CS6551 – COMPUTER NETWORKS

UNIT – 1: FUNDAMENTALS AND LINK LAYER

BUILDING A NETWORK

Introduction:

Most people know the Internet through its applications: the World Wide Web, email, streaming audio and video, chat rooms, and music (file) sharing. The Web, for example, presents an intuitively simple interface.

Users view pages full of textual and graphical objects, click on objects that they want to learn more about, and a corresponding new page appears. Most people are also aware that just under the covers, each selectable objection a page is bound to an identifier for the next page to be viewed.

Applications:

There are a variety of different classes of video applications. One class of video application is video- on-demand, which reads a preexisting movie from disk and transmits it over the network. Another kind of application is videoconferencing, which is in some way the more challenging (and, for networking people, interesting) case because it has very tight timing constraints. Just as when using the telephone, the interactions among the participants must be timely.

When a person at one end gestures, then that action must be displayed at the other end as quickly as possible. Too much delay makes the system unusable. Contrast this with video-on-demand where, if it takes several seconds from the time the user starts the video until the first image is displayed, the service is still deemed satisfactory.

Also, interactive video usually implies that video is flowing in both, while a video-on-demand application is most likely sending video in only one direction.

Although they are just two examples, downloading pages from the Web and participating in a videoconference demonstrate the diversity of applications that can be built on top of the Internet, and hint at the complexity of the Internet's design.

REQUIREMENTS

Introduction:

The first step is to identify the set of constraints and requirements that influence network design. Before getting started, however, it is important to understand that the expectations you have of a network depend on your perspective:

- ✓ An application programmer would list the services that his application needs, for example, a guarantee that each message the application sends will be delivered without error within a certain amount of time.

- ✓ A network designer would list the properties of a cost-effective design, for example, that network resources are efficiently utilized and fairly allocated to different users.

- ✓ A network provider would list the characteristics of a system that is easy to administer and manage, for example, in which faults can be easily isolated and where it is easy to account for usage.

Connectivity:

Starting with the obvious, a network must provide connectivity among a set of computers. Sometimes it is enough to build a limited network that connects only a few select machines. In fact, for reasons of privacy and security, many private (corporate) networks have the explicit goal of limiting the set of machines that are connected. In contrast, other networks (of which the Internet is the prime example) are designed to grow in a way that allows them the potential to connect all the computers in the world. A system that is designed to support growth to an arbitrarily large size is said to scale. Using the Internet as a model, this book addresses the challenge of scalability.

Links, Nodes, and Clouds:

Network connectivity occurs at many different levels. At the lowest level, a network can consist of two or more computers directly connected by some physical medium, such as a coaxial cable or an optical fiber. We call such a physical medium a link, and we refer to the computers it connects as nodes.

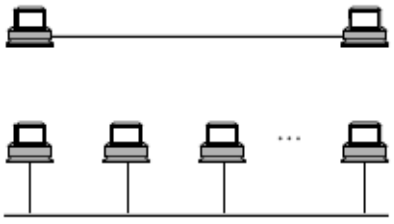


Figure: Direct links: point to point; multi-point

As illustrated in the above figure, physical links are sometimes limited to a pair of nodes (such a link is said to be point-to-point), while in other cases, more than two nodes may share a single physical link (such a link is said to be multiple-access).

Think of these blocks of data as corresponding to some piece of application data such as a file, a piece of email, or an image. Packet-switched networks typically use a strategy called store-and- forward. As the name suggests, each node in a store-and-forward network first receives a complete packet over some link, stores the packet in its internal memory, and then forwards the complete packet to the next node. In contrast, a circuit-switched network first establishes a dedicated circuit across a sequence of links and then allows the source node to send a stream of bits across this circuit to a destination node.

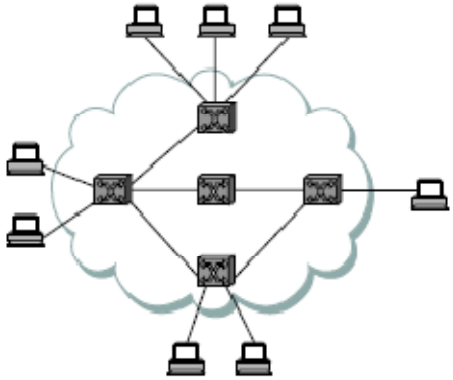


Figure: Switched network

The cloud in above figure distinguishes between the nodes on the inside that implement the network (they are commonly called switches, and their primary function is to store and forward packets) and the nodes on the outside of the cloud that use the network (they are commonly called hosts, and they support users and run application programs).

A node that is connected to two or more networks is commonly called a router or gateway, and it plays much the same role as a switch—it forwards messages from one network to another. Note that an internet can itself be viewed as another kind of network, which means that an internet can be built from an interconnection of internets. Thus, we can recursively build arbitrarily large networks by interconnecting clouds to form larger clouds.

Just because a set of hosts are directly or indirectly connected to each other does not mean that we have succeeded in providing host-to-host connectivity. The final requirement is that each node must be able to state which of the other nodes on the network it wants to communicate with. This is done by assigning an address to each node. An address is a byte string that identifies a node; that is, the network can use a node's address to distinguish it from the other nodes connected to the network.

NETWORK ARCHITECTURE

Introduction:

A computer network must provide general, cost effective, fair, and robust connectivity among a large number of computers. As if this weren't enough, networks do not remain fixed at any single point in time, but must evolve to accommodate changes in both the underlying technologies upon which they are based as well as changes in the demands placed on them by application programs. Designing a network to meet these requirements is no small task.

To help deal with this complexity, network designers have developed general blueprints—usually called network architectures—that guide the design and implementation of networks.

LAYERING AND PROTOCOLS

When a system gets complex, the system designer introduces another level of abstraction. The idea of an abstraction is to define a unifying model that can capture some important aspect of the system, encapsulate this model in an object that provides an interface that can be manipulated by other components of the system, and hide the details of how the object is implemented from the users of the object.

The challenge is to identify abstractions that simultaneously provide a service that proves useful in a large number of situations and that can be efficiently implemented in the underlying system. This is exactly what we were doing when we introduced the idea of a channel in the previous section: We were providing an abstraction for applications that hides the complexity of the network from application writers.

Abstractions naturally lead to layering, especially in network systems. The general idea is that you start with the services offered by the underlying hardware, and then add a sequence of layers, each providing a higher (more abstract) level of service. The services provided at the high layers are implemented in terms of the services provided by the low layers. Drawing on the discussion of requirements given in the previous section, for example, we might imagine a simple network as having two layers of abstraction sandwiched between the application program and the underlying hardware, as illustrated in the following figure.

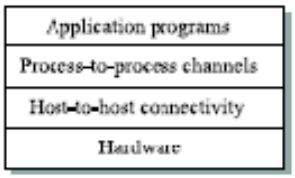


Figure: Example of a layered network system

The layer immediately above the hardware in this case might provide host-to-host connectivity, abstracting away the fact that there may be an arbitrarily complex network topology between any two hosts. The next layer up builds on the available host-to- host communication service and provides support for process-to-process channels, abstracting away the fact that the network occasionally loses messages, for example.

Layering provides two nice features. First, it decomposes the problem of building a network into more manageable components. Layers, each of which solves one part of the problem. Second, it provides a more modular design. If you decide that you want to add some new service, you may only need to modify the functionality at one layer, reusing the functions provided at all the other layers.

For example, a request/reply protocol would support operations by which an application an send and receive messages. An implementation of the HTTP protocol could support an operation to fetch a page of hypertext from a remote server. An application such as a web browser would invoke such an operation whenever the browser needs to obtain a new page, for example, when the user clicks on a link in the currently displayed page.

Second, a protocol defines a peer interface to its counterpart (peer) on another machine. This second interface defines the form and meaning of messages exchanged between protocol peers to implement the communication service. This would determine the way in which a request/reply protocol on one machine communicates with its peer on another machine.

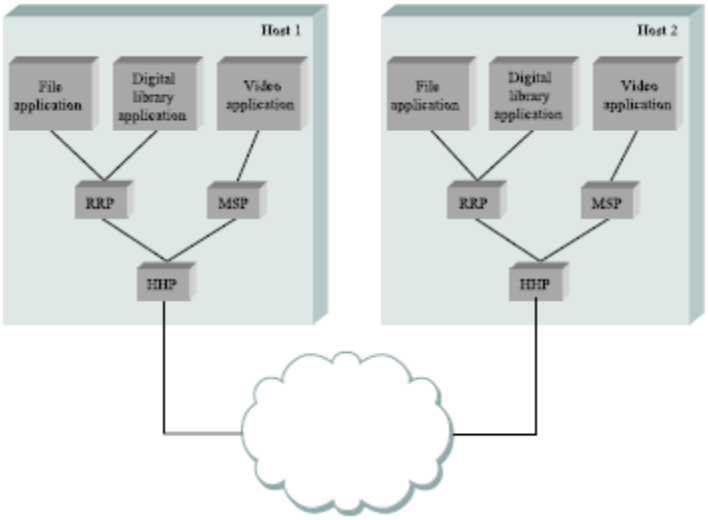


Figure: Example of a protocol graph

Consider what happens in the above figure when one of the application programs sends a message to its peer by passing the message to protocol RRP. From RRP's perspective, the message it is given by the application is an uninterrupted string of bytes. RRP does not care that these bytes represent an array of integers, an email message, a digital image, or whatever; it is simply charged with sending them to its peer.

As the name suggests, headers are usually attached to the front of a message. In some cases, however, this peer-to-peer control information is sent at the end of the message, in which case it is called a trailer. The exact format for the header attached by RRP is defined by its protocol specification. The rest of the message—that is, the data being transmitted on behalf of the application—is called the message's body or payload. We say that the application's data is encapsulated in the new message created by protocol RRP.

This process of encapsulation is then repeated at each level of the protocol graph; for example, HHP encapsulates RRP's message by attaching a header of its own. The message passed up from RRP to the application on host 2 is exactly the same message as the application passed down to RRP on host 1; the application does not see any of the headers that have been attached to it to implement the lower-level communication services.

Multiplexing and Demultiplexing:

Practically speaking, this entire means is that the header that RRP attaches to its messages contains an identifier that records the application to which the message belongs. We call this identifier RRP's demultiplexing key, or demux key for short. At the source host, RRP includes the appropriate demux key in its header.

When the message is delivered to RRP on the destination host, it strips its header, examines the demux key, and demultiplexes the message to the correct application. RRP is not unique in its support for multiplexing; nearly every protocol implements this mechanism.

For example, HHP has its own demux key to determine which messages to pass up to RRP and which to pass up to MSP. However, there is no uniform agreement among protocols—even those within a single network architecture—on exactly what constitutes a demux key. Some protocols use an 8-bit field (meaning they can support only 256 high-level protocols), and others use 16- or 32-bit fields.

Also, some protocols have a single demultiplexing field in their header, while others have pair of demultiplexing fields. In the former case, the same demux key is used on both sides of the communication, while in the latter case, each side uses a different key to identify the high-level protocol (or application program) to which the message is to be delivered.

INTERNET ARCHITECTURE

The ISO was one of the first organizations to formally define a common way to connect computers. The ISO, usually in conjunction with second standards organization known as the International Telecommunications Union (ITU),1 publishes a series of protocol specifications based on the OSI architecture. This is sometimes called the "X dot" series since the protocols are given names like X.25, X.400, X.500, and so on.

Starting at the bottom and working up, the physical layer handles the transmission of raw bits over a communications link. The data link layer then collects a stream of bits into a larger aggregate called a frame. Network adaptors, along with device drivers running in the node's OS, typically implement the data link level.

This means that frames, not raw bits, are actually delivered to hosts. The network layer handles routing among nodes within a packet-switched network. At this layer, the unit of data exchanged among is typically called a packet rather than a frame, although they are fundamentally.

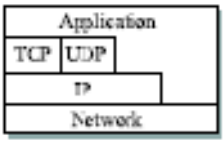


Figure: Alternate view of internet architecture

The lower three layers are implemented on all network nodes, including switches within the network and hosts connected along the exterior of the network. The transport layer then implements what we have up to this point been calling a process-to-process channel. Here, the unit of data exchanged is commonly called a message rather than a packet or a frame. The transport layer and higher layers typically run only on the end hosts and not on the intermediate switches or routers.

There is less agreement about the definition of the top three layers. Skipping ahead to the top (seventh) layer, we find the application layer. Application layer protocols include things like the File Transfer Protocol (FTP), which defines a protocol by which file transfer applications can interoperate.

Below that, the presentation layer is concerned with the format of data exchanged between peers, for example, whether an integer is 16, 32, or 64 bits long and whether the most significant byte is transmitted first or last, or how a video stream is formatted. Finally, the session layer provides a name space that is used to tie together the potentially different transport streams that are part of a single.

PERFORMANCE

Introduction:

Up to this point, we have focused primarily on the functional aspects of a network. Like any computer system, however, computer networks are also expected to perform well. This is because the effectiveness of computations distributed over the network often depends directly on the efficiency with which the network delivers the computation's data.

While the old programming adage "first get it right and then make it fast" is valid in many settings, in networking it is usually necessary to "design for performance." It is, therefore, important to understand the various factors that impact network performance.

Bandwidth and Latency:

Network performance is measured in two fundamental ways: bandwidth (also called throughput) and latency (also called delay). The bandwidth of a network is given by the number of bits that can be transmitted over the network in a certain period of time.

For example, a network might have a bandwidth of 10 million bits/second (Mbps), meaning that it is able to deliver10 million bits every second. It is sometimes useful to think of bandwidth in terms of how long it takes to transmit each bit of data. On a 10-Mbps network, for example, it takes 0.1 microseconds (μs) to transmit each bit.

First of all, bandwidth is literally a measure of the width of a frequency band. For example, a voice-grade telephone line supports a frequency band ranging from1.5 Performance 41300 to 3,300 Hz; it is said to have a bandwidth of 3,300 Hz − 300 Hz =3,000 Hz. If you see the word "bandwidth" used in a situation in which it is being measured in hertz, then it probably refers to the range of signals that can be accommodated. When we talk about the bandwidth of a communication link, we normally refer to the number of bits per second that can be transmitted on the link. We might say that the bandwidth of an Ethernet is 10 Mbps.

A useful distinction might be made, however, between the bandwidth that is available on the link and the number of bits per second that we can actually transmit over the link in practice. We tend to use the word "throughput" to refer to the measured performance of a system. Thus, because of a single physical link or a logical process-to-process channel. At the physical level, bandwidth is constantly improving, with no end in sight.

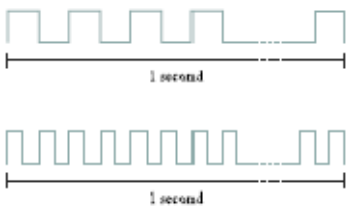


Figure: Bits transmitted at a particular bandwidth

Intuitively, if you think a second of time as a distance you could measure with a ruler and bandwidth as how many bits fit in that distance, then you can think of each bit as a pulse of some width. For example, each bit on a 1-Mbps link is 1 μs wide, while each bit on a 2-Mbps link is 0.5 μs wide, as illustrated in the above figure.

The more sophisticated the transmitting and receiving technology, the narrower each bit can become, and thus, the higher the bandwidth. For logical process-to-process channels, bandwidth is also influenced by other factors, including how many times the software that implements the channel has to handle, and possibly transform, each bit of data.

Third, there may be queuing delays inside the network, since packet switches generally need to store packets for some time before forwarding them on an outbound link. So, we could define the total latency as

Latency = Propagation + Transmit + Queue Propagation = Distance/Speed of Light
Transmit = Size/Bandwidth

where Distance is the length of the wire over which the data will travel, Speed of Light is the effective speed of light over that wire, Size is the size of the packet, and Bandwidth is the bandwidth at which the packet is transmitted.

LINK LAYER SERVICES

FRAMING

Introduction:

When node A wishes to transmit a frame to node B, it tells its adaptor to transmit a frame from the node's memory. This results in a sequence of bits being sent over the link. The adaptor on node B then collects together the sequence of bits arriving on the link and deposits the corresponding frame in B's memory.

There are several ways to address the framing problem. Note that while we framing in the context of point-to-point links, the problem is a fundamental one that must also be addressed in multiple-access networks like Ethernet and token rings.

