2015-2016

Computernetwerken

Ysaline de Wouters
KUL

Table of contents

CHAPTER 1 : FOUNDATION

1. Introduction

Network: set of serial lines used to attach dumb terminals to mainframe computers. Other important networks include the voice telephone network and the cable TV network used to disseminate video signals. The main things these networks have in common are that they are **specialized to handle one particular kind of data** and they typically connect to special-purpose devices.

>< computer network: built primarily from general-purpose programmable hardware. They are not optimized vor a particular application.

2. Applications

Internet: WorldWide Web email, online social networking, streaming audio and video, instant messaging, etc.

➔ We interact with the Internet as users of the network.
➔ But there are also people who create the applications, operate or manage network's, design and build the devices and protocols, …

2.1 The World Wide Web

= Internet application that catapulted the Internet from a somewhat obscure tool used mostly by scientist and engineers to the mainstream phenomenon that it is today.
The web presents an intuitively simple interface. Users view pages full of textual and graphical objects and click on objects that they want to learn more about.

To retrieve a page of information identified by an URL, we simply use a browser and a simple request/response protocol (HTTP).
Ex: http://www.mkp.com

By clicking on just one such URL, over a dozen messages may be exchanged over the internet.

- Up to 6 messages to translate the server name into its Internet Protocol (IP) 213.38.165.80
- 3 messages to set up a Transmission Control Protocol connection between the browser and the server (= request/response)
- 4 messages for your browser to send the THHP request and response
- 4 messages to tear down the TCP connection

/!\ The web is not a real time application since there are no real time boundaries >< Conference calls.

2.2 Streaming

Delivery of streaming audio and video. Ex: Services such as video on demand and internet radio use this technology. Streaming audio and video implies a **timelier transfer of messages** from sender to receiver, and the receiver displays the video or plays the audio pretty much as it arrives.

2.3 Videoconferencing

= Real-time bidirectional streaming multimedia. These applications have considerably tighter **timing constraints** than streaming applications.
Ex: when using a voice-over-IP application such as Skype, the interactions among the participants must be timely. Too much delay in this sort of environment makes the system unusable.

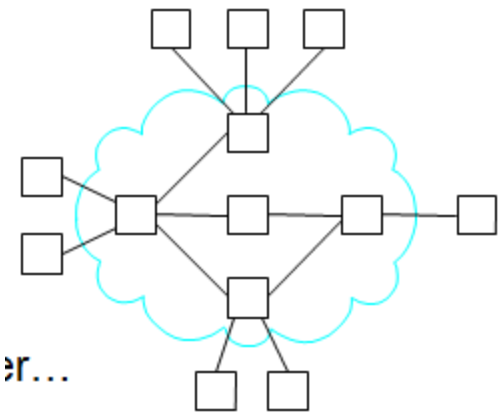
➔ 2 streams: voice and video stream.

3. Requirements

3.1 Scalable connectivity

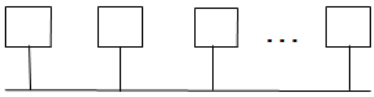
A network must **provide connectivity** among a set of computers. Sometimes it is enough to build a limited network that connects only a few select machines. Indeed, for reasons of privacy and security, many private (corporate) networks have the explicit goal of limiting the set of machines that are connected.
>< the **internet** that allows them the potential to **connect all the computers in the world.**

Scale = system that is designed to support growth to an arbitrarily large size.



- Link: physical medium that allows to connect computers.
  point-to-point: Physical links are sometimes limited to a pair of nodes

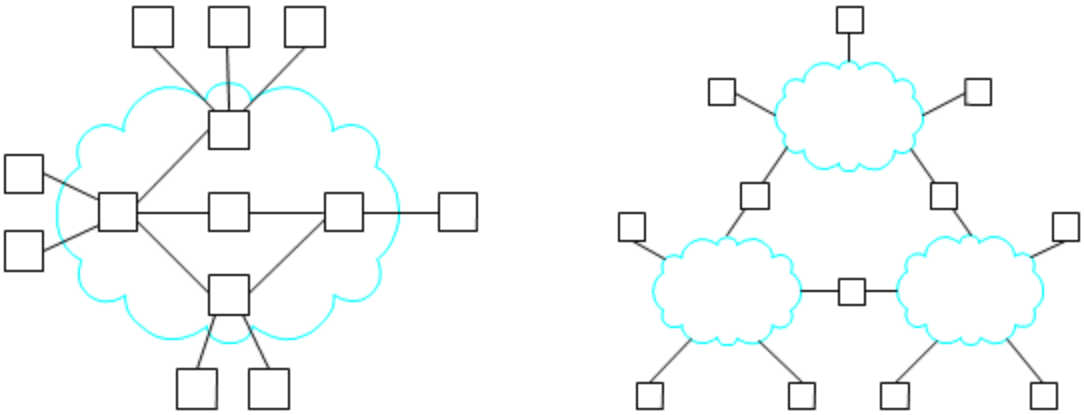
  - **Multiple access:** more than two nodes may share a single physical link. These are often limited in size, in terms of both the geographical distance they can cover and the number of nodes they can connect.



- **Nodes**: hosts and switches

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.

Fortunately, connectivity between two nodes does not necessarily imply a direct physical connection between them—indirect connectivity may be achieved among a set of cooperating nodes.



The left figure shows a set of nodes, each of which is attached to one or more point-to-point links. Those nodes that are attached to at least two links run software that forwards data received on one link out on another. If organized in a systematic way, these forwarding nodes form a **switched network**. There are numerous types of switched networks, of which the two most common are **circuit switched** and **packet switched** (the nodes in such a network send discrete blocks of data to each other).

- ⇨ Packet-switched networks: use a strategy called store-and-forward. 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. = **++ Efficient !** Another advantage regards capacity levels. I don't need to block a certain capacity. The packet is forwarded and the remaining links remain free.
- ⇨ Circuit-switched network: establishes a dedicated circuit across a sequence of links, and then allows the source nod to send a stream of bits across this circuit to a destination node.

The links represented on the picture don't necessarily need to by physical link. It may also be a wireless link.

The second figure shows a set of independent networks that are interconnected to form an internetwork.

A node that is connected to two or more networks is commonly called a router or gateway. It plays much the same role as a switch: it forwards messages from one network to another.

⇨ We can recursively build large networks by interconnecting clouds to form larger clouds.
Ex: internet is a network made off other networks.

Another requirement is that each node must be able to say which of the other nodes on the network it wants to communicate with. This is done by an address to each node.

Adress = byte string that identifies a node.

When a source node wants the network to deliver a message to a certain destination node, it specifies the adedress of the destination node. If the sending and receiving nodes are not directly connected, then the switches and routers of the network use this address to decide how to forward the message toward the destination.

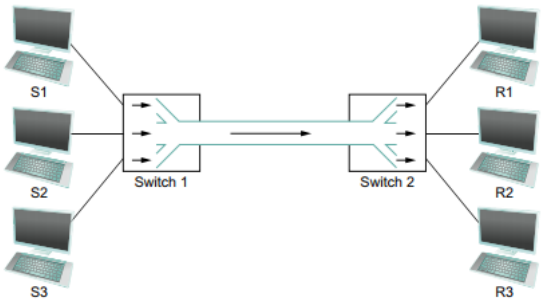
⇨ Process of determining how to forward messages toward the destination node based on its address is called **routing**.
- **UNICAST**: When the source node wants to send a message to **a single destination**.
- **MULTICAST**: send a message to **some subset of the other nodes** but not all of them.
- **BROADCAST**: Source node wants to broadcast a message to **all the nodes** on the network.
- **ANYCAST**: the message is sent to a group of nodes. And it is sufficient if one node of the group receives the message.

3.2 Cost-effective ressource sharing

Goal: provide all pairs of hosts with the ability to exchange messages.

⇨ How do all the hosts that want to communicate share the network, especially if they want to use it at the same time?
⇨ How do several hosts share the same link when they all want to use it at the same time?

Multiplexing: a system resource is shared among multiple users.
= Multiple data flows sharing a physical link.



Three hosts on the left side of the network are sending data to the three hosts on the right by sharing a switched network that contains only one physical link. Three flows of data (corresponding to the three pairs of hosts), are multiplexed onto a single physical link by switch 1 and then de-multiplexed back into separate flows by switch 2.

Different methods for multiplexing multiple flows onto one physical link

- **Synchronous time-division multiplexing (STDM):** divide time into equal-sized quant and, in a round-robin fashion, give each glow a chance to send its data over the physical link.
  Ex: during time quantum 1, data is transmitted from S1 to R1. during time quantum 2, data is transmitted from S2 to R2, …
- **Frequency-division multiplexing (FDM):** transmit each flow over the physical link at a different frequency.

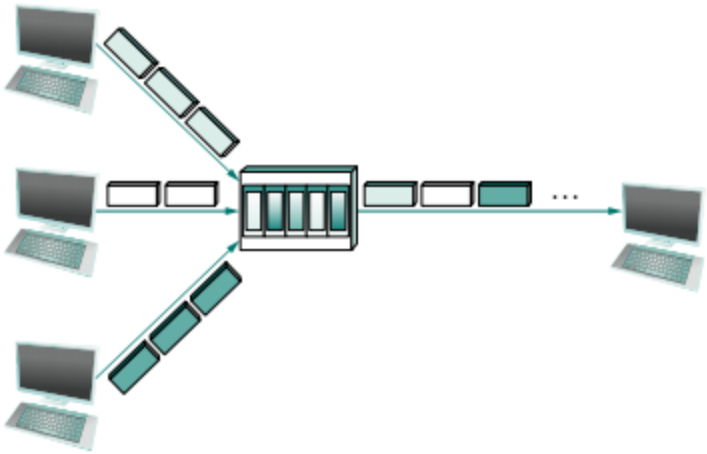
  Limitations:

  - If one of the flows does not have any data to send, its share of the physical link remains idle, even if one of the other flows has data to transmit.
  - STDM and FDM are limited to situations in which the maximum number of flows is fixed and known ahead of time.

⇨ **Statistical multiplexing**: data is transmitted from each flow on demand rather than during a predetermined time slot. So, if only one flow has data to send, it gets to transmit that data without waiting for its quantum to come around and thus without having to watch the quanta assigned to the other flows go unused.

>< it has no mechanism to ensure that all the flows eventually get their turn to transmit over the physical link. That is, once a flow begins sending data, we need some way to limit the transmission, so that the other flows can have a turn.

To account for this need, statistical multiplexing defines an upper bound on the size of the block of data that each flow is permitted to transmit at a given time.
This limited-size block of data is typically referred to as a packet, to distinguish it from the arbitrarily large message that an application program might want to transmit. As a consequence of this type of multiplexing, the source may need to fragment the message into several packets, with the receiver reassemble the packets back into the original message.



Three different computes want to send data. The different packets compete with each other as to use the single link. The switch may decide which packet can use the link first (ex: fifo).
Switches require buffers. These buffer is limited in size and capacity. Therefore, it might get overloaded.

- Causes delays
- Congestion
- Packets may get lost
- How to guarantee quality?

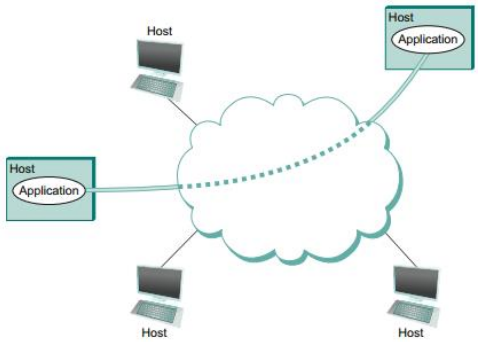
3.3 Support for common services

⇨ Application program running in the hosts connected to the network must be able to communicate in a meaningful way.

Since many applications need common services, it is much more logical to implement those common services once and then to let the application designer build the application using those services.

Challenge: identify the right set of common services.

Goal: hide complexity of the network from the application without overly constraining the application designer



What functionality must be provided?

- Guaranteed delivery? Guarantees that packets that got lost, still arrive at their final destination.
- Ensured order of arrival?

- Timing constraints?
- Ensured privacy and integrity

RELIABILITY: reliable message delivery is one of the most important functions that a network can provide.

Computer networks do not exist in a perfect world. Machines crash and later are rebooted, fibers are cut, electrical interference corrupts bits, switches run out of buffer space, …

- **Bit errors** may be introduced into the data: a 1 is turned into a 0 or vice versa. Sometimes single bits are corrupted, but more often than not a burst error occurs.
- **Packets might get lost:** one reason this can happen is that the packet contains an uncorrectable bit error and therefore has to be discarded. Besides, one of the nodes that has to handle the packet, for example, a switch that's is forwarding it from one link to another, is so overloaded that it has no place to store the packet and therefore is forced to drop it (= congestion).
- **Failure at the node and link level:** a physical link is cut, or the computer it is connected to crashes. Can be caused by software that crashes, a power failure, or a reckless backhoe operator.

  ⇨ Network must be able to mask certain kinds of failures/errors: make network appear more reliable than it really is
  - Error correction
  - Retransmission
  - rerouting

3.4 Manageability

Managing a network includes making changes as the network grows to carry more traffic or reach more users, and troubleshooting the network when things go wrong or performance isn't as desired.

Network management is no longer the province of experts but needs to be accomplished by consumers with little to ne special training. Therefore, networking devices should be plug-and-play.

4. Network Architecture

4.1 Layering and protocols

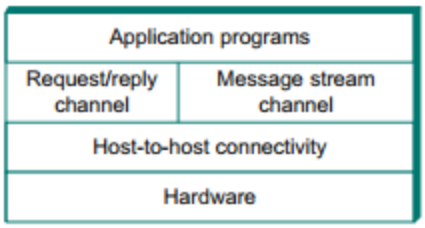
Abstraction is the fundamental tool used by system designers to manage complexity. It consists in hiding the details behind a well-defined interface. The idea is to define a model that can capture

some important aspects 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.

Abstractions naturally lead to layering. 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 level of service. The services provided at the high layers are implemented in terms of the services provided by the low layers.

Layering provides two nice features

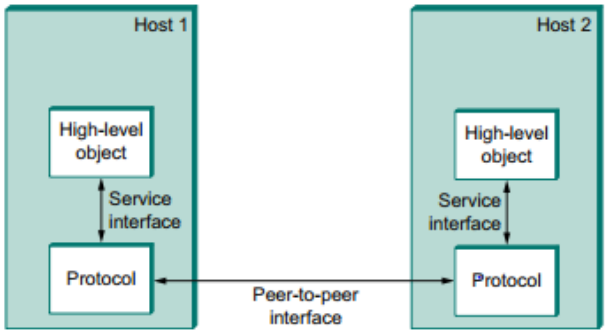
- **Decomposes** the problem of building a network into more manageable components. By implementing several layers, each layer solves one part of the problem.
- It provides a **more modular design**. If you want to add some new service, you may only need to modify the functionality at one layer, reusing the functions provided at all the other layers.



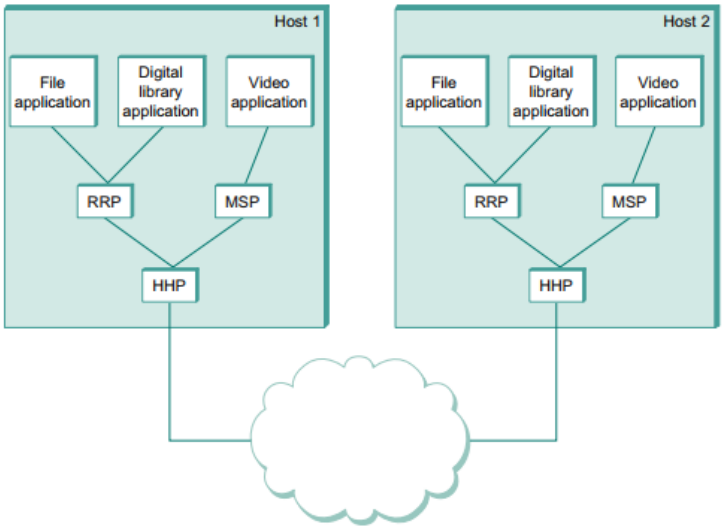
The abstract objects that make up the layers of a network system are called protocols. A protocol provides a communication service that higher-level objects use to exchange messages.

Each protocol defines two different interfaces.

- **Service interface** to the other objects on the same computer that want to use its communication services.
- **Peer interface:** defines the form and meaning of messages exchanged between protocol peers to implement the communication service.



Except at the hardware level, where peers directly communicate with each other over a link, peer-to-peer communication is indirect, each protocol communicates with its peer by passing messages to some lower-level protocol, which in turn delivers the message to its peer. There are potentially multiple protocols at any given level, each providing a different communication service.



Suite of protocols, represented by a **protocol graph**. The nodes of the graph correspond to protocols, and the edges represent a depends on relation.

Ex: Request/reply protocol (RRP) and MSP (message stream protocol) implement two different types of process-to-process channels, and both depend on the Host-to-host protocol (HHP) which provides a host-to-host connectivity service.
Suppose that the file access program on host 1 wants to send a message to its peer on host 2 using the communication service offered by RRP. In this case, the file application asks RRP to send the message on its behalf. To communicate with its peer, RRP invokes the services of HHP, which in turn transmits the message to its peer on the other machine. Once the message has arrived at the instance of HHP on host 2, HHP passes the message up to RRP, which in turn delivers the message to the file application.

⇨ The application is said to employ the services of the protocol stack RRP/HHP.

The term protocol is overloaded

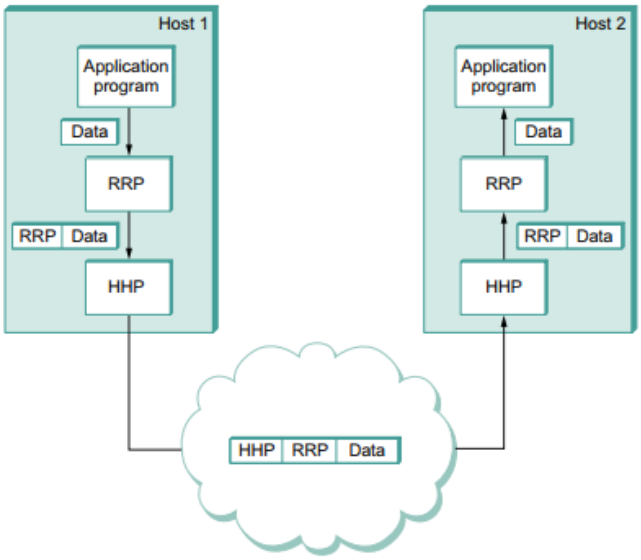
- Specification of abstract interfaces
- Module that implements these interfaces

ENCAPSULATION

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. However, RRP must communicate control information to its peer, instructing it how to handle the message when it is received. RRP does this by attaching a header to the message. Generally speaking, a header is a small data structure—from a few bytes to a few dozen bytes—that is used among peers to communicate with

each other. 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 RRP.

⇨ This process of encapsulation is then repeated at each level of the protocol graph. HHP encapsulates RRP's message by attaching a header of its own. When the message arrives at the destination host, it is processed in the opposite order.



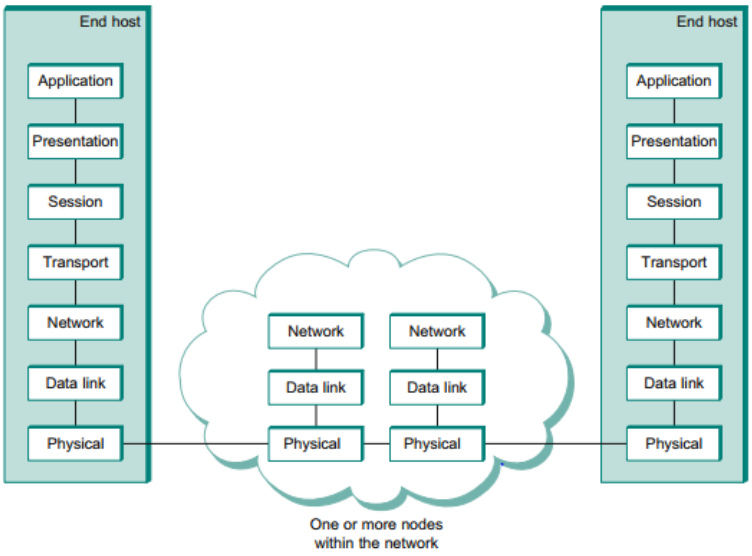
MULTIPLEXING AND DEMULTIPLEXING

The header that RRP attaches to its messages contains an identifier that records the application to which the message belongs. This identifier is called RRP's demultiplexing key or demux key. When the message is delivered to RRP on the destination host, its trips its header, examines the demux key, and demultiplexes the message to the correct application.

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, and others use 16- or 32-bit fields. Also some protocols have a single demultiplexing field in their header, while others have a pair of demultiplexing fields.

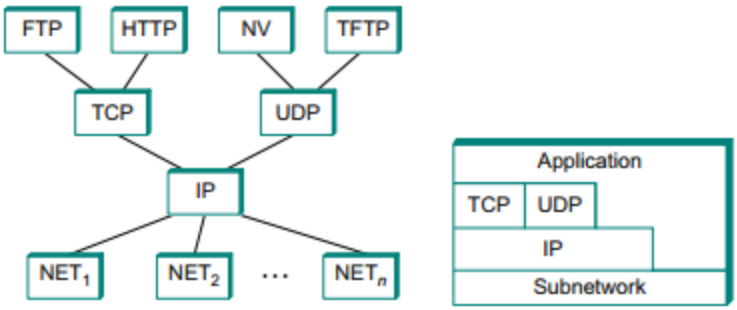
THE 7-LAYER MODEL

The Open Systems Interconnection (OS° architecture defines a partitioning of network functionality into seven layers, where one or more protocols implement the functionality assigned to a given layer.



- **Physical layer:** handles the transmission of raw bits over a communications link.
- **Data link layer:** collects a stream of bits into a larger aggregate called a frame
- **The network layer** handles routing among nodes within a packet-switched network. At this layer, the unit of data exchanged among nodes is typically called a packet rather than a frame.
- **The transport layer**: 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.
- **Application layer protocols**: include things like the HTTP, which is the basis of the WW and is what enables web browsers to request pages from web servers.
- **Presentation layer**: concerned with the format of data exchanged between peers, for instance, whether an integer is 16, 32 or 64 bits long, whether the most significant byte is transmitted first or last or how video stream is formatted.
- **Session layer**: provides a name space that is used to tie together the potentially different transport streams that are part of a single application.

4.2 Internet Architecture



While the 7-layer OSI model can, with some imagination, be applied to the Internet, a 4-layer mode is often used instead.

- At the lowest level is a wide variety of network protocols, denoted NT1, NET2, … these protocols are implemented by a combination of hardware and software. Ex: you might find Ethernet or wireless protocols at this layer.
- The second layer consists of a single protocol: The Internet Protocol (IP). This is the protocol that supports the interconnection of multiple networking technologies into a single, logical internetwork.
- The third layer contains two main protocols: The transport mission control protocol (TCP) and the User Datagram Protocol 5UDP). CP and UDP provide alternative logical channels to application programs.
  TCP and UDP are sometimes called end-to-end protocols, although it is equally correct to refer to them as transport protocols.
- Running above the transport layer is a range of application protolcs such as HTTP, FTP, Telnet, … that enable the interoperation of popular applications.

5. Implementing Network Software

5.1 Application Programming Interface (Sockets)

The place to start when implementing a network application is the interface exported by the network. Since most network protocols are implemented in software, and nearly all computer systems implement their network protocols as part of the operating system.
Although each operating system is free to define its own network API, over time certain of these API's have become widely supported; they have been ported to operating system other than their native system.

- Each protocol provides a certain set of services.
- The API provides a syntax by which those services can be invoked on a particular computer system.

The main abstraction of the socket interface, not surprisingly, is the socket. A good way to think of a socket is as the point where a local application process attaches to the network. The interface defines operations for creating a socket, attaching the socket to the network, sending/receiving messages through the socket, and closing the socket.

- Create a socket
- The next step depends on whether you are a client or a server. On a server machine, the application process performs a passive open, the server says that it is prepared to accept connections, but it does not actually establish a connection.

```
int bind(int socket, struct sockaddr *address, int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address, int *addr_len)
```

- The **binds** operation binds the newly created socket to the specified address. This is the network address of the local participant, the server.
- The **listen** operation defines how many connections can be pending on the specified socket.
- The **accept** operation carries out the passive open. It is a blocking operation that does not return until a remote participant has established a connection, and when it does complete it returns a new socket that corresponds to this just-established connection, and the address argument contains the remote participant's address.

- On the client machine, the application process performs an active open; that is, it says who it wants to communicate with by invoking the following single operation.
  This operation does not return until TCP has successfully established a connection, at which time the application is free to begin sending data.

6. Performance

⇨ In networking it is usually necessary to design for performance.

6.1 Bandwidth and Latency

Network performance is measured in two fundamental ways:

- **Bandwidth** (= throughput): given by the number of bits that can be transmitted over the network in a certain period of time. Ex: bandwidth of 10 million bits/second.
  We can talk either about the bandwidth of a network as a whole or about the bandwidth of a single physical link.
  Notation:
  - **Size**: KB = 2^{10} bytes, MB = 2^{20} bytes, GB = 2^{30} bytes, …
- **Latency** (= delay): corresponds to how ling it takes a message to travel from one end of a network to the other. We can focus either on the latency of a single link or an end-to-end channel. It is strictly measured in terms of time.

⇨ There are many situations in which it is more important to know how long it takes to send a message from one end of a network to the other and back, rather than the one-way latency. This is the **round-trip time (RTT) of the network. >< one-way delay**

Three components of latency

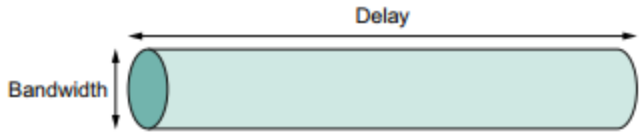
- Speed-of-light propagation delay. This delay occurs because nothing, including a bit on a wire, can travel faster than the speed of light.
- Amount of time it takes to transmit a unit of date. Transmit = Size/Bandwidth.
- Queuing delays inside the network since packet switches generally need to store packets for some time before forwarding them on an outbound link.

$$\text{Latency} = \text{Propagation} + \text{Transmit} + \text{Queue}$$
$$\text{Propagation} = \text{Distance/SpeedOfLight}$$
$$\text{Transmit} = \text{Size/Bandwidth}$$

⇨ Their relative importance depends on the application.

6.2 Delay x Bandwidth product

If we think of a channel between a pair of processes as a hollow pipe, where the latency corresponds to the length of the pipe and the bandwidth gives the diameter of the pipe, then the delay x bandwidth product gives the volume of the pipe. This is the maximum number of bits that could be in transit through the pipe at any given instant. In other words, if latency corresponds to the length of the pipe, then give the width of each bit, we can calculate how many bits fit in the pipe.



Ex: 100 ms x 45Mbps = 560 KB.

⇨ This product is important to know when constructing high-performance networks because it corresponds to how many bits the sender must transmit before the first bit arrives at the receiver.

⇨ The bits in the pipe are said to be “in flight”, which means that if the receiver tells the sender to stop transmitting it might receive up to one RTT x bandwidth’s worth of data before the sendere manages to respond.

