and IPv6 addresses are quadruply hard (because they are four times as long). To make it easier to describe a particular host, the domain name system (DNS) was invented. Besides being easier to remember, domain names have several other advantages over IP addresses. Domain names:

- Minimize service disruption when an IP address changes. The IP addresses of a given host can change over time—when a mobile phone goes out of Wi-Fi range, when a user reboots his or her cable modem at home, when the owner of a server changes ISPs, and so on. By updating the domain name record to point to the server’s new address, users can continue to access the server by the same name.
- Allow a host to be accessed by more than one address. To improve robustness and performance, a server might have multiple IPv4 and IPv6 addresses on multiple physical network connections. If they all share the same domain name, clients can connect to any of its addresses. And when new addresses are added, clients can automatically learn about them.
- Allow multiple physical hosts to pretend to be a single host. For example, a single domain name might be serviced by a dozen servers that are geographically distributed around the world to improve performance.
- Can adapt to changes in the underlying technology. For example, an app that connects to a host by name does not need to care whether that host is reachable by IPv4 or IPv6. It asks for the address, and the

domain name resolver returns an appropriate data structure for connecting to that host.

This section describes the parts of a domain name and the DNS lookup process at a high level.

## What’s in a (Domain) Name?

A domain name is a human-readable name that describes a particular host. Each domain name is made up of a series of parts separated by periods. The easiest way to explain these parts is by example. Consider the hostname `mail.example.com`:

- `com`—This part of the domain name is outside your administrative control. This may have multiple parts, but at a minimum, it contains a top-level domain (TLD). These top-level domains fall into one of the following categories:
    - A generic TLD such as `.com`, `.org`, or `.net`
    - A restricted TLD such as `.biz`, `.edu`, or `.gov`
    - A sponsored TLD such as `.mobi`, `.museum`, or `.travel`
    - A two-letter country code such as `.us`
- When the top-level domain contains a country code, the portion of the domain name outside your control may have multiple parts. For example, a business in the UK generally has an address in the `.co.uk` namespace.Each of the top-level domains, in turn, is part of the dot (`.`) root domain. In common usage, the root domain does not contribute any characters to the domain name itself. Under the hood, however, every fully qualified domain name ends in a period. For example, the name `apple.com.` can resolve to exactly one domain. The name `apple.com` typically resolves to `apple.com.`, but if that lookup fails, it could also resolve to `apple.com.example.com` if your computer is configured with `example.com` as its default search domain.
- `example`—The domain part. This part is under the administrative control of your company or organization, and applies to the company or organization as a whole.
- `mail`—The host part. This part allows you to uniquely identify multiple servers in a single domain. In addition, the domain name itself (with no host part) is also a valid hostname—apple.com, for example.

Domains can also contain any number of subdomains. For example, `www.david.example.com` would be the `www` host in the `david` subdomain of the `example.com` domain. For that matter, even the `example.com` domain is a subdomain of the `com` domain, which is itself a subdomain of the root domain. Every domain (except the root domain) is a subdomain of another domain.

Be aware that in the context of end-user discussions, the term domain is often used to refer to the largest part of the domain hierarchy that someone who wants to set up a service can actually buy or otherwise request. Thus, in the `com` hierarchy, a domain would contain two parts—`apple.com`, for example. Similarly, a domain in the `au` hierarchy would include at least three parts—`apple.com.au`, for example. It is not strictly correct to limit the use of these terms in this way, but this usage is not uncommon.
In general, domain names must be in ASCII (although there is a standard for internationalized domain names) and can contain only letters, numbers, and hyphens. Domain names are case-insensitive. For more information about internationalized domain names, see http://www.icann.org/en/resources/idn.