Byte-Oriented Protocols (PPP):

One of the oldest approaches to framing—it has its roots in connecting terminals to mainframes—is to view each frame as a collection of bytes (characters) rather than a collection of bits. Such a byte- oriented approach is exemplified by older protocols such as the Binary Synchronous Communication (BISYNC) protocol developed by IBM in the late 1960s.

Figure: PPP frame format

Message Protocol (DDCMP) used in Digital Equipment Corporation's DECNET. The more recent and widely used Point-to-Point Protocol (PPP) provides another example of this approach.

Figure: BISYNC frame format

The beginning of a frame is denoted by sending a special SYN (synchronization) character. The data portion of the frame is then contained between two more special characters: STX (start of text) and ETX (end of text). The SOH (start of header) field serves much the same purpose as the STX field. The problem with the sentinel approach, of course, is that the ETX character might appear in the data portion of the frame.

BISYNC overcomes this problem by "escaping" the ETX character by preceding it with a data-link-escape (DLE) character whenever it appears in the body of a frame; the DLE character is also escaped (by preceding it with an extra DLE) in the frame body.

This approach is often called character stuffing because extra characters are inserted in the data portion of the frame. The frame format also includes a field labeled cyclic redundancy check (CRC) that is used to detect transmission errors; various algorithms for error detection. Finally, the frame contains additional header fields that are used for, among other things, the link-level reliable delivery algorithm.

Clock-Based Framing (SONET):

T he next approach to framing is exemplified by the Synchronous Optical Network (SONET) standard. For lack of a widely accepted generic term, we refer to this approach simply as clock-based framing. SONET was first proposed by Bell Communications Research (Bellcore), and then developed under the American National Standards Institute (ANSI) for digital transmission over optical fiber; it has since been adopted by the ITU-T. Who standardized what and when is not the interesting issue, though.

The thing to remember about SONET is that it is the dominant standard for long- distance transmission of data over optical networks. An important point to make about SONET before we go any further is that the full specification is substantially larger than this book. Thus, the following discussion will necessarily cover only the high points of the standard. Also, SONET addresses both the framing problem and the encoding problem.

As with the previously discussed framing schemes, a SONET frame has some special information that tells the receiver where the frame starts and ends. However, that is about as far as the similarities go. Notably, no bit stuffing is used, so that a frame's length does not depend on the data being sent. So the question to ask is, "How does the receiver know where each frame starts and ends?" We consider this question for the lowest-speed SONET link, which is known as STS-1 and runs at 51.84 Mbps. An STS-1 frame is shown in the following figure.

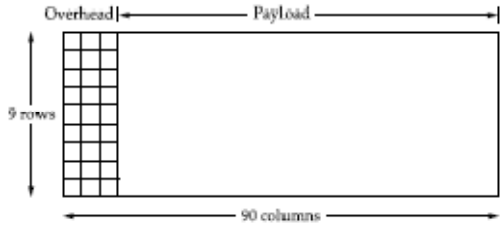


Figure: A SONET STS-1 frame

It is arranged as nine rows of 90 bytes each, and the first 3 bytes of each row are overhead, with the rest being available for data that is being transmitted over the link. The first 2 bytes of the frame contain a special bit pattern, and it is these bytes that enable the receiver to determine where the frame starts. However, since bit stuffing is not used, there is no reason why this pattern will not occasionally turn up in the payload portion of the frame. To guard against this, the receiver looks for the special bit pattern consistently, hoping to see it appearing once every 810 bytes, since each frame is 9×90 = 810 bytes long.

ERROR DETECTION

Introduction:

There are two basic approaches that can be taken when the recipient of a message detects an error. One is to notify the sender that the message was corrupted so that the sender can retransmit a copy of the message. If bit errors are rare, then in all probability the retransmitted copy will be error free.

Alternatively, there are some types of error detection algorithms that allow the recipient to reconstruct the correct message even after it has been corrupted; such algorithms rely on error correcting codes. One of the most common techniques for detecting transmission errors is a known as the cyclic redundancy check (CRC).

Before discussing that approach, we consider two simpler schemes that are also widely used: two-dimensional parity and checksums. The former is used by the BISYNC protocol when it is transmitting ASCII characters (CRC is used as the error code when BISYNC is used to transmit EBCDIC), and the latter is used by several Internet protocols.

The basic idea behind any error detection scheme is to add redundant information to a frame that can be used to determine if errors have been introduced. In the extreme, we could imagine transmitting two complete copies of the data.

If the two copies are identical at the receiver, then it is probably the case that both are correct. If they differ, then an error was introduced into one (or both) of them, and they must be discarded. This is a rather poor error detection scheme for two reasons.

First, it sends n redundant bits for an n-bit message. Second, many errors will go undetected—any error that happens to corrupt the same bit positions in the first and second copies of the message. Fortunately, we can do a lot better than this simple scheme.

In general, we can provide quite strong error detection capability while sending only k redundant bits for2.4 Error Detection 93an n-bit message, where k _ n. On an Ethernet, for example, a frame carrying up to12, 000 bits (1,500 bytes) of data requires only a 32-bit CRC code, or as it is commonly expressed, uses CRC-32. Such a code will catch the overwhelming majority of errors.

Two-Dimensional Parity:

Two-dimensional parity is exactly what the name suggests. It is based on “simple” (one dimensional parity, which usually involves adding one extra bit to a 7-bit code to balance the number of 1s in the byte. For example, odd parity sets the eighth bit to 1 if needed to give an odd number of 1s in the byte, and even parity sets the eighth bit to 1 if needed to give an even number of 1s in the byte.

Two- dimensional parity does a similar calculation for each bit position across each of the bytes contained in the frame. This results in an extra parity byte for the entire frame, in addition to a parity bit for each byte.

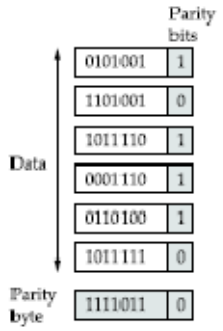


Figure: Two-dimensional party

The above figure illustrates how two-dimensional even parity works for an example frame containing 6 bytes of data. Notice that the third bit of the parity byte is 1 since there is an odd number of 1s in the third bit across the 6 bytes in the frame. It can be shown two-dimensional parity catches all 1-, 2-, and 3-bit errors, and most 4-bit errors. In this case, we have added 14 bits of redundant information to a 42-bit message, and yet we have stronger protection against common errors than the "repetition code" described above.

Internet Checksum Algorithm:

The idea behind the Internet checksum is very simple—you add up all the words that are transmitted and then transmit the result of that sum. The result is called the checksum. The receiver performs the same calculation on the received data and compares the result with the received checksum.

If any transmitted data, including the checksum itself, is corrupted, then the results will not match, so the receiver knows that an error occurred You can imagine many different variations on the basic idea of a checksum.

The exact scheme used by the Internet protocols works as follows. Consider the data being checksummed as a sequence of 16-bit integers. Add them together using 16-bit ones complement arithmetic (explained below) and then take the ones complement of the result. That 16-bit number is the checksum. In ones complement arithmetic, a negative integer −x is represented as the complement of x, that is, each bit of x is inverted.

When adding numbers in ones complement arithmetic, a carryout from the most significant bit needs to be added to the result. Consider, for example, the addition of −5 and −3 in ones complement arithmetic on 4-bitintegers: +5 is 0101, so −5 is 1010; +3 is 0011, so −3 is 1100. If we add 1010 and1100 ignoring the carry, we get 0110.

In ones complement arithmetic, the fact that this operation caused a carry from the most significant bit causes us to increment the result, giving 0111, which is the ones complement representation of −8 (obtained by inverting the bits in 1000), as we would expect.

Cyclic Redundancy Check:

It should be clear by now that a major goal in designing error detection algorithms is to maximize the probability of detecting errors using only a small number of redundant bits. Cyclic redundancy checks use some fairly powerful mathematics to achieve this goal.

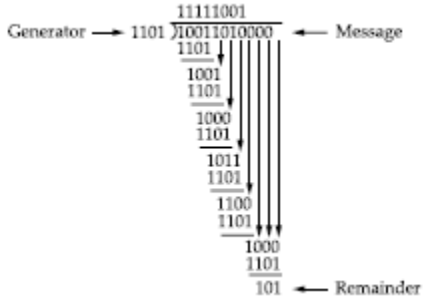


Figure: CRC calculation using polynomial long division

For example, a 32-bit CRC gives strong protection against common bit errors in messages that are thousands of bytes long. The theoretical foundation of the cyclic redundancy check is rooted in a branch of mathematics called finite fields. While this may sound daunting, the basic ideas can be easily understood.

To start, think of an (n+1)-bit message as being represented by an n degree polynomial, that is, a polynomial whose highest-order term is xn. The message is represented by a polynomial by using the value of each bit in the message as the coefficient.

UNIT – 2: MEDIA ACCESS AND INTERNETWORKING

MEDIA ACCESS CONTROL

ETHERNET (802.3)

Introduction:

The Ethernet is easily the most successful local area networking technology of the 20yearsDeveloped in the mid-1970s by researchers at the Xerox Palo Alto Research Center (PARC the Ethernet working example of the more general carrier sense, multiple access with collision detect (CSMA/CD) local area network technology. As indicated by the CSMA name, the Ethernet is a multiple-access network, meaning that a set of nodes send and receive frames over a shared link.

The "carrier sense" in CSMA/CD means that all the nodes can distinguish between an idle and a busy link, and "collision detect" means that a node listens as it transmits and can therefore detect when a frame it is transmitting has interfered (collided) with a frame transmitted by another node. The Ethernet has its roots in an early packet radio network, called Aloha, developed at the University of Hawaii to support computer communication across the Hawaiian Islands.

Like the Aloha network, the fundamental problem faced by the Ethernet is how to mediate access to a shared medium fairly and efficiently (in Aloha the medium was the atmosphere, while in Ethernet the medium is a coax cable). That is, the core idea in both Aloha and the Ethernet is an algorithm that controls when each node can transmit. Digital Equipment Corporation and Intel Corporation joined Xerox to define a10-Mbps Ethernet standard in 1978.

Physical Properties:

An Ethernet segment is implemented on a coaxial cable of up to 500 m. This cable is similar to the type used for cable TV, except that it typically has an impedance of 50 ohms instead of cable TV's 75 ohms. Hosts connect to an Ethernet segment by tapping into it; taps must be at least 2.5 m apart. A transceiver—a small device directly attached to the tap—detects when the line is idle and drives the signal when the host is transmitting.

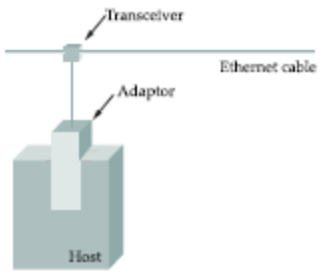


Figure: Ethernet transceiver and receiver

Multiple Ethernet segments can be joined together by repeaters. A repeater is a device that forwards digital signals, much like an amplifier forwards analog signals. in the same way as you would with 10Base5 cable. With 10Base2, a T-joint is spliced into the cable. In effect, 10Base2 is used to daisy-chain a set of hosts together.

With10BaseT, the common configuration is to have several point-to-point segments coming out of a multi-way repeater, sometimes called a hub, as illustrated in the following figure.

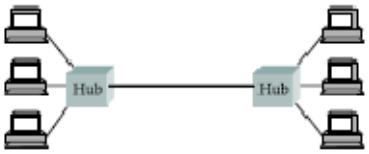


Figure: Ethernet hub

Access Protocol:

It is typically implemented in hardware on the network adaptor. First, however, we describe the Ethernet's frame format and addresses.

Finally, each frame includes a 32-bit CRC. Note that from the host's perspective, an Ethernet frame has a 14-byte header: two 6-byte addresses and a 2-byte type field. The sending adaptor attaches the preamble, CRC, and postamble before transmitting, and the receiving adaptor removes them.

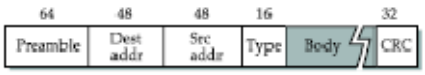


Figure: Ethernet frame format

The frame format just described is taken from the Digital-Intel-Xerox Ethernet standard. The 802.3 frame format is exactly the same, except it substitutes a 16-bit length This type field is the first thing in the data portion of the 802.3 frames, that is, it immediately follows the 802.3 header.

Addresses:

Each host on an Ethernet—in fact, every Ethernet host in the world—has a unique Ethernet address. Technically, the address belongs to the adaptor, not the host; it is usually burned into ROM. Ethernet addresses are typically printed in a form humans can read as a sequence of six numbers separated by colons. Each number corresponds to 1 byte of the 6-byte address and is given by a pair of hexadecimal digits, one for each of the4-bit nibbles in the byte; leading 0s are dropped.

Similarly, an address that has the first bit set to 1 but is not the broadcast address is called a multicast address. A given host can program its adaptor to accept some set of multicast addresses. Multicast addresses are used to send messages to some subset of the hosts on an Ethernet (e.g., all file servers).

- ✓ Frames addressed to the broadcast address; 2.6 Ethernet (802.3)
- ✓ Frames addressed to a multicast address, if it has been instructed to listen to that address;
- ✓ All frames, if it has been placed in promiscuous mode.

It passes to the host only the frames that it accepts.

Transmitter Algorithm:

The transmitter algorithm is defined as follows. When the adaptor has a frame to send and the line is idle, it transmits the frame immediately; there is no negotiation with the other adaptors. The upper bound of1, 500 bytes in the message means that the adaptor can occupy the line for only a fixed length of time. When an adaptor has a frame to send and the line is busy, it waits for the line to go idle and then transmits immediately.

The Ethernet is said to be a 1-persistent protocol because an adaptor with a frame to send transmits with probability 1 whenever a busy line goes idle. In general, a p-persistent algorithm transmits with probability $0 \leq p \leq 1$after a line becomes idle, and defers with probability $q = 1 - p$.

The reasoning behind choosing a $p < 1$ is that there might be multiple adaptors waiting for the busy line to become idle, and we don't want all of them to begin transmitting at the same time. If each adaptor transmits immediately with a probability of, say, 33%, then up to three adaptors can be waiting to transmit and the odds are that only one will begin transmitting when the line becomes idle.

Despite this reasoning, an Ethernet adaptor always transmits immediately after noticing that the network has become idle and has been very effective in doing so. To complete the story about p-persistent protocols for the case when $p < 1$, you might wonder how long a sender that loses the coin flip (i.e., decides to defer) has to wait before it can transmit. The answer for the Aloha network, which originally developed this style of protocol, was to divide time into discrete slots, with

BLUETOOTH (802.15.1)

Introduction:

Bluetooth fills the niche of very short-range communication between mobile phones, PDAs, notebook computers, and other personal or peripheral devices. For example, Bluetooth can be used to connect a mobile phone to a headset, or a notebook computer to a printer.

Roughly speaking, Bluetooth is a more convenient alternative to connecting two devices with a wire. In such applications, it is not necessary to provide much range or bandwidth. This is fortunate for some of the target battery- powered devices, since it is important that they not consume much power.

Bluetooth operates in the license-exempt band at 2.45 GHz. It has a range of only about 10 m. For this reason, and because the communicating devices typically belong to one individual or group, Bluetooth is sometimes categorized as a personal area network (PAN). Version 2.0 provides speeds up to 2.1 Mbps. Power consumptions are low.

Bluetooth is specified by an industry consortium called the Bluetooth Special Interest Group. It specifies an entire suite of protocols, going beyond the link layer to define application protocols, which it calls profiles, for a range of applications. For example, there is a profile for synchronizing a PDA with a personal computer. Another profile gives a mobile computer access to a wired LAN in the manner of 802.11, although this was not Bluetooth's original goal.