Table 1.1 Sample Delay × Bandwidth Products

Link type	Bandwidth (typical)	One-way distance (typical)	Round-trip delay	RTT × Bandwidth
Dial-up	56 kbps	10 km	87 μs	5 bits
Wireless LAN	54 Mbps	50 m	0.33 μs	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

6.3 High-Speed Networks

There is an eternal optimism that network bandwidth will continue to improve. This causes network designers to start thinking about what happens in the limit or, what is the impact on network design of having infinite bandwidth available?

- Bandwidth increases
- (propagation) delay does not decrease
- Delay x bandwidth product increases

6.4 Application Performance Needs

BANDWIDTH REQUIREMENTS

- Application programs want as much bandwidth as the network can provide. The more bandwidth that is available, the faster the program will be able to return the image to the user.
- Some applications are able to state an upper limit on how much bandwidth they need.
- Variable rate: Ex: compressed video stream. Each frame can also be compressed because not all the detail in a picture is readily perceived by a human eye. The compressed video does not flow at a constant rate, but varies with time according to factors such as the amount of action and detail in the picture and the compression algorithm being used.

DELAY REQUIREMENTS

- As little as possible is not always required
- Variation in delay (jitter) is often more important
  - Buffer needed to reduce jitter
  - Extra delay depends on upper/upper bounds on latency

CHAPTER 2: GETTING CONNECTED

1. Introduction

We need to address the problem of connecting a host to a cloud. This, in effect, is the problem every Internet Service Provider faces when it wants to connect a new customer to the network: how to connect one more node to the ISP's cloud?

Whether we want to construct a trivial two-node network with one link or connect the one-billionth host to an existing network like the Internet, we need to address a common set of issues.

- We need some physical medium over which to make the connection
- The medium may be a length of wire, a piece of optical fiber, or some less tangible medium (such as air) through which electromagnetic radiation can be transmitted.
- It may cover a small or wide area

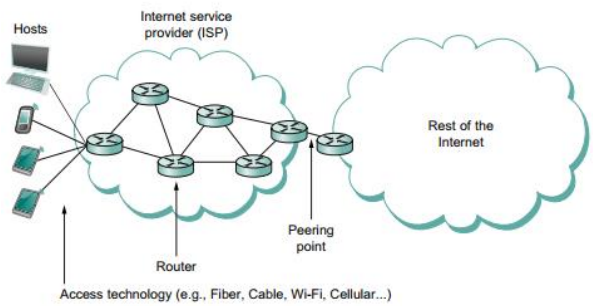
PROBLEM: physically connecting

2. Perspectives on connecting

⇨ In this chapter, we focus on what it takes to make a useful link, so that large, reliable networks containing millions of links can be built.

Operators of large networks deal with links that span hundreds or thousands of kilometers connecting refrigerator-sized routers.

>< the typical user of a network encounters links mostly as a way to connect a computer to the global Internet. This link will sometimes be a wireless link in a coffee shop; sometimes it is an Ethernet link in an office building or university.



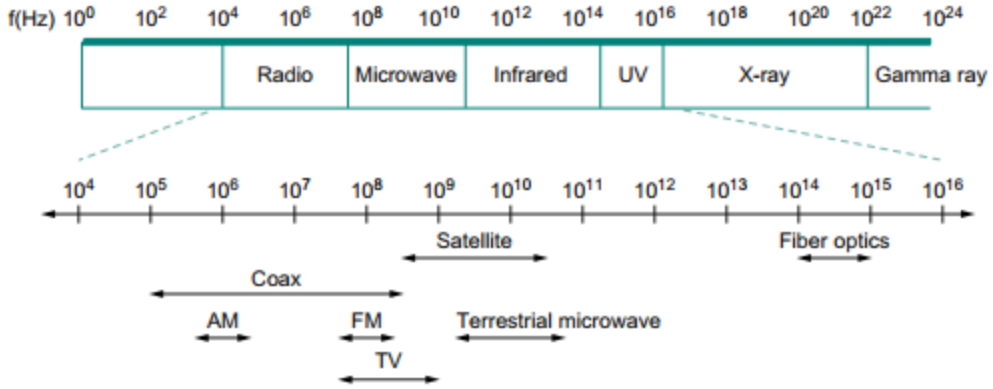
On the left, we see a variety of end-user devices ranging from mobile phones to PDAs to full-fledged computers connected by various means to an ISP. Links all look the same on the picture although they could be of any type as mentioned earlier. There are also some links that connect routers together inside the ISP and a link that connects the ISP to the rest of the Internet. These links all look alike.

⇨ The idea is that our laptop or smartphone doesn't have to care what sort of link it is connected to. The only thing that matters is that it has a link to the Internet.

2.1 Classes of Links

For a start, all practical links rely on some sort of electromagnetic radiation propagating through a medium or, in some cases, through free space.

- One way to characterize links then, is by the **medium** they use. Typically copper wire in some form, as in Digital Subscriber Line and coaxial cable. Optical fiber, as in both commercial fiber-to-the-home services and many long-distance links in the Internet's backbone.
- **Frequency**: measured in hertz, with which the electromagnetic waves oscillate.



The problem of encoding binary data onto electromagnetic signals is kind of complex. We can think of it as being divided into two layers.

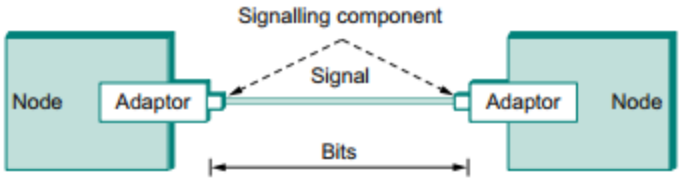
- **Modulation**: varying the frequency, amplitude, or phase of the signal to effect the transmission of information. Ex: vary the power (amplitude) of a single wavelength.
- **How links are used.**

3. Encoding

As mentioned before, signals propagate over **physical links**. The task, therefore, is to encode the binary data that the source node wants to send into the signals that the links are able to carry and then to decode the signal back into the corresponding binary data at the receiving node.

Most of the functions are performed by a network adaptor, a piece of hardware that connects a node to a link. This adaptor contains a signaling component that actually encodes bits into signals at the sending node and decodes signals into bits at the receiving node.

⇨ Signals travel over a link between two signaling components, and bits flow between network adaptors.

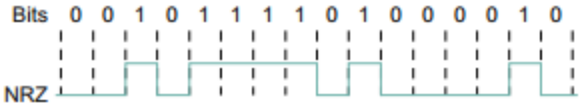


ENCODING BITS ONTO SIGNALS

The thing to do is to map the data value 1 onto the high signal and the data value 0 onto the low signal.

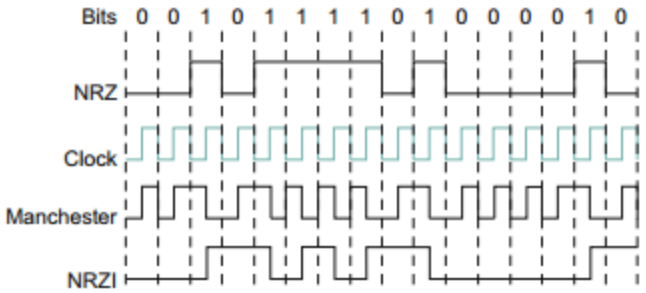
= non-return to zero (NRZ)

The problem with NRZ is that a sequence of several consecutive 1s means that the signal stays high on the link for an extended period of time. Similarly, several consecutive 0s means that the signal stays low for a long time.



⇨ Problems caused by long strings of 1s or 0s.

- **Baseline wander:** the receiver keeps an average of the signal it has seen so far and then uses this average to distinguish between low and high signals. Whenever the signal is significantly lower than this average, the receive concludes that it has just seen a 0. Likewise, a signal that is significantly higher than the average is interpreted to be a 1.
- **Clock recovery:** frequent transitions from high to low and vice versa are necessary to enable clock recovery. The clock recovery problem is that both the encoding and the decoding processes are driven by a clock. Every clock cycle the sender transmits a bit and the receiver recovers a bit. The sender's and the receiver's clocks have to be precisely synchronized in order for the receiver to recover the same bits the sender transmits.
  If the receiver's clock is even slightly faster or slower than the sender's clock, then it does not correctly decode the signal.
- **Low signal** may be interpreted as **no signal**.



SOLUTIONS

- **Non-return to zero inverted (NRZI).** The sender makes a transition from the current signal to encode a 1 and stay at the current signal to encode a 0. This solves the problem of consecutive 1s, but obviously does nothing for consecutive 0s.
- **Manchester Encoding:** The Manchester encoding results in 0 being encoded as a low-to-high transition and 1 being encoded as a high-to-low transition. Both 0s and 1s result in a transition to the signal. Therefore, the clock can be effectively recovered at the receiver.
  Problem: it doubles the rate at which signal transitions are made on the link, which means that the receiver has half the time to detect each pulse of the signal. → only 50% efficient. Bite rate = baud rate/2
  Baud rate = rate at which the signal changes.
- **4B/5B encoding:** inserts extra bits into the bit stream so as to break up long sequences of 0s or 1s. Very 4 bits of actual data are encoded in a 5-bit code that is then transmitted to the receiver.
  The 5-bit codes are selected in such a way that each one has no more than one leading 0 and no more than two trailing 0s. The resulting 5-bit code are then transmitted using the NRZI encode, which explains why the code is only concerned about consecutive 0s. → achieves 80% efficiency.

Table 2.2 4B/5B Encoding

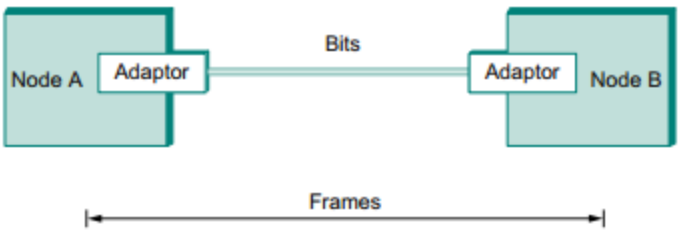
4-Bit Data Symbol	5-Bit Code
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

4. Framing

Blocks of data, not bit streams, are exchanged between nodes. It is the network adaptor that enables the nodes to exchange frames by breaking sequences of bits into frames. 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.

Challenge faced by adaptors = determining where the frame begins and ends

- Byte-oriented: BISYNC, PPP, DDCMP
- Bit-oriented: HDLC
- Clock-based: SONET

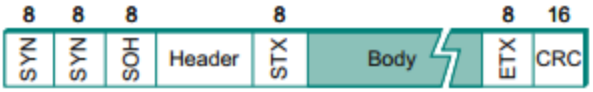


4.1 Byte-oriented protocols

⇨ View each frame as a collection of bytes rather than a collection of bits.

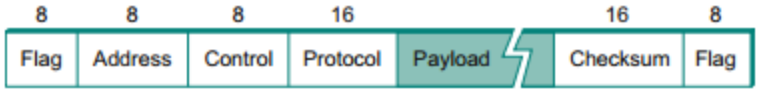
4.1.1 Sentinel-Based Approaches

A packet can be shown as a sequence of labeled files. Above each filed is a number indicating the length of that field in bits. The packets are transmitted beginning with the leftmost field.



1) SYNC uses special characters known as sentinel characters to indicate where frames start and end. The beginning of a frame is denoted by sending a special **SYN** (Synchronization) character.
2) The data portion of the frame is then contained between two more special characters: **STX** (Start of text) and **ETX** (end of text).
3) The **SOH** (Start of header) serves much the same purpose as the **STX** field.
4) CRC (cyclic redundancy check) is used to detect transmission errors.

Problem: 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 DLE (data-link-escape) character whenever it appears in the body of a frame. DLE is used to indicate that the following byte is information and not a control byte.



The point-to-point protocol, which is commonly used to carry Internet Protocol packets over various sorts of point-to-point links, is similar to BISYNC in that it also uses sentinels and character stuffing.

- The special start-of-text character, denoted as the Flag field is 01111110.
- The address and control fields usually contain default values and so are uninteresting.
- The Protocol field is used for demultiplexing. It identifies the high-level protocol such as IP or IPX.
- The checksum field is either 2 (by default) or 4 bytes long.

The PPP frame format is unusual in that several of the field sizes are negotiated rather than fixed.

4.1.2 Byte-Counting Approach

An alternative to detecting the end of a file with a sentinel value is to include the number of items in the file at the beginning of the file. The same is true in framing. The number of bytes contained in a frame can be included as a field in the frame header. The count field specifies how many bytes are contained in the frame's body.

Danger: a transmission error could corrupt the count field, in which case the end of the frame would not be correctly detected. Should this happen, the receiver will accumulate as many bytes as the bad count field indicates and then use the error detection field to determine that the frame is bad.

⇨ A framing error could possibly cause back-to-back frames to be incorrectly received.

4.2 Bit-Oriented Protocols

A bit-oriented protocol is not concerned with byte boundaries. It views the frame as a collection of bits. These bits might come from some character set. They might be pixel values in an image; or they could be instructions and operands from an executable file.

Ex: SDLC (Synchronous Data Link Control), HDLC (High-Level Data Link Control)

⇨ Delineate frame with special pattern: 01111110
⇨ **Problem**: Special pattern appears in the payload

HDLC denotes both the beginning and the end of a frame with the distinguished bit sequence **01111110**. This sequence is also transmitted during any times that the link is idle so that the sender and receiver can keep their clocks synchronized. Both protocols essentially use the sentinel approach. Because this sequence might appear anywhere in the body of the frame, the bits 01111110 might cross byte boundaries. Bit-oriented protocols use the analog of the DLE character. **= bit stuffing**

1) On the **sending side**, any time five consecutive 1s have been transmitted from the body of the message, the sender inserts a 0 before transmitting the next bit.
2) On the **receiving side**, should five consecutive 1s arrive, the receiver makes its decision based on the next bit it sees. If the next bit is a 0, it must have been stuffed, and so the receiver removes it. If the next bit is a 1, then one of two thigs is true

- Either this is the end-of-frame marker [01111110]
- An error has been introduced into the bit stream. The whole frame is then discarded. [01111111]

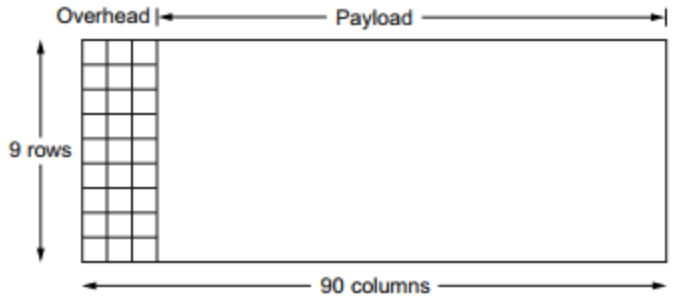
⇨ An interesting characteristic of bit stuffing, as well as character stuffing, is that the size of a frame is dependent on the data that is being sent in the payload of the frame. It is not possible to make all frames exactly the same size.

4.3 Clock-Based Framing

⇨ **Synchronous Optical Network (SONET).** It has been for many years the dominant standard for long-distance transmission of data over optical networks.
SONET addresses both the framing problem and the encoding problem.

SONET-frame has some special information that tells the receiver where the frame starts and ends. How does the receiver know where each frame starts and end?

Ex: STS-1 frame, which is the lowest-speed SONET link. It is arranged as 9 rows of 90 bytes each (frame = 810 bytes long), 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.



- SONET supports the multiplexing of multiple low-speed links in the following way. A given SONET link runs at one of a finite set of possible rates, ranging from 51.84 Mbps (STS-1) to 2488.32 Mbps (STS-48). The significance for framing is that a single SONET frame can contain sub-frames for multiple lower-rate channels.
- Each frame is 150µs long. This means that at STS-1 rates, a SONET frame is 810 bytes long, while at STS-3 rates, each SONET frame is 2430 bytes long.

5. Error detection

Bit errors are sometimes introduced into frames. This may happen because of electrical interference or thermal noise. Although errors are rare, especially on optical links, some mechanism is need to detect the errors so that corrective action can be taken.

Detecting errors is one part of the problem. The other part is correcting errors once detected. Two basic approaches can be taken when the recipient of a message detects an error:

- **Detection**: Notify the sender that the message was corrupted so that the sender can retransmit a copy of the message.
- **Correction**: Some types of error detection algorithms allow the recipient to reconstruct the correct message even after it has been corrupted.

5.1 Error-detecting codes

Basic idea: 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 of them, and they must be discarded.

⇨ Poor error detection
- It sends n redundant bits for an n-bit message. Common techniques for detection add only k extra bits.
- Many errors will go undetected

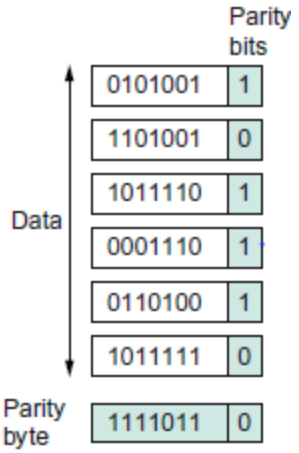
>< in general, the goal of error detecting odes is to provide a high probability of detecting errors combined with a relatively low number of redundant bits.

5.2 Two-dimensional parity

This is based on “simple” parity, which involves adding one extra bit to a 7-bit code to balance the number of 1s in the byte.

- Odd parity sets the eight bit to 1 if needed to give an odd number of 1s in the byte.
- Even parity sets the eight 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.



⇨ Two-dimensional parity catches all 1, 2, and 3-bit errors and most 4-bit errors. Not all 4-bit errors can be detected since in that case, one error could compensate another.

⇨ We must check both, horizontally and vertically to make sure to spot all the errors.

⇨ In this example we have added 14 bits of redundant information to a 42-bit message, and yet we have stronger protection against common error than the repletion code as seen earlier.

5.3 Internet Checksum Algorithm

⇨ Not used at the link level.

Basic idea: you add up all the words that are transmitted and then transmit the result of that sum. The result is 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.

Evaluation:

- Small number of redundant bits
- Does not score well for strength of error detection
- Easy to implement in software
- Compensating errors go undetected

5.4 Cyclic redundancy check

⇨ Uses some fairly powerful mathematics to achieve the goal of detecting errors using a small number of redundant bits.

Think of an (n+1) bit message represented by an n degree polynomial, that is, a polynomial whose highest-order term is x^n. the message is represented by a polynomial by using the value of each bit in the message as the coefficient for each term in the polynomial, starting with the most significant bit to represent the highest-order term.

$$\begin{aligned} M(x) &= 1 \times x^7 + 0 \times x^6 + 0 \times x^5 + 1 \times x^4 \\ &\quad + 1 \times x^3 + 0 \times x^2 + 1 \times x^1 \\ &\quad + 0 \times x^0 \\ &= x^7 + x^4 + x^3 + x^1 \end{aligned}$$

⇨ We can thus think of a sender and a receiver as exchanging polynomials with each other.

For the purposes of calculating a CRC, a sender and receiver have to agree on a divisor polynomial, C(x). C(x) is a polynomial of degree k. For example, suppose C(x) = x3 +x2 +1. In this case, k = 3. The answer to the question "Where did C(x) come from?" is, in most practical cases, "You look it up in a book." In fact, the choice of C(x) has a significant impact on what types of errors can be reliably detected, as we discuss below.

When a sender wishes to transmit a message M(x) that is n+1 bits long, what is actually sent is the (n+1)-bit message plus k bits. We call the complete transmitted message, including the redundant bits, P(x).

What we are going to do is contrive to make the polynomial representing P(x) exactly divisible by C(x); we explain how this is achieved below. If P(x) is transmitted over a link and there are no errors introduced during transmission, then the receiver should be able to divide P(x) by C(x) exactly, leaving a remainder of zero. On the other hand, if some error is introduced into P(x) during transmission, then in all likelihood the received polynomial will no longer be exactly divisible by C(x), and thus the receiver will obtain a nonzero remainder implying that an error has occurred.

- Transmit polynomial P(X) that is evenly divisible by C(x). Subtract remainder of $M(x)x^k/C(x)$ from $M(x)x^k$.
- Receiver polynomial P(x) + E(x)
  E(x) = 0 implies no errors
- Divide (P(x) + E(x)) by C(x); remainder zero if E(x) was zero (no error), or E(x) is exactly divisible by C(X).

Evaluation:

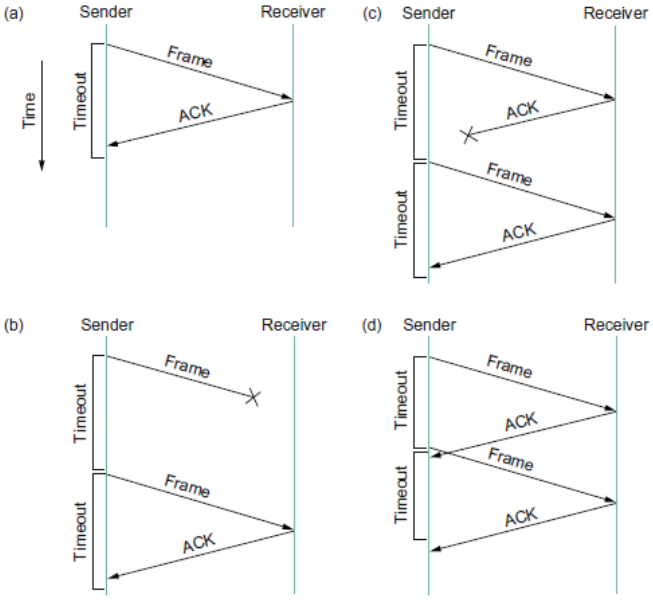
CRC will detect:

- All single-bit errors, as long as the x^k and x0 terms in C(x) have non-zero coefficients.
- All double-bit errors, as long as C(x) contains a factor with at least three terms
- Any odd number of errors, as long as C(x) contains the factor (x+1).
- Any "burst" error for which the burst length is less than k bits.
- Most (but not all) burst errors of larger than k bits.

6. Reliable transmission

Frames are sometimes corrupted while in transit, with an error code like CRC used to detect such errors. While some error codes are strong enough also to correct errors, in practice the overhead is typically too large to handle the range of bit and burst errors that can be introduced on a network link. Some errors may be severe to be corrected. As a result, some corrupt frames must be discarded.

- **Acknowledgement** is a small control frame that a protocol sends back to its peer saying that it has received an earlier frame. The receipt of an acknowledgment indicates to the sender of the original frame that its frame was successfully delivered.
- **Timeout**: action of waiting a reasonable amount of time. If the sender does not receive an acknowledgement after a reasonable amount of time, then it retransmits the original frame.
  ⇨ The strategy of using acknowledgements a timeouts to implement reliable delivery is sometimes called automatic repeat request (ARQ).



■ **FIGURE 2.17** Timeline showing four different scenarios for the stop-and-wait algorithm. (a) The ACK is received before the timer expires; (b) the original frame is lost; (c) the ACK is lost; (d) the timeout fires too soon.

a) Situation in which the ACK is received before the timer expires
b) The original frame is lost
c) The ACK gets lost
d) Situation in which the timeout fires too soon.

6.1 Stop-and-wait

After transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame. If the acknowledgment does not arrive after a certain period of time, the sender times out and retransmits the original frame.

There is one important subtlety in the stop-and-wait algorithm. Assume that the sender sends a frame and the receiver acknowledges it, but the acknowledgment is either lost or delayed in arriving. This is illustrated by c) and d). in both cases, the send times out and retransmits the original frame, but the receiver will think that it is the next frame, since it correctly received an acknowledged the first frame. → has the potential to cause duplicate copies of a frame to be delivered.

Solution: the header for a stop-and-wait protocol usually includes a 1-bit sequence number. That is, the sequence for each frame alternate. Thus, when the sender retransmits frame 0, the receiver can determine that it is seeing a second copy of frame 0 rather than the first copy of frame 1 and therefore can ignore it.

Evaluation

(-) It allows the sender to have only one outstanding frame on the link at a time, and this may be far below the link's capacity.

6.2 Sliding Window

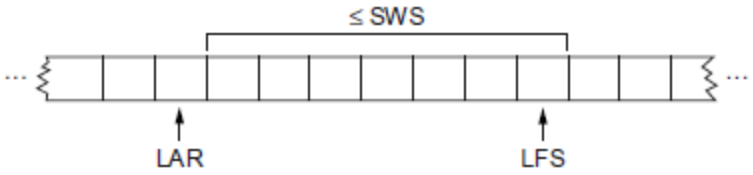
⇨ Allows multiple outstanding frames (un-ACKed)

First, the sender assigns a sequence number, denoted SeqNum, to each frame. The sender maintains three state variables

1) The **SWS** = send window size, gives the upper bound on the number of outstanding frames that the sender can transmit.
2) **LAR**: denotes the sequence number of the last acknowledgment received.
3) **LFS**: sequence of the last frame sent.

⇨ LFS – LAR <= SWS

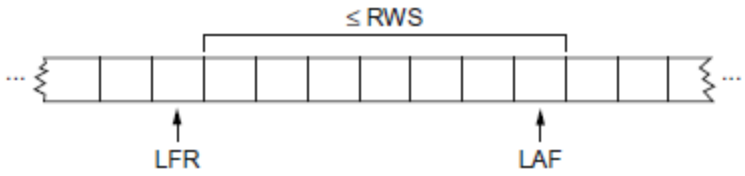
When an acknowledgement arrives, the sender moves LAR to the right, thereby allowing the sender to transmit another frame. Also the sender associates a timer with each frame it transmits, and it retransmits the frame should be timer expire before an ACK is received.



The receiver maintains the following three variables

- The receiver window siz, RWS: gives the upper bound on the number of out-of-order frames that the receiver is willing to accept
- LAT denotes the sequence number of the largest acceptable frame
- LFR denotes the sequence number of the last frame received.

⇨ LAF – LFR <= RWS



When a frame with sequence number SeqNum arrives, the receiver takes the following action.

- ➢ If SeqNum <= LFR or SeqNum > LAF, then the frame is outside the receiver's window and it is discarded.
- ➢ If LFR < SeqNum <= LAF, then the frame is within the receiver's window and it is accepted.

The receiver then needs to decide whether or not to send an ACK. Let SeqNumToAck denote the largest sequence number not yet acknowledged, such that all frames with sequence numbers less than or equal to SeqNumToAck have been received. The receiver acknowledges the receipt of SeqNumToAck, even if higher numbered packets have been received. This acknowledgement is said to be cumulative. It then sets LFR = SeqNumToAck and adjusts LAF = LFR + RWS.

Improvements

- Negative acknowledgement
- Duplicate ACKs
- Selective Acknowledgement (SACK)

Sliding window serves three different roles

- Reliable delivery of frames over an unreliable link or network
- Order preservation: higher level protocols receive data in correct order
- Flow control: receiver controls speed of sender

6.3 Concurrent Logical Channels

⇨ An important consequence of previous approaches is that the frames sent over a given like are not kept in any particular order. The protocol also implies nothing about flow control.

Basic idea: multiplex several logical channels onto a single point-to-point link and to run the stop-and-wait algorithm on each of these logical channels. There is no relationship maintained among the frames sent on any of the logical channels, yet because a different frame can be outstanding on each of the several logical channels the send can keep the link full.

The sender keeps 3 bits of stat for each channel

- A Boolean saying whether the channel is currently busy
- The 1-bit sequence number to use the next time a frame is sent on this logical channel
- The next sequence number to expect on a frame that arrives on this channel

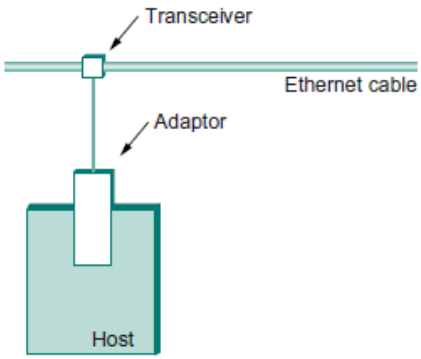
When the node has a frame to send, it uses the lowest idle channel, and otherwise it behaves just like stop-and-wait.