## Looking Up Names

Before a computer or other device can contact the host associated with a domain name, it must first look up the name to obtain the corresponding IP addresses. To do this, it contacts a DNS server. After the server returns one or more IP addresses, the computer or device can then connect to the remote host using any of those addresses.
Because a number of different agencies in different countries work together to manage the domain name system, the process for looking up addresses is somewhat complicated. A typical domain lookup includes the following steps:

1. Your computer or other device sends a query to a DNS server.
2. Your local DNS server sends a query to one of several central servers (called the root servers), asking what server is authoritative for the domain.
3. The root servers generally delegate responsibility for the lookup. Instead of providing a response with the IP address for `www.david.example.com`, the root servers instead tell you to ask another server at a specific IP address.
4. Your local DNS server sends a query to that TLD server, asking what server is authoritative for the domain.
5. The TLD server consults a database to see which servers answer requests for the specified domain.
6. The TLD server delegates responsibility for the lookup to the authoritative servers for the domain (and provides their IP addresses).
7. Your local DNS server sends a query to one or more of the servers provided by the TLD. That query asks for the host's IP addresses.
8. That server either returns one or more IP addresses or further delegates responsibility to another server.

For example, if you are looking up a host in the `com` top-level domain, your computer makes two requests: one asking for a list of IPv4 addresses and one asking for a list of IPv6 addresses. The root server then delegates

responsibility to a gTLD (generic top-level domain) server, which in turn delegates responsibility to a server that is authoritative for the domain. That authoritative server may either return the answer or further delegate responsibility to another server for a specific subdomain.
This process of asking other servers to provide information is called recursion. In general, DNS servers that are intended for use by end users (caching servers) support recursion, whereas DNS servers that are authoritative for a specific domain do not. For this reason, your local DNS server may have to talk to several DNS servers before it finally reaches one that answers for a specific domain or subdomain.
You can use the `whois` tool to learn about the domains within most registries, including which domain name servers are authoritative for the domain. For more information, see the `a target="_self" whois/a` man page. You can also use the `a target="_self" nslookup/a` and `a target="_self" dig/a` tools to perform traditional unicast DNS lookups, and `a target="_self" dns-sd/a` to browse, resolve, and advertise Bonjour services.

## Other Uses of DNS

DNS lookups can provide more than just IP addresses. The DNS record for a hostname can contain various record types that each provide different kinds of information. A few of the more interesting record types include:

- `A`—An IPv4 address.
- `AAAA`—An IPv6 address.
- `CNAME`—A canonical name (mapping one hostname onto another hostname).
- `DNSKEY`—An encryption key used by DNSSEC (a cryptographically secure enhancement to the domain name system that is in the process of being phased in) when verifying the authenticity of a DNS reply.
- `MX` (mail exchanger)—The mail server (or servers) that should accept mail on behalf of the specified domain.
- `NS`—The name server delegation for a particular record (indicating that a request for that record should be answered by another server).
- `PTR`—A pointer to a canonical name. Similar to a `CNAME` record except that resolving typically stops at this point, and the client must then resolve the resulting `CNAME`, if desired. This is primarily used for reverse DNS lookups (because the goal is to get a name from an IP address, not to get the IP address back again). It is also used by DNS Service Discovery to store the human-readable name for a service.
- `SOA` (start of authority)—Used primarily to indicate how long clients should cache the results and which other servers are authoritative for the domain.

- `SRV`—Contains the hostname and port for a provided service. This record type is used by DNS Service Discovery.
- `TXT`—Contains a series of informational attributes used by DNS Service Discovery.

Next
Previous

Next
Previous

# Addressing Schemes and Domain Names

At every level of networking, each host is assigned one or more numeric identifiers that uniquely represent it within a particular network (though they are not always globally unique). The exact nature of those identifiers varies depending on whether you are working with networking at a high level (IP, for example) or a low level (raw Ethernet packets, for example).