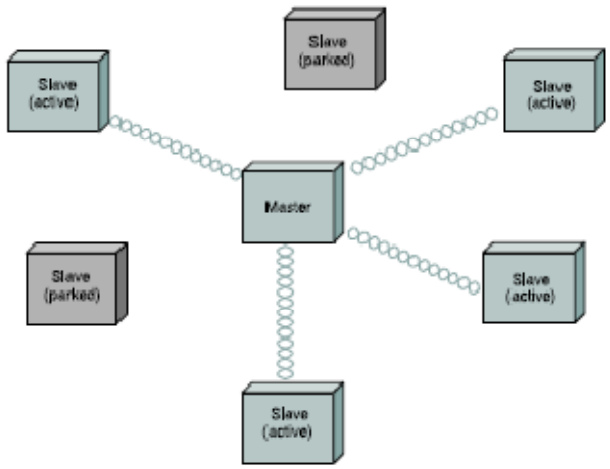


Figure: A Bluetooth piconet

The basic Bluetooth network configuration, called a piconet, consists of a master device and up to seven slave devices, as in Figure 2.40. Any communication is between the master and a slave; the slaves do not communicate directly with each other. Because slaves have a simpler role, their Bluetooth hardware and software can be simpler and cheaper.

The master can start to transmit in odd-numbered slots. A slave can start to transmit in an even- numbered slot, but only in response to a request from the master during the previous slot, thereby preventing any contention between the slave devices.

A slave device can be parked: set to an inactive, low-power state. A parked device cannot communicate on the piconet; it can only be reactivated by the master. A piconet can have up to 255 parked devices in addition to its active slave devices.

ZigBee is a newer technology that competes with Bluetooth to some extent. Devised by the ZigBee alliance and standardized as IEEE 802.15.4, it is designed for situations where the bandwidth requirements are low and power consumption must be very low to give very long battery life.

It is also intended to be simpler and cheaper than Bluetooth, making it financially feasible to incorporate in cheaper devices such as a wall switch that wirelessly communicates with a ceiling-mounted fan.

WI-FI (802.11)

Introduction:

This section takes a closer look at a specific technology centered around the emerging IEEE 802.11 standard, also known as Wi-Fi.6Wi-Fi is technically a trademark, owned a trade group called the Wi- Fi alliance, that certifies product compliance with 802.11.

Like its Ethernet and token ring siblings, 802.11 is designed for use in a limited geographical area (homes, office buildings, campuses), and its primary challenge is to mediate access to a shared communication medium—in this case, signals propagating through space. 802.11 supports additional features (e.g., time-bounded services, power management, and security mechanisms), but we focus our discussion on its base functionality.

Physical Properties:

802.11 run over six different physical layer protocols (so far). Five are based on spread spectrum radio, and one on diffused infrared (and is of historical interest only at this point). The fastest runs at a maximum of 54 Mbps. The original 802.11 standard defined two radio-based physical layers standards, one using frequency hopping (over 79 1-MHz-wide frequency bandwidths) and the other using direct sequence (with an 11-bit chipping sequence).

Both provide up to2 Mbps. Then physical layer standard 802.11b was added. Using a variant of direct sequence, 82.11b provides up to 11 Mbps. These three standards run in the licenseexempt2.4 GHz frequency band of the electromagnetic spectrum. Then came 802.11a, which delivers up to 54 Mbps using a variant of FDM called orthogonal frequency division multiplexing (OFDM).

802.11a runs in the license-exempt 5-GHz band. On one hand, this band is less used, so there is less interference. On the other hand, there is more absorption of the signal and it is limited to almost line of sight. The most recent standard is 802.11g, which is backward compatible with 802.11b (and returns to the 2.4-GHzband).

802.11g uses OFDM and delivers up to 54 Mbps. It is common for commercial products to support all three of 802.11a, 802.11b, and 802.11g, which not only ensures compatibility with any device that supports any one of the standards, but also makes it possible for two such products to choose the highest bandwidth option for a particular environment.

Collision Avoidance:

At first glance, it might seem that a wireless protocol would follow the same algorithm as the Ethernet—wait until the link becomes idle before transmitting and back off should a collision occur— and to a first approximation, this is what 802.11 does. The additional complication for wireless is that, while a node on an Ethernet receives every other node's transmissions, a node on an 802.11 network may be too far from certain other nodes to receive their transmissions (and vice versa).

The peculiar thing about the 802.11 frame format is that it contains four, rather than two, addresses. How these addresses are interpreted depends on the settings of the To DS and From DS bits in the frame's Control field.

This is to account for the possibility that the frame had to be forwarded across the distribution system, which would mean that the original sender is not necessarily the same as the most recent transmitting node. Similar reasoning applies to the destination address.

In the simplest case, when one node is sending directly to another, both the DS bits are 0, Addr1 identifies the target node, and Addr2 identifies the source node. In the most complex case, both DS bits are set to 1, indicating that the message went from a wireless node onto the distribution system, and then from the distribution system to another wireless node.

WIMAX (802.16)

Introduction:

WiMAX, which stands for Worldwide Interoperability for Microwave Access, was designed by the WiMAX Forum and standardized as IEEE 802.16. It was originally conceived as a last-mile technology. In WiMAX's case that "mile" is typically1 to 6 miles, with a maximum of about 30 miles, leading to WiMAX being classified as a metropolitan area network (MAN).

In keeping with a last-mile role, WiMAX does not incorporate mobility at the time of this writing, although efforts to add mobility are nearing completion as IEEE 802.16e. Also in keeping with the last-mile niche, WiMAX's client systems, called subscriber stations, are assumed to be not end-user computing devices, but rather systems that multiplex all the communication of the computing devices being used in a particular building.

WiMAX provides up to 70 Mbps to a single subscriber station. In order to adapt to different frequency bands and different conditions, WiMAX defines several physical layer protocols. The original WiMAX physical layer protocol is designed to use frequencies in the 10- to 66- GHz range. In this range waves travel in straight lines, so communication is limited to line-of-sight (LOS). A WiMAX base station uses multiple antennas pointed in different directions; the area covered.

Cell Phone Technologies:

Cell phone technology seems an obvious approach to mobile computer communication, and indeed data services based on cellular standards are commercially available. One drawback is the cost to users, due in part to cellular's use of licensed spectrum (which has historically been sold off to cellular phone operators for astronomical sums).

The frequency bands that are used for cellular telephones (and now for cellular data) vary around the world. In Europe, for example, the main bands for cellular phones are at 900 and 1,800 MHz

In North America, 850- and 1,900-MHz bands are used. This global variation in spectrum usage creates problems for users who want to travel from one part of the world to another, and has created a market for phones that can operate at multiple frequencies (e.g., a tri-band phone can operate at three of the four frequency bands mentioned above). That problem, however, pales in comparison to the proliferation of incompatible standards that have plagued the cellular communication business.

Only recently have some signs of convergence on a small set of standards appeared. And finally, there is the problem that most cellular technology was designed for voice communication, and is only now starting to support moderately high-bandwidth data communication. Like 802.11 and WiMAX, cellular technology relies on the use of base stations that are part of a wired network.

The geographic area served by a base station's antenna is called a cell. A base station could serve a single cell, or use multiple directional antennas to serve multiple cells. Cells don't have crisp boundaries, and they overlap. Where they overlap, a mobile phone could potentially communicate with multiple base stations.

The current base station senses the weakening signal from the phone, and gives control of the phone to whichever base station is receiving the strongest signal from it. If the phone is involved in a call at the time, the call must be transferred to the new base station in what is called a handoff. As we noted above, there is not one unique standard for cellular, but rather a collection of competing technologies that support data traffic in different ways and deliver different speeds.

These technologies are loosely categorized by "generation." The first generation (1G) was analog, and thus of limited interest from a data communications perspective. Most of the cell phone technology currently deployed is considered second generation (2G) or "2.5G" (not quite worthy of being called 3G, but more advanced than 2G). The 2G and later technologies are digital.

The most widely deployed 2G technology is referred to as GSM—the Global System for Mobile Communications, which is used in more than 200 countries. North America, however, is a late adopter of GSM, which helped prolong the proliferation of competing standards. 2G technologies use one of two approaches sharing a limited amount of spectrum between simultaneous calls. One way is a combination of FDM and TDM.

The spectrum available is divided into disjoint frequency bands, and each band is subdivided into time slots. A given call is allocated every nth slot in one of the bands. The other approach is code division multiple access (CDMA). CDMA does not divide the channel in either time or frequency, but rather uses different chipping codes to distinguish the transmissions of different cell phone users.

The 2G and later cell phone technologies use compression algorithms tailored to human speech to compress voice data to about 8 Kbps without losing quality. Since 2G technologies focus on voice communication, they provide connections with just enough bandwidth for that compressed speech—not enough for a decent data link.

SWITCHING AND FORWARDING

In the simplest terms, a switch is a mechanism that allows us to interconnect links to form a larger network. A switch is a multi-input, multi output device, which transfers packets from an input to one or more outputs. Thus, a switch adds the star topology to the point-to-point link, bus (Ethernet), and ring (802.5, 802.17, and FDDI) topologies established in the last chapter. A star topology has several attractive properties:

- ✓ Even though a switch has a fixed number of inputs and outputs, which limits the number of hosts that can be connected to a single switch, large networks can be built by interconnecting a number of switches;

- ✓ We can connect switches to each other and to hosts using point-to-point links, which typically means that we can build networks of large geographic scope;

- ✓ Adding a new host to the network by connecting it to a switch does not necessarily reduce the performance of the network for other hosts already connected.

A switch is connected to a set of links and, for each of these links, runs the appropriate data link protocol to communicate with the node at the other end of the link.

A switch's primary job is to receive incoming packets on one of its links and to transmit them on some other link. This function is sometimes referred to as either switching or forwarding, and in terms of the OSI architecture, it is the main function of the network layer.

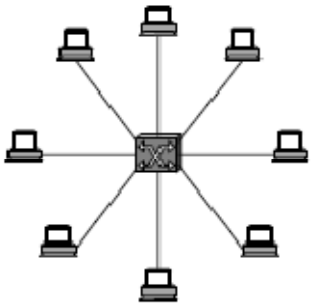


Figure: A switch provides a star topology

Source routes are sometimes categorized as "strict" or "loose." In a strict source route, every node along the path must be specified, whereas a loose source route only specifies a set of nodes to be traversed, without saying exactly how to get from one node to the next.

A loose source route can be thought of as a set of waypoints rather than a completely specified route. The loose option can be helpful to limit the amount of information that a source must obtain to create a source route. In any reasonably large network, it is likely to be hard for a host to get the complete path information it needs to correctly construct a strict source route to any destination.

Bridges and LAN Switches:

A class of switch that is even some newer types of optical switch that use microscopic, electronically controlled mirrors to deflect all the light from one switch port to another, so that there could be an uninterrupted optical channel from point A to point B. The technology behind these devices is called MEMS (Micro electro-mechanical Systems).

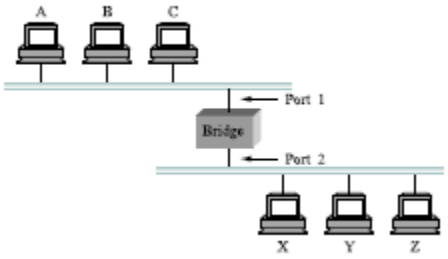


Figure: Illustration of a learning bridge

This would not be a workable solution, however, if doing so exceeded the physical limitations of the Ethernet.(Recall that no more than four repeaters between any pair of hosts and no more than a total of 2,500 m in length are allowed.)

An alternative would be to put a node between the two Ethernets and have the node forward frames from one Ethernet to the other. This node would be in promiscuous mode, accepting all frames transmitted on either of the Ethernets, so it could forward them to the other. The node we have just described is typically called a bridge, and a collection of LANs connected by one or more devices.

INTERNETWORKING

SIMPLE INTERNETWORKING (IP)

Internetwork:

The term “internetwork” or sometimes just “internet” with a lowercase i, to refer to a arbitrary collection of networks interconnected to provide some sort of host-to-host packet delivery service. For example, a corporation with many sites might construct a private internetwork by interconnecting the LANs at their different sites with point-to-point links leased from the phone company.

For now, we use network to mean either a directly connected or a switched network of the kind that was discussed in the last two chapters. Such a network uses one technology, such as 802.5, Ethernet, or ATM.

An internetwork is an interconnected collection of such networks. Sometimes, to avoid ambiguity, we refer to the underlying networks that we are interconnecting as physical networks. An internet is a logical network built out of a collection of physical networks. In this context, a collection of Ethernets connected by bridges or switches would still be viewed as a single network.

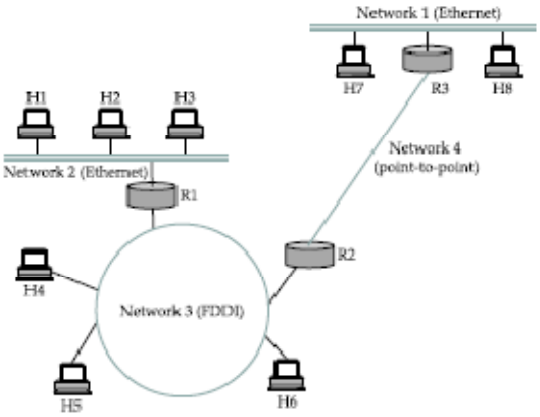


Figure: A simple network

The above figure shows an example internetwork. An internetwork is often referred to as a network of networks because it is made up of lots of smaller networks. In this figure, we see Ethernets, an FDDI ring, and a point-to-point link. Each of these is a single technology network. The nodes that interconnect the networks are called routers.

Service Model:

A good place to start when you build an internetwork is to define its service model, that is, the host-to- host services you want to provide. The main concern in defining a service model for an internetwork is that we can provide a host-to-host service only if this service can somehow be provided over each of the underlying physical networks.

For example, it would be no good deciding that our internetwork service model was going to provide guaranteed delivery of every packet in 1 ms or less if there were underlying network technologies that could arbitrarily delay packets.

Datagram Delivery:

Every datagram carries enough information to let the network forward the packet to its correct destination; there is no need for any advance setup mechanism to tell the network what to do when the packet arrives.

The important thing about IP addresses is that they are what is carried in the headers of IP packets, and it is those addresses that are used in IP routers to make forwarding decisions.

Datagram Forwarding in IP:

The discussion here focuses on forwarding; we take up routing.

The main points to bear in mind as we discuss the forwarding of IP datagram are the following:

- ✓ Every IP datagram contains the IP address of the destination host;
- ✓ The "network part" of an IP address uniquely identifies a single physical network that is part of the larger Internet;
- ✓ All hosts and routers that share the same network part of their address are connected to the same physical network and can thus communicate with each other by sending frames over that network.

Every physical network that is part of the Internet has at least one router that, by definition, is also connected to at least one other physical network; this router can exchange packets with hosts or routers on either network. Forwarding IP datagrams can therefore be handled in the following way.

A datagram is sent from a source host to a destination host, possibly passing through example that meant that R2 could store the information needed to reach all the hosts in the network (of which there were eight) in a four-entry table. Even if there were 100 hosts on each physical network, R2 would still only need those same four entries.

ADDRESS TRANSLATION (ARP)

Introduction:

ARP takes advantage of the fact that many link-level network technologies, such as Ethernet and token ring, support broadcast. If a host wants to send an IP datagram to a host (or router) that it knows to be on the same network (i.e., the sending and receiving node have the same IP network number), it first checks for a mapping in the cache.

If no mapping is found, it needs to invoke the Address Resolution Protocol over the network. It does this by broadcasting an ARP query onto the network. This query contains the IP address in question (the target IP address). Each host receives the query and checks to see if it matches its IP address. If it does match, the host sends a response message that contains its link-layer address back to the originator of the query.

The originator adds the information contained in this response to its ARP table. The query message also includes the IP address and link-layer address of the sending host. Thus, when a host broadcasts a query message, each host on the network can learn the sender's link-level and IP addresses and place that information in its ARP table.

However, not every host adds this information to its ARP table. If the host already has an entry for that host in its table, it "refreshes" this entry, that is, it resets the length of time until it discards the entry. If that host is the target of the query, then it adds the information about the sender to its table, even if it did not already have an entry for that host.

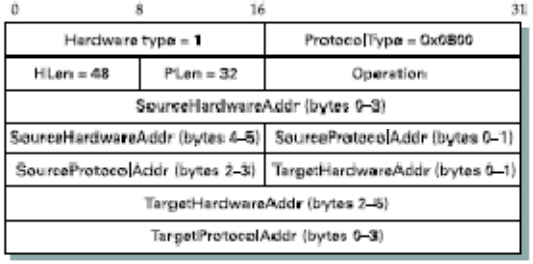


Figure: ARP packet format

In fact, ARP can be used for lots of other kinds of mappings—the major differences are in the address sizes. In addition to the IP and link-layer addresses of both sender and target, the packet contains

- ✓ A Hardware Type field, which specifies the type of physical network (e.g., Ethernet);
- ✓ A Protocol Type field, which specifies the higher-layer protocol (e.g., IP); the same administrative entity. The division of the ATM network into a number of LISs also improves scalability by limiting the number of nodes that must be supported by a single ARP server.