7. Ethernet and multiple access networks

- ⇨ Became the dominant local area networking technology.
- ⇨ Extremely popular in campus networks and data centers
- ⇨ The Ethernet is a multiple-access network, meaning that a set of nodes sends and receives frames over a shared link. It can be seen like a bus that has multiple stations plugged into it.

Fundamental question: how to mediate access to a shared medium fairly and efficiently. → algorithm that controls when each node can transmit.

7.1 Physical properties

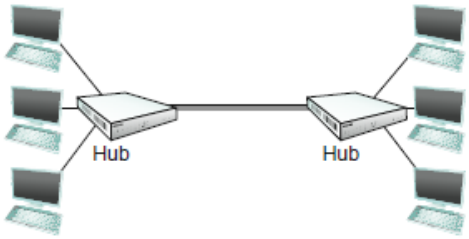


Ethernet segments were originally implemented using coaxial cable of length up to 500m. Hosts connected to an Ethernet segment by tapping into it. A **transceiver**, a small device directly attached to the tap, detected when the line was idle and drove the signal when the host was transmitting. It also received incoming signals. The transceiver, in turn, connected to an Ethernet adaptor, which was plugged into the host.

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. Repeaters understand only bits, not frames; however, no more than four repeaters could be positioned between any pair of hosts, meaning that a classical Ethernet had a total reach of only 2500m.

Why? The farther apart two nodes are, the longer it takes for a frame sent by one to reach the other, and the network is vulnerable to a collision during this time.

Any signal placed on the Ethernet by a host is broadcast over the entire network. That is, the signal is propagated in both directions, and repeaters and hubs forward the signal on all outgoing segments. Terminators attached to the end of each segment absorb the signal and keep it from bouncing back and interfering with trailing signals.



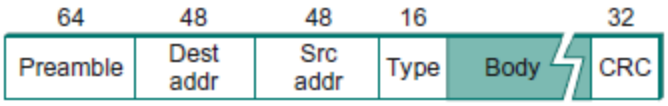
It is also possible to create a multiway repeater, sometimes called a hub. A hub just repeats whatever it hears on one port out all its other ports.

7.2 Access Protocol

The Ethernet's Media Access Control (MAC) controls access to a shared Ethernet link.

FRAME FORMAT

- The 64-bit preamble allows the receiver to synchronize with the signal. It is a sequence of alternating 0s and 1s.
- Source and destination hosts are identified with a 48-bit address.
- Packet type field serves as the demultiplexing key.
- 32-bit CRC
- Body: each frame contains up to 1500 bytes of data. Minimally, a frame must contain at least 46 bytes of data, even if this means the host has to pad the frame before transmitting it.



ADDRESS

- **Unicast address:** Each host on an Ethernet has a unique Ethernet address. Technically, the address belongs to the adaptor, not the host. Ethernet addresses are printed in form humans can read as a sequence of six numbers separated by colons. Each number corresponds to 1byte of the 6-byte address and is given by a pair of hexadecimal digits.

Each frame transmitted on an Ethernet is received by every adaptor connected to that Ethernet. Each adaptor recognizes those frames addressed to its address and passes only those frames on to the host.

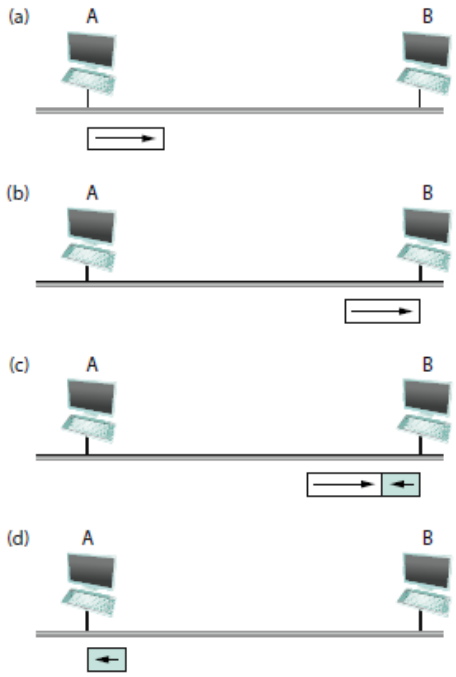
- In addition to these unicast addresses, an Ethernet address consisting of all 1s is treated as a **broadcast address**. All adaptors pass frames addressed to the broadcast address up to the host.
- An address that has the first bit set to 1 bit is not the broadcast address is called a **multicast address**.
- Promiscuous mode: when the adaptor delivers all received frames to the host.

TRANSMITTER ALGORITHM

- 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 of 1500 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.

It is possible for two (or more) adaptors to begin transmitting at the same time, either because both found the line to be idle or because both had been waiting for a busy line to become idle. When this happens, the two (or more) frames are said to collide on the network. Each sender, because the Ethernet supports collision detection, is able to determine that a **collision** is in progress.
When an adaptor detects that its frame is colliding with another, it first makes sure to transmit a 32-bit jamming sequence and then stops the transmission. Thus, a transmitter will minimally send 96 bits in the case of a collision: 64-bit preamble + 32-bit jamming sequence.

Illustration of the worst-case scenario, where hosts A and B are at opposite ends of the network.



a) Host A begins transmitting a frame at time t.
b) It takes it one link latency (d) for the frame to reach host B. Thus, the first bit of A's frame arrives at B at time t + d.
c) An instant before host A's frame arrives (B still sees an idle line), host B begins to transmit its own frame. B's frame will immediately collide with A's frame, and this collision will be detected by host B.
d) Host B will send the 32-bit jamming sequence as described above. B's frame will be a runt. Unfortunately, host A will not know that the collision occurred until B's frame reaches it, which will happen one link latency later, at time t + 2 x d. Host A must continue to transmit until this time in order to detect the collision.

⇨ Once an adaptor has detected a collision and stopped its transmission, it waits a certain amount of time and tries again. Each time it tries to transmit but fails, the adaptor doubles the amount of time it waits before trying again. = **Exponential backoff.**

7.3 Experience with Ethernet

- Ethernets work best under lightly loaded conditions. This is because under heavy loads too much of the network's capacity is wasted by collisions. A utilization of over 30% is considered heavy on an Ethernet.
- Most Ethernets are used in a conservative way, having fewer than 200 hosts connected to them, which is far fewer than the maximum of 1024. They were also far shorter than 2500m.

8. Wireless

⇨ Wireless technologies differ from wired links!

- Like wired links, issues of **bit errors** are of great concern, due to the unpredictable noise environment of most wireless. Framing and reliability also have to be addressed.
- Unlike wired links, **power** is a big issue for wireless, especially because wireless links are often used by small mobile devices that have limited access to power.

Restricted power output to avoid interference. Ex: only 4 Watt allowed in license-exempt band. **Result**: limited distance.

- Concerns about interference with other devices and usually regulations about how much power a device may emit at any given frequency.
- Wireless media are inherently multi-access. It is difficult to direct your radio transmission to just a single receiver or to avoid receiving radio signals from any transmitter with enough power in the neighborhood.

⇨ Media access control is a central issue for wireless links.

⇨ Since it is hard to control who receives the signal when you transmit over the air, issues of eavesdropping may also be addressed.

Challenge: Because wireless links all share the same medium, the challenge is to share that medium efficiently, without unduly interfering with each other. Most if this sharing is accomplished by dividing it up along the dimensions of frequency and space.

➔ Exclusive use of a particular frequency in a particular geographic area may be allocated to an individual entity such as a corporation. It is feasible to limit the area covered by an electromagnetic signal because such signals weaken, or attenuate, with the distance from their origin.
Devices that use license-exempt frequencies are still subject to certain restrictions to make that otherwise unconstrained sharing work. Most important of these is a limit on transmission power. This limits the range of a signal, making it less likely to interfere with another signal. Ex: a cordless phone might have a range of about 100 feet.

SPREAD SPECTRUM

⇨ When spectrum is shared among many devices and applications

⇨ **Spread the signal over a wider frequency band,** as to minimize the impact of interference from other devices.

Frequency hopping (FHSS) involves transmitting the signal over a (pseudo-) random sequence of frequencies. The first transmitting at one frequency, then a second, then a third, and so on.

This scheme reduces interference by making it unlikely that two signals would be using the same frequency for more than the infrequent isolated bit.

ADVANTAGES:

- Simple implementation
- Tolerance of interference

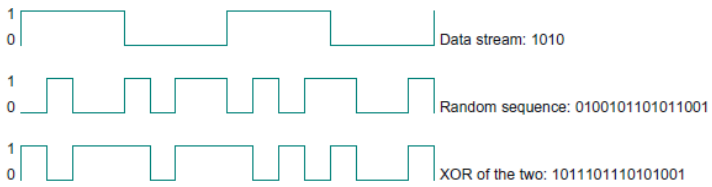
- Protection against eavesdropping
- Simultaneous transmission over multiple access points
- Scalable

DRAWBACKS

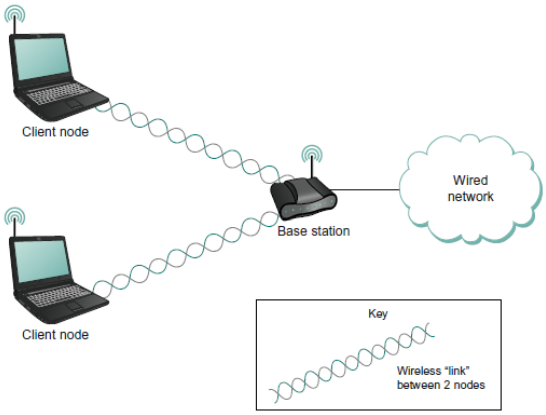
- Lower rate and range
- Complex hand-off from cell to cell
- Vendors seem to prefer DSSS

Direct sequence DSSS adds redundancy for greater tolerance of interference. Each bit of date is represented by multiple bits in the transmitted signal so that, if some of the transmitted bits are damaged by interference, there is usually enough redundancy to recover the original bit.

For each bit, send XOR of that bit and n pseudo-random bits. The sender and receiver know RN generator to generate pseudo-random bits. Transmitted values are spread over frequency band that is n times wider.



In many wireless networks today, we observe that there are **two different classes of endpoints**.



- **Base station:** usually has no mobility but has a wired (or at least high-bandwidth) connection to the Internet or other networks.
- **Client node:** this is often mobile and relies on its link to the base station for all of its communication with other nodes.

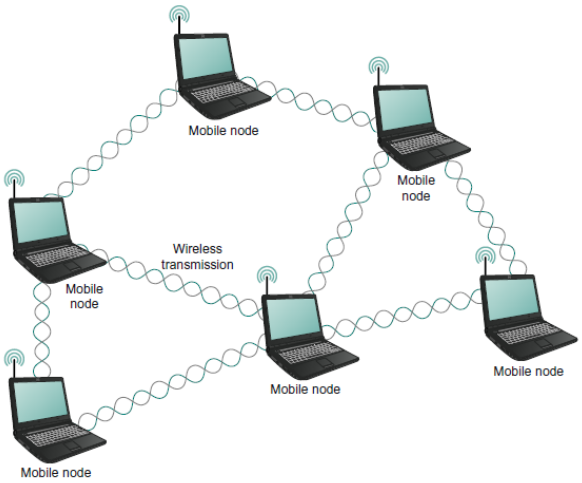
One of the interesting aspects of wireless communication is that it naturally supports point-to-point communication, because radio waves sent by one device can be simultaneously received by many devices. However, it is often useful to create a point-to-point link abstraction for higher layer protocols.

The topology represented above implies three qualitatively different **levels of mobility**:

- **No mobility:** when a receiver must be in a fixed location to receive a directional transmission from the base station.
- **Mobility within the range of a base**, as is the case with Bluetooth.
- **Mobility between bases**, as is the case with cell phones, 3G and Wi-Fi.

Alternative topology: Mesh or ad hoc network.

In a wireless mesh, nodes are peers. That is, there is no special base station node. Messages many be forwarded via a chain of peer nods as long as each node is within range of the preceding node. This allows the wireless portion of a network to extend beyond the limited range of a single radio.



8.1 802.11/WI-FI

WI-FI is technically a trademark owned by a trade group called the WI-FI alliance, which certifies product compliance with 802.11. Like Ethernet, 802.11 is designed for use in a limited geographical area (home, office buildings, campuses), and its primary challenge is to mediate access to a shared communication medium.

PHYSICAL PROPERTIES

- 802.11 defines a number of physical layers that operate in various frequency bands and provide a range of different data rates.
- The original 802.11 defined two radio-based physical layers standards, one using frequency hopping and the other using

direct sequence spread spectrum. Both provided data rates in the 2Mbps range.

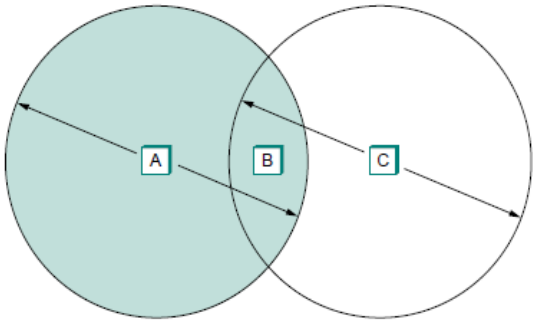
- Orthogonal frequency division multiplexing (OFDM)
  This band is less used, so there is less interference. On the other hand, there is more absorption of the signal and it is limited to almost line of sight.
- 802.11n has appeared on the scene. It achieves considerable advances in maximum possible data rate using multiple antennas and allowing greater wireless channel bandwidths. The use of multiple antennas is often called MIMO for multiple-input, multiple-output.

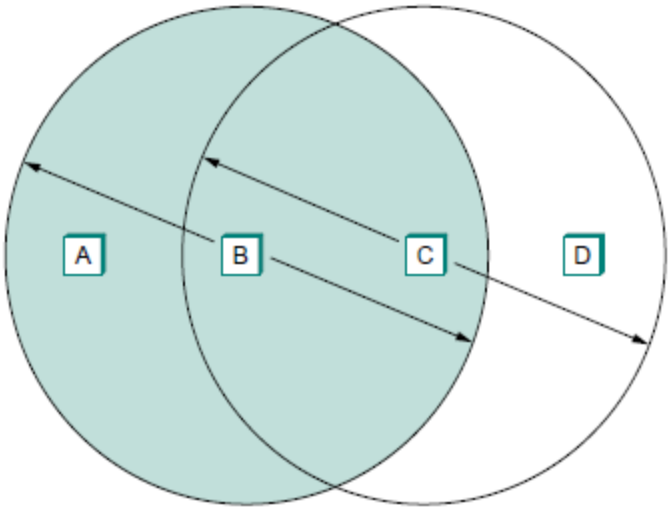
  It is common for commercial products to support more than one flavor of 802.11. Some base stations support all four variants. This not only ensures compatibility with any device that supports any one of the standards but also makes it possible for two such products to choose the highest bandwidths option for a particular event.

COLLISION AVOIDANCE

- ⇨ At the 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 collusion occurs, and, to a first approximation, this is what 802.11 does.
- ⇨ Additional complication: while a node on an Ethernet receives every other nodes transmission and can transmit and receive at the same time, neither of these conditions holds for wireless nodes. → makes detection of collisions more complex.
  The reason why wireless nodes cannot usually transmit and receive at the same time is that the power generated by the transmitter is much higher than any received signal is likely to be and so swamps the receiving circuitry.
  The reason why a node may not receive transmissions from another node is because that node may be too far away or blocked by an obstacle.

A and C are both within range of B but not each other. Suppose both A and C want to communicate with B and so they each send it a frame. A and C are unaware of each other since their signals do not carry that far. These two frames collide with each other at B, but unlike an Ethernet, neither A nor C is aware of this collision. A and C are said to be **hidden nodes** with respect to each other.





Exposed node problem: occurs under circumstances where each of the four nodes is able to send and receive signals that reach just the nodes to its immediate left and right. For instance, B can exchange frames with A and C but it cannot reach D, while C can reach B and D but not A. If B is sending to A, node C is aware of this communication because it hears B's transmission. It would be a mistake, however for C to conclude that it cannot transmit to anyone just because it can hear B's transmission. If C wants to transmit to node D, this is not a problem since C's transmission to D will not interfere with A's ability to receive from B.

802.11 addresses these problems by using **CSMA/CA**, where the CA stands for collision avoidance, in contrast to the collision detection of **CSMA/CD** used on Ethernets.

The carrier sense part seems simple enough: before sending a packet, the transmitter checks if it can hear any other transmissions. If not, it sends. However, because of the hidden terminal problem, just waiting for the absence of signals from other transmitter **does not guarantee that a collision will not occur** from the perspective of the receiver.

⇨ For this reason, one part of CSMA/CA is an explicit ACK from the receiver to the sender. If the packet was successfully decoded and passed its CRC at the receiver, the receiver sends an ACK back to the sender. Neighbors must remain silent until they see this ACK.

If a collision does occur, it will render the entire packet useless. For this reason, 802.11 adds an optional mechanism called RTS-CTS (Ready to Send-Clear to send). This goes some way toward addressing the hidden terminal problem.

The sender sends an RTS, a short packet, to the intended receiver, and if that packet is received successfully the receiver responds with another short packet, the CTS. Even though the RTS may not have been heard by a hidden terminal, the CTS probably will be. This tells the nodes within range of

the receiver that they should not send anything for a while, the amount of time of the intended transmission is included in the RTS and CTS packets.

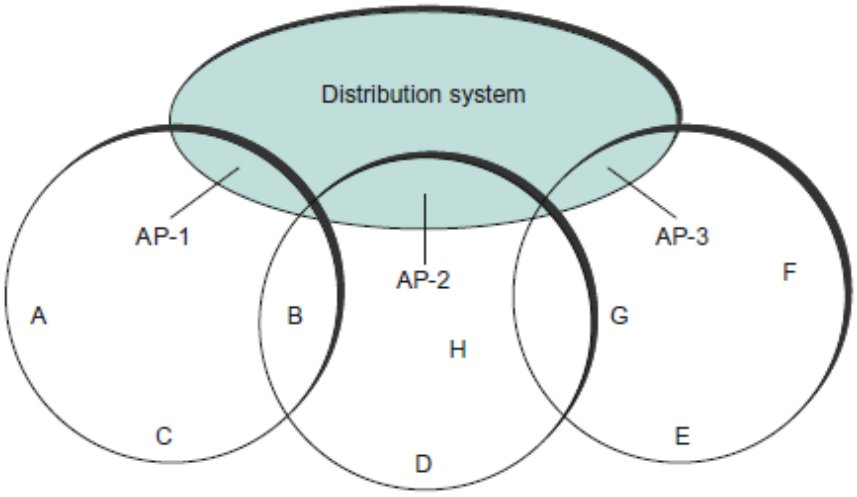
Collision

- If two nodes send an RTS simultaneously
- No collision detection in 802.11
- When senders don't receive CTS they realize collision has happened
- Try again after exponential back off

DISTRIBUTION SYSTEM

At the current time, nearly all 802.11 networks use a **base-station-oriented topology**. Instead of all nodes being created equal, some nodes are allowed to roam and some are connected to a wired network infrastructure. 802.11 calls these base stations **access points**, and they are connected to each other by a so-called **distribution system**.

This is a distribution system that connects three access points, each of which services the nodes in some region. Each access point operates on some channel in the appropriate frequency range, and each AP will typically be on a different channel than its neighbors.



Although two nodes can communicate directly with each other if they are within reach of each other, the idea behind this configuration is that **each node associates itself with one access point**. For node A to communicate with node E, for instance, A first sends a frame to its access point AP1, which forwards the frame across the distribution system to AP3, which finally transmits the frame to E.

AP1 may have used the **bridging protocol** to forward the message to AP3.

The **technique for selecting an AP** is called **scanning** and involves the following four steps:

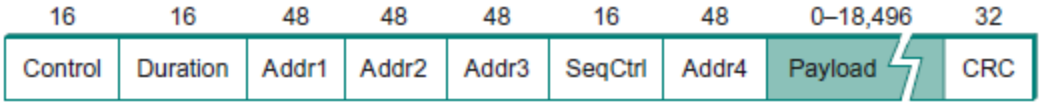
- The node sends a Probe frame while moving
- All AP's within reach reply with ProbeResponse frame
- Node selects one AP, sends it AssociateRequest frame
- AO replies with AssociationResponse frame
- New AO informs old AO via Ds.

- ⇨ A node engages this protocol whenever it joins the network, as well as when it becomes unhappy with its current AP. This might happen, for example, because the signal from its current AP has weakened due to the node moving away from it. Whenever a node acquires a new AP, the new AP notifies the old AP of the change via the distribution system.

FRAME FORMAT

1) Source and destination node addresses (each = 48 bits long)
2) 32-bit CRC.
3) Up to 2312 bytes of data
4) The control field contains three subfields of interest.
5) A 6-bit field that indicates whether the frame carries data, is an RTS or CTS frame or is being used by the scanning algorithm and a pair of1-bit fields.

- ⇨ The 802.11 frame format contains four, rather than two addresses.
- ⇨ Remark: no preamble? No frame delineation.



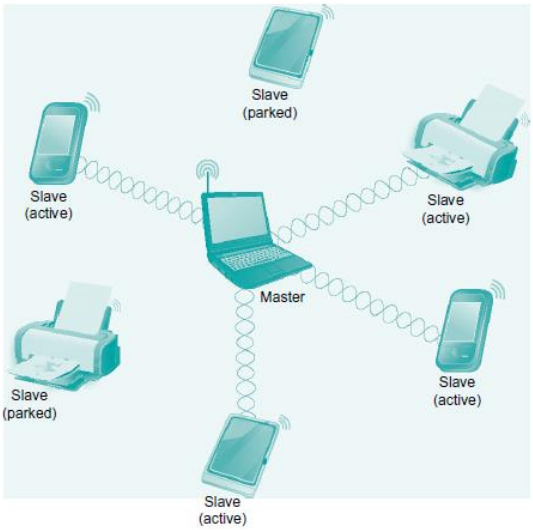
8.2 Bluetooth

- ⇨ Fills the niche of very short range communication between mobile phones, PDAs, notebook computers, and other personal or peripheral devices.
- ⇨ Ex: can be used to connect a mobile phone to a headset or a notebook computer to a keyboard.
- ⇨ In such application it is not necessary to provide much range or bandwidth.

The basic Bluetooth network configuration, called a piconet, consists of a master device and up to seven devices. 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.

Since Bluetooth operates in a license-exempt band, it is required to use a spread spectrum technique to deal with possible interference in the band. It uses frequency-hopping with 79 channels, using each for 625µs at a time.

A slave device can be parked. That is, it is 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 salve devices.



8.3 Cell Phone technologies

- ⇨ Data services based on cellular standards have become increasingly popular. One drawback compared to the technologies just described has tended to be the cost to users, due in part to cellar's use of licenses spectrum, which has been sold off to cellular phone operators for astronomical sums.
- ⇨ 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.

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.

SECURITY OF WIRELESS LINKS

One of the obvious problems of wireless links compared to wires or fibers is that you can't be too sure where your data has gone. You can probably figure out if it was received by the intended receiver, but there is no telling how many other receivers might have also picked up your transmission.

Users might be able to consume resources that you would prefer to consumer yourself, such as the finite bandwidth between your house and your ISP.

⇨ That's why wireless networks typically come with some sort of mechanism to control access to both the link itself and the transmitted data.
= wireless security.

CHAPTER 3 : INTERNETWORKING

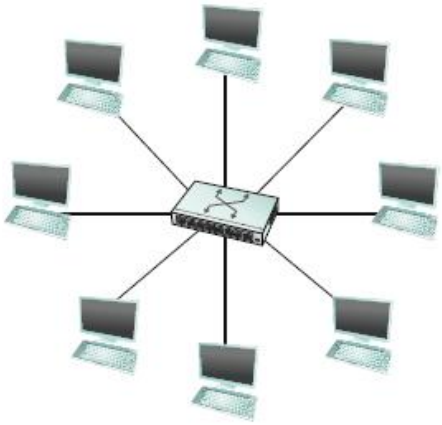
⇨ How do we build networks of **global scale**?

Internetworking = concept of interconnecting different types of networks.

1) We need a way to interconnect links.
2) Devices that interconnect **links of the same type** are often called **switches**. → important class of switches today are those used to **interconnect Ethernet** segments. These switches are also sometimes called **bridges**.
   Core job of a bridge is to take packets that arrive on an input and forward them to the right output so that they will reach their appropriate destination.
3) We need a way to interconnect disparat networks and links.
   = **gateways** or **routers**
4) Finding a suitable path or route through a network.

1. Switching and bridging

Switch = mechanism that allows us to interconnect links to form a larger network. It is a multi-input, multi-output device that transfers packets from an input to one or more outputs.



- 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.**
- 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.

Every host on a switched network has its own link to the switch, so it may be entirely possible for many hosts to transmit at the full link speed (bandwidth), provided that the switch is designed with enough aggregate capacity.

⇨ Switched networks are considered more scalable than shared-media networks because of this ability to support many hosts at full speed.

⇨ 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.

Switching or forwarding = receiving incoming packets on one of its links and transmit them on some other links

⇨ Which output link to place each packet on? It looks at the header of the packet for an identifier that it uses to make the decision. Details of how it uses to make the decision vary.

1) Datagram/connectionless approach
2) Virtual circuit/connection-oriented approach
3) Source routing

We need to have a way to identify the end nodes. Such identifiers are usually called **addresses**. Another assumption is that there is some way to identify the input and output ports of each switch. There are at least two sensible ways to identify ports

4) Number of each port
5) Identify the port by the name of the nod to which it leads.

1.1 Datagrams

Basic idea: include in every packet enough information to enable any switch to decide how to get it to its destination. That is, every packet contains the complete destination address.

To decide how to forward a packet, a switch consults a forwarding table. This table shows the forwarding information that switch needs to forward datagrams in the example network.

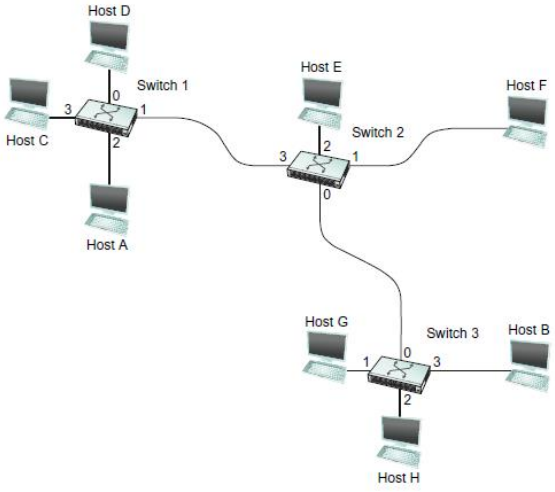


Table 3.1 Forwarding Table for Switch 2

Destination	Port
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

This table is easy to build when you have a complete overview of the network. On the contrary, it is a lot harder to create the forwarding tables in large, complex netwoks with dynamically changing topologies and multiple paths between destinations. That harder problem is known as routing. This is a process that takes place in the background so tahat, when a data packet turns up, we will have the right information in the forwarding table to be able to fforward, or swithc, the packet.