## Link-Layer Addressing

At the link layer (physical layer), each network interface is usually identified by a globally unique hardware ID:

- Ethernet—MAC address (short for Media Access Control)
- Wi-Fi—MAC address (short for Media Access Control)
- Bluetooth—Bluetooth hardware ID (similar to a MAC address, but it does not share the same namespace)
- GSM Cellular—IMEI (International Mobile Equipment Identity)
- CDMA Cellular—ESN (Electronic Serial Number) or MEID (Mobile Equipment Identifier)

Because the identifier uniquely identifies a single network interface, you should never encounter two physical devices that share the same hardware address on the same physical network. More importantly, if a single host has more than one network interface of the same type, each interface has a different hardware ID.

The hardware ID is used to determine whether a particular device should listen to a packet or ignore it. Some physical networks also limit which packets get sent down a particular wire, based on whether an interface matching that hardware ID was seen on that wire previously. These

networks are referred to as switched networks. Networks in which every host sees every packet, by contrast, are called shared networks.
**Note:** Most programs do not need to use link-layer addresses directly.

## IP-Layer Addressing

At the IP layer and above, hosts are identified by an IP address (which is also sometimes called an IP number). An IP address can be in one of two forms: IPv4 or IPv6.
An IPv4 address consists of four bytes, and is usually represented to the user as a series of four numbers separated by decimal points. For example, the loopback address that always sends data back to the sending machine is `127.0.0.1`.
Because the IPv4 address space is limited in size (and rapidly running out), the IPv6 standard was created. This standard greatly expands the number of available addresses.
An IPv6 address is a 128-bit value, and is usually written as eight groups of 16-bit hexadecimal numbers separated by colons. Leading zeros in each group can be omitted as long as there is at least one digit in each group. For example, the IPv6 loopback address, `0000:0000:0000:0000:0000:0000:0000:0001`, can be simplified to `0:0:0:0:0:0:0:1`.
Also, if an IPv6 address contains several groups of zeros in a row, you can omit those groups and replace them with a double colon. You can do this only once per address (because otherwise the address would be ambiguous). Thus, the IPv6 loopback address, `0:0:0:0:0:0:0:1`, can be further simplified to `::1` (by replacing all the leading zero groups with a double colon).

## Domain Name System (DNS)

IPv4 addresses are hard to remember, and IPv6 addresses are quadruply hard (because they are four times as long). To make it easier to describe a particular host, the domain name system (DNS) was invented. Besides being easier to remember, domain names have several other advantages over IP addresses. Domain names:

- Minimize service disruption when an IP address changes. The IP addresses of a given host can change over time—when a mobile phone goes out of Wi-Fi range, when a user reboots his or her cable modem at home, when the owner of a server changes ISPs, and so on. By updating the domain name record to point to the server’s new address, users can continue to access the server by the same name.

- Allow a host to be accessed by more than one address. To improve robustness and performance, a server might have multiple IPv4 and IPv6 addresses on multiple physical network connections. If they all share the same domain name, clients can connect to any of its addresses. And when new addresses are added, clients can automatically learn about them.
- Allow multiple physical hosts to pretend to be a single host. For example, a single domain name might be serviced by a dozen servers that are geographically distributed around the world to improve performance.
- Can adapt to changes in the underlying technology. For example, an app that connects to a host by name does not need to care whether that host is reachable by IPv4 or IPv6. It asks for the address, and the domain name resolver returns an appropriate data structure for connecting to that host.

This section describes the parts of a domain name and the DNS lookup process at a high level.

## What’s in a (Domain) Name?

A domain name is a human-readable name that describes a particular host. Each domain name is made up of a series of parts separated by periods. The easiest way to explain these parts is by example. Consider the hostname `mail.example.com`:

- `com`—This part of the domain name is outside your administrative control. This may have multiple parts, but at a minimum, it contains a top-level domain (TLD). These top-level domains fall into one of the following categories:

  A generic TLD such as `.com`, `.org`, or `.net`
  A restricted TLD such as `.biz`, `.edu`, or `.gov`
  A sponsored TLD such as `.mobi`, `.museum`, or `.travel`
  A two-letter country code such as `.us`
- When the top-level domain contains a country code, the portion of the domain name outside your control may have multiple parts. For example, a business in the UK generally has an address in the `.co.uk` namespace.Each of the top-level domains, in turn, is part of the dot (`.`) root domain. In common usage, the root domain does not contribute any characters to the domain name itself. Under the hood, however, every fully qualified domain name ends in a period. For example, the name `apple.com.` can resolve to exactly one domain. The name `apple.com` typically resolves to `apple.com.`, but if that lookup fails, it could also resolve to `apple.com.example.com` if your computer is configured with `example.com` as its default search domain.