- ✓ The basic job of an ARP server is to enable nodes on a LIS to resolve IP addresses to ATM addresses without using broadcast. Each node in the LIS must be configured with the ATM addresses of the ARP server so that it can establish a VC to the server when it boots. Once it has a VC to the server, the node sends a registration message to the ARP server that contains both the IP and ATM addresses of the registering node.

An interesting consequence of the classical IP over ATM model is that two nodes on the same ATM network cannot establish a direct VC between themselves if they are on different subnets. This would violate the rule that communication from one subnet to another must pass through a router.

On the issue of heterogeneity, IP begins by defining a best-effort service model that makes minimal assumptions about the underlying networks; most notably, this service model is based on unreliable datagrams.

IP then makes two important additions to this starting point: (1) a common packet format (fragmentation/reassembly is the mechanism that makes this format work over networks with different MTUs), and (2) a global address space for identifying all hosts (ARP is the mechanism that makes this global address space work over networks with different physical addressing schemes).

On the issue of scale, IP uses hierarchical aggregation to reduce the amount of information needed to forward packets. Specifically, IP addresses are partitioned into network and host components, with packets first routed toward the destination network and then delivered to the correct host on that network.

HOST CONFIGURATION (DHCP)

Dynamic Host Configuration Protocol:

The primary method uses a protocol known as the Dynamic Host Configuration Protocol (DHCP). DHCP relies on the existence of a DHCP server that is responsible for providing configuration information to hosts. There is at least one DHCP server for an administrative domain.

At the simplest level, the DHCP server can function just as a centralized repository for host configuration information. Consider, for example, the problem of administering addresses in the internet work of a large company. DHCP saves the network administrators from having to walk around to every host in the company with a list of addresses and network map in hand and configuring each host manually.

A more sophisticated use of DHCP saves the network administrator from even having to assign addresses to individual hosts. In this model, the DHCP server maintains a pool of available addresses that it hands out to hosts on demand.

This considerably reduces the amount of configuration an administrator must do, since now it is only necessary to allocate a range of IP addresses (all with the same network number) to each network. Since the goal of DHCP is to minimize the amount of manual configuration required for a host to function, it would rather defeat the purpose if each host had to be configured with the address of a DHCP server.

Thus, the first problem faced by DHCP is that of server discovery. To contact a DHCP server, a newly booted or attached host sends a DHCPDISCOVER message to a special IP address (255.255.255.255) that is an IP broadcast address.

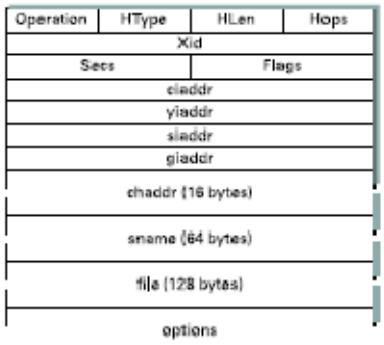


Figure: DHCP packet format

There is at least one relay agent on each network, and it is configured with just one piece of information: the IP address of the DHCP server. When a relay agent receives a DHCPDISCOVER message, it unicasts it to the DHCP server and awaits the response, which it will then send back to the requesting client.

The process of relaying a message from a host to a remote DHCP server. The message is actually sent using a protocol called the User Datagram Protocol (UDP) that runs over IP. UDP is discussed in detail in the next chapter, but the only interesting thing it does in this context is to provide a demultiplexing key that says, “This is a DHCP packet.”

DHCP is derived from an earlier protocol called BOOTP, and some of the packet fields are thus not strictly relevant to host configuration. When trying to obtain configuration information, the client puts its hardware address (e.g., its Ethernet address) in the chaddr field. The DHCP server replies by filling in the yiaddr (“your” IP address) field and sending it to the client.

Other information such as the default router to be used by this client can be included in the options field. In the case where DHCP dynamically assigns IP addresses to hosts, it is clear that hosts cannot keep addresses indefinitely, as this would eventually cause the server to exhaust its address pool.

.

ERROR REPORTING (ICMP)

IP is always configured with a companion protocol, known as the Internet Control Message Protocol (ICMP) that defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully.

For example, ICMP defines error messages indicating that the destination host is unreachable (perhaps due to a link failure), that the reassembly process failed, and that the TTL had reached 0 that the IP header checksum failed, and so on.

ICMP also defines a handful of control messages that a router can send back to a source host. One of the most useful control messages, called an ICMP-Redirect, tells the source host that there is a better route to the destination. ICMP-Redirects are used in the following situation.

Suppose a host is connected to a network that has two routers attached to it, called R1 and R2, where the host uses R1 as its default router. Should R1 ever receive a datagram from the host, where based on its forwarding table it knows that R2 would have been a better choice for a particular destination address, it sends an ICMP-Redirect back to the host, instructing it to use R2 for all future datagrams addressed to that destination. The host then adds this new route to its forwarding table.

UNIT – 3: ROUTING

DISTANCE VECTOR (RIP)

Each node constructs a one- dimensional array (a vector) containing the "distances" (costs) to all other nodes and distributes that vector to its immediate neighbors. The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors. A link that is down is assigned an infinite cost. To see how a distance-vector routing algorithm works, it is easiest to consider an example like the one depicted in the following figure.

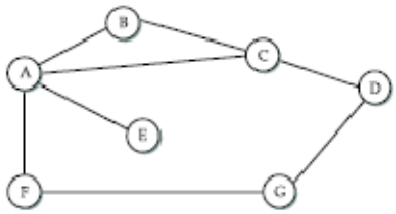


Figure: Distance-vector routing an example network

In this example, the cost of each link is set to 1, so that a least-cost path is simply the one with the fewest hops.

Destination	Cost	Next Hop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Figure: Initial routing table at node A

F tells node A that it can reach node G at a cost of 1; A also knows it can reach F at a cost of 1, so it adds these costs to get the cost of reaching G by means of F. This total cost of 2 is less than the current cost of infinity, so A records that it can reach G at a cost of 2 by going through F.

Similarly, A learns from C that D can be reached from C at a cost of 1; it adds this to the cost of reaching C (1) and decides that D can be reached via C at a cost of 2, which is better than the old cost of infinity. At the same time, A learns from C that B can be reached from C at a cost of 1, so it concludes that the cost of reaching B via C is 2.

The second mechanism, sometimes called a triggered update, happens whenever a node receives an update from one of its neighbors that causes it to change one of the routes in its routing table.

That is, whenever a node's routing table changes, it sends an update to its neighbors, which may lead to a change in their tables, causing them to send an update to their neighbors. Now consider what happens when a link or node fails.

The nodes that notice first send new lists of distances to their neighbors, and normally the system settles down fairly quickly to a new state. As to the question of how a node detects a failure, there are a couple of different answers.

In one approach, a node continually tests the link to another node by sending a control packet and seeing if it receives an acknowledgment. In another approach, a node determines that the link (or the node at the other end of the link) is down if it does not receive the expected periodic routing update for the last few update cycles.

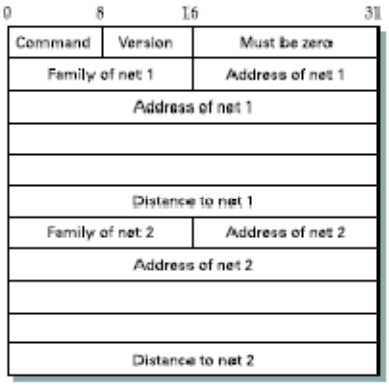


Figure: RIP packet format

Suppose, for example, that the link from A to E goes down. In the next round of updates, A advertises a distance of infinity to E, but B and C advertise a distance of 2One technique to improve the time to stabilize routing is called split horizon.

The idea is that when a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor. For example, if B has the route (E, 2, A) in its table, then it knows it must have learned this route from A, and so whenever B sends a routing update to A, it does not include the route (E, 2) in that update.

In a stronger variation of split horizon, called split horizon with poison reverse, B actually sends that route back to A, but it puts negative information in the route to ensure that A will not eventually use B to get to E. For example, B sends the route (E, ∞) to A.

LINK STATE (OSPF)

Introduction:

Link-state routing is the second major class of intra-domain routing protocol. The starting assumptions for link-state routing are rather similar to those for distance-vector routing. Each node is assumed to be capable of finding out the state of the link to its neighbors (up or down) and the cost of each link.

The basic idea behind link-state protocols is very simple: Every node knows how to reach its directly connected neighbors, and if we make sure that the totality of this knowledge is disseminated to every node, then every node will have enough knowledge of the network to build a complete map of the network.

Reliable Flooding:

Reliable flooding is the process of making sure that all the nodes participating in the routing protocol get a copy of the link-state information from all the other nodes. As the term "flooding" suggests, the basic idea is for a node to send its link-state information out on its entire directly connected links, with each node that receives this information forwarding it out on its entire links.

This process continues until the information has reached all the nodes in the network. More precisely, each node creates an update packet, also called a link-state packet (LSP) that contains the following information:

The ID of the node that created the LSP;

- ✓ A list of directly connected neighbors of that node, with the cost of the link to each one;
- ✓ A sequence number;
- ✓ A time to live for this packet.

The first two items are needed to enable route calculation; the last two are used to make the process of flooding the packet to all nodes reliable. Reliability includes making sure that you have the most recent copy of the information, since there may be multiple, contradictory LSPs from one node traversing the network.

In addition, it is clearly desirable to minimize the total amount of routing traffic that is sent around the network; after all, this is just "overhead" from the perspective of those who actually use the network for their applications. The next few paragraphs describe some of the ways that these goals are accomplished.

One easy way to reduce overhead is to avoid generating LSPs unless absolutely necessary. This can be done by using very long timers—often on the order of hours—for the periodic generation of LSPs. Given that the flooding protocol is truly reliable when topology changes, it is safe to assume that messages saying "nothing has changed" do not need to be sent very often.

Each time a node generates a new LSP, it increments the sequence number by 1. Unlike most sequence numbers used in protocols, these sequence are not expected to wrap, so the field needs to be quite large (say, 64 bits). If a node goes down and then comes back up, it starts with a sequence number of 0.

If the node was down for a long time, all the old LSPs for that node will have timed out (as described below); otherwise, this node will eventually receive a copy of its own LSP with a higher sequence number, which it can then increment and use as its own sequence number.

A node always decrements the TTL of a newly received LSP before flooding it to its neighbors. It also "ages" the LSP while it is stored in the node. When the TTL reaches 0, the node refloods the LSP with a TTL of 0, which is interpreted by all the nodes in the network as a signal to delete that LSP.

Route Calculation:

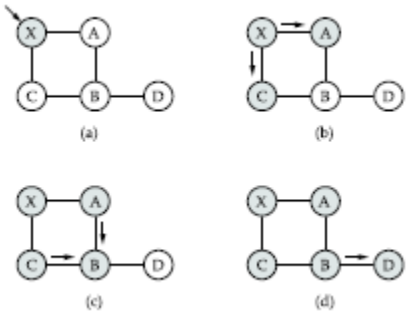


Figure: Routing of link-state packets

We first define Dijkstra's algorithm in graph-theoretic terms. Imagine that a node takes all the LSPs it has received and constructs a graphical representation of the network, in which N denotes the set of nodes in the graph, and l(i, j) denotes the nonnegative cost (weight) associated with the edge between nodes i, j ∈ N, and l(i, j)=∞ if no edge connects i and j.

In the following description, we let s ∈ N denote this node, that is, the node executing the algorithm to find the shortest path to all the other nodes in N. Given these definitions, the algorithm is defined as follows:

M = {s}

for each n in N − {s}

$C(n) = l(s, n)$ while $(N \_= M)$

$M = M \cup \{w\}$ such that $C(w)$ is the minimum for all w in $(N - M)$ for each n in $(N - M)$

$C(n) = MIN(C(n), C(w) + l(w, n))$

Each of these lists contains a set of entries of the form (Destination, Cost, NextHop). The algorithm works as follows:

1. Initialize the Confirmed list with an entry for myself; this entry has a cost of 0.

2. For the node just added to the Confirmed list in the previous step, call it node Next, select its LSP.

3. For each neighbor (Neighbor) of Next, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor.

The link-state routing algorithm has many nice properties: It has been proven to stabilize quickly, it does not generate much traffic, and it responds rapidly to topology changes or node failures. On the downside, the amount of information stored at each node (one LSP for every other node in the network) can be quite large.

METRICS

The ARPANET was the testing ground for a number of different approaches to link-cost calculation. (It was also the place where the superior stability of link-state over distance-vector routing was demonstrated; the original mechanism used distance vector while the later version used link state.)

The original ARPANET routing metric measured the number of packets that were queued waiting to be transmitted on each link, meaning that a link with 10 packets queued waiting to be transmitted was assigned a larger cost weight than a link with 5packets queued for transmission.

Using queue length as a routing metric did not work well, however, since queue length is an artificial measure of load—it moves packets toward the shortest queue rather than toward the destination, a situation all too A second version of the ARPANET routing algorithm, sometimes called the "new routing mechanism," took both link bandwidth and latency into consideration and used delay, rather than just queue length, as a measure of load. This was done as follows.

First, each incoming packet was timestamped with its time of arrival at the router (ArrivalTime); its departure time from the router (Depart Time) was also recorded. Second, when the link-level ACK was received from the other side, the node computed the delay for that packet as

Delay = (Depart Time− ArrivalTime) +Transmission Time +Latency

where Transmission Time and Latency were statically defined for the link an captured the link's bandwidth and latency, respectively. Notice that in this case, DepartTime − ArrivalTime represents the amount of time the packet was delayed (queued) in the node due to load.

Finally, the weight assigned to each link was derived from the average delay experienced by the packets recently sent over that link. Although an improvement over the original mechanism, this approach also had a lot of problems. Under light load, it worked reasonably well, since the two static factors of delay dominated the cost.

Under heavy load, however, a congested link would start to advertise a very high cost. This caused all the traffic to move off that link, leaving it idle, so then it would advertise a low cost, thereby attracting back all the traffic, and so on. The smoothing was achieved by several mechanisms.

First, the delay measurement was transformed to link utilization, and this number was averaged with the last reported utilization to suppress sudden changes.

Second, there was a hard limit on how much the metric could change from one measurement cycle to the next. By smoothing the changes in the cost, the likelihood that all nodes would abandon a route at once is greatly reduced.

The compression of the dynamic range was achieved by feeding the measured utilization, the link type, and the link speed into a function that is shown graphically in the following figure.

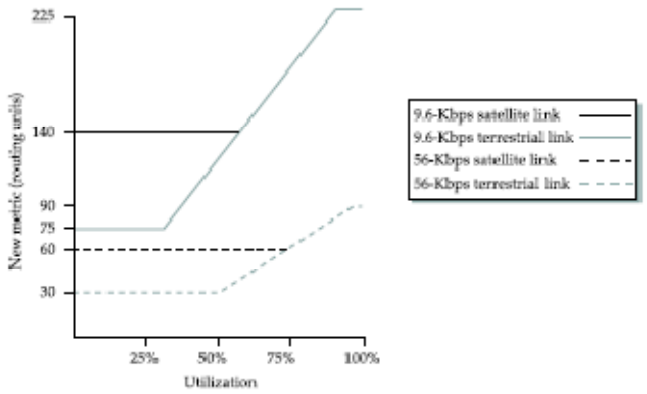


Figure: Revised ARPANET routing metric versus link utilization

Observe the following:

- ✓ A highly loaded link never shows a cost of more than three times its cost when idle;
- ✓ The most expensive link is only seven times the cost of the least expensive;
- ✓ A high-speed satellite link is more attractive than a low-speed terrestrial link;
- ✓ Cost is a function of link utilization only at moderate to high loads.