Characteristics of datagram networks

6) A host can send a packet anywhere at any time, since any packet that turns up at a switch can be immediately forwarded.
   Datagram networks = connectionless >< connection-oriented networks, in which some connection state needs to be established before the first data packet is sent.
7) When a host sends a packet, it has no way of knowing if the network is capable of delivering it or if the destination host is even up and running.
8) Each packet is forwarded independently of previous packets that might have been sent to the same destination. Thus, two successive packets from host A to host B may follow completely different paths.
9) A switch or link failure might not have any serious effect on communication if it is possible to find an alternate route around the failure and to update the forwarding table accordingly.

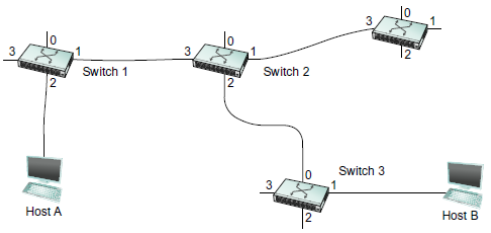
1.2 Virtual Circuit Switching

= connection-oriented model

It requires setting up a virtual connection from the source host to the destination host before any data is sent.

We suppose that host A wants to send packets to host B. We can think of this as a **two-stage process.**

1) **Connection setup:** necessary to establish a connection state in each of the switches between the source and destination hosts. The connection state for a single connection consists of an entry in a VC table in each switch through which the connection passes.



One entry in the VC table on a single switch contains:

10) A virtual circuit identifier that uniquely identifies the connection at this switch and which will be carried inside the header of the packets that belong to this connection.
11) An incoming interface on which packets for this VC arrive at the switch
12) An outgoing interface in which packets for this VC leave the switch
13) A potentially different VC that will be used for outgoing packets

⇨ If a packet arrives on the designated incoming interface and that packet contains the designated VCI value in its header, then that packet should be sent out this specified outgoing interface with the specified outgoing VCI value having been first placed in its header.

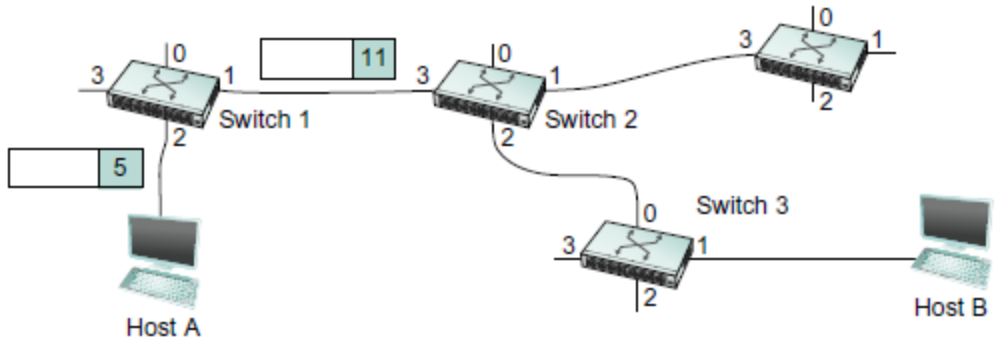
There may be many virtual connections established in the switch at one time. Also, we observe that the incoming and outgoing VCI values are generally not the same. Thus the VCI is not a globally significant identifier for the connection. It has as significance only on a given link.

There are two broad approaches to establishing connection state

- Have a network administrator configure the state, in which case the virtual circuit is permanent.
- A host can send messages into the network to cause the state to be established. = **signaling**. The resulting virtual circuits are said to be **switched**.

Table 3.2 Virtual Circuit Table Entry for Switch 1

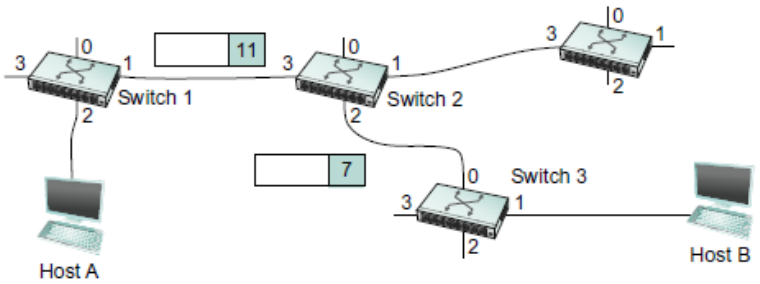
Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
2	5	1	11



2) Data transfer

Once the VC tables have been set up, the data transfer phase can proceed. For instance, for any packet that it wants to send to host B, A puts the VCI values of 5 in the header of the packet an sends it to switch 1. Switch 1 receives any such packet on interface 2 and it uses the combination of the interface and the VCI in the packet header to find the appropriate VC table entry.

⇨ In real networks of reasonable size, the burden of configuring VC tables correctly in a large number of switches would quickly become excessive using the above procedures.



To start the signaling process, host A sends a setup message into the network, that is, to switch 1. The setup message contains, among other things, the complete destination address of host B. The setup message needs to get all the way to B to create the necessary connection state in very switch along the way.

Getting the setup message to B is a lot like getting a datagram to B, in that the switches have to know which output to send the setup message to so that it eventually reaches B.

When switch 1 receives the connection request, in addition to sending it on to switch 2, it creates a new entry in its virtual circuit table for this new connection. The entry is exactly the same as shown previously. The main difference is that now the task of assigning an unused VCI value on the interface is performed by the switch of that port.

When switch 2 receives the setup message, it performs a similar process. It picks the value 11 as the incoming VCI value. Etc.

Finally, the setup message arrives as host B. Assuming that B is healthy and willing to accept a connection from host A, it too allocates an incoming VCI value, in this case 4. This VCI value can be used by B to identify all packets coming from host A.

To complete the connection, everyone needs to be told what their downstream neighbor is using as the VCI for this connection. Host B sends an acknowledgement of the connection setup to switch 3 and includes in that message the VCI that it chose. And we repeat this until we reach host A.

⇨ At this point, everyone knows all that is necessary to allow traffic to flow from host A to host B. Each switch has a complete virtual circuit table entry for the connection. Moreover, host A has a firm acknowledgement that everything is in place all the way to host B. At this point, the connection table entries are in place in all three switches just as in the administratively configured example above.

When host A no longer wants to send data to host B, it tears down the connection by sending a teardown message to switch 1. The switch removes the relevant entry from its table and forwards the message on to the other switches in the path, which similarly delete the appropriate table entries.

Characteristics of VCI

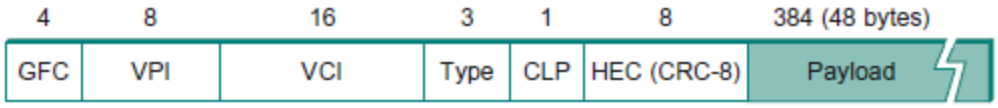
- Since host A has to wait for the connection request to reach the far side of the network and return before it can send its first data packet, there is at least one RTT of delay before data is sent.
- While the connection request contains the full address for host B, each data packet contains only a small identifier, which is only unique on one link. Thus, the per-packet overhead caused by the header is reduced relative to the datagram model.
- If a switch or link in a connection fails, the connection is broken and a new one will need to best established. Also, the old one needs to be torn down to free up table storage space I the switches.
- The issue of how a switch decides which link to forward the connection request on has been glossed over. In essence, this is the same problem as building up the forwarding table for datagram forwarding, which requires some sort of routing algorithm.

⇨ One of the nice aspects of virtual circuits is that by the time the host gets the go-ahead to send data, it knows quite a lot about the network, for instance, that there really is a route to the receiver and that the receiver is willing and able to receive data.

>< datagram network has no connection establishment phase, and each switch processes each packet independently, making it less obvious how a datagram network would allocate resources in a meaningful way. Instead, each arriving packet competes with all other packets for buffer space. If there are no free buffers, the incoming packet must be discarded.

ASYNCHRONOUS TRANSFER MODE (ATM)

⇨ Most well-known virtual circuit-based networking technology.



1) Generic Flow Control (GFC)
2) 24 bits labelled VPI and VCI. They correspond to the virtual circuit identifier introduced previously. Breaking the field into two parts allows for a level of hierarchy.
3) 8-bit cyclic redundancy check. This is known as the header error check (HEC). It uses the CRC-8-bit error correction capability on the cell header only.

⇨ Protecting the cell header is particularly important because an error in the VCI will cause the cell to be misdelivered.

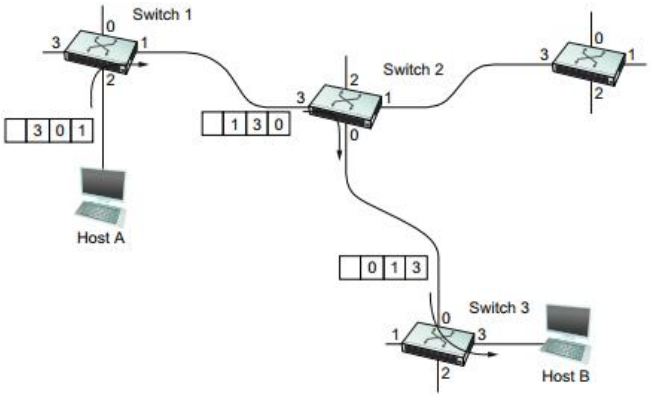
/!\ ATM comes in only one size: 53 bytes.

1.3 Source routing

⇨ Third approach to switching that uses neither virtual circuits nor conventional datagrams.
⇨ **Various ways to implement source routing.**

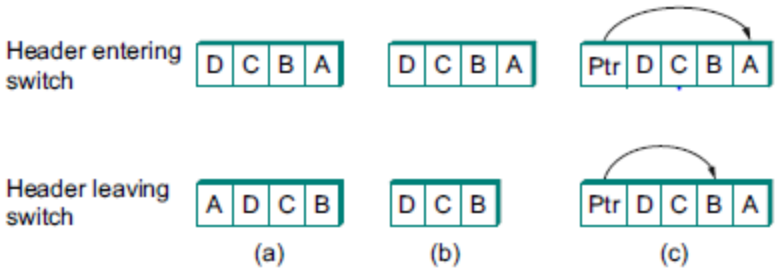
One way would be to **assign a number to each output of each switch** and to **place that number in the header of the packet**. The switching function is then very simple: For each packet that arrives on an input, the switch would read the port number in the header and transmit the packet on that output. However, since there will in general be more than one switch in the path between the sending and the receiving host, the header for the packet needs to contain enough information to allow every switch in the path to determine which output the packet needs to be placed on.

⇨ One way of doing is to put an ordered list of switch ports in the header and to rotate the list so that the next switch in the path is always at the front of the list.



We assume that each switch reads the rightmost element of the list.

- This approach assumes that host **A knows enough about the topology of the network** to form a header that has all the right direction sin it for every switch in the path.
- We **cannot predict how big the header needs to be**, since it must be able to hold one word of information for every switch on the path. → headers are of variable length with no upper bound, unless we can predict with absolute certainty the maximum number of switches through which a packet will ever need to pass.
- Some **variations** on this approach. Rather than rotate the header, each switch could just strip the first element as it uses it. **Rotation** has an advantage over **stripping**. Rotation helps keeping a copy of the complete header, which may help it figure out how to get back to host A.
  We could also think of a **pointer**, so that each switch just updates the point rather than rotating the header. This may be more efficient to implement.



⇨ Source routing can be used in both datagram networks and virtual circuit networks.

Source routes are sometimes categorized as strict or loos.

1) In a **strict source route**, every node along the path must be specified

2) **Loose source** only specifies a set of nodes to be traversed, without saying exactly how to get from one node to the next. = set of waypoints.
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 construct a correct strict source route to any destination.

1.4 Bridges and LAN Switches

⇨ Class of switch used to forward packets between LANs such as Ethernets.
= Lan Switches
⇨ Widely used in campus and enterprise networks.

Suppose you have a pair of Ethernets that you want to interconnect

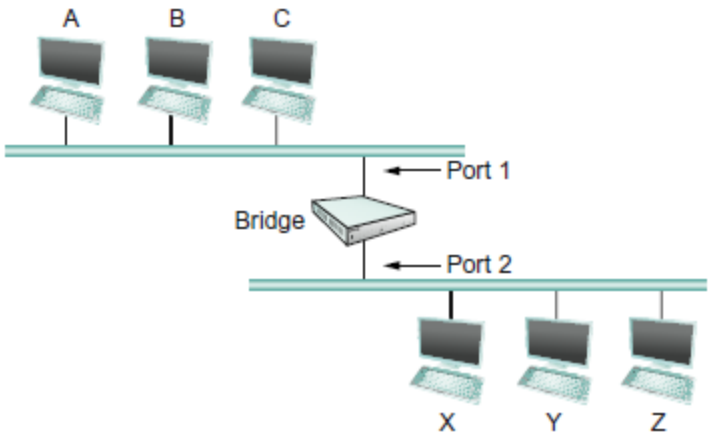
- You might try to put a repeater between them.
- An alternative would be to put a node with a pair of Ethernet adaptors between the two Ethernets and have the node forward frames from one Ethernet to the other. This node would differ from a repeater, which operates on bits, not frames, and just blindly copies the bits received on one interface to another. This node would differ from a repeater, which operates on bits, not frames, and just blindly copes the bits received on one interface to another. Instead, this node would fully implement the Ethernet's collision detection and media access protocols on each interface.
= bridge node
A collection of LANs connected by one or more bridges is usually said to form an extended LAN

- Multi input device - Multi output device - Increases the total bandwidth of a network

LEARNING BRIDGES

- The first optimization: observe that it need not forward all frames that it receives.

Ex: a frame from host A that is addressed to host B arrives on port 1. There is no need for the bridge to forward the frame out over port 2.



⇨ How does a bridge come to lean on which port the various hosts reside?

1) **Human downloading** a table into the bridge. Then, whenever the bridge receives a frame on port 1 that is addressed to host A, it would not forward the frame out on port 2. → too burdensome.
2) Bridge can learn this for itself. The idea is for each bridge to inspect the source address in all the frames It receives. So, when host A sends a frame to a host on either side of the bridge, the bridge receives this frame and records the fact that a frame from host A was just receive on port 1. In this way the bridge can build the table.

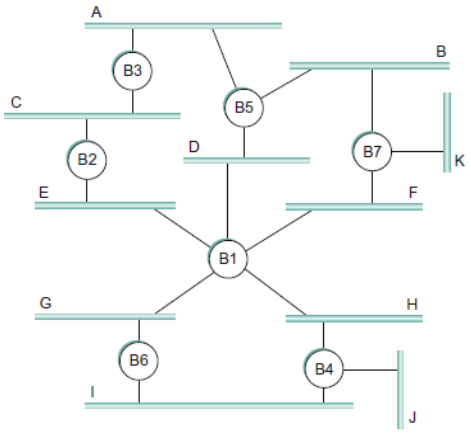
Table 3.4 Forwarding Table Maintained by a Bridge

Host	Port
A	1
B	1
C	1
X	2
Y	2
Z	2

- ⇨ Each packet carries a global address and the bridge decides which output to send a packet on by looking up that address in a table.
- ⇨ When a bridge first boots, this table is empty. Entries are added over time. Also, a timeout is associated with each entry. And the bridge discards the entry after a specified period of time. This is to protect against the situation in which a host, and as a consequence, its LAN address, is moved from one network to another.

SPANNING TREE ALGORITHM

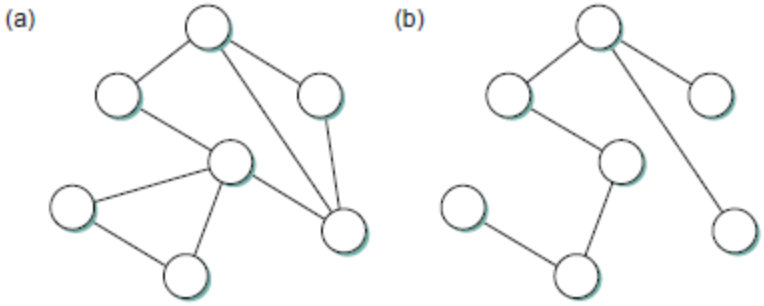
- ⇨ The preceding method works just fine until the extended LAN has a loop in it, in which case it fails. **Frames may loop through the extended LAN forever.**



Bridges B1, B4 and B6 form a loop. Suppose that a packet enters bridge B4 from Ethernet J and that the destination address is one not yet in any bridge's forwarding table: B4 sends a copy of the packet out to Ethernets H and I. Now bridge B6 forwards the packet to Ethernet G, where B1 would see it and forward it back to Ethernet H. B4 still doesn't have this destination in its table. So it forwards the packet back to Ethernets I and J. There is nothing to stop this cycle from repeating endlessly, with packets looping in both directions among B, B4 and B6.

- ⇨ Why would an extended LAN come to have a loop in it?
  1) Network managed by more than one administrator (Ex: because it spans multiple departments in an enterprise).
  2) Loops are built into the network on purpose, to provide redundancy in case of failure.
- ⇨ Whatever the cause, bridges must be able to correctly handle loops. This is addressed by having the bridges run a distributed spanning tree algorithm.

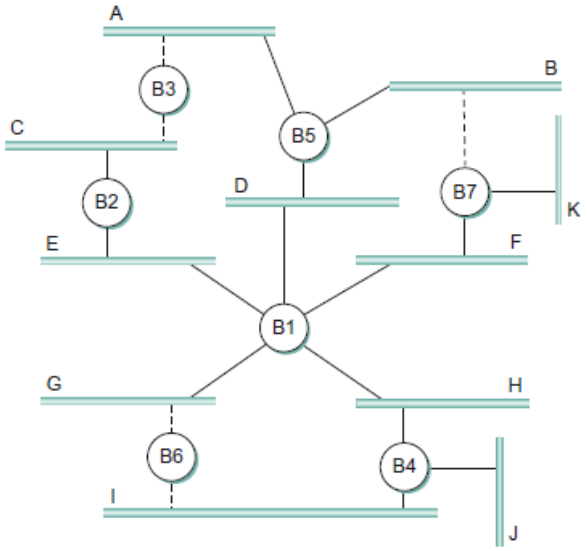
  If you think of the extended LAN as being represented by a graph that possibly has loops (cycles), then a spanning tree is a subgraph of this graph that covers (spans) all the vertices but contains no cycles. → keeps all of the vertices of the original graph but throws out some of the edges.



Spanning tree algorithm = protocol used by a set of bridges to agree upon a spanning tree for a particular extended LAN. In practice this means that each bridge decides the ports over which it is and is not willing to forward frames.

⇨ Dynamic algorithm: bridges are always prepared to reconfigure themselves into a new spanning tree should some bridge fail, and so those unused ports and bridges proved the redundant capacity needed to recover from failures.

Main idea of spanning tree algorithm → bridges select the ports over which they will forward frames. The algorithm selects ports as follows. Each bridge has a unique identifier. fThe algorithm first elects the bridge with the smallest ID as the root of the spanning tree. This root bridge always forwards frames out over all of its ports. Next, each bridge computes the shortest path to the root and notes which of its ports is on this path.



BROADCAST AND MULTICAST

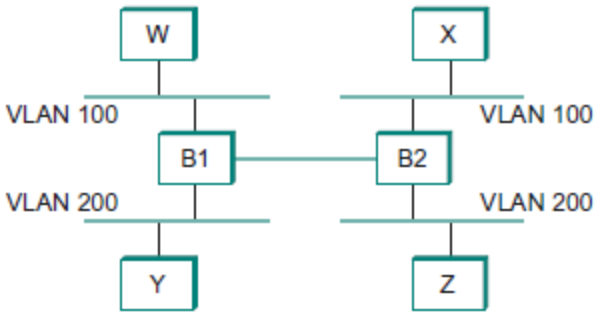
⇨ Goal of a bridge is to transparently extend a LAN across multiple networks, and since most LANs support both broadcast and multicast, then bridges must also support thes two features.

1) **Broadcast**: each bridge forwards a frame with a destination broadcast address out on each active port other than the one on which the frame was received.
2) **Multicast**: each host deciding for itself whether or not to accept the message.

LIMITATIONS OF BRIDGES

⇨ Main issues become apparent when we consider the issues of **scale and heterogeneity.**

1) **Scale**: it is not realistic to connect more than a few LANs by means of bridges. (few means tens of). The spanning tree algorithm scales linearly.
   One approach to increasing the scalability of extended LANs is the virtual LAN. VLANs allow a single extended LAN to be portioned into several seemingly separate LANs. Each virtual LAN is assigned an identifier, and packets can only travel from one segment to another if both segments have the same identifier.



Four hosts on four different LAN segments. In the absence of VLANs, any broadcast packet from any host will reach all the other hosts. Low let's suppose that we define the segments connected to hosts W and X as being one VLAN: VLAN 100. And another VLAN 200 between Y and Z. To do this, we need to configure a VLAN ID on each port of bridges B1 and B2. The link between B1 and B2 is considered to be in both VLANs.

When a packet sent by host X arrives at bridge B2, the bridge observes that it came in a port that was configured as being in VLAN 100. It inserts a VLAN header between the Ethernet header and its payload. The interesting part of the VLAN header is the VLAN ID; in this case, that ID is set to 100. The bridge now applies its normal rules for forwarding to the packet, with the extra restriction that the packet may not be sent out an interface that is not part of VLAN 100. Thus, under no circumstances will the packet—even a broadcast packet—be sent out the interface to host Z, which is in VLAN 200. The packet is, however, forwarded on to bridge B1, which follows the same rules and thus may forward the packet to host W but not to host Y.

⇨ Attractive feature of VLANs is that it is possible to change the logical topology without moving any wires or changing any addresses.

- **Heterogeneity**: bridges are fairly limited in the kinds of networks they can interconnect. Bridges make use of the network's frame header and so can support only networks that have exactly the same format for addresses.
  Can be used to connect Ethernets to Ethernets or token rings to token rigs, etc.

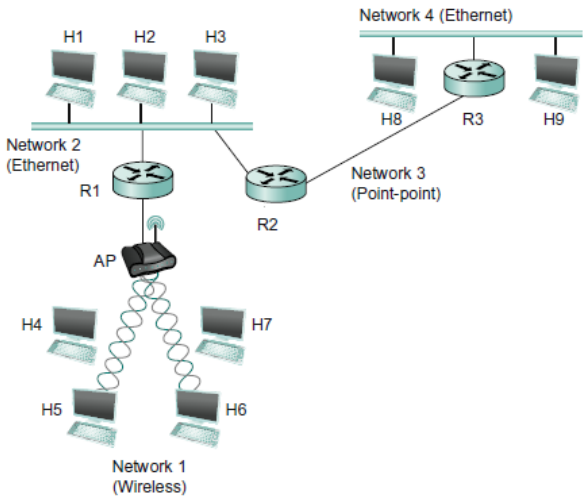
2. Basic Internetworking (IP)

⇨ Explore some ways to go beyond the limitations of bridged networks, enabling us to build large, highly heterogeneous networks with reasonably efficient routing.
= **internetworks**.

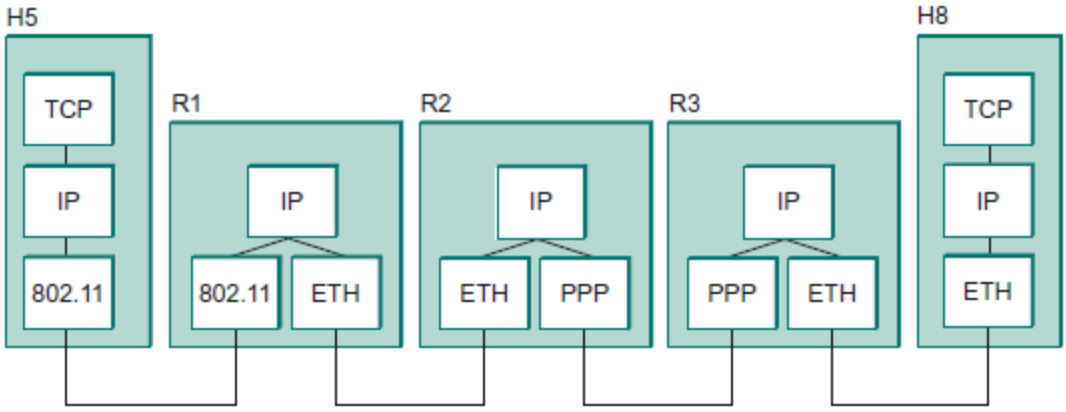
2.1 What is an Internetwork?

⇨ Refers to an arbitrary **collection of networks interconnected** to provide some sort of host-to-host packet delivery service.
⇨ When we are talking about the widely used global internetwork to which a large percentage of networks are now connected, we call it the **Internet**.

- Network: to mean either a directly connected or a switched network.
- An internetwork: interconnected collection of such networks.
  An internet is a logical network built out of a collection of physical networks. A collection of Ethernets connected by bridges or switches would still be viewed as a single network.



The Internet Protocol is the key too used today to build scalable heterogeneous internetworks. One way to think of IP is that it runs on all the nodes in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single logical internetwork.



This figure shows how hots H5 and H8 are logically connected by the internet as a reference to previous figure, including the protocol graph running on each node.

2.2 Service model

- When building an internetwork, a good place to start is to define its service model = host-to-host service you want to provide.

  Main concern: provide a host-to-host service only if this service can somehow be provided over each of the underlying physical networks.

  ⇨ Make it undemanding enough that just about any network technology that might turn up in an internetwork would be able to provide the necessary service.

IP model can be thought of as having two parts:

1) An **addressing scheme**, which provides a way to identify all hosts in the internetwork
2) A **datagram** model of data delivery.

⇨ **Best effort because**, although IP makes every effort to deliver datagrams, it makes no guarantees.
If something goes wrong and the packet gets lost, corrupted, misdelivered, or in any way fails to reach its intended destination, the network does nothing. It made its best effort, and that is all It has to do.
= **unreliable service**. It does not make any attempt to recover from the failure.

DATAGRAM DELIVERY

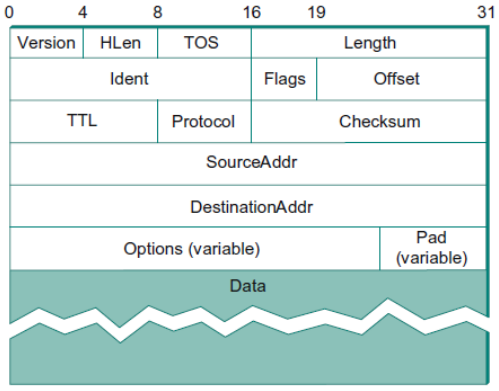
Datagram = type of packet that happens to be sent in a connectionless manner over a network. Every datagram carries enough information to let the network forward the packet to its correct destination. We just send the packet, and the network makes its best effort to gget it to the desired destination.

Best-effort is the simplest service you could ask for from an internetwork.

The ability of IP to run over anything is frequently cited as one of its most important characteristics.

PACKET FORMAT

A key part of the IP service model is the type of packets that can be carried. The IP datagram, liked most packets, consists of a header followed by a number of bytes of data.



IP Header

- The version field specifices the version of IP. The current version of IP is, and it is sometimes called IPv4.

The header processing software starts off by looking at the version and then branches off to process the rest of the packet according to the appropriate format.