- `example`—The domain part. This part is under the administrative control of your company or organization, and applies to the company or organization as a whole.
- `mail`—The host part. This part allows you to uniquely identify multiple servers in a single domain. In addition, the domain name itself (with no host part) is also a valid hostname—apple.com, for example.

Domains can also contain any number of subdomains. For example, `www.david.example.com` would be the `www` host in the `david` subdomain of the `example.com` domain. For that matter, even the `example.com` domain is a subdomain of the `com` domain, which is itself a subdomain of the root domain. Every domain (except the root domain) is a subdomain of another domain.

Be aware that in the context of end-user discussions, the term domain is often used to refer to the largest part of the domain hierarchy that someone who wants to set up a service can actually buy or otherwise request. Thus, in the `com` hierarchy, a domain would contain two parts—`apple.com`, for example. Similarly, a domain in the `au` hierarchy would include at least three parts—`apple.com.au`, for example. It is not strictly correct to limit the use of these terms in this way, but this usage is not uncommon.

In general, domain names must be in ASCII (although there is a standard for internationalized domain names) and can contain only letters, numbers, and hyphens. Domain names are case-insensitive. For more information about internationalized domain names, see http://www.icann.org/en/resources/idn.

## Looking Up Names

Before a computer or other device can contact the host associated with a domain name, it must first look up the name to obtain the corresponding IP addresses. To do this, it contacts a DNS server. After the server returns one or more IP addresses, the computer or device can then connect to the remote host using any of those addresses.

Because a number of different agencies in different countries work together to manage the domain name system, the process for looking up addresses is somewhat complicated. A typical domain lookup includes the following steps:

1. Your computer or other device sends a query to a DNS server.
2. Your local DNS server sends a query to one of several central servers (called the root servers), asking what server is authoritative for the domain.
3. The root servers generally delegate responsibility for the lookup. Instead of providing a response with the IP address for `www.david.example.com`, the root servers instead tell you to ask another server at a specific IP address.

4. Your local DNS server sends a query to that TLD server, asking what server is authoritative for the domain.
5. The TLD server consults a database to see which servers answer requests for the specified domain.
6. The TLD server delegates responsibility for the lookup to the authoritative servers for the domain (and provides their IP addresses).
7. Your local DNS server sends a query to one or more of the servers provided by the TLD. That query asks for the host's IP addresses.
8. That server either returns one or more IP addresses or further delegates responsibility to another server.

For example, if you are looking up a host in the `com` top-level domain, your computer makes two requests: one asking for a list of IPv4 addresses and one asking for a list of IPv6 addresses. The root server then delegates responsibility to a gTLD (generic top-level domain) server, which in turn delegates responsibility to a server that is authoritative for the domain. That authoritative server may either return the answer or further delegate responsibility to another server for a specific subdomain.

This process of asking other servers to provide information is called recursion. In general, DNS servers that are intended for use by end users (caching servers) support recursion, whereas DNS servers that are authoritative for a specific domain do not. For this reason, your local DNS server may have to talk to several DNS servers before it finally reaches one that answers for a specific domain or subdomain.

You can use the `whois` tool to learn about the domains within most registries, including which domain name servers are authoritative for the domain. For more information, see the `a target="_self" whois/a` man page. You can also use the `a target="_self" nslookup/a` and `a target="_self" dig/a` tools to perform traditional unicast DNS lookups, and `a target="_self" dns-sd/a` to browse, resolve, and advertise Bonjour services.