GLOBAL INTERNET

Today's Internet has tens of thousands of networks connected to it. Routing protocols such as those we have just discussed do not scale to those kinds of numbers. This section looks at a variety of techniques that greatly improve scalability and that have enabled the Internet to grow as far as it has. Before getting to these techniques, we need to have a general picture in our heads of what the global Internet looks like. It is not just a random interconnection of Ethernets, but instead it takes on a shape that reflects the fact that it interconnects many different organizations. The following figure gives a simple depiction of the state of the Internet in 1990.

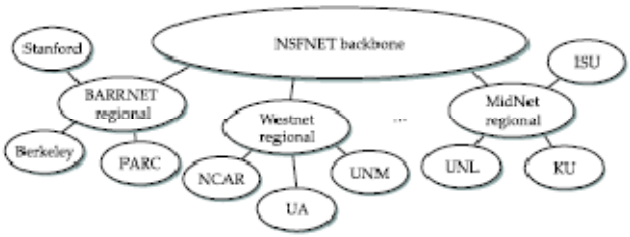


Figure: The tree structure of the internet in 1990

Since that time, the Internet's topology has grown much more complex than this figure suggests—we One of the salient features of this topology is that it consists of "end user" sites (e.g., Stanford University) that connect to "service provider" networks (e.g., BARRNET was a provider network that served sites in the San Francisco Bay area).

In 1990, many providers served a limited geographic region and were thus known as regional networks. Systems in the Internet is routing domains, we refer to the two parts of the routing problem as inter-domain routing and intra-domain routing. In addition to improving scalability, the AS model decouples the intra-domain routing that takes place in one AS from that taking place in another. Thus, each AS can run whatever intra-domain routing protocols it chooses. It can even use static routes or multiple protocols if desired.

The inter-domain routing problem is then one of having different ASs share reachability information— descriptions of the set of IP addresses that can be reached via a given AS—with each other. Perhaps the most important challenge of inter-domain routing today is the need for each AS to determine its own routing policies. A simple example routing policy implemented at a particular AS might look like this: Whenever possible, I prefer to send traffic via AS X than via AS Y, but I'll use AS Y if it is the only path, and I never want to carry traffic from AS X to AS Y or vice versa.

Note that in this simple treelike structure, there is a single backbone, and autonomous systems are connected only as parents and children and not as peers. The replacement for EGP is the Border Gateway Protocol (BGP), which is in its fourth version at the time of this writing (BGP-4).

BGP is also known for being rather complex. This section presents the highlights of BGP-4. As a starting position, BGP assumes that the Internet is an arbitrarily interconnected set of ASs.

Unlike the simple tree-structured Internet. Today's Internet consists of an interconnection of multiple backbone networks (they are usually called service provider networks, and they are operated by private companies rather than the government), and sites are connected to each other in arbitrary ways.

Given this rough sketch of the Internet, if we define local traffic as traffic that originates at or terminates on nodes within an AS, and transit traffic as traffic that passes through an AS, we can classify ASs into three types:

- ✓ Stub AS: an AS that has only a single connection to one other AS; such an AS will only carry local traffic. The small corporation in Figure 4.29 is an example of a stub AS.

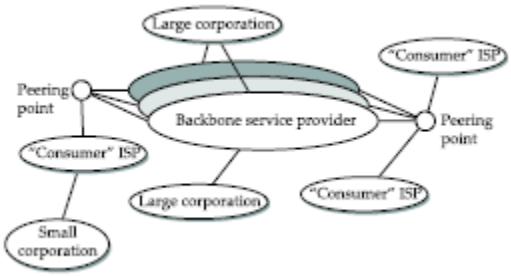


Figure: Today's multi-backbone internet

- ✓ Multihomed AS: an AS that has connections to more than one other AS but that refuses to carry transit traffic; for example, the large corporation.

- ✓ Transit AS: an AS that has connections to more than one other AS and that is designed to carry both transit and local traffic, such as the backbone providers.

First, it is necessary to find some path to the intended destination that is loop-free. Second, paths must be compliant with the policies of the various ASs along the path—and as we have already seen, those policies might be almost arbitrarily complex.

Thus, while intra-domain focuses on a well-defined problem of optimizing the scalar cost of the path, intra-domain focuses on finding the best, non-looping, policy-compliant path—a much more complex optimization problem.

There are additional factors that make inter-domain routing hard. The first is simply a matter of scale. An Internet backbone router must be able to forward any packet A second challenge in inter-domain routing arises from the autonomous nature of the domains. Note that each domain may run its own interior routing protocols, and use any scheme they choose to assign metrics to paths.

This means that it is impossible to calculate meaningful path costs for a path that crosses multiple ASs. A cost of 1,000 across one provider might imply a great path, but it might mean an unacceptably bad one from another provider.

As a result, inter-domain routing advertises only reachability. The concept of reachability is basically a statement that "you can reach this network through this AS." This means that for inter-domain routing to pick an optimal path is essentially impossible.

The third challenge involves the issue of trust. Provider A might be unwilling to believe certain advertisements from provider B for fear that provider B will advertise erroneous routing information.

For example, trusting provider B when he advertises a great route to anywhere in the Internet can be a disastrous choice if provider B turns out to have made a mistake configuring his routers or to have insufficient capacity to carry the traffic.

That BGP speaker establishes BGP sessions to other BGP speakers in other ASs. These sessions are used to exchange reachability information among ASs. In addition to the BGP speakers, the AS has one or more border gateways, which need not be the same as the speakers. The border gateways are the routers through which packets enter and leave the AS. In our simple example in routers R2 and R4 would be border gateways.

IP VERSION 6 (IPV6)

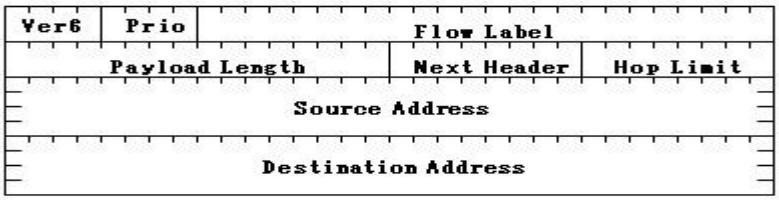
In many respects, the motivation for a new version of IP is the same as the motivation for the techniques described so far in this section: to deal with scaling problems caused by the Internet's massive growth. Subnetting and CIDR have helped to contain the rate at which the Internet address space is being consumed (the address depletion problem) and have also helped to control the growth of routing table information needed in the Internet's routers (the routing information problem).

However, there will come a point at which these techniques are no longer adequate. In particular, it is virtually impossible to achieve 100% address utilization efficiency, so the address space will be exhausted well before the 4 billionth host is connected to the Internet.

All of these possibilities argue that a bigger address space than that provided by 32 bits will eventually be needed. Support for real-time services;

- ✓ Security support;
- ✓ Autoconfiguration (i.e., the ability of hosts to automatically configure themselves with such information as their own IP address and domain name);
- ✓ Enhanced routing functionality, including support for mobile hosts.

IPv6 Header format



In addition to the wish list, one absolutely nonnegotiable feature for IPng was that there must be a transition plan to move from the current version of IP (version 4) to the new version.

With the Internet being so large and having no centralized control, it would be completely impossible to have a flag day on which everyone shut down their hosts and routers and installed a new version of IP. Thus, there will probably be a long transition period in which some hosts and routers will run IPv4 only, some will run IPv4 and IPv6, and some will run IPv6 only.

Addresses and Routing:

First and foremost, IPv6 provides a 128-bit address space, as opposed to the 32 bits of version 4. Thus, while version 4 can potentially address 4 billion nodes if address assignment efficiency reaches 100%, IPv6 can address 3.4×1038 nodes, again assuming 100% efficiency.

Address Space Allocation:

Drawing on the effectiveness of CIDR in IPv4, IPv6 addresses are also classless, but the address space is still subdivided in various ways based on the leading bits. Rather than specifying different address classes, the leading bits specify different uses of the IPv6 address. The current assignment of prefixes.

The multicast address space is (obviously) for multicast, thereby serving the same role as class D addresses in IPv4. Note that multicast addresses are easy to distinguish— they start with a byte of all 1s.

Prefix	Use
00...0 (128 bits)	Unspecified
00...1 (128 bits)	Loopback
1111 1111	Multicast addresses
1111 1110 10	Link local unicast
1111 1110 11	Site local unicast
Everything else	Global unicast

\

Figure: Address prefix assignments for IPv6

Within the global unicast address space are some important special types of addresses. A node may be assigned an IPv4-compatible IPv6 address by zero-extending a 32-bit IPv4 address to 128 bits. A node that is only capable of understanding IPv4 can be assigned an IPv4-mapped IPv6 address by prefixing the 32-bit IPv4 address with 2 bytes of all 1s and then zero-extending the result to 128 bits.

Address Notation:

Just as with IPv4, there is some special notation for writing down IPv6 addresses. The standard representation is x:x:x:x:x:x:x:x where each “x” is a hexadecimal representation of a 16-bit piece of the address. An example would be 47CD:1234:4422:ACO2:0022:1234:A456:0124 Any IPv6 address can be written using this notation. Since there are a few special types of IPv6 addresses, there are some special notations that may be helpful in certain circumstances

Thus,

47CD:0000:0000:0000:0000:0000:A456:0124 could be written

47CD::A456:0124

Packet Format:

Despite the fact that IPv6 extends IPv4 in several ways, its header format is actually simpler. This simplicity is due to a concerted effort to remove unnecessary functionality from the protocol.

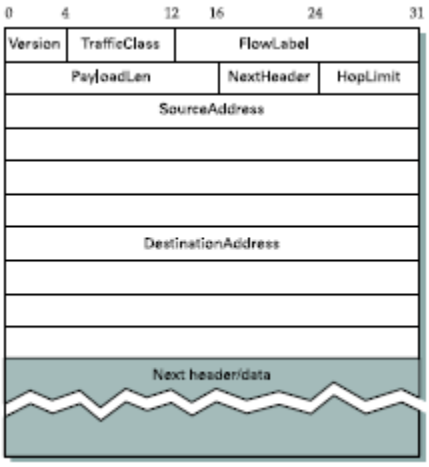


Figure: IPv6 packet header

The PayloadLen field gives the length of the packet, excluding the IPv6 header, measured in bytes. The NextHeader field cleverly replaces both the IP options and the Protocol field of IPv4.

If options are required, then they are carried in one or more special headers following the IP header, and this is indicated by the value of the NextHeader field. If there are no special headers, the NextHeader field is the demux key identifying the higher-level protocol running over IP.

In contrast, IPv6 treats options as extension headers that must, if present, appear in a specific order. This means that each router can quickly determine if any of the options are relevant to it; in most cases, they will not be. Usually this can be determined by just looking at the NextHeader field.

The end result is that option processing is much more efficient in IPv6, which is an important factor in router performance. In addition the new formatting of options as extension headers means that they can be of arbitrary length, whereas in IPv4 they were limited to 44 bytes at most.

MULTICAST-ADDRESSES

IP has a sub range of its address space reserved for multicast addresses. In IPv4, these addresses are assigned in the class D address space, and IPv6 also has a portion of its address reserved for multicast group addresses.

Some sub ranges of the multicast ranges are reserved for intra-domain multicast, so they can be reused independently by different domains.

Thus, there are 28 bits of possible multicast addresses in IPv4 when we ignore the prefix shared by all multicast addresses. This presents a problem when attempting to take advantage of hardware multicasting on a LAN.

Let's take the case of Ethernet. Ethernet multicast addresses have only 23 bits when we ignore their shared prefix. In other words, to take advantage of Ethernet multicasting, IP has to map 28-bit IP multicast addresses into 23-bit Ethernet multicast addresses.

This is implemented by taking the low- order 23 bits of any IP multicast address to use as its Ethernet multicast address, and ignoring the high-order 5 bits. Thus, 32 (25) IP addresses map into each one of the Ethernet addresses.

When a host on an Ethernet joins an IP multicast group, it configures its Ethernet interface to receive any packets with the corresponding Ethernet multicast address. Unfortunately, this causes the receiving host to receive not only the multicast traffic it desired, but also traffic sent to any of the other 31 IP multicast groups that map to the same Ethernet address, if they are routed to that Ethernet.

Therefore, IP at the receiving host must examine the IP header of any multicast packet to determine whether the packet really belongs to the desired group.

MULTICAST ROUTING (DVMRP, PIM)

Introduction:

A router's unicast forwarding tables indicate, for any IP address, which link to use to forward the unicast packet. To support multicast, a router must additionally have multicast forwarding tables that indicate, based on multicast address, which links—possibly more than one—to use to forward the multicast packet (the router duplicates the packet if it is to be forwarded over multiple links).

Thus, where unicast forwarding tables collectively specify a set of paths, multicast forwarding tables collectively specify a set of trees: multicast distribution trees. Furthermore, to support source-specific multicast (and, it turns out, for some types of any source multicast), the multicast forwarding tables must indicate which links to use based on the combination of multicast address and the (unicast) IP address of the source, again specifying a set of trees.

DVMRP:

DVMRP was the first multicast routing protocol to see widespread use. Recall that, in the distance- vector algorithm, each router maintains a table of _Destination, Cost, NextHop_ tuples, and exchanges a list of _Destination, Cost_ pairs with its directly connected neighbors. Extending this algorithm to support multicast is a two-stage process.

First, we create a broadcast mechanism that allows a packet to be forwarded to all the networks on the internet.

Second, we need to refine this mechanism so that it prunes back networks that do not have hosts that belong to the multicast group. Consequently, DVMRP is one of several multicast routing protocols described as flood-and-prune protocols. Given a unicast routing table, each router knows that the current shortest path to a given destination goes through NextHop.

Thus, whenever it receives a multicast packet from source S, the router forwards the packet on all outgoing links (except the one on which the packet arrived) if and only if the packet arrived over the link that is on the shortest path to S (i.e., the packet came from the NextHop associated with S in the routing table). This strategy effectively floods packets outward from S, but does not loop packets back toward S.

There are two major shortcomings to this approach. The first is that it truly floods the network; it has no provision for avoiding LANs that have no members in the multicast group.

The second limitation is that a given packet will be forwarded over a LAN by each of the routers connected to that LAN. This is due to the forwarding strategy of flooding packets on all links other than the one on which the packet arrived, without regard to whether or not those links are part of the shortest-path tree rooted at the source.

The solution to this second limitation is to eliminate the duplicate broadcast packets that are generated when more than one router is connected to a given LAN. One way to do this is to designate one router as the "parent" router for each link, relative to the source, where only the parent router is allowed to forward multicast packets from that source over the LAN.

Determining if any group members reside on the network is accomplished by having each host that is members of group G periodically announce this fact over the network, as described in our earlier description of link-state multicast. The router then uses this information to decide whether or not to forward a multicast packet addressed to G over this LAN.

The second stage is to propagate this "no members of G here" information up the shortest-path tree. This is done by having the router augment the _Destination, Cost _ pairs it sends to its neighbors with the set of groups for which the leaf network is interested in receiving multicast packets. This information can then be propagated from router to router, so that for each of its links, a given router knows for what groups it should forward multicast packets.

Note that including all of this information in the routing update is a fairly expensive thing to do. In practice, therefore, this information is exchanged only when some source starts sending packets to that group.

In other words, the strategy is to use RPB, which adds a small amount of overhead to the basic distance-vector algorithm, until a particular multicast address becomes active. At that time, routers that are not interested in receiving packets addressed to that group speak up, and that information is propagated to the other routers.

PIM-SM:

Protocol-independent multicast, or PIM, was developed in response to the scaling problems of earlier multicast routing protocols. In particular, it was recognized that the existing protocols did not scale well in environments where a relatively small proportion of routers want to receive traffic for a certain group.

For example, broadcasting traffic to all routers until they explicitly ask to be removed from the distribution is not a good design choice if most routers don't want to receive the traffic in the first place. This situation is sufficiently common that PIM divides the problem space into sparse mode and dense mode, where sparse and dense refer to the proportion of routers that will want the multicast.

PIM dense mode (PIM-DM) uses a flood-and-prune algorithm like DVMRP, and suffers from the same scalability problem. PIM sparse mode (PIM-SM) has become the dominant multicast routing protocol and is the focus of our discussion here.

A multicast forwarding tree is built as a result of routers sending Join messages to the RP. PIM-SM allows two types of tree to be constructed: a shared tree, which may be used by all senders, and a source-specific tree, which may be used only by a specific sending host.