- HLen specifies the length of the header in 32-bit words.
- TOS (Type of service). Its basic function is to allow packets to be treated differently based on application needs.
- Length of the datagram, including the header. It counts bytes rather than words.
- The second word of the header contains information about fragmentation,
- The third word of the header contains a TTL field (Time to Live). Originally, TTL was set to a specific number of seconds that the packet would be allowed to live. Routers along the path would decrement this field until it reached 0.
- Protocol field: demultiplexing key that identifies the higher-level protocol to which this IP packet should be passed.
- The checksum is calculated by considering the entire IP header as a sequence of 16-bit words, adding them up using ones' complement arithmetic, and taking the ones complement of the result. It makes sense to discard any packet that fails the checksum.
- SourceAddr: required to allow recipients to decide if they want to accept the packet and to enable them to reply.
- DestinationAddr: key to datagram delivery: every packet contains a full address for its intended destination so that forwarding decisions can be made at each router.
- At the end of the header there may be a number of options. The presence or absence of options may be determined by examining the header length field.

FRAGMENTATION AND REASSEMBLY

One of the problems of providing a uniform host-to-host service model over a heterogeneous collection of networks is that each network technology tends to have its own idea of how large a packet can be. Ex: an Ethernet can accept packets up to 1500 bytes long.

Two choices for the IP service model

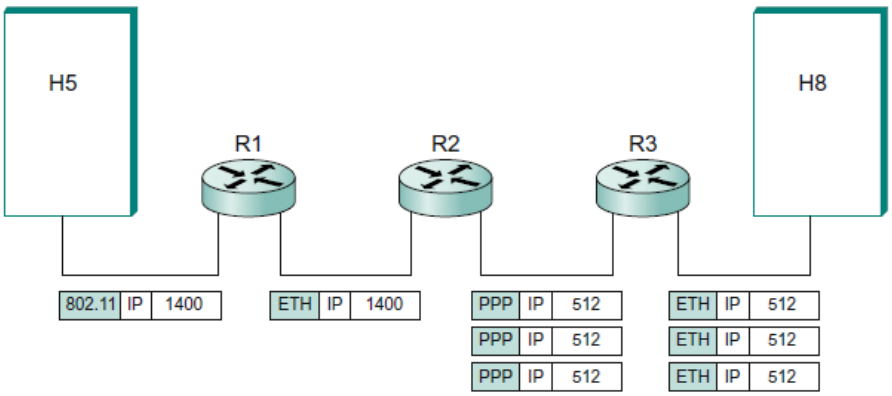
- Make sure that all IP datagrams are **small enough to fit inside one packet** on any network technology
- Provide means by which **packets can be fragmented and reassembled** when they are too big to go over a given network technology.
    - ⇨ Good choice, especially when considering the fact that new network technologies are always turning up, and IP needs to run over all of them.

Central idea: every network type has a maximum transmission unit **(MTU),** which is the largest IP datagram that it can carry in a frame. This value is smaller than the largest packet size on that network because the IP datagram needs to fit in the payload of the link-layer frame.

1) Ethernet 1500 bytes
2) WI-FI 2312 bytes
3) PPP 532 bytes

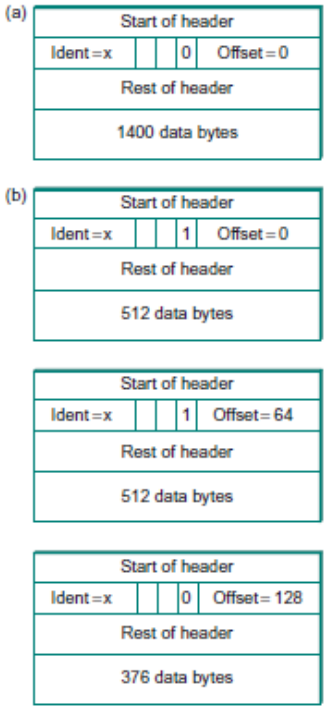
Fragment when necessary (MTU < Datagram) Fragmentation occurs in a router when it receives a datagram that it wants to forward over a network that has an MTU that is smaller than the received datagram. To enable these fragments to be reassembled at the receiving host, they all carry the same identifier in the Ident field. This identifier is intended to be unique among all the datagrams that might arrive at the destination from this source over some reasonable time period.

⇨ Should all the fragments not arrive at the receiving host, the host gives up on the reassembly process and discards the fragments that did arrive. IP does not attempt to recover from missing fragments.



⇨ Host H5 sends a datagram to host H8.
Assumptions: MTU is 1500 bytes for the two Ethernets and the 802.11 network, and 532 bytes for the PP network.
A 1420-byt datagram sent from H5 makes it across the 802.11 network and the first Ethernet without fragmentation but must be fragmented to three datagrams at router R2. These three fragments are then forwarded by router R3 across the second Ethernet to the destination host.

- Each fragment is itself a self-contained IP datagram that is transmitted over a sequence of physical networks, independent of other fragments
- Each IP datagram is re-encapsulated for each physical network over which it travels.



a) The unfragmented packet has 1400 bytes of data and a 20-byte IP header. When the packet arrives at router R2, which has an MTU of 532 bytes, it has to be fragmented.
b) A 532-byte MTU leaves 512 bytes for data after the 20-byte IP header, so the first fragment contains 512 bytes of data. The router sets the M bit in the Flags field, meaning that there are more fragments to follow, and it sets the Offset to 0, since this fragment contains the first part of the original datagram.
c) The data carried on the second fragment starts with the 513th byte of the original data, so the Offset field in this header is set to 64 (512/8).
d) The third fragment contains the last 376 bytes of data, and the offset is now 5 * 512 / 8 = 128.

- The fragmentation process is done in such a way that it could be repeated if a fragment arrived at another network with an even smaller MTU. Fragmentation produces smaller, valid IP datagrams that can be readily reassemble into the original datagram upon receipt, independently of the order of their arrival.
  - ⇨ Reassembly is done at the receiving host and not at each router.

    Reassembly is far from a simple process

    1) If a single fragment is lost, the receiver will still attempt to reassemble the datagram, and it will eventually give up and have to garbage-collect the resources that were used to perform the failed reassembly
    2) IP delays reassembly until destination host reached.
  - ⇨ Try to avoid fragmentation at source host using path MTU discovery.

2.3 Global addresses

⇨ **Global uniqueness** is the first property that should be provided in an addressing scheme

Ethernet addresses are globally unique, but that alone does not suffice for an addressing scheme in a large internetwork. Ethernet addresses are also flat, which means that they have no structure and provide very few clues to routing protocols.

>< IP addresses are hierarchical, which means that they are made up of several parts that correspond to some sort of hierarchy in the internetwork. They consist of two parts

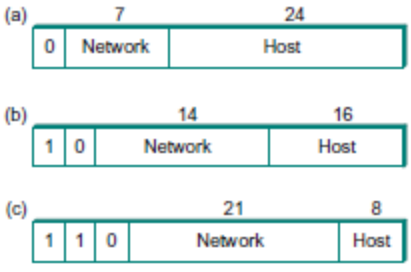
- **Network part:** identifies the network to which the host is attached. All hots attached to the same network have the same network part in their IP address.
  Routers attached to several networks need to have an address on each network, one for each interface;
- **Host part:** identifies each host uniquely on that particular network.

HIERARCHICAL ADDRESSES

⇨ Unlike some other forms of hierarchical address, the sizes of the two parts are not the same for all addresses. Originally, IP addresses were divided into three different classes, each of which defines different-sized network and host parts.
In all cases, the address is **32-bits long.**

The class of an IP address is identified in the most significant few bits.

- If the first bit is 0, it is a class A address. 126 networks of ~ 16.000.000 hots.
- If the first is 1 and the second 0, it is a class B address. ~16.000 networks of ~65.000 hosts.
- If the first two bits are 1 and the third is 0, it is a class C address. ~2.000.000 networks of 254 hosts.



This addressing scheme has a lot of **flexibility**, allowing networks of vastly different sizes to be accommodated fairly efficiently. The original idea was that the Internet would consist of a small number of wide area networks, a most number of site-sized networks, and a large number of LANs. However, it turned out not to be flexible enough.

⇨ IP addresses are written as **four decimal integers** separated by dots. Each integer represents the decimal value contained in 1 byte of the address, starting at the most significant.
Ex: 171.69.210.245
/!\ not to confuse IP addresses with Internet domain names, wich are also hierarchical. Domain names tends to be strings separated by dots. Ex: cs.princeton.edu.

2.4 Datagram forwarding in IP

⇨ **Forwarding** is the process of taking a packet from an input and sending it out on the appropriate output, while **routing** is the process of building up the tables that allow the correct output for a packet to be determined.

Strategy

1) Every datagram contains destination's address
2) If directly connected to destination network, then forward to host.
3) If not directly connected to destination network, then forward to some router
4) Forwarding table maps network number into next hop.
5) Each router maintains a forwarding table
6) Each host has a default router

Forwarding IP datagrams

A datagram is sent from a source host to a destination host, possibly passing through several routers along the way. Any node, whether it is a host or a router, first tries to establish whether it is connected to the same physical network as the destination. To do this, it compares the network part of the destination address with the network part of the address of each of its network interfaces.

- If a match occurs, then that means that the destination lies on the same physical network as the interface, and the packet can be directly delivered over that network.
- If the node is not connected to the same physical network as the destination node, then it needs to send the datagram to a router. In general, each node will have a choice of several routers, and so it needs to pick the best one, or at least one that has a reasonable chance of getting the datagram closer to its destination.
  Router that it chooses = **the next hop router**.

The router finds its next hop by consulting its forwarding table.
There is normally also a default router that is used if none of the entries in the table matches the destination's network numberf. For a host, it may be quite acceptable to have a default router and nothing else. This means that all datagrams destined for host not on the physical network to wich the sending host is attached will be sent out thourgh the default router.

Table 3.5 Example Forwarding Table for Router R2 in Figure 3.14

NetworkNum	NextHop
1	R1
4	R3

Table 3.6 Complete Forwarding Table for Router R2 in Figure 3.14

NetworkNum	NextHop
1	R1
2	Interface 1
3	Interface 0
4	R3

- Suppose that H1 wants to send a datagram to H2. Since they are on the same physical network, H1 and H2 have the same network number in their IP address. Thus, H1 deduces that it can deliver the datagram directly to H2 over the Ethernet. The one issue that needs to be resolved is how H1 finds out the correct Ethernet address for H2.
- Suppose H5 wants to send a datagram to H8. These hosts are on different physical networks, so they have different network numbers, H5 deduces that it needs to send the datagram to a router. R1 is the only choice, the default router, so H1 sends the datagram over the wireless network to R1.

R1 know that it cannot deliver a datagram directly to H8 because neither of R1's interfaces is on the same network as H8. Suppose R1's default router is R2; R1 then sends the datagram to R2 over the Ethernet. Assuming R2 has the forwarding table shown above, it looks up H8's network number and forwards the datagram over the PP network to R3. Finally, R3, since it is on the same network as H8, forwards the datagram directly to H8.

- Central Routers (R2)
- Edge Routers (R1, R3). They don't have to know all networks. They use R2 as default router
- Host (with one interface). The forwarding table contains one interface and default router. It directly delivers packet to destination, or delivers it to the default router.

2.5 Subnetting and Classless Addressing

The original intent of IP addresses was that the network part would uniquely identify exactly one physical network. It runs out that this approach has a couple of drawbacks.
Ex: large campus that has lots of internal networks and decides to connect to the Internet. For every network, no matter how small the site needs at least a class C network address. Even worse, for any network with more than 255 hosts, they need a class B address.

⇨ There are only a finite number of network numbers, and there are far fewer class B addresses than class Cs. Class B addresses tend to be in particularly high demand because you never know if your network might expand beyond 255 nodes, so it is easier to use a class B address from the start than to have to renumber every host when you run out of room on a class C network.

Assigning one network number per physical network, uses up the IP address space potentially much faster than we would like. While we would need to connect over 4 billion hosts to use up all the valid addresses, we only need to connect 2^{14} (+/- 16000) class B networks before that part of the address space runs out.

⇨ We would like to find some way to use the network numbers more efficiently.

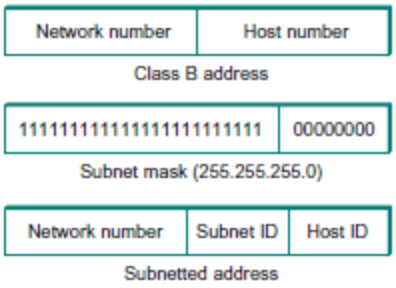
The amount of stat that is stored in a node participating in a routing protocol is proportional to the number of other nodes, and that routing in an internet consists of building up forwarding tables that tell a router how to reach different networks. The more network numbers there are in use, the bigger the forwarding tables get.

➔ Big forwarding tables add **costs** to routers, and thy are potentially **slower** to search tan smaller tables for a given technology, so they degrade router performance.

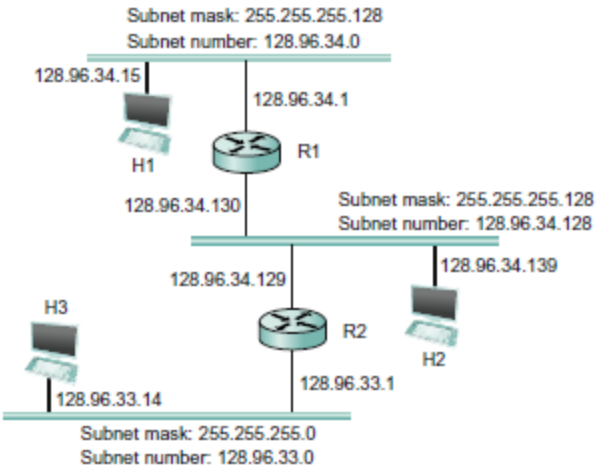
Subnetting provides a first stop to reducing total number of network numbers that are assigned. The idea is to take a single IP network number and allocate the IP addresses with that network number to several physical networks, which are now referred to as **subnets**.

- The **subnets** should be close to each other. This is because at a distant point in the Internet, they will all look like a single network, having only one network number between them. This means that a router will only be able to select one route to reach any of the subnets, so they had better all be in the same general direction.
  ⇨ Perfect situation in which to use Subnetting is a large campus or corporation that has many physical networks.

  Subnet mask defines variable portions of host part. The subnet mask enables us to introduce a subnet number. All hosts on the same physical network will have the same subnet number. Which means that hosts may be on different physical networks but share a single network number.



Subnets are not visible from the resto of the Internet. As result, routers in the outside word see a single network.



CLASSLESS ADDRESSING

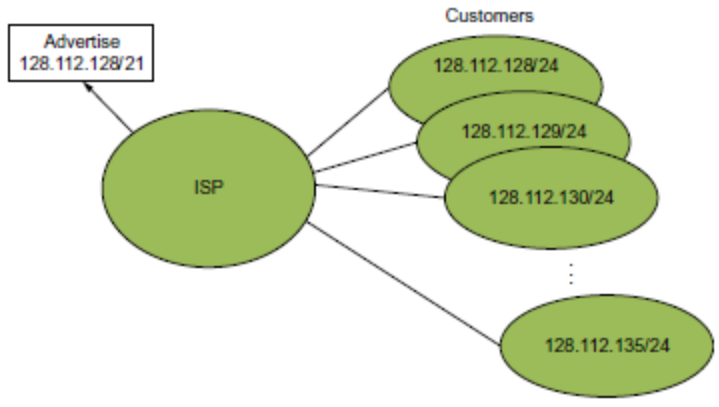
Classless Interdomain Routing (CIDR) takes the Subnetting idea to its logical conclusion by essentially doing away with address classes altogether. Subnetting only allows us to split a classful address among multiple subnets, while CIDR allows us to coalesce several classful addresses into a single supernet.

- ⇨ Keeps the routing system from being overloaded.
- ⇨ Issues of address space efficiency and scalability routing system are coupled.
  Ex: case of a company whose network has 256 hosts on it. That is too many for a class C address, you we would be tempted to assign a class B. However, this would be inefficient! Even though Subnetting can help us to assign addresses carefully, it does not get around the face that any organization with more than 255 hosts, or an expectation of eventually having that many, wants a class B address.

The first way you might deal with this issue would be to refuse to give a class B address to any organization that requests one unless they can show a need for something close to 64K addresses, and instead giving them an appropriate number of class C addresses to cover the expected number of hosts. Since we would now be handing out address space in chunks of 256 addresses at a time, we could more accurately match the amount of address space consumed to the size of the organization. For any organization with at least 256 hosts, we can guarantee an address utilization of at least 50%, and typically much more.
This solution, however, raises a problem that is at least as serious: **excessive storage requirements at the routers**. If a single site has, say, 16 class C network numbers assigned to it, that means every Internet backbone router needs 16 entries in its routing tables to direct packets to that site. This is true even if the path to every one of those networks is the same. If we had assigned a class B address to the site, the same routing information could be stored in one table entry. However, our address assignment efficiency would then be only 16×255/65,536 = 6.2%.

CIDR tries to balance the desire **to minimize the number of routes** that a router needs to know against the **need to hand out addresses efficiently**. To do this, CIDR helps us to **aggregate routes**. It lets us use a single entry in a forwarding table to tell us how to reach a lot of different networks. It does this by breaking the rigid boundaries between address classes.

CIDR requires a new type of notation to represent network numbers, or prefixes as they are known, because the prefixes can be of any length. The convention is to place a /X after the prefix, where X is the prefix length in bits. So for example, the 20-bit prefix for all the networks 192.4.16 through 192.4.31 is represented as a 192.4.16/20.



IP FORWARDING REVISITED

So far, we have assumed that we could find the network number in a packet and then look up that number in a forwarding table. Now that we have introduced CIDR, we need to reexamine this

assumption. CIDR means that prefixes may be of any length, from 2 to 32 bits. → Not restricted to aggregation of class C networks. Besides, prefixes in forwarding tables can overlap, in the sense that some addresses may match more than one prefix.

- ⇨ **Longest match algorithm needed**. Ex: we might find both 171.69 and 171.69.10 in the forwarding table of a single router. In this case, a packet destined to, 171.69.10.5 clearly matches both prefixes. The packet matches the longest prefix, which would be 171.69.10.

ADDRESS TRANSLATION (ARP)

- ⇨ **Recall**: we previously talked about how to get IP datagrams to the right physical network but glossed over the issue of how to get a datagram to a particular host or router on that network.
  Main issue: IP datagrams contain IP addresses, but the physical interface hardware on the host or router to which you want to send the datagram only understands the addressing scheme of that particular network. → We need to translate the IP address to a link-level address that makes sense on this network.

One simple way to map an IP address into a physical network address is to encode a host's physical address in the host part of its IP address. For example, a host with physical address 00100001 01001001 (which has the decimal value 33 in the upper byte and 81 in the lower byte) might be given the IP address 128.96.33.81. While this solution has been used on some networks, it is limited in that the network's physical addresses can be no more than 16 bits long in this example; they can be only 8 bits long on a class C network. This clearly will not work for 48-bit Ethernet addresses.

A more general solution would be for each host to maintain a table of address pairs, that is, the table would map IP addresses into physical addresses.

- ⇨ Can be accomplished using the **Address Resolution Protocol (ARP).**
  Goal: enable each host on a network to build up a table of mappings between IP addresses and link-level addresses. Since these mappings may change over time, the entries are timed out periodically and removed.

ARP takes advantage of the fact that many link-level network technologies, such as Ethernet, 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.

2.6 Host configuration (DHCP)

IP addresses not only must be unique on a given internetwork but also must reflect the structure of the internetwork. They contain a network part and a host part. The network part must be the same for all hosts on the same network. So, it is not possible for the IP address to be configured once into a host when it is manufactured, since that would imply that the manufacturer knew which hosts were going to end up on which networks and it would mean that a host, once connected to one network, could never move to another. → IP addresses need to be **reconfigurable**.

In addition to an IP address, there are some other pieces of information a host needs to have before it can start sending packets. The most notable is the **address of a default router.**
Most host operating systems prove a way for a system administrator, or even a user, to manually configure the IP information needed by a host.
Drawbacks:

- It is a lot of work to configure all the hosts in a large network directly
- The configuration process is very error prone, since it is necessary to ensure that every host gets the correct network number and that not two hosts receive the same IP address.
  - ⇨ Automated configuration methods are required. 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.

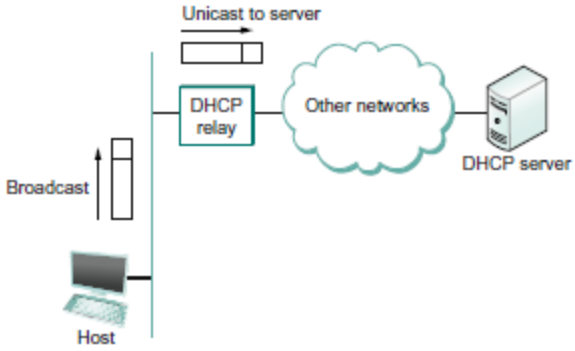
DHCP saves the network administrators from having to walk around to every hosts in the company with a list of addresses and network map in hand and configuring each host manually. The configuration information for each host could be stored in the **DHCP server** and automatically retrieved by each host when it is booted or connected to the network. However, the administrator would still pick the address that each host is to receive. He would just store that in the server. In this model, the configuration information for each host is stored in a table that is indexed by some form of unique client identifier, typically the hardware address.

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 and 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 to each network.

⇨ Goal of DHCP is to minimize the amount of manual configuration required for a host to function.

To contact a DHCP server, a newly booted or attached host sends a **DHCPDISCOVER message** to a special IP address that is an IP broadcast address. It will be received by all hosts and routers on the network. Routers do not forward such packets onto other networks, preventing broadcast to the entire Internet.

The server would then reply to the host that generated the discovery message. However, it is not really desirable to require one DHCP server on every network, because this still creates a potentially large number of servers that need to be correctly and consistently configured. Therefore, DHCP uses the concept of a relay agent. There is a 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 with it will then send back to the requesting client.



2.7 Error Reporting (ICMP)

⇨ How does the Internet treats errors?

⇨ IP does not necessarily fail silently.

- 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. Ex: error messages indicating that the destination host is unreachable (perhaps due to a link failure), that the reassembly process failed, 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 also provides the basis for two widely used debugging tools
  - Ping: uses ICMP echo messages to determine if a node is reachable and alive

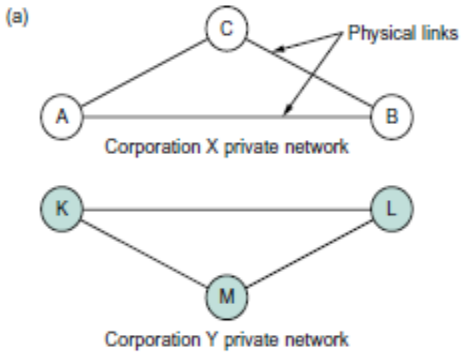
- Traceroute: uses a slightly non-intuitive technique to determine the set of routers along the path to a destination.

2.8 Virtual Networks and Tunnels

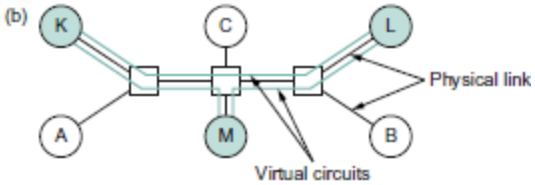
⇨ We previously focused on making it possible for nodes on different networks to communicate with each other in an unrestricted way.
Goal in Internet: everybody wants to be able to send email to everybody

⇨ There are many situations where more controlled connectivity is required. → **Virtual Private network (VPN)**
Ex: Corporations with many sites often build private networks by leasing transmission lines from the phone companies and using those lines to interconnect sites.

To make a private network virtual, the leased transmission lines, which are not shared with any other corporations, would be replaced by some sort of shared network. A **virtual circuit** (VC) is a very reasonable replacement for a leased line because it still provides a logical point-to-point connection between the corporation's sits. Ex: if corporation X has a VC from site A to site B, then clearly it can send packets between sites A and B. But there is no way that corporation Y can get its packets delivered to site B without first establishing its own virtual circuit to site B, and the establishment of such a VC can be administratively prevented, thus preventing unwanted connectivity between corporation X and corporation Y.



a) Shows to private networks for two separate corporations.
b) They are both migrated to a virtual circuit network. The limited connectivity of a real private network is maintained, but since the private networks now share the same transmission facilities and switches we say that two virtual private networks have been created.



⇨ It is also possible to provide a similar function using an IP network, an internetwork, to provide the connectivity. However, we cannot just connect the various corporations' sites to a single internetwork because that would provide connectivity between corporation X and corporation Y, which we wish to avoid.
Solution: **IP tunnel**

IP TUNNEL

= virtual point-to-point link between a pair of nodes that are actually separated by an arbitrary number of networks.

The virtual link is created within the router at the entrance to the tunnel by providing it with the IP address of the router at the far end of the tunnel. Whenever the router at the entrance of the tunnel wants to send a packet over this virtual link, it encapsulates the packet inside an IP datagram. The destination address in the IP header is the address of the router at the far end of the tunnel, while the source address is that of the encapsulating router.

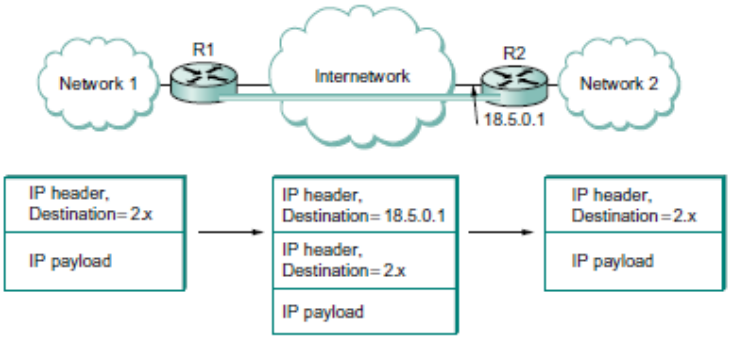


Table 3.8 Forwarding Table for Router R1 in Figure 3.27

NetworkNum	NextHop
1	Interface 0
2	Virtual interface 0
Default	Interface 1

Ex: a tunnel has been configured from R1 to R2 and assigned a virtual interface number of 0. R1 has two physical interfaces. Interface 0 connects to network 1. Interface 1 connects to a large internetwork and is thus the default for all traffic that does not match something more specific in the forwarding table.

In addition, R1 has a virtual interface, which is the interface to the tunnel. Suppose R1 receives a packet from network 1 that contains an address in network 2. The forwarding table says this packet should be sent out virtual interface 0. In order to send a packet out this interface, the router takes the packet, adds an IP header addressed to R2, and then proceeds to forward the packet as if it had just been received. R2's address is 18.5.0.1; since the network number of this address is 18, not 1 or 2, a packet destined for R2 will be forwarded out the default interface into the internetwork.

Once the packet leaves R1, it looks to the rest of the world like a normal IP packet destined to R2, and it is forwarded accordingly. All the routers in the internetwork forward it using normal means, until it arrives at R2. When R2 receives the packet, it finds that it carries its own address, so it removes the IP header and looks at the payload of the packet.

While R2 is acting as the endpoint of the tunnel, there is nothing to prevent it from performing the normal functions of a router. For example, it might receive some packets that are not tunneled, but that are addressed to networks that it knows how to reach, and it would forward them in the normal way.

Downsides

- **Increases the length of packets.** This might represent a significant waste of bandwidth for short packets. Longer packets might be subject to fragmentation, which has its own set of drawbacks.
- **Performance implications** for the routers at either end of the tunnel, since they need to do more work than normal forwarding as they add and remove the tunnel header.
- **Management cost** for the administrative entity that is responsible for setting up the tunnels and making sure they are correctly handled by the routing protocols.