## Other Uses of DNS

DNS lookups can provide more than just IP addresses. The DNS record for a hostname can contain various record types that each provide different kinds of information. A few of the more interesting record types include:

- `A`—An IPv4 address.
- `AAAA`—An IPv6 address.
- `CNAME`—A canonical name (mapping one hostname onto another hostname).
- `DNSKEY`—An encryption key used by DNSSEC (a cryptographically secure enhancement to the domain name system that is in the process of being phased in) when verifying the authenticity of a DNS reply.

- `MX` (mail exchanger)—The mail server (or servers) that should accept mail on behalf of the specified domain.
- `NS`—The name server delegation for a particular record (indicating that a request for that record should be answered by another server).
- `PTR`—A pointer to a canonical name. Similar to a `CNAME` record except that resolving typically stops at this point, and the client must then resolve the resulting `CNAME`, if desired. This is primarily used for reverse DNS lookups (because the goal is to get a name from an IP address, not to get the IP address back again). It is also used by DNS Service Discovery to store the human-readable name for a service.
- `SOA` (start of authority)—Used primarily to indicate how long clients should cache the results and which other servers are authoritative for the domain.
- `SRV`—Contains the hostname and port for a provided service. This record type is used by DNS Service Discovery.
- `TXT`—Contains a series of informational attributes used by DNS Service Discovery.

Next
Previous

Next
Previous

# Dynamic Address Assignment

In the early days of the Internet, every host was assigned a unique IP address that was hard-coded into its configuration files. As the Internet grew in popularity among less technically savvy users, and diskless devices became prevalent, it became impractical to manually change the IP address of each machine every time you changed networks. (This became even more important when laptops and wireless networking became commonplace.) To solve this problem, a number of protocols were developed culminating in the modern Dynamic Host Configuration Protocol (DHCP) standard, IPv6 neighbor discovery, DHCPv6, and link-local addressing. These protocols are described further in the sections that follow.

## Dynamic Host Configuration Protocol (DHCP) and DHCPv6

In IPv4, DHCP provides a means for a computer or other device to ask a central server for an IPv4 address suitable for use on the network. Depending on the server, the IPv4 address can be either assigned randomly from a pool of available addresses or assigned manually by an administrator based on the requesting host's MAC address.

In IPv6, the DHCPv6 protocol can optionally be used in a similar way, though IPv6 addresses can also be acquired through stateless address autoconfiguration, as described in the next section.

A DHCP server can also provide additional information needed for routing—the subnet mask and router address, for example—as well as domain name servers, directory servers, and other arbitrary data as defined by various extensions to the protocol.

The way DHCP works (at a high level) can be summarized as follows: the client broadcasts a request for an IPv4 address. After the server assigns one, the client is said to own a "lease" for that IP address that lasts a particular period of time. The client may renew that lease at any time until the expiration date. The server is not obligated to agree to extend the lease, but most servers do.

Generally speaking, if the client is connecting for the first time, this exchange is a series of broadcast UDP packets. If the client is renewing an existing (still valid) IP lease, the exchange can be sent with unicast UDP packets (at the client's discretion).

The client may continue to send requests for additional data (DNS servers, network volume mounts, Active Directory domain controllers, and so on) by sending requests with extra options and waiting for the reply.

## Neighbor Discovery and IPv6 Address Assignment

The IPv6 protocol (built on top of ICMPv6) provides built-in support for neighbor discovery. Neighbor discovery serves two purposes:

- Discovering the hardware address of other hosts on the same physical network. This takes the place of ARP.
- Stateless address autoconfiguration (SLAAC). This is an alternative to much of the functionality provided by DHCP.

When a host first connects to a network, it initially uses a self-assigned IPv6 address, which is globally unique by design. This address allows the host to send neighbor discovery requests to solicit a router. The router then provides the same sort of information that would appear in a DHCP offer under IPv4—the IP address of the router, the network prefix (which is roughly comparable to the network address and subnet mask), the IP addresses of DNS servers, and so on. From there, the client can construct a routable IPv6 address to use when communicating with the public Internet.