The normal mode of operation creates the shared tree first, followed by one or more source-specific trees if there is enough traffic to warrant it. Because building trees installs state in the routers along the tree, it is important that the default is to have only one tree for a group, not one for every sender to a group. When a router sends a Join message toward the RP for a group G, it is sent using normal IP unicast transmission. This is illustrated in the following figure in which router R4 is sending a Join to the rendezvous point for some group.

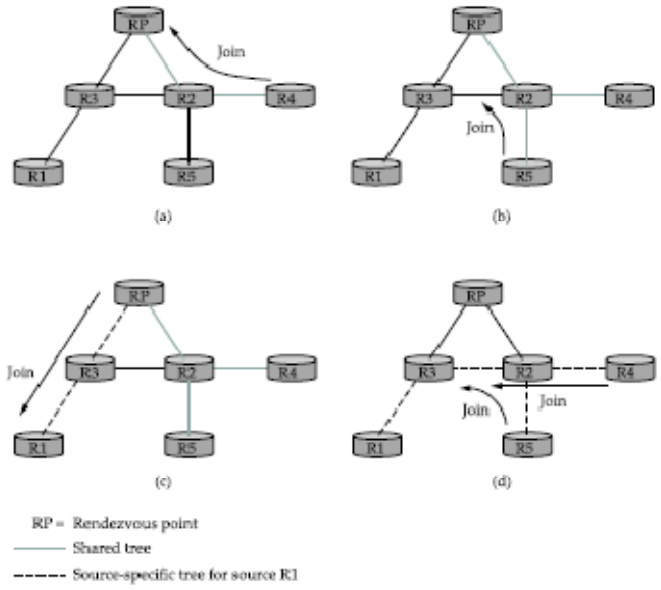


Figure: PIM operation

The initial Join message is "wildcarded," that is, it applies to all senders. A Join message clearly must pass through some sequence of routers before reaching the RP (e.g., R2). Each router along the path looks at the Join and creates a forwarding table entry for the shared tree, called a (*, G) entry (* meaning "all senders").

To create the forwarding table entry, it looks at the interface on which the Join arrived and marks that interface as one on which it should forward data packets for this group. It then determines which interface it will use to forward the Join toward the RP. This will be the only acceptable interface for incoming packets sent to this group. It then forwards the Join toward the RP. Eventually, the message arrives at the RP, completing the construction of the tree branch. The shared tree thus constructed is shown as a solid line from the RP to R4 in the figure.

As more routers send Joins toward the RP, they cause new branches to be added to the tree, as illustrated in the figure Note that in this case, the Join only needs to travel to R2, which can add the new branch to the tree simply by adding a new outgoing interface to the forwarding table entry created for this group. R2 need not forward the Join on to the RP. Note also that the end result of this process is to build a tree whose root is the RP.

UNIT – 4: TRANSPORT LAYER

OVERVIEW OF TRANSORT LAYER

The following list itemizes some of the common properties that a transport protocol can be expected to provide:

- ✓ Guarantees message delivery.
- ✓ Delivers messages in the same order they are sent.
- ✓ Delivers at most one copy of each message.
- ✓ Supports arbitrarily large messages.
- ✓ Supports synchronization between the sender and the receiver.
- ✓ Allows the receiver to apply flow control to the sender.
- ✓ Supports multiple application processes on each host.

From below, the underlying network upon which the transport protocol operates has certain limitations in the level of service it can provide.

Some of the more typical limitations of the network are that it may

- ✓ Drop messages.
- ✓ Reorder messages.
- ✓ Deliver duplicate copies of a given message.
- ✓ Limit messages to some finite size.
- ✓ Deliver messages after an arbitrarily long delay.

UDP

Simple Demultiplexer (UDP):

The client learns the server's port in the first place. A common approach is for the server to accept messages at a well-known port. That is, each server receives its messages at some fixed port that is widely published, much like the emergency telephone service available at the well-known phone number 911.

In the Internet, for example, the domain name server (DNS) receives messages at well-known port 53 on each host, the mail service listens for messages at port 25, and the Unix talk program accepts messages at well-known port 517, and so on.

This mapping is published periodically in an RFC and is available on most Unix systems in file /etc/services. Sometimes a well-known port is just the starting point for communication: The client and server use the well-known port to agree on some other port that they will use for subsequent communication, leaving the well-known port free for other clients.

An alternative strategy is to generalize this idea, so that there is only a single well-known port—the one at which the "port mapper" service accepts messages. A client would send a message to the port mapper's well-known port asking for the port it should use to talk to the "whatever" service, and the port mapper returns the appropriate port.

This strategy makes it easy to change the port associated with different services over time, and for each host to use a different port for the same service.

As just mentioned, a port is purely an abstraction. Exactly how it is implemented differs from system to system, or more precisely, from OS to OS. Typically, a port is implemented by a message queue, as illustrated in the following figure.

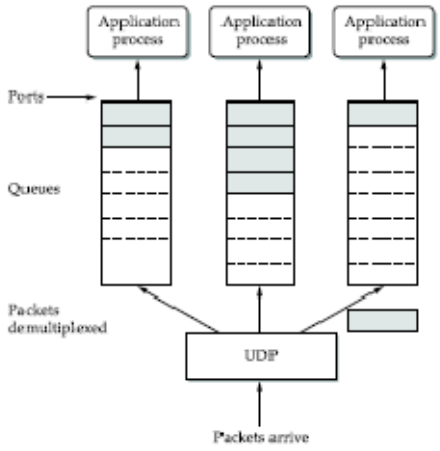


Figure: UDP message queue

When a message arrives, the protocol (e.g., UDP) appends the message to the end of the queue. Should the queue be full, the message is discarded. There is no flow-control mechanism that tells the sender to slow down.

When an application process wants to receive a message, one is removed from the front of the queue. If the queue is empty, the process blocks until a message becomes available.

Finally, although UDP does not implement flow control or reliable/ordered delivery, it does a little more work than to simply demultiplex messages to some application process—it also ensures the correctness of the message by the use of a checksum

UDP uses the same checksum algorithm as IP. The motivation behind having the pseudo header is to verify that this message has been delivered between the correct two endpoints. For example, if the destination IP address was modified while the packet was in transit, causing the packet to be misdelivered, this fact would be detected by the UDP checksum.

RELIABLE BYTE STREAM (TCP)

Introduction:

In contrast to a simple demultiplexing protocol like UDP, a more sophisticated transport protocol is one that offers a reliable, connection-oriented, byte-stream service. Such a service has proven useful to a wide assortment of applications because it frees the application from having to worry about missing or reordered data. The Internet's Transmission Control Protocol (TCP) is probably the most widely used protocol of this type; it is also the most carefully tuned.

In terms of the properties of transport protocols given in the problem statement at the start of this chapter, TCP guarantees the reliable, in-order delivery of a stream of bytes. It is a full-duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. It also includes a flow-control mechanism for each of these byte streams that allow the receiver to limit how much data the sender can transmit at a given time.

Segment Format:

TCP is a byte-oriented protocol, which means that the sender writes bytes into a TCP connection and the receiver reads bytes out of the TCP connection. Although "byte stream" describes the service TCP offers to application processes, TCP does not itself transmit individual bytes over the Internet.

Instead, TCP on the source host buffers enough bytes from the sending process to fill a reasonably sized packet and then sends this packet to its peer on the destination host. TCP on the destination host then empties the contents of the packet into a receive buffer, and the receiving process reads from this buffer at its leisure. This situation is illustrated in the following figure, which, for simplicity, shows combine to uniquely identify each TCP connection.

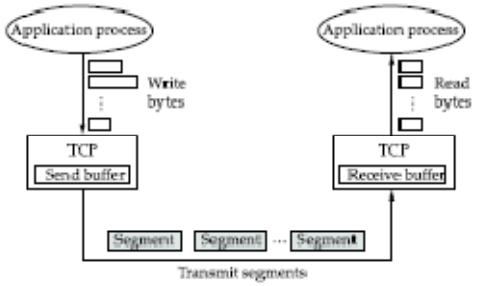


Figure: TCP manages byte stream

That is, TCP's demux key is given by the 4-tuple

_ SrcPort, SrcIPAddr, DstPort, DstIPAddr_

Note that because TCP connections come and go, it is possible for a connection between a particular pair of ports to be established, used to send and receive data, and closed, and then at a later time for the same pair of ports to be involved in a second connection. We sometimes refer to this situation as two different incarnations of the same connection.

The Acknowledgment, SequenceNum, and AdvertisedWindow fields are all involved in TCP's sliding window algorithm. Because TCP is a byte-oriented protocol, each byte of data has a sequence number; the SequenceNum field contains the sequence number for the first byte of data carried in that segment.

The Acknowledgment and AdvertisedWindow fields carry information about the flow of data going in the other direction. To simplify our discussion, we ignore the fact that data can flow in both directions, and we concentrate on data that has a particular SequenceNum flowing in one direction and Acknowledgment and AdvertisedWindow.

Three-Way Handshake:

The algorithm used by TCP to establish and terminate a connection is called a three-way handshake. We first describe the basic algorithm and then show how it is used by TCP. The three-way handshake involves the exchange of three messages between the client and the server, as illustrated by the timeline.

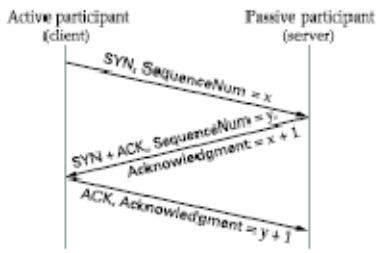


Figure: Timeline for three-way handshake algorithm

The idea is that two parties want to agree on a set of parameters, which, in the case of opening a TCP connection, are the starting sequence numbers the two sides plan to use for their respective byte streams. In general, the parameters might be any facts that each side wants the other to know about.

First, the client (the active participant) sends a segment to the server (the passive participant) stating the initial sequence number it plans to use (Flags = SYN, SequenceNum = x). then responds with a single segment that both acknowledges the client's sequence number (Flags = ACK, Ack = x + 1) and states its own beginning sequence number (Flags = SYN, SequenceNum = y). That is, both the SYN and ACK bits are set in the Flags field of this second message. Finally, the client responds with a third segment that acknowledges the server's sequence number (Flags = ACK, Ack = y + 1).

FLOW CONTROL

Most of the above discussion has the fact that data arriving from an upstream node was filling the send buffer, and data being transmitted to downstream node was emptying the receive buffer. You should make sure you understand this much before proceeding because now comes the point where the two algorithms differ more significantly.

In what follows, we reintroduce the fact that both buffers are of some finite size, denoted MaxSendBuffer and MaxRcvBuffer, although we don't worry about the details of how they are implemented.

In other words, we are only interested in the number of bytes being buffered, not in where those bytes are actually stored.

End-to-End Protocols:

All the while this is going on, the send side must also make sure that the local application process does not overflow the send buffer; that is, that LastByteWritten−LastByteAcked ≤ MaxSendBuffer

If the sending process tries to write y bytes to TCP, but (LastByteWritten−LastByteAcked) + y > MaxSendBuffer then TCP blocks the sending process and does not allow it to generate more data.

TCP on the receive side does not spontaneously send non-data segments; it only sends them in response to an arriving data segment. TCP deals with this situation as follows. Whenever the other side advertises a window size of 0, the sending side persists in sending a segment with 1 byte of data every so often.

It knows that this data will probably not be accepted, but it tries anyway, because each of these 1-byte segments triggers a response that contains the current advertised window. Eventually, one of these 1-byte probes triggers a response that reports a nonzero advertised window.

Note that the reason the sending side periodically sends this probe segment is that TCP is designed to make the receive side as simple as possible—it simply responds to segments from the sender, and it never initiates any activity on its own. This is an example of a well-recognized (although not universally applied) protocol design.

RETRANSMISSION

Adaptive Retransmission:

Because TCP guarantees the reliable delivery of data, it retransmits each segment if an ACK is not received in a certain period of time. TCP sets this timeout as a function of the RTT it expects between the two ends of the connection.

Unfortunately, given the range of possible RTTs between any pair of hosts in the Internet, as well as the variation in RTT between the same two hosts over time, choosing an appropriate timeout value is not that easy.

To address this problem, TCP uses an adaptive retransmission mechanism. We now describe this mechanism and how it has evolved over time as the Internet community has gained more experience using TCP.

The solution, which was proposed in 1987, is surprisingly simple. Whenever TCP retransmits a segment, it stops taking samples of the RTT; it only measures SampleRTT.

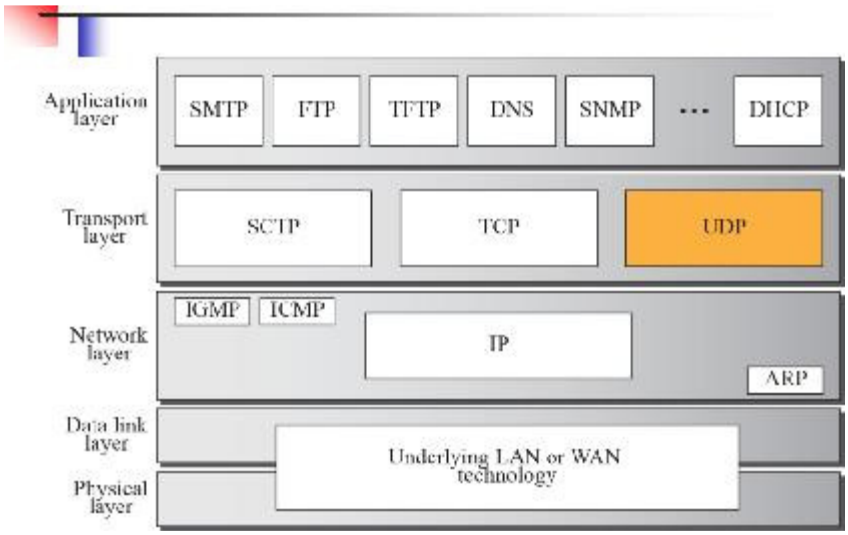


Figure: Position of UDP in the TCP/IP protocol suite

Congestion control:

Packets contend at a router for the use of a link, with each contending packet placed in a queue waiting its turn to be transmitted over the link. When too many packets are contending for the same link, the queue overflows and packets have to be dropped. When such drops become common events, the network is said to be congested. Most networks provide a congestion-control mechanism to deal with just such a situation.

Congestion control and resource allocation involve both hosts and network elements such as routers. In network elements, various queuing disciplines can be used to control the order in which packets get transmitted and which packets get dropped. The queuing discipline can also segregate traffic; that is, to keep one user's packets from unduly affecting another user's packets.

At the end hosts, the congestion-control mechanism paces how fast sources are allowed to send packets. This is done in an effort to keep congestion from occurring in the first place, and should it occur, to help eliminate the congestion.

A policy specifies a particular setting of those knobs, but does not know (or care) about how the black box is implemented. In this case, the mechanism in question is the queuing discipline, and the policy is a particular setting of which flow gets what level of service (e.g., priority or weight).

TCP CONGESTION CONTROL

Introduction:

The essential strategy of TCP is to send packets into the network without a reservation and then to react to observable events that occur. TCP assumes only FIFO queuing in the network's routers, but also works with fair queuing. TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson; roughly eight years after the TCP/IP protocol stack had become operational.

Immediately preceding this time, the Internet was suffering from congestion collapse—hosts would send their packets into the Internet as fast as the advertised window would allow, congestion would occur at some router (causing packets to be dropped), and the hosts would time out and retransmit their packets, resulting in even more congestion.

Broadly speaking, the idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit.

Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network, and that it is therefore safe to insert a new packet into the network without adding to the level of congestion.

Additive Increase/Multiplicative Decrease:

TCP maintains a new state variable for each connection, called Congestion Window, which is used by the source to limit how much data it is allowed to have in transit at a given time. The congestion window is congestion control's counterpart to flow control's advertised window.

TCP is modified such that the maximum number of bytes of unacknowledged data allowed is now the minimum of the congestion window and the advertised window. Thus, TCP's effective window is revised as follows:

MaxWindow = MIN(CongestionWindow,AdvertisedWindow) EffectiveWindow = MaxWindow−(LastByteSent− LastByteAcked)

That is, MaxWindow replaces AdvertisedWindow in the calculation of EffectiveWindow. Thus, a TCP source is allowed to send no faster than the slowest component—the network or the destination host—can accommodate.