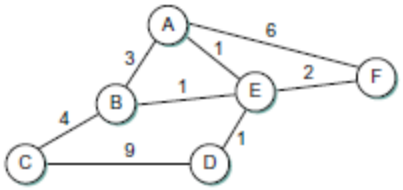
3. Routing

⇨ How do switches and routers acquire the information in their forwarding tables?

1) **The forwarding table** is used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function. This means that a row in the forwarding table contains the mapping from a network prefix to an outgoing interface and some MAC information, such as the Ethernet address of the next hop.
2) **The routing table**, on the other hand, is the table that is built up by the routing algorithms as a precursor to building the forwarding table. It generally contains mapping from network prefixes to next hops.

3.1 Network as a Graph

Graph representing a network. The nodes of the graph, may be hosts, switches, routers, or networks. Here, we will focus on the case where the nodes are routers. The edges of the graph correspond to the network links. Each edge has an associated cost, which gives some indication of the desirability of sending traffic over that link.



⇨ Find the **lowest-cost path** between any two nodes, where the cost of a path equals the sum of the costs of all the edges that make up the path.

Static approach has some severe shortcomings

1) Does not deal with changes in topology (new nodes, links)
2) Does not deal with failures (nodes, links)
3) Does not deal with changing costs (dynamic loads)

⇨ Hence, routing is achieved in most practical networks by running routing protocols among the nodes. These protocols provide a distributed, dynamic way to solve the problem of finding the lowest-cost path in the presence of link and node failures and changing edge costs.

3.2 Distance-Vector (RIP)

⇨ Each node constructs a one-dimensional array containing the "distances" to all other nodes and distributes that vector to its immediate neighbors.
Assumption: each node knows the cost of the link to each of its directly connected neighbors. These costs may be provided when the router is configured by a network manager. A link that is down is assigned an infinite cost.

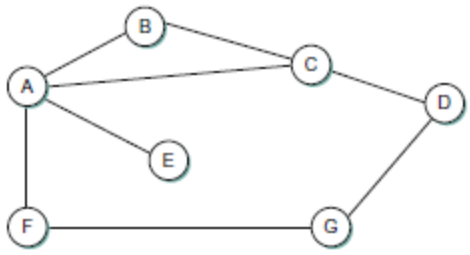


Table 3.10 Initial Distances Stored at Each Node (Global View)

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

⇨ Each node knows only the information in one row of the table. The global view that is presented here is not available at any single point in the network.

The next step in distance-vector routing is that every node sends a message to its directly connected neighbors containing its personal list of distance. Ex: node 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, etc. Based on the exchanged information, A can update its routing table with costs and next hops for all nodes in the network.

In a nutshell, each node maintains a routing table of triples (Destination, Cost, NextHop).

- Node updates local table if it receives a "better" route (smaller cost)
- Refresh existing routes
- Delete route if they time out
  - ⇨ Convergence: all nodes achieve a consistent view after a number of steps, without the help of a central authority.

Table 3.11 Initial Routing Table at Node A

Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

There are **two different circumstances** under which a given node decides **to send a routing update** to its neighbors.

- **Periodic update:** each node automatically sends an update message every so often, even if nothing has changed. This serves to let the other nodes know that this node is still running. It also makes sure that they keep getting information that they may need if their current routes become unviable. The frequency of these periodic updates varies from protocol to protocol, but it is typically on the order of several seconds to several minutes.
- **Triggered update:** happens whenever a node notices a link failure or receives an update from one of its neighbors that causes it to change one of the routes in its routing table. 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.

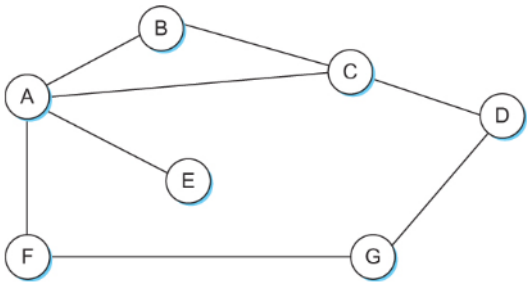
  - ⇨ **What happens when a link or node fails**? The nodes that notice first send new lists of distances to their neighbors, and normally the system settles down fairly quickly to a new state.
  - ⇨ **How does a node detect failure?**

- A node may continually test the link to another node by sending a control packet and seeing if it receives an acknowledgement.
- A node determines that the link is down if it does not receive the expected periodic routing update for the last few update cycles.

LINK FAILURE

⇨ What happens when a node detects a **link failure?** Consider what happens when F detects that its link to G has failed.

1) F sets its new distance to G to infinity and passes that information along to A.
2) Since A knows that its 2-hop path to G is through F, A would also set its distance to G to infinity.
3) With the next update from C, A would learn that C has a 2-hop path to G. thus A would know that it could reach G in 3 hops through C, which is less than infinity, and so A would update its table accordingly.
4) When it advertises this to F, node F would learn that it can reach G at a cost of 4 through A, which is less than infinity, and the system would again become stable.



Initial situation in F

Dest	Cost	NextHop
A	1	A
B	2	A
C	2	A
D	2	G
E	2	A
G	1	G

F after convergence

Dest	Cost	NextHop
A	1	A
B	2	A
C	2	A
D	3	A
E	2	A
G	4	A

COUNT TO INFINITY

⇨ Slightly different circumstances can prevent the network from stabilizing.
Ex: 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 2 to E. depending on the exact timing of events, the following might happen: Node B, upon hearing that E can be reached in 2 hops from C, concludes that it can reach E in 4 hops and advertises this to C. Node C concludes that it can reach E in 5 hops. And so on … This cycle stops only when the distances reach some number that is large enough to be considered infinite. In the meantime, none of the nodes actually knows that E is unreachable, and the routing tables for the network do not stabilize.

Initial situation in A

Dest	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

A after 3 rounds

Dest	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	6	B
F	1	F
G	2	F

⇨ **No convergence!**
⇨ **Loop-breaking heuristics:** technique to improve the time to stabilize routing. 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.
- split horizon: do not send updates back to NextHop
- set infinity to 16

3.3 Link State Routing (OSPF)

⇨ Second major class of intra domain routing protocol
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 and the cost of each link.
We want to provide each node with enough information to enable it to find the least-cost path to any destination.

Ex: Opens Shortest Path First (OSPF)

- Fast distribution of new information
- Fast convergence
- No count-to-infinity (every node knows complete topology
- Refresh period = hours (little traffic)
- Amount of link state information
- Complex protocol (authentication, load balancing, multiple metrics…)

Idea: 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. This is clearly a sufficient condition for finding the shortest path to any point in the network.

Two mechanisms:

- Reliable dissemination of link-state information
- Calculation of routes from the sum of all the accumulated link-state knowledge.

RELIABLE FLOODING

= 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.

Basic idea: for a node to send its link-state information out on all of its directly connected links. This process continues until the information has reached all the nodes in the network.

⇨ Each node creates an update packet, also called a link-state packet (LSP)
  - 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

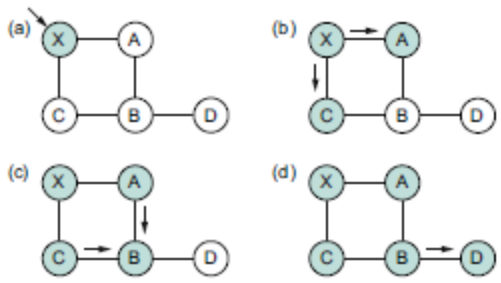
making sure that you have the most recent copy of the information, since there may be multiple contradictory LSPs from one node traversing the network. Making the flooding reliable has proven to be quite difficult!

⇨ How does flooding work?

1) The **transmission of LSPs** between adjacent routers is made reliable using acknowledgements and retransmissions just as in the reliable link layer protocol.

Ex: consider a node X that receives a copy of an LSP that originated at some other node Y. Note that Y may be any other router in the same routing domain as X. X checks to see if it has already stored a copy of an LSP from Y. If not, it stores the LSP. If it already has a copy, it compares the sequence numbers. If the new LSP has a larger sequence number, it is assumed to be the more recent, and that LSP is stored, replacing the old one. A smaller (or equal) sequence number would imply an LSP older (or not newer) than the one stored, so it would be discarded and no further action would be needed.

If the received LSP was the newer one, X then sends a copy of that LSP to all of its neighbors except the neighbor from which the LSP was just received. The fact that the LSP is not sent back to the node from which it was received helps to bring an end to the flooding of an LSP. Since X passes the LSP on all its neighbors, who then turn around and do the same thing, the most recent copy of the LSP eventually reaches all nodes.



⇨ Each node becomes shaded as it stores the new LSP

a) The LSP arrives at node X, which sends it to neighbors A and C.
b) A and C do not send it back to X, but send It on to B
c) . Since B receives two identical copies of the LSP, it will accept whichever arrived first and ignore the second as a duplicate.
d) It then passes the LSP onto D, which has no neighbors to flood it to. The process is complete.

⇨ Just as in RIP, each node generates LSPs under two circumstances.

1) Expiry of a periodic timer

2) A change in topology can cause a node to generate a new LSP. However, the only topology-based reason for a node to generate an LSP is if one of its directly connected links or immediate neighbors has gone down.

Important goal: the newest information must be flooded to all nodes as quickly as possible, while old information must be removed from the network and not allowed to circulate. Moreover, it is clearly desirable to minimize the total amount of routing traffic that is sent around the network. This is just overhead from the perspective of those who actually use the network for their applications.

⇨ Easy way to reduce overhead: 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.

LSPs also carry a time to live. This is used to ensure that old link state information is eventually removed from the network. A node always decrements the TTL of a newly received LSP before flooding it to its neighbors.

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

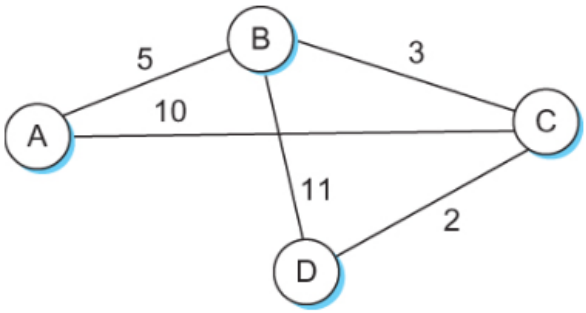
Once a given node has a copy of the LSP from every other node, it is able to compute a complete map for the topology of the network, and from this map it is able to decide the best route to each destination.

⇨ How does it calculate routes from this information?
Dijkstra's shortest-path algorithm.

```
M = {s}
for each n in N-{s}
    C(n) = l(s,n)
while (N != M)
    M = M union {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))
```

Let

- N denotes set of nodes in the graph
- L(I, j) denotes non-negative cost (weight) for edge (I,j)
- S denotes the starting node (this node where algorithm is executed)
- M denotes the set of nodes incorporated so far by the algorithm
- C(n) denotes cost of the path from s to each node n



Link State info at node D

From\To	A	B	C	D
A	0	5	10	∞
B	5	0	3	11
C	10	3	0	2
D	∞	11	2	0

```
starting node s=D    → M={D}, N-M={A,B,C}
[ ∞   11   2   0] → select w=C; M={C,D}
interesting to deviate traffic via C?
[2+10 2+3  2   0] ← N-M={A,B}
[ 12   5   2   0] → select w=B; M={B,C,D}
interesting to deviate traffic via B?
[5+5   5   2   0] ← N-M={A}
[ 10   5   2   0] → select w=A; N-M={Ø}
```

Resulting table of D

Dest	Cost	NextHop
A	10	C
B	5	C
C	2	C

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 and select its LSP.
3) For each neighbor of next, calculate the cost to reach this neighbor as the sum of the cost from myself to next and from next to neighbor.
    a. If neighbor is currently on neither the confirmed nor the tentative list, then add it to the tentative list, where NextHop is the direction I go to reach next.
    b. If neighbor is currently on the tentative list, and the cost is less than the currently listed cost for neighbor, then replace the current entry. NextHop is the direction I go to reach next
4) If the tentative list is empty, stop. Otherwise, pick the entry from the tentative list with the lowest cost, move it to the confirmed list, and return to step 2.

⇨ The link-state routing algorithm 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 can be quite large. This is one of the fundamental problems of routing and is an instance of the more general problem of **scalability**.

3.4 The metrics

⇨ We previously assumed that link costs, or metrics, were known when we execute the routing algorithm. In this section, we look

at some ways to calculate link costs that have proven effective in practice.
Ex: reasonable and very simple: assign a cost of 1 to all links. The least-cost route will then be the one with the fewest hops.
Drawbacks:

- A satellite link with 205-ms latency looks just as attractive to the routing protocol as a terrestrial link with 1-ms latency.
- It does not distinguish between routes on a capacity basis, making a 9.6-kbps link look just as good as a 45-Mbps.

ARPANET

= Testing ground for a number of different approaches to link-cost calculation.

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 5 packets 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.
⇨ The original ARPANET mechanism suffered from the fact that it did not take either the bandwidth or the latency of the link into consideration.

➔ New routing mechanism
This second version of the ARPANET routing algorithm takes both link bandwidth and latency into consideration and used delay, rather than just queue length, as a measure of load.

- Each incoming packet was timestamped with its time of arrival (ArrivalTime) at the router, its departure time from the router was also recorded (DepartTime)
- When the link-level ACK was received from the other side, the node computed the delay for that packs as

$$Delay = (DepartTime - ArrivalTime) + TransmissionTime + Latency$$

If the ACK did not arrive, but instead the packet timed out, the DepartTime was rest to the time the packet was retransmitted.

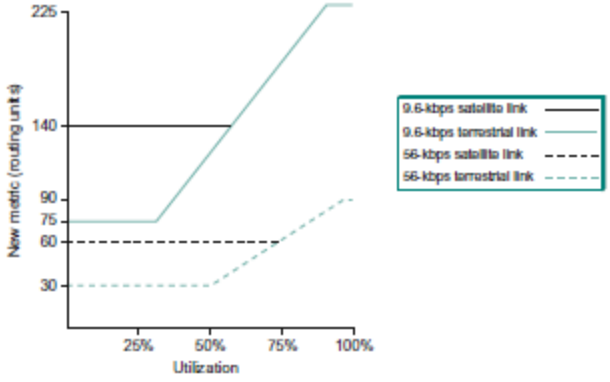
Improvement over the original mechanism but it also had **a lot of problems**.

- **Unstable under heavy load:** Under heavy load, 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.
  Effect: many links would spend a great deal of time being idle.
- **Enormous range of possible values:** Ex: a heavily loaded 9.6 kbps link could look 127 times costlier than a lightly loaded 56-kbps link.

➔ **Third approach:** the revised ARPANET routing metric
The major changes were to compress the dynamic range of the metric considerably, to account for the link type, and to smooth the variation of the metric with time.

The smoothing was achieved by several mechanisms

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

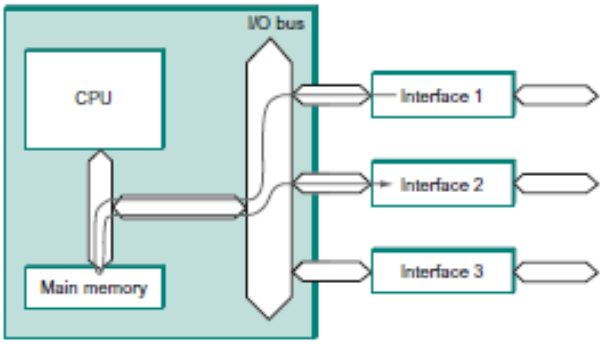
- A highly loaded link never shows a cost of more than three times its cost when idle
- The most expensive link is only seven times the cost of the least expensive.
- A high-speed satellite link is more attractive than a low-speed terrestrial link.
- Cost is a function of link utilization only at moderate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

Dynamically changing metrics are unstable. Static metrics seem often used in reality. One common approach to setting metrics is to use a constant multiplied by (1/link_bandwidth).

4. Implementation and performance

4.1 Switch basics

⇨ Switches and routers use similar implementation techniques.
RMQ: the word switch will be used to cover both types of devices, router and switches, since their internal designs are so similar.



This figure shows a processor with three network interfaces used as a switch. The figure shows a path that a packet might take from the time it arrives on interface 1 until it is output on interface 2. We have assumed here that the processor has a mechanism to move data directly copied by the CP, this technique is called **direct memory access (DMA).** Once the packet is in memory, the CPU examines its header to determine which interface the packet should be sent out on. It then uses DMA to move the packet out to the appropriate interface. This figure does not show the packet going to the CPU because the CPU inspects only the header of the packet. It does not have to read every byte of data in the packet.

The main problem with using a general purpose processor as a switch is that its performance is limited by the fact that all packets must pass through a single point of contention. Ex: each packet crosses the I/O bus twice and is written to and read from main memory once.

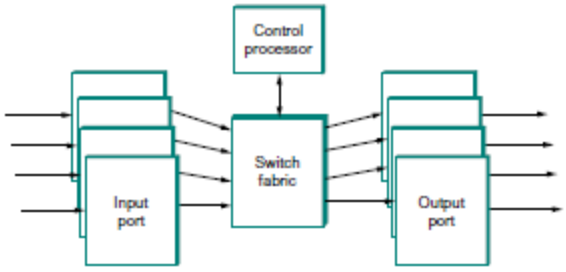
The upper bound on aggregate throughput of such a device is, thus, either half the main memory bandwidth or half the I/O bus bandwidth, whichever is less.

Moreover, this upper bound also assumes that moving data is the only problem, a fair approximation for long packets but a bad one when packets are short. In the latter case, the cost of processing each packet, parsing its header and deciding which output link to transmit it on, is likely to dominate.

Exam: How could we improve the given structure? A good idea would be to put one bus for each interface.

⇨ Limited packets-per-second processing.
The CPU must be able to process all packets.

4.2 Ports



Most switches look conceptually similar to the one shown above. They consist of a number of **input and output ports** and a **fabric**. Usually, there is at least one **control processor** in charge of the whole switch that communicates with the ports either directly or, as shown here, via the switch fabric. The ports communicate with the outside world. They may contain fiber optic receivers and lasers, buffers to hold packets that are waiting to be switched or transmitted, and often a significant amount of other circuitry that enables the switch to

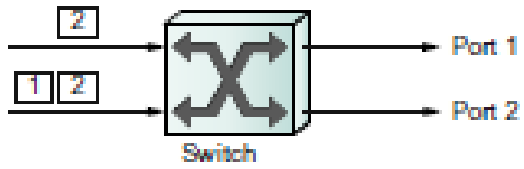
function. The **fabric** as a simple job: when presented with a packet, deliver it to the right output port.

One of the jobs of the **ports** is to **deal with the complexity of the real world** in such a way that the fabric can do its relatively simple job.

Ex: assume that this switch is supporting a virtual circuit model of communication. In general, the virtual circuit mapping tables are located in the ports. The ports maintain lists of virtual circuit identifiers that are currently in use, with information about what output a packet should be sent out on for each VCI and how the VCI needs to be remapped to ensure uniqueness on the outgoing link.

⇨ Fabrics that switch packets by looking only at the information in the packet are referred to as **self-routing**, since they require no external control to route packets.

⇨ Another key function of ports is **buffering**.



Buffering can happen in either the input or the output port. It can also happen within the fabric (sometimes called internal buffering). Simple input buffering has some serious **limitations**.

Ex: Consider an input buffer implemented as a FIFO. As packets arrive at the switch, they are placed in the input buffer. The switch then tries to forward the packets at the front of each FIFO to their appropriate output port. If the packets at the front of several different input ports are destined for the same output port at the same time, then only one of them can be forwarded. The rest must stay in their input buffers.

⇨ **Drawback:** Those packets left at the front of the input buffer prevent other packets further back in the buffer from getting a chance to go to their chosen outputs, even though there may be no contention for those outputs.
= head-of-line blocking.

Previous figure illustrates this phenomenon. We see a packet destined for port 1 blocked behind a packet contending for port 2. It can be shown that when traffic is uniformly distributed among outputs, head-of-line blocking limits the throughput of an input-buffered switch to 59% of the theoretical maximum. Thus, the majority of switches use either pure output buffering or a mixture of internal and output buffering.

- ⇨ Those that rely on input buffers use more advanced buffer management schemes to avoid head-of-line blocking.
- ⇨ Buffers are the main source of delay in a switch, and also the place where packets are most likely to get dropped due to lack of space to store them.
- ⇨ The buffers are the main place where the quality of service characteristics of a switch are determined.

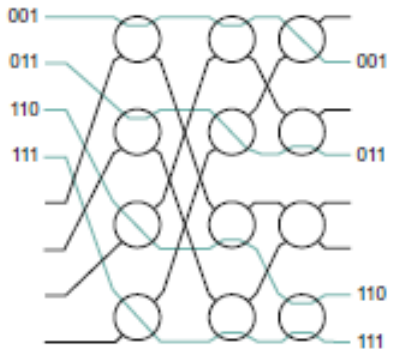
4.3 Fabrics

- ⇨ A switch fabric should be able to move packets from input ports to output ports with **minimal delay** and in a way that **meets the throughput goals** of the switch.
- ⇨ The fabrics display some degree of parallelism. A high – performance fabric with n ports can often move one packet from each of its n ports to one of the output ports at the same time.

1) **Shared bus:** type of fabric found in a conventional processor used as a switch. Because the bus bandwidth determines the throughput of the switch, high-performance switches usually have specially designed busses rather than the standard busses found in PCs.
2) **Shared Memory:** packets are written into a memory location by an input port and then read from memory by the output ports. A shared memory switch is similar in principle to the shared bus switch, except it usually uses a specially designed, high-speed memory bus rather than an I/O bus.
3) **Crossbar**: this switch is a matrix of pathways that can be configured to connect any input fort to any output port.

   Problem: they require each output port to be able to accept packets from all inputs at once, implying that each port would have a memory bandwidth equal to the total switch throughput.
4) **Self-routing**: this kind of fabrics relies on some information in the packet header to direct each packet to its correct output. Usually a special self-routing

header is appended to the packet by the input port after it has determined which output the packet needs to go to.
These are among the most scalable approaches to fabric design.
Ex: Banyan network.



Many self-routing fabrics resemble the one shown on this figure. It consists of regularly interconnected 2x2 switching elements. These switches perform a simple task. They look at 1 bit in each self-routing header and route packets toward the upper output if it is zero or toward the lover output if it is one. If two packets arrive at the banyan element at the same time and both have the bit set to the same value, then they want to be routed to the same output and a collision will occur.

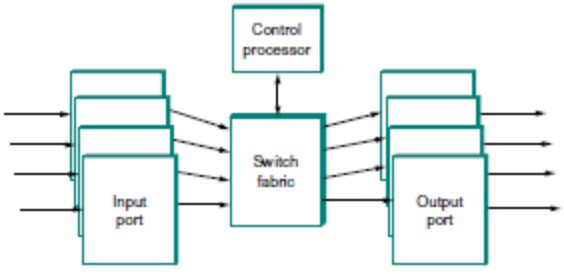
⇨ Either preventing or dealing with these collisions is a main challenge for self-routing switch design.

On this figure we can see the self-routing header that contains the output port number encoded in binary. The switch elements in the first column look at the most significant bit of the output port number and route packets to the top if that bit is 0 or the bottom if it is a 1.

Switch elements in the second column look at the second bit in the header, and those in the last column look at the least significant bit.

⇨ One of the interesting things about switch design is the wide range of different types of switches that can be built using the same basic technology.
Ex: Ethernet switches, ATM switches, and Internet routers, they all have been built using designs such as those outlined in this section.

4.4 Router implementation



- ⇨ **The control processor** is responsible for running the routing protocols discussed previously, among other things, and generally acts as the central point of control of the router.
- ⇨ The **switching fabric** transfers packets from one port to another, just as in a switch, and the ports provide a range of functionality to allow the router to interface to links of various types.

➔ A few points are worth noting about router design and how it differs from switch design.

- Routers must be designed to **handle variable-length packets.**
  Many high-performance routers are designed using a switching fabric that is cell based. In such cases, the ports must be able to convert variable length packets into cells and back again.
  = segmentation and re-assembly (SAR)
- As a consequence of the variable length of IP datagrams, it can be harder to characterize the performance of a router than a switch that forwards only cells.
  Routers can usually forward a certain number of packets per second, and this implies that the total throughput in bits per second depends on packet size.
- When it comes to the task of forwarding IP packets, routers can be broadly characterized as having either a **centralized** or **distributed** forwarding model.
  - **Centralized**: the IP forwarding algorithm is done in a **single processing engine** that handles the traffic from all ports.
  - **Distributed**: there are **several processing engines**, perhaps one per port, or more often one per line car, where a line card may serve one or more physical ports.
    All things equal, a distributed forwarding model should be able to forward more packets per second through the router as a whole, because there is more processing power in total.
    But a distributed model also complicates the software architecture, because each forwarding engine typically needs its own copy of the forwarding table, and thus it is necessary for the control processor to ensure that the forwarding tables are updated consistently and in a timely manner.
- Another different aspect is the IP forwarding algorithm itself.
- **Network processor.** It is intended to be a device that is just about as programmable as a standard PC processor, but that is **more highly optimized for networking tasks**.
  Ex: a network processor might have instructions that are particularly well suited to performing lookups on IP addresses, or calculating checksums on IP datagrams. Such devices could be used I routers and other networking devices.

CHAPTER 4: ADVANCED INTERNETWORKING

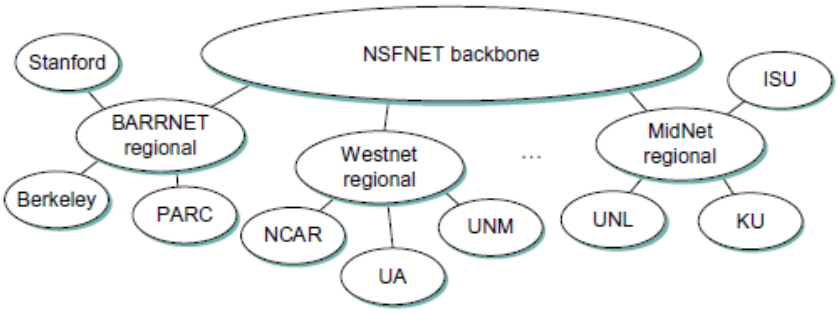
1. Introduction

- ⇨ In previous chapter we dealt with the problem of **heterogeneity**.
- ⇨ The second critical problem in internetworking is the problem of **scale**. To fully understand this problem, it is worth considering the growth of the Internet, which has roughly doubled in size each year for 30 years.
- ⇨ How to build a routing system that can handle hundreds of thousands of networks and billions of end nodes?

2. The global Internet

Today's Internet has hundreds of thousands of networks connected to it. Routing protocols such as those we have just discussed do not scale to those kinds of numbers.

- ⇨ What does the global Internet look 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.



1) One of the salient features of this topology is that it consists of **end-users sites** (Ex: Stanford University) that connect to **service provider** (Ex: BARRNET).
   In 1990, many providers served a limited geographic region and were thus known as regional networks.
2) **Each provider and end-user** is likely to be an **administratively independent** entity. This has some significant consequences on routing.

- Different providers will have different ideas about the best routing protocol to use within their networks and on how metrics should be assigned to links in their network.