In addition to the self-assigned link-local address, each host constructs at least two additional IPv6 addresses: a permanent address and a privacy address.
The host constructs its permanent address based on the host's physical (link-layer) address. This results in an address whose host part generally remains unchanged as the computer moves from network to network. This address is intended to provide a consistent location where other hosts on the network can send data to the host in question.
Additionally, each host constructs one or more privacy addresses. These addresses are generated randomly and change over time. Thus, they are not tied to a particular piece of hardware. By default, outgoing connections are sent using a privacy address.
**Note:** When registering services for public consumption, however, apps should not use the privacy address because it will change. If you use Bonjour to register your service, it will handle these details for you. Bonjour is described in the next section.

## Link-Local Addressing and Bonjour

Domain names are a great solution for permanent servers, but they cost money and require a technically savvy user to configure them. Also, for non-public servers, a globally published domain name makes little sense. Further, standard domain names are fairly challenging to use with dynamic IP addresses assigned by DHCP servers and similar schemes. With an increasing number of dynamically configured networks, another solution was needed.
To provide an alternative, OS X and iOS support Bonjour, an implementation of zero-configuration networking. Bonjour consists of three parts:

- Link-local addresses in IPv4—A means for self-assignment of IP addresses in the absence of a DHCP server or other means of IP address assignment. (Link local address assignment is built in to the IPv6 protocol itself, and thus there is no need for Bonjour to duplicate this functionality.)
- Multicast DNS—A technology for providing DNS resolution when an infrastructure server is not present or when local, unmanaged names are more convenient.
- DNS Service Discovery—A means of registering and discovering services.

The combination of these technologies allows multiple hosts on the same physical network to advertise services and discover one another without the need for a permanent DNS infrastructure.
Typically, Bonjour uses multicast DNS to send a DNS Service Discovery query to every machine on a local network, asking whether any of them provide a particular service. Each machine that provides the requested

service then sends a message back to the original machine. The result is that those machines can discover and use services provided by other machines on the network without needing to tell each machine about those services ahead of time.

DNS Service Discovery can be combined with link-local addresses to allow zero-configuration networking—discovering networked devices even if you have no network infrastructure configured at all. For example, you might share files between two computers over an ad hoc wireless network.

The way link-local addresses work is fairly straightforward.

- For IPv4, if a host fails to obtain an address from a DHCP server, the computer chooses a random IP address in a particular range (`169.254.*.*`), then uses a link-specific protocol (such as ARP) to ask who has that address. If another machine responds, it tries a different address. If no machine responds, the host claims that number as its own.
- For IPv6, every interface has a link-local address within the `fe80::` prefix, regardless of whether the interface has another address assigned through SLAAC, DHCPv6, or other mechanisms. This address is computed based on the interface's hardware address if the interface has one (an Ethernet MAC address, for example). If it does not, the address is chosen randomly within that prefix, and SLAAC's duplicate address detection protocol is used to guarantee uniqueness. See RFC 4862 for details.

Because these IP addresses are all in the same subnet, every host with a link-local address on a given local area network can talk to every other host without going through a router. Thus, they can all discover one another without the need for any additional infrastructure.

**Note:** These three Bonjour technologies can also be used independently of one another. In particular, multicast DNS can be used to look up the name of a machine on the local network without discovering services. Similarly, traditional infrastructure DNS servers can provide information about services through Wide-Area Bonjour.

Next
Previous

Previous

# Glossary

**Address Resolution Protocol (ARP)** A protocol for determining the hardware address of a computer or other device based on its IP address.

**application layer** The topmost layer of the networking protocol stack. This layer consists of data formats and protocols specific to a given application. For example, the HTTP (hypertext transport protocol) standard is an application-layer protocol.