The answer is that the TCP source sets the CongestionWindow based on the level of congestion it perceives to exist in the network. This involves decreasing the congestion window when the level of congestion goes up and increasing the congestion window when the level of congestion goes down.

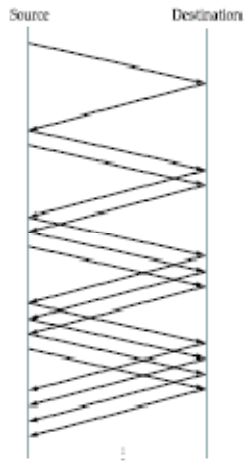


Figure: Packet in transit

This linear increase is illustrated in above figure. Note that in practice, TCP does not wait for an entire window's worth of ACKs to add 1 packet's worth to the congestion window, but instead increments CongestionWindow by a little for each ACK that arrives. Specifically, the congestion window is incremented as follows each time an ACK arrives:

Increment = MSS × (MSS/CongestionWindow) CongestionWindow + = Increment

Fast Retransmit and Fast Recovery:

The mechanisms described so far were part of the original proposal to add congestion control to TCP. It was soon discovered, however, that the coarse-grained implementation of TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire. Because of this, a new mechanism called fast retransmit was added to TCP.

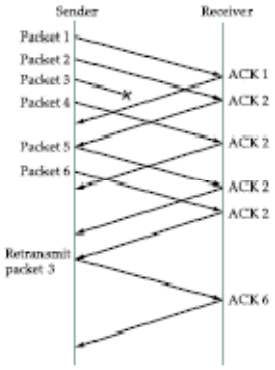


Figure: Fast retransmit based on duplicate ACKs

Fast retransmit is a heuristic that sometimes triggers the retransmission of a dropped packet sooner than the regular timeout mechanism. The fast retransmit mechanism does not replace regular timeouts; it just enhances that facility. The idea of fast retransmit is straightforward.

Congestion-Avoidance Mechanisms:

It is important to understand that TCP's strategy is to control congestion once it happens, as opposed to trying to avoid congestion in the first place. In fact, TCP repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and then it backs off from this point.

Said another way, TCP needs to create losses to find the available bandwidth of the connection. An appealing alternative, but one that has not yet been widely adopted, is to predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded. We call such a strategy congestion avoidance, to distinguish it from congestion control.

DECBIT

The first mechanism was developed for use on the Digital Network Architecture (DNA), a connectionless network with a connection-oriented transport protocol. This mechanism could, therefore, also be applied to TCP and IP. As noted above, the idea here is to more evenly split the responsibility for congestion control between the routers and the end nodes.

Each router monitors the load it is experiencing and explicitly notifies the end nodes when congestion is about to occur. This notification is implemented by setting a binary congestion bit in the packets that flow through the router; hence the name DECbit.

The destination host then copies this congestion bit into the ACK it sends back to the source. Finally, the source adjusts its sending rate so as to avoid congestion.

A single congestion bit is added to the packet header. A router sets this bit in a packet if its average queue length is greater than or equal to 1 at the time the packet arrives. This average queue length is measured over a time interval that spans the last busy + idle cycle, plus the current busy cycle. (The router is busy when it is transmitting and idles when it is not.)

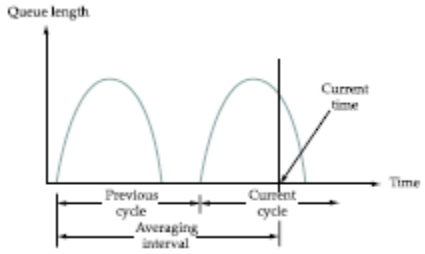


Figure: Computing average queue length at a router

The above figure shows the queue length at a router as a function of time. Essentially, the router calculates the area under the curve and divides this value by the time interval to compute the average queue length.

Using a queue length of 1 as the trigger for setting the congestion bit is a trade-off between significant queuing (and hence higher throughput) and increased idle time (and hence lower delay).

In particular, the source maintains a congestion window, just as in TCP, and watches to see what fraction of the last window's worth of packets resulted in the bit being set.

If less than 50% of the packets had the bit set, then the source increases its congestion window by one packet. If 50% or more of the last window's worth of packets had the congestion bit set, then the source decreases its congestion window to 0.875 times the previous value.

RED

A second mechanism, called random early detection (RED), is similar to the DECbit scheme in that each router is programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window.

RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways.

The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it implicitly notifies the source of congestion by dropping one of its packets. The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK.

As the "early" part of the RED acronym suggests, the gateway drops the packet earlier than it would have to, so as to notify the source that it should decrease its congestion window sooner than it would normally have.

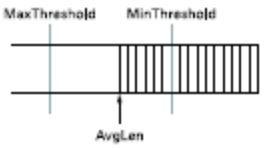


Figure: RED thresholds on a FIFO queue

In other words, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on.

Avg-Len is computed as

$$\text{AvgLen} = (1-\text{Weight})\times\text{AvgLen} + \text{Weight}\times\text{SampleLen}$$

where $0 < \text{Weight} < 1$ and Sample-Len is the length of the queue when a sample measurement is made. In most software implementations, the queue length is measured every time a new packet arrives at the gateway.

QUALITY OF SERVICE

Introduction:

For many years, packet-switched networks have offered the promise of supporting multimedia applications, that is, those that combine audio, video, and data. After all, once digitized, audio and video information become just another form of data—a stream of bits to be transmitted.

There is more to transmitting audio and video over a network than just providing sufficient bandwidth, however. Participants in a telephone conversation, for example, expect to be able to converse in such a way that one person can respond to something said by the other and be heard almost immediately.

Thus, the timeliness of delivery can be very important. We refer to applications that are sensitive to the timeliness of data as real-time applications.

Even file transfer applications can have timeliness constraints, such as a requirement that a database update complete overnight before the business that needs the data resumes on the next day.

Application Requirements:

Before looking at the various protocols and mechanisms that may be used to provide quality of service to applications, we should try to understand what the needs of those applications are. To begin, we can divide applications into two types: real-time and nonreal-time. The latter are sometimes called “traditional data” applications, since they have traditionally been the major applications found on data networks.

Figure: An audio application

They include most popular applications like Telnet, FTP, email, web browsing, and so on. All of these applications can work without guarantees of timely delivery of data. Another term for this nonreal-time class of applications is elastic, since they are able to stretch gracefully in the face of increased delay.

Note that these applications can benefit from shorter-length delays, but they do not become unusable as delays increase. Also note that their delay requirements vary from the interactive applications like Telnet to more asynchronous ones like email, with interactive bulk transfers like FTP in the middle.

Real-Time Audio Example:

At the receiving host, the data must be played back at some appropriate rate. For example, if the voice samples were collected at a rate of one per 125 μs, they should be played back at the same rate. Thus, we can think of each sample as having a particular playback time: the point in time at which it is needed in the receiving host.

In the voice example, each sample has a playback time that is 125 μs later than the preceding sample. If data arrives after its appropriate playback time, either because it was delayed in the network or because it was dropped and subsequently retransmitted, it is essentially useless. It is the complete worthlessness of late data that characterizes real-time applications. In elastic applications, it might be nice if data turns up on time, but we can still use it when it does not.

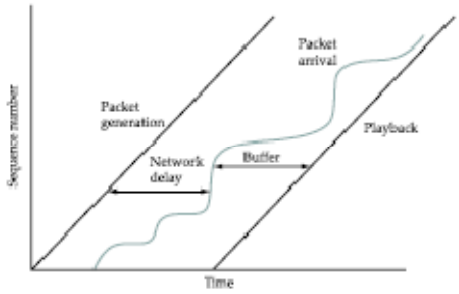


Figure: A playback buffer

One way to make our voice application work would be to make sure that all samples take exactly the same amount of time to traverse the network. Then, since samples are injected at a rate of one per 125 μs, they will appear at the receiver at the same rate, ready to be played back. However, it is generally difficult to guarantee that all data traversing a packet-switched network will experience exactly the same delay.

Packets encounter queues in switches or routers and the lengths of these queues vary with time, meaning that the delays tend to vary with time, and as a consequence, are potentially different for each packet in the audio stream.

The way to deal with this at the receiver end is to buffer up some amount of data in reserve, thereby always providing a store of packets waiting to be played back at the right time. If a packet is delayed a short time, it goes in the buffer until its playback time arrives. If it gets delayed a long time, then it will not need to be stored for very long in the receiver's buffer before being played back.

As denoted by the cumulative percentages given across the top of the graph, 97% of the packets in this case had a latency of 100 ms or less. This Approach to QoS Support Considering this rich space of application requirements, what we need is a richer service model that meets the needs of any application.

UNIT – 5: APPLICATION LAYER

TRADITIONAL APPLICATIONS

The discussion about applications is by focusing on two of the most popular—the World Wide Web and email. We then turn to the domain name system (DNS)—not an application that users normally invoke explicitly, but nevertheless an application that all other applications depends upon. This is because the name server is used to translate host names into host addresses; the existence of such an application allows the users of other applications to refer to remote hosts by name rather than by address.

In other words, a name server is usually used by other applications rather than by humans. Our final example in this section is network management, which although not so familiar to the average user, is the application of choice for system administrators.

All of these application classes use the request/reply paradigm—users send requests to servers, which then respond accordingly. We refer to these as traditional applications because they typify the sort of applications that have existed since the early days of computer networks. By contrast, later sections will look at a class of applications that have become feasible only relatively recently: streaming applications (e.g., multimedia applications like video and audio) and various overlay-based applications.

Before taking a close look at each of these applications, there are three general points that we need to make. The first is that it is important to distinguish between application programs and application protocols.

For example, the Hyper Text Transport Protocol (HTTP) is an application protocol that is used to retrieve web pages from remote servers. There can be many different application programs—that is, web clients like Internet Explorer, Netscape, Firefox, and Safari—that provide users with a different look and feel, but all of them use the same HTTP protocol to communicate with web servers over the Internet. This section focuses on four application protocols:

- ✓ SMTP: Simple Mail Transfer Protocol is used to exchange electronic mail.
- ✓ HTTP: Hyper Text Transport Protocol is used to communicate between web browsers and web servers.
- ✓ DNS: Domain Name System protocol is used to query name servers and send the responses.
- ✓ SNMP: Simple Network Management Protocol is used to query the state of remote network nodes. The second point is that since all of the application protocols described in this section follow the same request/reply communication pattern, we would expect that they are all built on top of an RPC transport protocol. This is not the case, however, as they are all implemented on top of either TCP or UDP.

ELECTRONIC MAIL (SMTP, MIME, IMAP)

Introduction:

Email is one of the oldest network applications. In fact, the pioneers of the ARPANET had not really envisioned email as a key application when the network was created—remote access to computing resources was the main design goal—but it turned out to be a surprisingly successful application.

Out of this work evolved the Internet's email system, which is now used by hundreds of millions of people every day. As with all the applications described in this section, the place to start in understanding how email works is to (1) distinguish the user interface (i.e., your mail reader) from the underlying message transfer protocol (in this case, SMTP), and (2) to distinguish between this transfer protocol and a companion protocol (RFC 822 and MIME) that defines the format of the messages being exchanged.

Message Format:

RFC 822 defines messages to have two parts: a header and a body. Both parts are represented in ASCII text. Originally, the body was assumed to be simple text. This is still the case, although RFC 822 has been augmented by MIME to allow the message body to carry all sorts of data. This data is still represented as ASCII text, but because it may be an encoded version of, say, a JPEG image, it's not necessarily readable by human users.

The message header is a series of <CRLF> terminated lines. (<CRLF> stands for carriage-return + line-feed, which are a pair of ASCII control characters.

Applications:

For example, the To: header identifies the message recipient, and the Subject: header says something about the purpose of the message. Other headers are filled in by the underlying mail delivery system. Examples include Date: (when the message was transmitted), From: (what user sent the message), and Received: (each mail server that handled this message). There are, of course, many other header lines; the interested reader is referred to RFC 822.

MIME consists of three basic pieces.

The first piece is a collection of header lines that augment the original set defined by RFC 822. These header lines describe, in various ways, the data being carried in the message body. They include MIME-Version: (the version of MIME being used), Content-Description: (a human-readable description of what's in the message, analogous to the Subject: line), Content-Type: (the type of data contained in the message), and Content-Transfer-Encoding (how the data in the message body is encoded).

The second piece is definitions for a set of content types (and subtypes). For example, MIME defines two different still-image types, denoted image/gif and image/jpeg, each with the obvious meaning.

As a third example, MIME defines an application type, where the subtypes correspond to the output of different application programs (e.g., application/postscript and application/msword).

MIME also defines a multipart type that says how a message carrying more than one data type is structured. This is like a programming language that defines both base types (e.g., integers and floats) and compound types (e.g., structures and arrays). One possible multipart subtype is mixed, which says that the message contains a set of independent data pieces in a specified order. Each piece then has its own header line that describes the type of that piece.

Plain text, a JPEG image, and a PostScript file would look something like this:

MIME-Version: 1.0 Content-Type: multipart/mixed;

boundary="-------417CA6E2DE4ABCAFBC5" From: Alice Smith <Alice@cisco.com> To: Bob@cs.Princeton.edu

Subject: promised material

Date: Mon, 07 Sep 1998 19:45:19 -0400

---------417CA6E2DE4ABCAFBC5

Content-Type: text/plain; charset=us-ascii Content-Transfer-Encoding: 7bit

Bob,

Here’s the jpeg image and draft report I promised.

--Alice

---------417CA6E2DE4ABCAFBC5 Content-Type: image/jpeg Content-Transfer-Encoding: base64

... unreadable encoding of a jpeg figure

---------417CA6E2DE4ABCAFBC5

Content-Type: application/postscript; name="draft.ps" Content-Transfer-Encoding: 7bit

... readable encoding of a PostScript document

In this example, the Content-Type line in the message header says that this message contains various pieces, each denoted by a character string that does not appear in the data itself. Each piece then has its own Content-Type and Content-Transfer- Encoding lines.

Message Transfer:

Next, we look at SMTP—the protocol used to transfer messages from one host to another. To place SMTP in the right context, we need to identify the key players.

First, users interact with a mail reader when they compose, file, search, and read their email. There are countless mail readers available, just like there are many web browsers to choose from. In fact, most web browsers now include a mail reader.

Second, there is a mail daemon (or process) running on each host. You can think of this process as playing the role of a post office: Mail readers give the daemon messages they want to send to other users, the daemon uses SMTP running over TCP to transmit the message to a daemon running on another machine, and the daemon puts incoming messages into the user's mailbox (where that user's mail reader can later find it).

Since SMTP is a protocol that anyone could implement, in theory there could be many different implementations of the mail daemon.

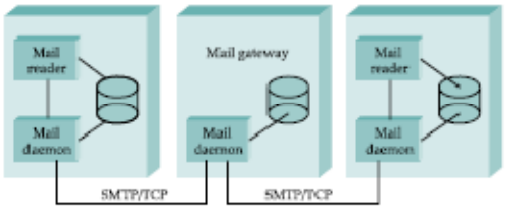


Figure: Sequence of mail gateways

It turns out, though, that the mail daemon running on most hosts is derived from the sendmail program originally implemented on While it is certainly possible that the sendmail program on a sender's machine establishes SMTP/TCP connection to the send mail program on the recipient's machine, in many cases the mail traverses one or more mail gateways on its route from the sender's host to the receiver's host.

Like the end hosts, these gateways also run a sendmail process. It's not an accident that these intermediate nodes are called "gateways" since their job is to store and forward email messages, much like an "IP gateway" (which we have referred to as a router) stores and forwards IP datagrams.

The only difference is that a mail gateway typically buffers messages on disk and is willing to try retransmitting them to the next machine for several days, while an IP router buffers datagrams in memory and is only willing to retry transmitting them for a fraction of a second.

Mail Reader:

The final step is for the user to actually retrieve her messages from the mailbox, read them, reply to them, and possibly save a copy for future reference. The user performs all these actions by interacting with a mail reader.

In many cases, this reader is just a program running on the same machine as the user's mailbox resides, in which case it simply reads and writes the file that implements the mailbox.

In other cases, the user accesses her mailbox from a remote machine using yet another protocol, such as the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP).