Because of this independence, each provider's network is usually a **single autonomous system (AS).**

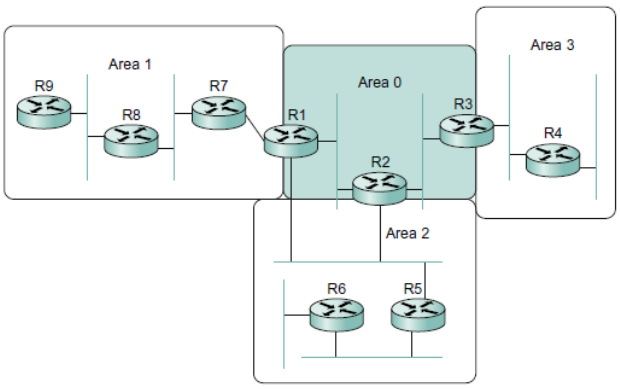
⇨ We need to deal with **two related scaling issues**

- The **scalability of routing**. We need to find ways to minimize the number of network numbers that get carried around in routing protocols and stored in the routing tables of routers.
- **Address utilization:** making sure that the IP address space does not get consumed too quickly.

2.1 Routing Areas

⇨ How can link-state routing protocols be used to partition a routing domain into subdomains called areas?

An area is a set of routers that are administratively **configured to exchange link-state information** with each other. There is one special area, the **backbone area**, also known as **area 0.**



Routers R1, R2, R3 are members of the backbone area. They are also members of at least one nonbackbone area. R1 is actually a member of both area 1 and area 2. A router that is a member of both the backbone are and a nonbackbone area is an **area border router (ABR).** These are distinct from the routers that are at the edge of an AS, which are referred to as **AS border router**.

⇨ **Routing within a single area**: All the routers in the area send link-state advertisements to each other and thus develop a complete consistent map of the area. However, the link-state advertisements of routers that are not area border routers do not leave the area in which they originated. This has the effect of making the flooding and route calculation process considerably more scalable.

Ex: router R4 in area 3 will never see a link-state advertisement from router R8 in area 1. As a consequence, it will know nothing about the detailed topology of areas other than its own.

⇨ How, then does a router in one area determine the right next hop for a packet destined to a network in another area?
Imagine the path of a packet that has to travel form one nonbackbone area to another as being split into three parts.

1) First it travels from its source network to the backbone area.
2) Then it crosses the backbone.
3) Then it travels from the backbone to the destination network.

To make this work, the area border routers summarize routing information that they have learned from one area and make it available in their advertisements to other areas.
Ex: R1 receives link-state advertisemnets from all the routers in area 1 and can thus determine the cost of reaching any network in area 1.
When R1 sends link-state advertisements into area 0, it advertises the costs of reaching the networks in area 1 much as if all those networks were directly connected to R1. This enables all the area 0 routers to learn the cost to rach all networks in area 1.
The area border routers then summarize this information and advertise it inot the nonbackbone areas. Thus,a ll routers lean how to reach all networks in the domain.

In area 2, there are two ABRs. Therefore, routers in area 2 will have to make a choice as to which one they use to reach the backbone. Since both R1 and R2 will be advertising costs to various networks, it will become clear which is the better choice as the routers in area 2 run their shortest-path algorithm.

Ex: pretty clear that R1 is going to be a better choice than R2 for destinations in area 1.

2.2 Interdomain Routing (BGP)

- ⇨ Internet is organized as autonomous systems, each of which is under the control of a **single administrative entity**.
- ⇨ The basic idea behind autonomous systems is to provide an additional way to **hierarchically aggregate routing information** in a large internet, thus improving scalability.

Routing problems

1) Within a single autonomous system
   = **Intradomain**
2) Between autonomous systems
   = **Interdomain**

In addition to improving scalability, the AS mode decouples the Intradomain routing that takes place in one AS from that takin place in another. So, each AS can run whatever Intradomain routing protocols it chooses.

The Interdomain routing problem is 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.

CHALLENGES IN INTERDOMAIN ROUTING

The most important challenge of Interdomain routing today is the need for each AS to determine its **own routing policies**.

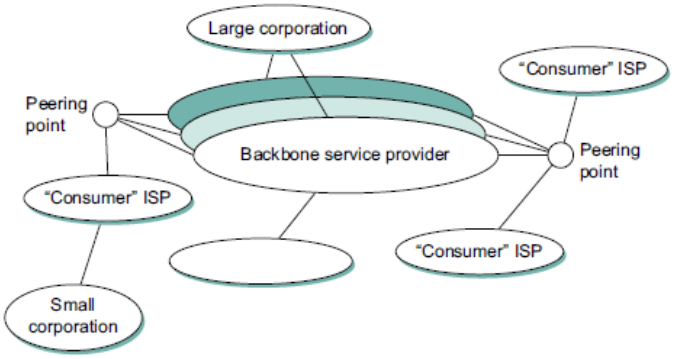
Ex: whenever possible, I prefer to send traffic via AS X than via AS Y, but I'll use AS Y if it is the only path, and I never want to carry traffic from AS X to AS Y or vice versa.
The more autonomous systems I connect to, the more complex policies I might have, especially when I consider backbone providers, who may interconnect with dozens of other providers and hundreds of customers and have different economic arrangements with each one.
I need to be able to implement such a policy without any help from other autonomous systems, and in the face of possible misconfiguration or malicious behavior by other autonomous systems.
There is also often a desire to keep the policies **private**. Since the entities that run the autonomous systems, mostly ISPs, are often in competition with each other, they don't want their economic arrangements to be made public.

Two major interdomain routing protocols in the history of Internet

- **Exterior Gateway Protocol (EGP)**

Limitations: it constrained the topology of the Internet rather significantly. EGP was designed when the Internet had a treelike topology and it did not allow for the topology to become more general. In the simple treelike structure there is a single backbone, and autonomous systems are connected only as parents and children and not as peers.

- **Border Gateway Protocol (BGP)**
  This one is often regarded as one of the more complex parts of the Internet.
  Unlike its predecessor EGP, BGP makes virtually no assumptions about how autonomous systems are interconnected. They form an arbitrary graph.
  ⇨ General enough to accommodate non-tree-structured internetworks.



Today's Internet consists of a richly interconnected set of networks, mostly operated by private companies (ISPs) rather than governments.

Many Internet Service Providers mainly exist to provide service to "consumers", while others offer something more like the old backbone service, interconnecting other providers and sometimes larger corporations.
Often many providers arrange to interconnect with each other at a single **peering point.**

Local traffic: Traffic that **originates** at or **terminates on nodes** within an AS.

Transit traffic: Traffic that passes through an AS.

We can classify autonomous systems into **three broad types**.

- **Stub AS:** an AS that has only a single connection to one other AS. This AS will only carry local traffic.
  Ex: small corporation (on the previous figure)
- **Multihomed AS:** AS that has connections to more than one other AS but that refuses to carry transit traffic.

Ex: large corporation

- **Transit AS:** AS that has connections to more than one other AS and that is designed to carry both transit and local traffic.
  Ex**:** the backbone providers

- Routing: finding optimal paths based on minimizing some sort of link metric.
- Interdomain routing:
  - 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 autonomous systems along the path, and those policies migh be almost arbitrarily complex.

Interdomain routing = hard!

- Matter of scale. An Internet backbone router must be able to forward any packet destined anywhere in the Internet. That means having a routing table that will provide a match for any valid IP address.

Another challenge in Interdomain routing arises from the autonomous nature of the domains. Interdomain 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 Interdomain routing to pick an optimal path is essentially impossible.

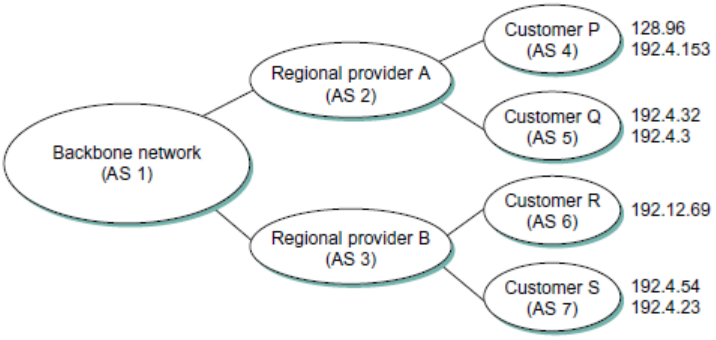
The autonomous nature of Interdomain raises issue of trust. → Provider A might be unwilling to believe certain advertisements from provider B for fear that provider B will advertise erroneous routing information.
Ex: trusting provider B when he advertises a great route to anywhere in the Internet can 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.

BASICS OF BGP

Each AS has one or more border routers through which packets enter and leave the AS. A border router is simply an IP router that is charged with the task of forwarding packets between autonomous systems.

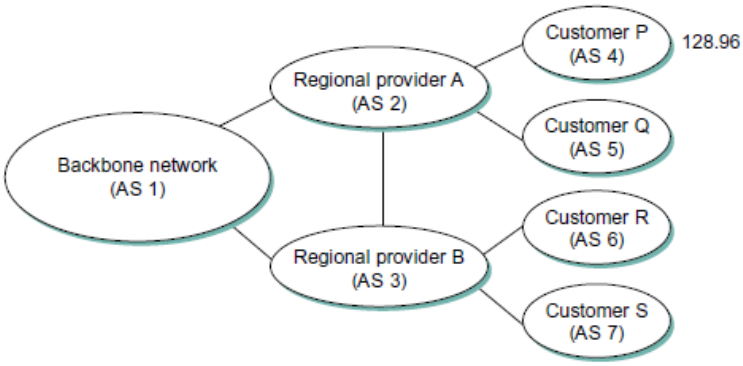
Each AS tat participates in BGP must also have at least one BGP speaker, a router that "speaks" BGP to other BGP speakers in other autonomous systems. It is common to find that border routers are also BGP speakers, but that does not have to be the case.

BGP advertises complete paths as an enumerated list of autonomous systems to reach a particular network. Hence, it is sometimes called a path-vector protocol. The advertisement of complete paths is necessary to enable the sorts of policy decisions described above to be made in accordance with the wishes of a particular AS.



Assume that the providers are transit networks, while the customer networks are stubs. A BGP speaker for the AS of provider A would be able to advertise reachability information for each of the network numbers assigned to customers P and Q.
The networks 128.96, 192.4.153, 192.4.32 and 192.4.3 can be reached directly from AS 2.
The backbone network, on receiving this advertisement, can advertise: "The networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path <AS 1, AS 2>. Similarly, it could advertise, the networks 192.12.69, 192.4.54, and 192.4.23 can be reached along the path <AS 1, AS 3>.

⇨ An important job of BGP is to prevent the establishment of looping paths.



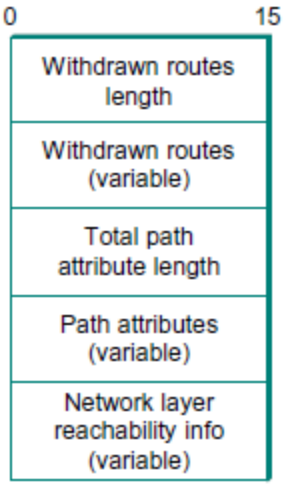
This figure differs from previous one, only in the addition of an extra link between AS2 and AS3. But the effect now is that the graph of autonomous systems has a loop in it.
Suppose AS 1 learns that it can reach network 128.96 through AS 2, so it advertises this fact to AS 3, who in turn advertises it back to AS 2. In the absence of any loop prevention mechanism, AS 2 could now decide that AS 3 was the preferred route for packets destined for 128.96. If AS 2 starts sending packets addressed to 128.96 to AS 3, AS 3 would send them to AS 1; AS 1 would send them back to AS 2; and they would loop forever. This is prevented by carrying the complete AS path in the routing messages. In this case, the advertisement for a path to 128.96 received by AS 2 from AS 3 would contain an AS path of <AS 3, AS 1, AS 2, AS 4>. AS 2 sees itself in this path, and thus concludes that this is not a useful path for it to use.

⇨ In order for this loop prevention technique to work, the AS numbers carried in BGP clearly need to be unique.

Ex: AS 2 can only recognize itself in the AS path in the above example if no other AS identifies itself in the same way.

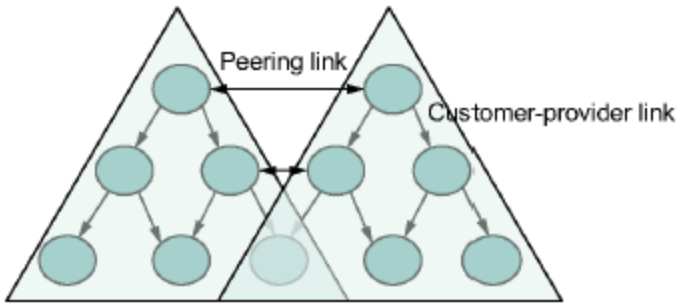
A given AS will only advertise routes that it considers good enough for itself. That is, if a BGP speaker has a choice of several different routes to a destination, it will choose the best one according to its own local policies, and then that will be the route it advertises. Moreover, a BGP speaker is under no obligation to advertise any route to a destination, even if it has one.

Given that links fail and policies change, BGP speakers need to be able to cancel previously advertised paths. This is done with a form of negative advertisement, known as a withdrawn route. Both positive and negative reachability information are carried in a BGP update message.



BGP is defined to run on top of TCP, the reliable transport protocol. Because BGP speakers can count on TCP to be reliable, this means that any information that has been sent from one speak to another does not need to be sent again. Thus, as long as nothing has changed, a BGP speaker can simply send an occasional keep alive message that says: "I'm still here and nothing has changed".

COMMON AS RELATIONSHIPS AND POLICIES



This figure represents the common relationships.

- **Provider-Customer:** Providers are in the business of connecting their customers to the rest of the Internet. A customer might be a corporation or it might be a smaller ISP.
  Policy: advertise all the routes I know about to my customer, and advertise routes I learn from my customer to everyone.
- **Customer-Providers:** the customer wants to get traffic directed to him by his provider, and he wants to be able to send traffic to the rest of the Internet through his provider.
  Policy: advertise my own prefixes and routes learned from my customers to my provider, advertise routes learned from my provider to my customers, but don't advertise routes learned from one provider to another provider.
- **Peer**: symmetrical peering between autonomous systems. Two providers who view themselves as equals usually peer so that they can get access to each other's customers without having to pay another provider.
  Policy: Advertise routes learned from my customers to my peer. Advertise routes learned from my peer to my customers, but don't advertise routes from my peer to any provider or vice versa.

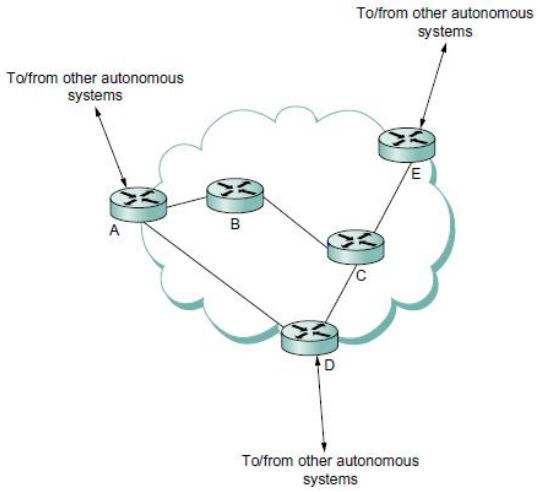
  ⇨ This figure brings back some structure to the apparently unstructured Internet.
    - **Bottom**: stub networks that are customers of one or more providers
    - As we move up the hierarchy, we see providers who have other providers as their customers.
    - **Top**: providers who have customers and peers but are not customers of anyone.
      = Tier-1 providers.

INTEGREGATING INTERDOMAIN AND INTRADOMAIN ROUTING

⇨ How do all the other routers in a domain get routing information?

- Stub AS that only connects to other autonomous systems at a single point, the border router is clearly the only choice for all routes that are outside the AS.
  Such a router can inject a default route into the Intradomain routing protocol. Indeed, this is a statement that any network that has not been explicitly advertised in the Intradomain protocol is reachable through the border router.
- Next step: have the border routers inject specific routes they have learned from outside the AS.

- Final level of complexity: comes in backbone networks, which learn so much routing information from BGP that it becomes too costly to inject it into the Intradomain protocol.
  Ex: if a border router wants to inject 10.000 prefixes that it learned about from another AS, it will have to send very big link-state packets to the other routers in that AS, and their shortest-path calculations are going to become very complex.



Three border routers: A, D, E. They speak eBGP (exterior BGP) to other autonomous systems and learn how to reach various prefixes.
They communicate with other with the interior routers B and C by building a mesh of iBGP sessions among all the routers in the AS.
Ex: Router B. how does it build up its complete view of how to forward packets to any prefix?
The upper left table below shows the information that router B learns from its iBGP sessions.
The top right table shows how to reach other nodes inside the domain.
Ex: to reach router E, B needs to send packets toward router C.
In the bottom table, B puts the whole picture together, combining the information about external prefixes learned from iBGP with the information about interior routes to the border routers learned from the IGP.
Ex: if a prefix like 18.0/16 is reachable via border router E, and the best interior path to E is via C, then it follows that any packet destined for 18.0/16 should be forwarded toward C.

⇨ Any router in the AS can build up a complete routing table for any prefix that is reachable via some border router of the AS.

Prefix	BGP Next Hop
18.0/16	E
12.5.5/24	A
128.34/16	D
128.69./16	A

BGP table for the AS

Router	IGP Path
A	A
C	C
D	C
E	C

IGP table for router B

Prefix	IGP Path
18.0/16	C
12.5.5/24	A
128.34/16	C
128.69./16	A

Combined table for router B

2.3 IP Version 6 (IPv6)

⇨ New version of IP to deal with exhaustion of the IP address space. CIDR helped considerably to contain the rate at which the Internet address space is being consumed and also helped to control the growth of routing table information ended in the Internet's routers. >< it is impossible to achieve 100% address utilization efficiency. Therefore, the address space will be exhausted well before the 4 billionth host is connected to the Internet. → Bigger address space than that provided by 32 bits will eventually be needed.

HISTORICAL PERSPECTIVE

Since the IP address is carried in the heard of every IP packet, increasing the size of the address dictates a change in the packet header. This means a new version of the Internet Protocol and, as a consequence, a need for a new software for every host and router in the Internet.

⇨ Effort to define a new version of IP is known as **IP Next Generation**, or **IPng**.
= IPv6

Wish list for IPng

- Support for **real-time services**
- **Security** support
- **Autoconfiguration**: 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.

In addition to this wish list, one absolutely non-negotiable 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! → 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

- IPv6 provides a 128-bit address space,a s opposed to the 32 b it version 4. So, while version 4 can potentially address 4 billion

nodes if address assignment efficiency reaches 100%, IPv6 can address 3.4 x 10^{38} nodes, again assuming a 100% efficiency.

ADDRESS SPACE ALLOCATION

Drawing on the effectiveness of CIDR in PVv4, 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.

Table 4.1 Address Prefix Assignments for IPv6

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

ADDRESS NOTATION

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.

```
47CD:1234:4422:AC02:0022:1234:A456:0124
```

- ⇨ Since there are a few special types of IPv6 addresses, there are some special notations that may be helpful in certain circumstances.
  Ex: an address with a large number of contiguous 0s can be written more compactly by omitting all the 0 fields.

  ```
  47CD:0000:0000:0000:0000:0000:A456:0124
  ```

  Becomes

  ```
  47CD::A456:0124
  ```

- ⇨ This form of shorthand can only be used for one set of contiguous 0s in an address to avoid ambiguity;

GLOBAL UNICAST ADDRESSES

- ⇨ Most important sort of addressing that IPv6 must provide is plain old unicast addressing. It must do this in a way that supports the rapid rate of addition of new hosts to the Internet and that

allows routing to be done in a scalable way as the number of physical networks in the Internet grows.

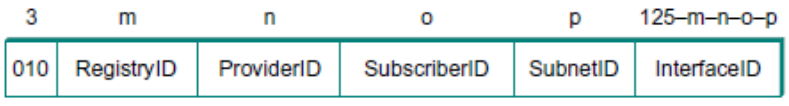
⇨ At the heart of IPv6 is the **unicast address allocation plan** that determines how unicast addresses will be assigned to service providers, autonomous systems, networks, hosts, and routers.

We may think of a non-transit AS as a **subscriber**, and of a transit AS as a **provider** (**direct** >< **indirect** (= backbone networks)).

⇨ The goal of the IPv6 address allocation plan is to **provide aggregation of routing information** to reduce the burden on Intradomain routers.
Key idea: use an **address prefix**, a set of contiguous bits at the most significant end of the address, to aggregate reachability information to a large number of networks and even to a large number of autonomous systems.
How to achieve this? Assign an address prefix to a direct provider and then for that direct provider to assign longer prefixes that begin with that prefix to its subscribers.
Drawback: if a site decides to change providers, it will need to obtain a new address prefix and renumber all the nodes in the site. This could be a colossal undertaking, enough to dissuade most people from ever changing providers.

⇨ **Does it make sense for hierarchical aggregation to take place at other levels in the hierarchy**? Ex: should all providers obtain their address prefixes from within a prefix allocated to the backbone to which they connect?
Given that most providers connect to multiple backbones, this probably doesn't make sense Also, since the number of providers is much smaller than the number of sites, the benefits of aggregating at this level are much fewer.

One place where aggregation may make sense is at the **national** or **continental level**. Continental boundaries form natural divisions in the Internet topology.
Ex: if all addresses in Europe had a common prefix, then a great deal of aggregation could be done, and most routers in other continents would only need one routing table entry for all networks with the Europe prefix.
Providers in Europe would all select their prefixes such that they began with the European prefix.

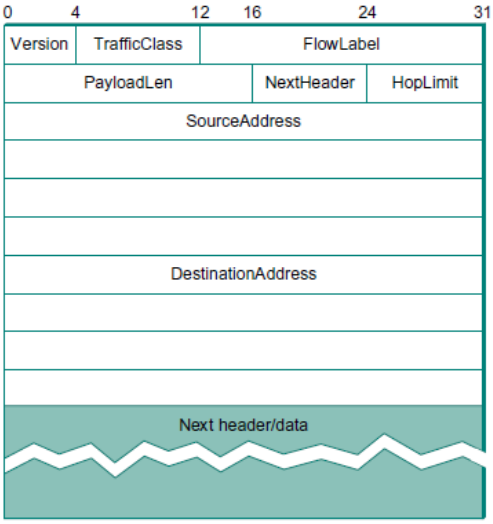


The RegistryID might be an identifier to a European address registry, with different IDs assigned to other continents or countries.

⇨ One tricky situation could occur when a subscriber is connected to more than one provider. Which prefix should the subsricber use for his or her site?
Ex: suppose a subscriber is connected to two providers, X and Y. If the subscriber takes his prefix from X, then Y has to advertise a prefix that has no relationship to its other subscribers and that as a consequence cannot be aggregated. If the subscriber numbers part of his AS with the prefix of X and part with the prefix of Y, he runs the risk of having half his site become unreachable if the connection to one provider goes down. One solution that works fairly well if X and Y have a lot of subscribers in common is for them to have three prefixes between them: one for subscribers of X only, one for subscribers of Y only, and one for the sites that are subscribers of both X and Y.

PACKET FORMAT

⇨ The header format of IPv6 is simpler than IPv4. This is due to a concerted effort to remove unnecessary functionality from the protocol.

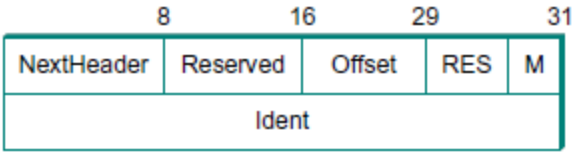


1) **Version field**: set to 6 for IPv6. This is in the same place relative to the start of the header as IPv4's version field so that header-processing software can immediately decide which header format to look for.
2) **Traffic class** and **flow label** fields both relate to quality of service issues.

3) **Pay load Len** field gives the length of the packet, excluding the IPv6 header, measured in bytes.
4) **NextHeader** field replaces both the IP options and Protocol field of IPv4. If there are no special headers, the NextHeader field is the demux key identifying the high-level protocol running over IP. That is, it serves the same purpose as the IPv4 Protocol field.
5) **Fragmentation** is now handled as an optional header. This means that the fragmentation related fields of IPv4 are not included in the IPv6 header.
6) **HopLimit** is the TTL of IPv4.
7) The bulk of the header is taken up with the source and destination addresses, each of which is 16 bytes (128 bits) long. So, the IPv6 header is always 40 bytes long.

Each option has its own type of extension header. The type of each extension header is identified by the value of the NextHeader field in the header that precedes it, and each extension header contains a NextHeader field to identify the header following it.

The last extension header will be followed by a transport-layer header (TCP) and in this case the value of the NextHeader field is the same as the value of the Protocol field would be in IPv4 header.



This header provides functionality similar to the fragmentation fields in IPv4, but it is only present if fragmentation is necessary.

AUTOCONFIGURATION

⇨ One factor that has inhibited faster acceptance of the technology is the fact that getting connected to the Internet has typically required a fair amount of system administration expertise.
Every host that is connected to the Internet needs to be configured with a certain minimum amount of information, such as a valid IP address, a subnet mask for the link to which it attaches, and the address of a name server.
⇨ It has not been possible to unpack a new computer and connect it to the Internet without some preconfiguration.

⇨ One goal of IPv6, is to provide support for autoconfiguration, sometimes referred to as **plug-and-play operation.**
IPv4: autoconfiguration is possible but it depends on the existence of a server that is configured to hand out addresses and other configuration information to DHCP clients.
><
IPv6: helps to provide a useful, new form of autoconfiguration called **stateless autoconfiguration**, which does not require a server.

IPv6 unicast addresses are hierarchical, and the least significant portion is the interface ID. Thus, we can subdivide the autoconfiguration problem into two parts:
1) Obtain an interface ID that is unique on the link to which the host is attached.
2) Obtain the correct address prefix for this subnet.

ADVANCED ROUTING CAPABILITIES

Another of IPv6's extension header is the **routing header**. In the absence of this header, routing for IPv6 differs very little from that of IPv4 under CIDR. The routing header contains a list of IPv6 addresses that represent nodes or topological areas that the packet should visit and route to its destination.

A host could say that it wants some packets to go through a provider that is cheap, others through a provider that provides high reliability, and still others through a provider that the host trusts to provide security.

⇨ To provide the ability to specify topological entities rather than individual nodes, IPv6 defines an any cast address. An any cast address is assigned to a set of interfaces, and packets sent to that address will go to the "nearest" of those interfaces. The nearest is determined by routing protocols.

OTHER FEATURES

IPv6 includes several additional features:

- Security
- Mobility

Main driver of IPv6: need for larger addresses.

3. Multicast

⇨ Multi-access networks like Ethernet implement multicast in hardware. There are, however, applications that need a broader multicasting capability that is effective at the scale of internetworks.
Ex: when a radio station is broadcast over the Internet, the same data must be sent to all the hosts where a user has tuned in that station.

1) One-to-many communication
2) Many-to-many

⇨ Normal IP communication in which each packet must be addressed and sent to a single host, is not well suited to such applications.
If an application has data to send to a group, it would have to send a **separate packet** with the identical data to each member of the group.
This **redundancy** consumes more bandwidth than necessary. Besides, the redundant traffic is not distributed evenly but rather is focused around the sending host, and may easily exceed the capacity of the sending host and the nearby networks and routers.