**broadcast address** A special address that sends a packet simultaneously to every device on a local area network.

**core router** A router that provides service for major Internet backbone routes. Core routers are powerful devices that must handle a large volume of traffic and usually must manage a large number of simultaneous routes. Core routers participate in route advertisements to discover or announce changes in the network topology.

**default gateway** The default router used for outgoing traffic if there is no explicit route for the destination IP in the system's routing table.

**domain name** A human-readable name that identifies an Internet or intranet site; for example, developer.apple.com is a domain name. By resolving a domain name, an application can obtain a corresponding IP address that is suitable for sending data to that site.

**edge router** A router that provides connectivity between a customer site and an upstream ISP. Edge routers generally route between only two or three different networks, and thus usually do not participate in route advertisements.

**encapsulation** The act of wrapping one packet inside another packet (usually of a different type). For example, on a local area network, your IP packets are encapsulated within Ethernet packets. The Ethernet packets provide information about their destination within the local area network. The IP packets inside them provide information about what to do with the packets once they reach the public Internet.

**firewall** A router that limits the type of traffic that can pass. A firewall may block certain ports, perform network address translation to hide the IP addresses of hosts on one side, block malformed packets, or perform various other packet rewriting operations.

**fragmentation** The process of breaking up a packet into smaller pieces to accommodate network connections with a smaller maximum packet size (referred to as the maximum transmission unit, or MTU).

**header** In the context of packets, the first part of a packet (before the actual payload) that contains information about where the packet should be sent.In the context of HTTP, a series of values that provide information about the content of a request or reply, such as the hostname, caching policies, and so on.

**hop** Any one of a series of physical links that make up the route from one host to another.

**host** Any device that is connected to a network. It may be a client computer, a server, a mobile phone, or even a network-attached printer.

**hostname (or host name)** A DNS name that points to a specific host (or a group of hosts that mimic a single host).

**infrastructure device** Any device that provides support for a network's basic operation—for example, a router, a Wi-Fi access point, or an Ethernet switch.

**Internet Control Message Protocol (ICMP)** A low-level networking protocol that provides out-of-band control messages that are used by the operating system when making TCP connections. ICMP is used mainly to deliver connection failure notifications—"connection refused" and "host unreachable" messages, for example. However, it is also used by some network diagnostic tools, such as `a target="_self" ping/a` and `a target="_self" traceroute/a`.

**IP (Internet Protocol) layer** The networking layer that provides basic transport of packets across the Internet. It sits above the physical layer (hardware interconnects) and below the transport layer (TCP and UDP, for example).

**IP address** A number that uniquely identifies a single host on the Internet (short for Internet Protocol address). An IP address can be in one of two forms: an IPv4 address or an IPv6 address.

**IPv4 address** An IP address consisting of four 8-bit numbers (for a total of 32 bits). For example, the IP address for developer.apple.com is 17.254.2.129.

**IPv6 address** An IP address consisting of eight groups of 16-bit hexadecimal numbers (for a total of 128 bits). If several groups in a row are all zero, you can omit those groups and replace them with a double colon (but only once per IP address). For example, the IPv6 address for example.com is 2001:500:88:200::10.

**latency** The amount of time it takes for a packet to reach its destination, usually measured in milliseconds. Latency is usually expressed as round-trip latency, which refers to the amount of time for a packet to reach its destination and for the response packet to reach the original host. Latency is important for two reasons. First, it increases the amount of time it takes to establish a connection. Second, it dramatically reduces performance when using protocols that require the client to wait for a response before sending subsequent requests.

**link** A physical connection between two hosts on a network (or a virtual connection that emulates a physical connection) with no intermediate routers (except for link-layer switches).

**link layer** The lowest layer of the network protocol stack. This layer provides support for the physical transport of packets from one host to another across a local area network or other physical link.

**listening socket (or listen socket)** A socket configured to listen for incoming connections.

**Maximum Transmission Unit (MTU)** The largest packet size that can be delivered across a particular link. The MTU is limited by the actual communication hardware, and usually represents the maximum payload size supported by the largest physical packet that the hardware supports. However, in some cases (such as gigabit Ethernet), the default MTU may be further limited in software to maintain backwards compatibility with legacy hardware that does not support larger packets.