IMAP is similar to SMTP in many ways. It is a client/server protocol running over TCP, where the client (running on the user's desktop machine) issues commands in the form of <CRLF> terminated ASCII text lines and the mail server (running on the machine that maintains the user's mailbox) responds in-kind. The exchange begins with the client authenticating herself, and identifying the mailbox she wants to access.

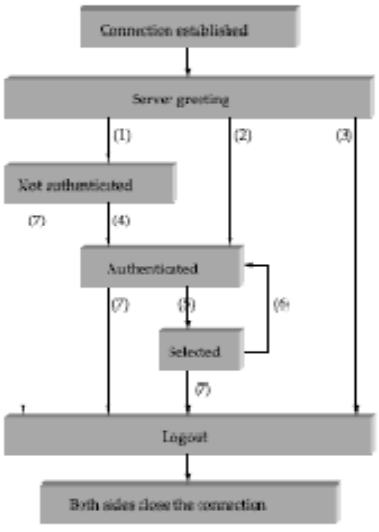


Figure: Sequence of mail gateways

LOGIN, AUTHENTICATE, SELECT, EXAMINE, CLOSE, and LOGOUT are example commands that the client can issue, while OK is one possible server response. Other common commands include FETCH, STORE, DELETE, and EXPUNGE, with the obvious meanings. Additional server responses include NO (client does not have permission to perform that operation) and BAD (command is ill formed).

When the user asks to FETCH a message, the server returns it in MIME format and the mail reader decodes it. In addition to the message itself, IMAP also defines a set of message attributes that are exchanged as part of other commands, independent of transferring the message itself.

Message attributes include information like the size of the message, but more interestingly, various flags associated with the message, such as Seen, Answered, Deleted, and Recent. These flags are used to keep the client and server synchronized, that is, when the user deletes a message in the mail reader, the client needs to report this fact to the mail server.

Later, should the user decide to expunge all deleted messages, the client issues an EXPUNGE command to the server, which knows to actually remove all earlier deleted messages from the mailbox.

Finally, note that when the user replies to a message, or sends a new message, the mail reader does not forward the message from the client to the mail server using IMAP, but it instead uses SMTP.

This means that the user's mail server is effectively the first mail gateway traversed along the path from the desktop to the recipient's mailbox.

HTTP

The World Wide Web has been so successful and has made the Internet accessible to so many people that sometimes it seems to be synonymous with the Internet. One helpful way to think of the Web is as a set of cooperating clients and servers, all of whom speak the same language: HTTP.

When you select to view a page, your browser (the client) fetches the page from the server using HTTP running over TCP. Like SMTP, HTTP is a text-oriented protocol.

At its core, each HTTP message has the general form

START_LINE <CRLF> MESSAGE_HEADER <CRLF> <CRLF>

MESSAGE_BODY <CRLF>

whereas before, <CRLF> stands for carriage-return-line-feed. The first line (START_LINE) indicates whether this is a request message or a response message. In effect, it identifies the "remote procedure" to be executed (in the case of a request message), or the "status" of the request (in the case of a response message).

The next set of lines specifies a collection of options and parameters that qualify the request or response. There are zero or more of these MESSAGE_HEADER lines—the set is terminated by a blank line—each of which looks like a header line in an email message.

HTTP defines many possible header types, some of which pertain to request messages, some to response messages, and some to the data carried in the message body. Instead of giving the full set of possible header types, though, we just give a handful of representative examples. Finally, after the blank line comes, the contents of the requested message (MESSAGE_BODY); this part of the message is typically empty for request messages.

Request Messages:

The first line of an HTTP request message specifies three things: the operation to be performed, the web page the operation should be performed on, and the version of HTTP being used.

Although HTTP defines a wide assortment of possible request operations—including "write" operations that allow a web page to be posted on a server—the two most common operations are GET (fetch the specified web page) and HEAD (fetch status information about the specified web page).

The former is obviously used when your browser wants to retrieve and display a web page. The latter is used to test the validity of a hypertext link or to see if a particular page has been modified since the browser last fetched it.

Operation Description:

OPTIONS Request information about available options GET Retrieve document identified in URL

HEAD Retrieve meta information about document identified in URL POST Give information (e.g., annotation) to server

PUT Store document under specified URL DELETE Delete specified URL

TRACE Loopback request message CONNECT For use by proxies

For example, the START_LINE

GET http://www.cs.princeton.edu/index.html HTTP/1.1

says that the client wants the server on host www.cs.princeton.edu to return the page named index.html.

This particular example uses an absolute URL. It is also possible to use a relative identifier and specify the host name in one of the MESSAGE_ HEADER lines; for example,

GET index.html HTTP/1.1 Host: www.cs.princeton.edu

Here, Host is one of the possible MESSAGE_HEADER fields. One of the more interesting of these is If-Modified-Since, which gives the client a way to conditionally request a web page—the server returns the page only if it has been modified since the time specified in that header line.

Response Messages:

Like request messages, response messages begin with a single START_LINE.

In this case, the line specifies the version of HTTP being used, a three-digit code indicating whether or not the request was successful, and a text string giving the reason for the response.

Example, the START_LINE

HTTP/1.1 202 Accepted

indicates that the server was able to satisfy the request.

Code Type Example Reasons:

1xx Informational Request received, continuing process

2xx Success Action successfully received, understood, and accepted 3xx Redirection Further action must be taken to complete the request 4xx Client error Request contains bad syntax or cannot be fulfilled 5xx Server error Server failed to fulfill an apparently valid request

Code	Type	Example Reasons
1xx	Informational	Request received, continuing process
2xx	Success	Action successfully received, understood, and accepted
3xx	Redirection	Further action must be taken to complete the request
4xx	Client error	Request contains bad syntax or cannot be fulfilled
5xx	Server error	Server failed to fulfill an apparently valid request

Table: Five types of HTTP result codes

HTTP/1.1 404 Not Found indicates that it was not able to satisfy the request because the page was not found. There are five general types of response codes, with the first digit of the code indicating its type. Also similar to request messages, response messages can contain one or more MESSAGE_HEADER lines.

These lines relay additional information back to the client. For example, the Location header line specifies that the requested URL is available at another location. Thus, if the Princeton CS Department web page had moved from http://www.cs.princeton.edu/index.html to http://www.princeton. edu/cs/index.html, for example, then the server at the original address might respond with

HTTP/1.1 301 Moved Permanently

Location: http://www.princeton.edu/cs/index.html

In the common case, the response message will also carry the requested page. This page is an HTML document, but since it may carry non-textual data (e.g., a GIF image); it is encoded using MIME (see Section 9.1.1). Certain MESSAGE_HEADER lines give attributes of the page contents, including Content-Length (number of bytes in the contents), Expires (time at which the contents are considered stale), and Last- Modified (time at which the contents were last modified at the server).

DNS

Host names differ from host addresses in two important ways. First, they are usually of variable length and mnemonic, thereby making them easier for humans to remember. (In contrast, fixed-length numeric addresses are easier for routers to process.)

Second, names typically contain no information that helps the network locate (route packets toward) the host. Addresses, in contrast, sometimes have routing information embedded in them; flat addresses (those not divisible into component parts) are the exception.

Before getting into the details of how hosts are named in a network, we first introduce some basic terminology. First, a namespace defines the set of possible names. A namespace can be either flat (names are not divisible into components), or it can be hierarchical (Unix file names are an obvious example).

Second, the naming system maintains a collection of bindings of names to values. The value can be anything we want the naming system to return when presented with a name; in many cases it is an address.

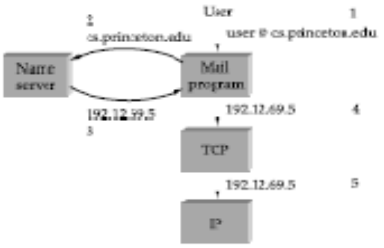


Figure: Names translated into addresses

Finally, a resolution mechanism is a procedure that, when invoked with a name, node in the tree corresponds to a domain, and the leaves in the tree correspond to the hosts being named.

The relevance of a zone is that it corresponds to the fundamental unit of implementation in DNS—the name server. Specifically, the information contained in each The Name and Value fields are exactly what you would expect, while the Type field specifies how the Value should be interpreted.

For example, Type = A indicates that the Value is an IP address. Thus, A records implement the name- to-address mapping we have been assuming. Other record types include

- ✓ NS: The Value field gives the domain name for a host that is running a name server that knows how to resolve names within the specified domain.

- ✓ CNAME: The Value field gives the canonical name for a particular host; it is used to define aliases.

- ✓ MX: The Value field gives the domain name for a host that is running a mail server that accepts messages for the specified domain.

The Class field was included to allow entities other than the NIC to define useful record types.

It is used by servers that cache resource records from other servers; when the TTL expires, the server must evict the record from its cache. Taken together, these two records effectively implement a pointer from the root name server to one of the TLD servers.

_ edu, a3.nstld.com, NS, IN_

_ a3.nstld.com, 192.5.6.32, A, IN _ _ com, a.gtld-servers.net, NS, IN_

_ a.gtld-servers.net, 192.5.6.30, A, IN _

Moving our way down the hierarchy by one level, the a3.nstld.com server has records for .edu domains like this:

_ princeton.edu, dns.princeton.edu, NS, IN _ _ dns.princeton.edu, 128.112.129.15, A, IN_

In this case, we get an NS record and an A record for the name server that is responsible for the princeton.edu part of the hierarchy. That server might be able to directly resolve some queries (e.g., for email.princeton.edu) while it would redirect others to a server at yet another layer in the hierarchy (e.g., for a query about penguins.cs.princeton.edu):

_ email.princeton.edu, 128.112.198.35, A, IN _

_ penguins.cs.princeton.edu, dns1.cs.princeton.edu, NS, IN _ _ dns1.cs.princeton.edu, 128.112.136.10, A, IN _

The mail exchange (MX) records serve the same purpose for the email application—it allows an administrator to change which host receives mail on behalf of the domain without having to change everyone's email address.

_ penguins.cs.princeton.edu, 128.112.155.166, A, IN _ _www.cs.princeton.edu, coreweb.cs.princeton.edu, CNAME, IN _ _ coreweb.cs.princeton.edu, 128.112.136.35, A, IN _

_ cs.princeton.edu, mail.cs.princeton.edu, MX, IN _ _ mail.cs.princeton.edu, 128.112.136.72, A, IN _

Note that although resource records can be defined for virtually any type of object, DNS is typically used to name hosts (including servers) and sites.

SNMP

A network is a complex system, both in terms of the number of nodes that are involved and in terms of the suite of protocols that can be running on any one node. Even if you restrict yourself to worrying about the nodes within a single administrative domain, such as a campus, there might be dozens of routers and hundreds—or even thousands—of hosts to keep track of.

The MIB defines the specific pieces of information—the MIB variables— that you can retrieve from a network node. The current version of MIB, called MIB-II, organizes variables into 10 different groups. You will recognize that most of the groups correspond to one of the protocols described in this book, and nearly all of the variables defined for each group should look familiar.

For example:

- ✓ System: general parameters of the system (node) as a whole, including where the node is located, how long it has been up, and the system's name.
- ✓ Interfaces: information about all the network interfaces (adaptors) attached to this node, such as the physical address of each interface, or how many packets have been sent and received on each interface.
- ✓ Address translation: information about the Address Resolution Protocol (ARP) and in particular, the contents of its address translation table.
- ✓ IP: variables related to IP, including its routing table, how many datagrams it has successfully forwarded, and statistics about datagram reassembly. Includes counts of how many times IP drops a datagram for one reason or another.
- ✓ TCP: information about TCP connections, such as the number of passive and active opens, the number of resets, the number of timeouts, default timeout settings, and so on. Per-connection information persists only as long as the connection exists.
- ✓ UDP: information about UDP traffic, including the total number of UDP datagrams that have been sent and received.

There are also groups for ICMP, EGP, and SNMP itself. The tenth group is used by different media.

WEB SERVICES

Network applications, even those that cross organization boundaries, are not new—we have just seen some examples in the preceding section. What is new about this problem is the scale. Not scale in the size of the network, but scale in the number of different kinds of network applications.

Both the protocols' specifications and the implementations of those protocols for traditional applications like electronic mail and file transfer have typically been developed by a small group of networking experts.

Here is a simple example of what we are talking about. Suppose you buy a book at an online retailer like Amazon.com. Once your book has been shipped, Amazon could send you the tracking number in an email, and then you could head over to the website for the shipping company— http://www.fedex.com, perhaps—and track the package. However, you can also track your package directly from the Amazon.com website.

Custom Application Protocols (WSDL, SOAP):

The architecture informally referred to as SOAP is based on Web Services Description Language (WSDL) and SOAP. Both of these standards are issued by the World Wide Web Consortium (W3C). This is the architecture that people usually mean when they use the term Web Services.

Defining Application Protocols:

WSDL has chosen a procedural operation model of application protocols. An abstract web service interface consists of a set of named operations, each representing a simple interaction between a client and the web service. An operation is analogous to a remotely callable procedure in an RPC system.

An example from W3C's WSDL Primer is a hotel reservation web service with two operations, CheckAvailability and MakeReservation.

Several MEPs are predefined and new custom MEPs can be defined, but it appears that in practice only two MEPs are being used:

MEPs are templates that have placeholders instead of specific message types or formats, so part of the definition of an operation involves specifying which message formats to map into the placeholders in the pattern. Message formats are not defined at the bit-level that is typical of protocols we have discussed

XML Schema provides a set of primitive data types and ways to define compound data types. Data that conforms to an XML Schema-defined format—its abstract data model—can be concretely represented using XML, or it can use another representation, such as the "binary" representation Fast.

Defining Transport Protocols:

SOAP uses many of the same strategies as WSDL, including message formats defined using XML Schema, bindings to underlying protocols, MEPs, and reusable specification elements identified using XML namespaces.

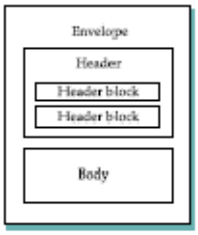


Figure: SOAP message structure

SOAP is used to define transport protocols with exactly the features needed to support a particular application protocol. SOAP aims to make it feasible to define many such protocols by using reusable components. Each component captures the header information and logic that go into implementing a particular feature.

To define a protocol with a certain set of features, just compose the corresponding components. Of course it's not quite that simple. Let's look more closely at this aspect of SOAP. SOAP 1.2 introduced a feature abstraction, which the specification describes thus:

A SOAP feature is an extension of the SOAP messaging framework. Although SOAP poses no constraints on the potential scope of such features, example features may include "reliability," "security," "correlation," "routing," and message exchange patterns (MEPs) such as request/ response, one-way, and peer-to-peer conversations. A SOAP feature specification must include:

- ✓ A URI that identifies the feature;
- ✓ The state information and processing, abstractly described, that is required at each SOAP node to implement the feature;
- ✓ The information to be relayed to the next node;
- ✓ If the feature is a MEP, the life cycle and temporal/causal relationships of the messages exchanged (e.g., responses follow requests and are sent to the originator of the request).

A Generic Application Protocol (REST):

The WSDL/SOAP Web Services architecture is based on the assumption that the best way to integrate applications across networks is via protocols that are customized to each application.

That architecture is designed to make it practical to specify and implement all those protocols. In contrast, the REST Web Services architecture is based on the assumption that the best way to integrate applications across networks is by applying the model underlying the World Wide Web architecture.

This model, articulated by Web architect Roy Fielding, is known as REpresentational State Transfer (REST). There is no need for a new REST architecture for web services—the existing web architecture is suitable, although a few extensions are probably necessary.

In the web architecture, individual web services are regarded as resources identified by URIs and accessed via HTTP—a single generic application protocol with a single generic addressing scheme.

Signers using WSDL/SOAP need to design such extensibility into each of their custom protocols. Of course, the designers of state representations in REST architecture also have to design for evolvability.

An area where WSDL/SOAP may have an advantage is in adapting or wrapping previously written, legacy applications to conform to web services. This is an important point since most web services will be based on legacy applications for the near future at least.

These applications usually have a procedural interface that maps more easily into WSDL's operations than REST states.

The REST versus WSDL/SOAP competition may very well hinge on how easy or difficult it turns out to be to devise REST-style interfaces for individual web services.