**multicast** A special type of packet that is simultaneously delivered to a multitude of hosts on the network, but not to every host (broadcast).

**neighbor discovery protocol (NDP)** A protocol used by IPv6 over Ethernet to learn about other devices on the physical network. Among other things, neighbor discovery can be used to learn the hardware addresses of nearby devices, discover routers and name servers, and determine information about upstream links, such as their Maximum Transmission Unit (MTU).

**netblock** See subnet.

**netmask** A collection of bits indicating which portion of an IPv4 address is the network part and which portion is the host part. If the network part of the destination address is the same as the network part of the source address, the two hosts are considered to be within the same subnet.

**network address** A special reserved address within each IPv4 network in which the host part is all zeros. This address was used by older operating systems as the broadcast address, so for historical compatibility reasons, this number is reserved.

**network address translation (NAT)** A form of packet rewriting performed by a firewall in which packets are modified to contain a different source or destination IP address before passing them on. NAT is most commonly used to make traffic from multiple devices appear to come from a single device, often for security or load balancing purposes.

**network interface** A piece of hardware (or virtual hardware) that represents the endpoint of a link.

**packets** A discrete unit of data that is sent across a computer network.

**path MTU discovery** A process by which one host determines the largest packet that can be sent to a destination without fragmenting it. This allows the host to fragment the data ahead of time, which prevents packets from potentially being fragmented more than once before reaching their final destination. Path MTU discovery works by sending packets with the “Don't Fragment” bit set. If any router along the path responds by sending an ICMP packet with the Fragmentation Needed bit set, the host then tries progressively smaller sizes until the packet reaches its destination successfully.
See also Maximum Transmission Unit (MTU).

**payload** The data contents of a packet (as distinct from the structure of the packet itself).

**physical layer** See link layer.

**port numbers** A number that uniquely identifies a particular service on a given host. Port numbers are further divided according to whether they are TCP or UDP ports.

**recursion** The use of recursive queries. A recursive query asks the domain name server to perform recursion on the client's behalf. If the domain name server allows recursive queries, it then sends a query to the root name server asking which server knows the answer, then asks that server, and so on, until it reaches a server that actually knows the answer to the query. See also recursion.

**route** The path that packets take from one host to another host. If the two hosts are on the same physical network, the route consists of a single link; if not, it passes through one or more routers.

**router** A device that routes packets between two or more networks. A router determines which network should receive each packet based on a set of routing rules. Most routers also communicate with other routers to optimize those rules as network links are added and removed.

**router address** The IP address of your router.

**routing** The process of taking a packet on one physical network and retransmitting it on a different physical network, using a set of rules to determine which network should receive each packet. A device that performs routing is called a router.

**shared network** A network in which every packet is received by every device on the network. This is the opposite of a switched network.

**subnet** A range of IP addresses in which packets from one host can be sent directly to another host without going through an intermediate router.

**switched network** A physical network in which an infrastructure device (called a switch) directs packets based on their destination. This improves network performance by ensuring that only the hosts that need to receive a given packet actually see it. This is the opposite of a shared network.

**trailer** The last part of a packet (after the payload) that usually contains a checksum of the payload data.

**Transmission Control Protocol (TCP)** A transport-layer protocol that provides bidirectional, stream-based delivery of data, with flow control and delivery guarantees (automatic retry). Contrast with User Datagram Protocol (UDP).

**transport layer** The networking layer that sits on top of the IP layer and can provide such features as port numbers, delivery guarantees, flow control, and checksums. The two most common transport-layer protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

**User Datagram Protocol (UDP)** A transport-layer protocol that provides unidirectional, packet-based delivery of data, with best-effort delivery (no retransmission). Contrast with Transmission Control Protocol (TCP).

Previous