The basic IP multicast model is a many-to-many model based on multicast groups, where each group has its own IP multicast address. The hosts that are members of a group receive copies of any packets ent to that group's multicast address.
A host can be in multiple groups, and it can join and leave groups freely by telling its local router using a protocol.

- Unicast addresses: associate with a node or an interface
- Multicast addresses: associate with an abstract group, the membership of which changes dynamically over time.

Compared to using unicast IP to deliver the same packets to many receivers, IP multicast is more scalable because it eliminates the redundant traffic (packets) that would have been sent many times over the same links, especially those near to the sending host.

⇨ Many-to-many multicast has been supplemented with support for a form of one-to-many multicast. This one is called **source-specific** multicast (SSM).
A receiving host specifies both a multicast group and a specific sending host.

A host signals its desire to join or leave a multicast group by communicating with its local router using a special protocol for just that purpose.
IPv4: that protocol is the Internet Group Management Protocol (IGMP)
IPv6: it is multicast Listener discovery (**MLD**). The router then has the responsibility for making multicast behave correctly with regard to that host.
Because a host may fail to leave a multicast group when it should, the router periodically polls the LAN to determine which groups are still of interest to the attached hosts.

3.1 Multicast Addresses

IP has a subrange 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 space reserved for multicast group addresses. Some subranges of the multicast ranges are reserved for Intradomain multicast, so they can be reuses independently by different domains.

There are thus 28 bits of possible multicast address in IPv4 when we ignore the prefixes shared by all multicast addresses.

3.2 Multicast Routing (DVMRP, PIM, MSDP)

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. Thus, where **unicast forwarding** tables collectively specify a **set of paths**, **multicast forwarding tables** collectively specify a set of **trees**: multicast distribution trees.
Moreover, to support Source-Specific Multicast, the multicast forwarding tables must indicate which links to use based on the combination of multicast address and the IP address of the source, again specifying a set of trees.

Multicast routing: process by which the multicast distribution trees are determined or, more concretely, the process by which the multicast forwarding tables are built. A multicast routing should not only work, it must also scale reasonably well as the network grows, and it must accommodate the autonomy of different routing domains.

DVMRP

Distance-vector routing can be extended to support multicast.

- ⇨ The resulting protocol is called Distance Vector Multicast Routing Protocol, or DVMRP. This was the first multicast routing protocol to see widespread use.
  Extending the algorithm to support multicast is a two-stage process.
  1) Create a broadcast mechanism that allows a packet to be forwarded to all the networks on the internet.
  2) Refine this mechanism so that it prunes back networks that do not have hosts that belong to the multicast group.
- ⇨ **DVMRP = flood-and-prune protocol.**

Given a unicast routing table, each router knows that the current shortest path to a given destination goes through NextHop. So, whenever it receives a multicast packet from source S, the router forwards the packet on all outgoing links if and only –if the packet arrived over the link that is on the shortest path to S.

- ⇨ This strategy floods packets outward from S but does not loop packets back toward S.

  Two major shortcomings
  1) It truly floods the network. It has no provision for avoiding LANs that have no members in the multicast group.
  2) 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 kinks 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.
     Solution: **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.
     The router that has the shortest path to source S is selected as the parent. A tie between two routers would be broken

according to which router has the smallest address.

Reverse Path Broadcast (RPB). The path is reverse because we are considering the shortest path toward the source when making our forwarding decisions, as compared to unicast routing, which looks for the shortest path to a given destination.

⇨ We want to prune the set of networks that receives ach packet addressed to group G to exclude those that have no hosts that are members of G. This can be accomplished in **two stages**.

1) **Recognize when a leaf network has no group members.**
   Determining that a network is a leaf is easy, if the parent router as described above is only router in the network, then the network is a leaf. Determining if any group members reside on the network is accomplished by having each host that is a member of group G periodically announce this fact over the network.
2) **Propagate this "no members of 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.

PIM-SM

PIM = Protocol Independent Multicast.

PIM was developed in response to the **scaling problems** of earlier multicast routing protocols. 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.

Ex: 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.

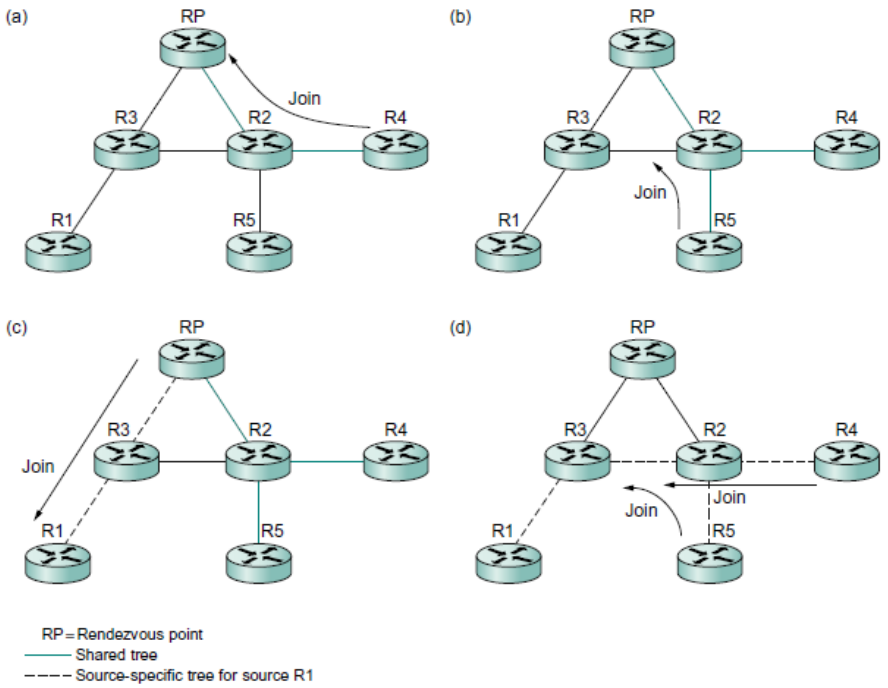
1) **Sparse mode:** has become the dominant multicast routing protocol.
2) **Dense mode:** uses a flood-and-prune algorithm like DVMRP and suffers from the same scalability problem.

In PIM-SM, routers explicitly join the multicast distribution tree using PIM protocol messages known as **join messages**. The contrast to DVMRP's approach of creating a broadcast tree first and then pruning the uninterested routers.

⇨ Where to send those Join messages. After all, any host could send to the multicast group.
To address this, PIM-SM assigns to each group a special router known as the rendezvous point (RP). In general, a number of routers in a domain are configured to be candidate RP's and PIM-SM defines a set of procedures by which all the routers in a domain can agree on the router to use as the RP for a given group.

A multicast forwarding tree is built as a result of routers sending join messages to the RP. PIM-SM allows two types of trees to be constructed:

- Shared tree: may be used by all senders
- Source-specific tree: may be used only by a specific sending host.



a) Router R4 is sending a join to the rendezvous point for some group. A join message clearly must pass trhough some sequence of routers before reaching the RP.
Each router along the path looks at the join and creates a forwarding table entry for the shared tree, called a (*, G), where * means all senders.

b) As more routers send joins toward the RP, they cause new branches to be added to the tree. In this case, the join only need to travel to 2, 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 needs not forward the Join on the RP. The end result of this process is to build a tree whose root is the RP.

c) In this figure we see a source-specific route from R1 to the RP and a tree that is valid for all senders from the RP to the receivers.
The next possible optimization is to replace the shared tree with a source-specific tree. This is desirable because the path from sender to receiver via the RP might be significantly longer than the shortest possible path.

INTERDOMAIN MULTICAST (MSDP)

PIM-SM has some significant shortcomings when it comes to Interdomain multicast.

- Existence of a single RP for a group goes **against** the principle that **domains are autonomous**. For a given multicast group, all the participating domains would be dependent on the domain where the RP is located.
- PIM-SM protocol is typically not used across domains, only **within a domain.**

⇨ To extend multicast across domains using PIM-SM, the **Multicast Source Discovery Protocol (MSDP)** was devised.
MSDP is used to connect different domains, each running PIM-SM internally, with its own RPs by connecting the RPs of the different domains.
Each RP has one or more MSDP peer RPs in other domains. Each pair of MSDP peers is connected by a TCP connection over which the MSDP protocol runs. Together, all the MSDP peers for a given multicast group form a loos mesh that is used as a broadcast network. MSDP messages are broadcast through the mesh of peer RPs using the Reverse Path Broadcast algorithm.

⇨ What information does MSDP broadcast through the mesh of RPs? Not group membership information. When a host joins a group, the furthest that information will flow is its own domain's RP. Instead, it is source, multicast sender, information.

SOURCE-SPECIFIC MULTICAST (PIM-SSM)

The original service model of PIM was a many-to-may model. Receivers joined a group, and any host could send to the group.

⇨ Was recommended to add a one-to-many model.
Once the need for a one-to-many service model was recognized, it was decided to make the source-specific routing capability of PIM-SM explicitly available to hosts.
It turns out that this mainly required changes to IGMP and its IPv6 analog, MDL, rather than PIM itself.

⇨ New capability = PIM-SSM.

New concept: the **channel**. This is the combination of a source address S and a group address G. The group address G looks just like a normal IP multicast address, and both IPv4 and IPv6 have allocated subranges of the multicast address space for SSM.

Benefits:

- Multicasts travel **more directly to receivers**.
- The address of a channel is effectively a multicast group address plus a source address. Therefore, given that a certain range of multicast group addresses will be used for SSM exclusively, multiple domains can use the same multicast group address independently and without conflict, as long as they use it only with sources in their own domains.
- Because only the specified source can send to an SSM group, there is less risk of attacks based on malicious hosts overwhelming the routers or receivers with bogus multicast traffic.
- PIM-SSM can be used across domains exactly as it is used witching a domain, without reliance on anything like MSDP.

BIDIRECTIONAL TREES (BIDIR-PIM)

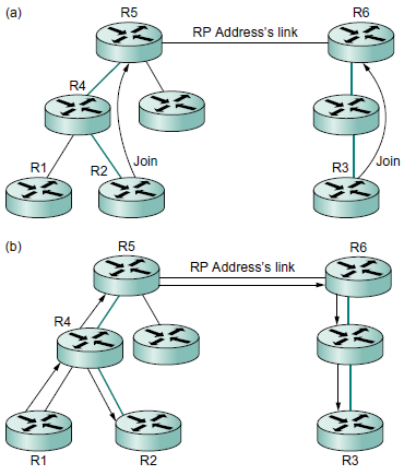
BIDR-PIM is a recent variant of PIM-SM that is well suited to many-to-many multicasting within a domain, especially when senders and receivers to a group may be the same, as in a multiparty videoconference.

As in PIM-SM, would-be receivers join groups by sending IGMP Membership Report messages, and a shared three rooted at an RP is used to forward multicast packets to receivers. However, unlike PIM-SM, the shared tree also has branches to the sources. That wouldn't make any sense with PIM-SM's unidirectional tree, but BIDIR-PIM's threes are bidirectional. A router that receives a multicast packet from a downstream branch can forward it both up the tree and down other branches.

A surprising aspect of BIDR-PIM is that there need not actually be an RP. All that is needed is a routable address, which is known as an RP address even though it need not be the address of an RP or anything at all.

⇨ How can this be?

A join from a receiver is forwarded toward the RP address until it reaches a router with an interface on the link where the RP address would reside, where the join terminates.



The upper figure shows a join from R2 terminating at R5, and a join from R3 terminating at R6. The upstream forwarding of a multicast packet similarly flows toward the RP address until it reaches a router with an interface on the link where the RP address would reside. But then, the router forwards the multicast packet onto that link as the final step of upstream forwarding, ensuring that all other routers on that link receive the packet.

The second figure illustrates the flow of multicast traffic originating at R1.

⇨ Multicast is a difficult problem space in which to find optimal solutions. You need to decide which criteria you want to optimize and what sort of application you are trying to support before you can make a choice of the "best" multicast mode for the task.

4. Multiprotocol label switching (MPLS)

MPLS combines some of the properties of virtual circuits with the flexibility and robustness of datagrams.

1) MPLS is very much associate with the Internet Protocol's datagram-based architecture, it **relies on IP addresses** and **IP routing protocols** to do its job.
2) On the other hand, MPLS-enabled routers also forward packets by examining relatively **short, fixe-length labels**, and the labels have **local scope,** just like in a virtual circuit network.

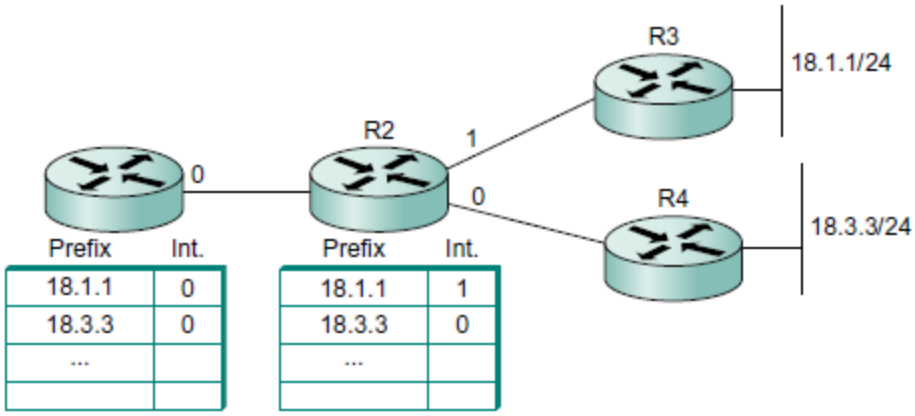
⇨ **What is it good for?**

1) To enable IP capabilities on devices that do not have the capability to forward IP datagrams in the normal manner.
2) To forward IP packets along explicit routes, pre-calculated routes that don't

necessarily match those that normal IP routing protocols would select.

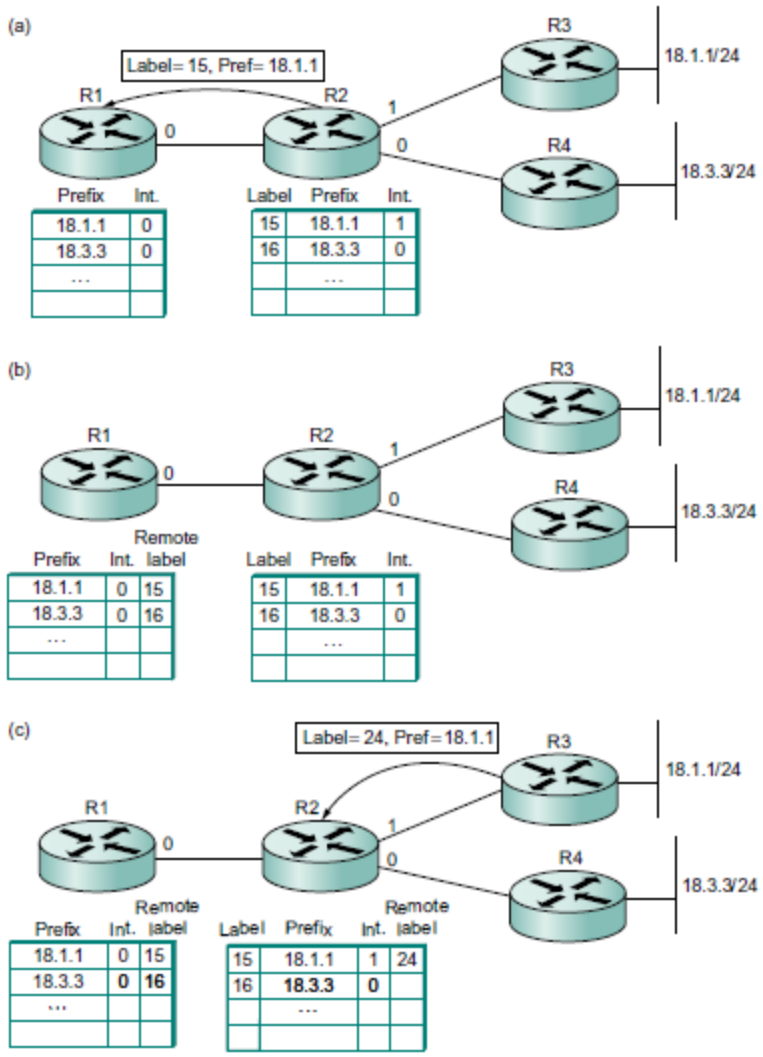
3) To support certain types of virtual private network services.

4.1 Destination-Based Forwarding



Each of the two routers on the far right (R3 and R4) has one connected network, with prefixes 18.1.1/24 and 28.3.3/24. The remaining routers R1 and R2 have routing tables that indicate which outgoing interface each router would use when forwarding packets to one of those two networks.

When MPLS is enable on a router, the router allocates a label for each prefix in its running table and advertises both the label and the prefix that it represents to its neighboring routers.



a) Router R2 has allocated the label value 15 for the prefix 188.1.1 and the label value 16 for the prefix 18.3.3. These labels can be chosen at the convenience of the allocating router and can be thought of as indices into the routing table. After allocating the labels, R2 advertises the label bindings to its neighbors. In this case, we see R2 advertising a binding between the label 15 and the prefix 18.1.1 to R1.

b)

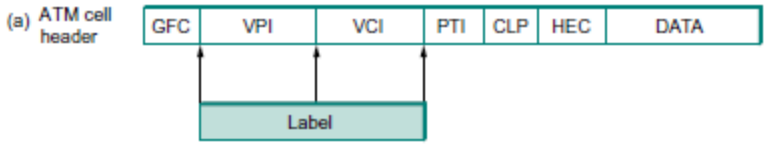
c) We see another label advertisement from router R3 to R2 for the prefix 18.1.1. and R2 places the remote label that it learned from R3 in the appropriate place in its table.

Main idea: We replace the normal IP destination address lookup with a label lookup. Although IP addresses are always the same length, IP prefixes are of variable length, and the IP destination address lookup algorithm needs to find the longest match, the longest prefix that matches the high order bits in the IP address of the packet being forwarded. By contrast, the label forwarding mechanism just described is an exact match algorithm. It is possible to implement a very simple exact match algorithm. For instance, by using the label as an index into an array where each element in the array is one line in the forwarding table.

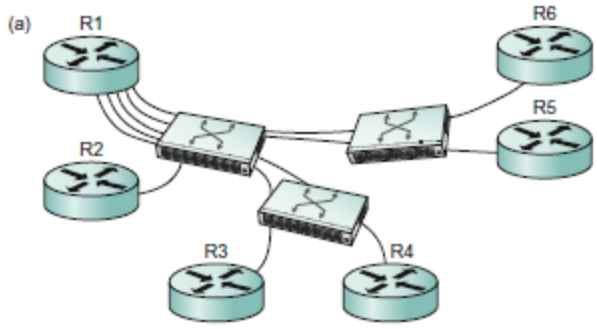
We have just said that labels are attached to packets, but where exactly are they attached? This depends on the type of link on which packets are carried.

4) When IP packets are carried as complete frames, as they are on most link types including Ethernet and PPP, the label is inserted as a "shim" between the layer 2 header and the IP header.

5) If an ATM switch is to function as an MPLS, then the label needs to be in a place where the switch can use it, and that means it needs to be in the ATM cell header, exactly where one would normally find the virtual circuit identifier and virtual path identifier fields.

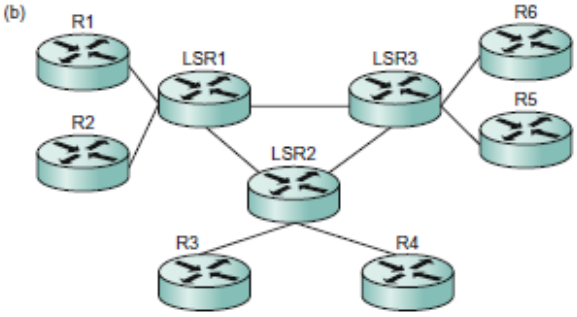


⇨ We can build a network that uses a mixture of conventional IP routers, label edge routers, and ATM switches functioning as LSRs, and they would all use the same routing protocols.



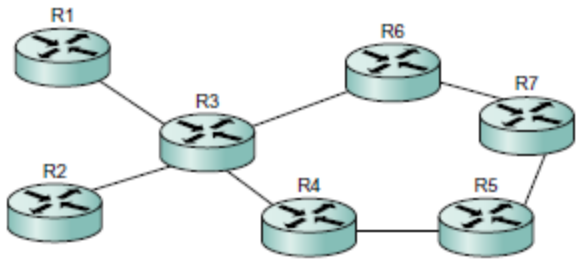
Set of routers interconnected by virtual circuits over an ATM network, a configuration called an overlay network. At one point in time, networks of this type were often built because commercially available ATM switches supported higher total throughput than routers. Today, networks like this are less common because routers have caught up with and even surpassed ATM switches.

The ATM switches have been replaced with LSRs. There are no longer virtual circuits interconnecting the routers. Thus, R1 has only one adjacency, with LSR1. In large networks, running PLS on the switches leads to a significant reduction in the number of adjacencies that each router must maintain and can greatly reduce the amount of work that the routers have to do to keep each other informed of topology changes.



By running the same routing protocols on edge routers and on the LSRs, an additional benefit is that the edge routers now have a **full view of the topology** of the network. This means that if some link or node fails inside the network, the edge routers will have a better chance of picking a good new path than if the ATM switches rerouted the affected VCs without the knowledge of the edge routers.

4.2 Explicit routing



Fish network

Suppose that the operator of the network has determined that any traffic flowing from R1 to R7 should follow the path R1-R3-R6-R7 and that any traffic going from R2 to R7 should follow the path R2-R3-R4-R5-R7. One reason for such a choice would be to make **good use** of the **capacity available** along the two distinct paths from R3 to R7.
We can think of the R1-to-R7 traffic as constituting one forwarding equivalence class, and the R2-to-R7 traffic constitutes a second FEC.

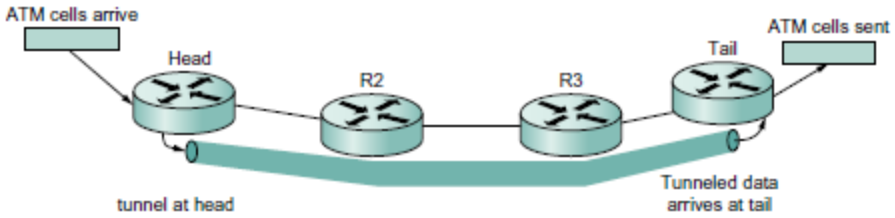
Because MPLS uses label swapping to forward packets, it is easy enough to achieve the desired routing if the routers are MPLS enabled. If R1 and R2 attach distinct labels to packets before sending them to R3, thus identifying them as being in different FECs, then R3 ca, forward packets from R1 and R2 along different paths.

⇨ How do all the routers in the network agree on what labels to use and how to forward packets with particular labels? → **Resource Reservation Protocol (RSVP).**

One of the applications of explicit routing is traffic engineering, which refers to the task of ensuring that sufficient resources are available in a network to meet the demands placed on it. Controlling exactly which paths the traffic flows on is an important part of traffic engineering. Explicit routing can also help to make networks more resilient in the face of failure, using a capability called fast reroute.

Finally, explicit routes need not be calculated by a network operator as in the above example. Routers can use various algorithms to calculate explicit routes automatically. The most common of these is constrained shortest path first (CSPF), which is like the link-state algorithms. It also takes various constraints into account.

4.3 Virtual Private Networks and Tunnels



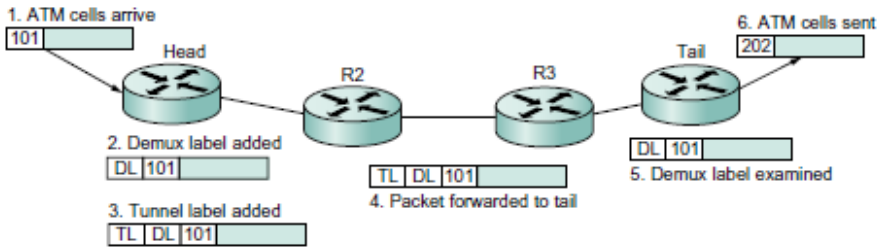
We previously talked about virtual private networks (VPNs). We noted that one way to build them was using tunnels. It turns out that MPLS can be thought of as a way to build tunnels, and this makes it suitable for building VPNs of various types.

The simplest form of MPLS VPN to understand is a layer 2 VPN. In this type of VPN, MPLS is used to tunnel layer 2 data across a network of MPLS-enabled routers.
One reason for tunnels is to provide some sort of network service that is not supported by some routers in the network.
IP routers are not ATM switches so you cannot provide an ATM virtual circuit service across a network of conventional routers. However, if you had a pair of routers interconnected by a tunnel, they could send ATM cells across the tunnel and emulate an ATM circuit.
= pseudo wire emulation

⇨ How are IP tunnels built?
The router at the entrance of the tunnel wraps the data to be tunneled in an IP header (the **tunnel header**), which represent the address of the router at the far end of the tunnel and sends the data like any other IP packet. The receiving router receives the packet with its own address in the header, strips the tunnel header and finds the data that was tunneled, which it then processes.

An MPLS tunnel is not too different from an IP tunnel, except that the tunnel header consists of an MPLS header rather than an IP header.

⇨ How would an ATM cell be forwarded?



1) An ATM cells arrives on the designated input port with the appropriate VCI value (here: 101).

2) The head router attaches the demultiplexing label that identifies the emulated circuit.
3) The head router then attaches a second label, which is the tunnel label that will get the packet to the tail router.
4) Routers between the head and tail forward the packet using only the tunnel label.
5) The tail router removes the tunnel label, finds the demultiplexing label and recognizes the emulated circuit.
6) The tail router modifies the ATM VCI to the correct value and sends it out the correct port.

The packet has two labels attached to it. Labels may be stacked on a packet to any depth.

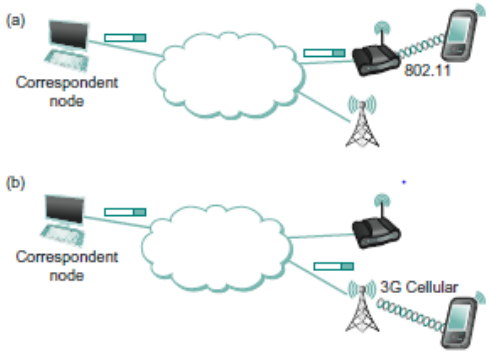
5. Routing among mobile devices

⇨ Mobile devices present some challenges for the Internet architecture. The Internet was designed in an era when computers were large, immobile devices, and while the Internet's designers probably had some notion that mobile devices might appear in the future, it's fair to assume it was not a top priority to accommodate them.
⇨ Today, mobile computers are everywhere, in the forms of laptops and IP-enabled mobile phones, and increasingly in other forms such as sensors.

5.1 Challenges for Mobile Networking

One key enabling technology that made the hotspot feasible is DHCP. You can settle in at a café, open your laptop, obtain an IP address for your laptop, and get your laptop talking to default router and a Domain Name System (DNS).

When you move from one access network to another, you need to get a new IP address. One that corresponds to the new network. But the computer or telephone at the other end of your conversation does not immediately know where you have moved or what your new IP address is. Consequently, in the absence of some other mechanism, packets would continue to be sent to the address where you used to be, not where you are now.



As the mobile node moves from the 802.11 network to the cellular network in (b), somehow packets from the correspondent node need to find their way to the new network and then on to the mobile node.
There are many different ways to tackle the problem just described.

Assuming that there is some way to redirect packets so that they come to your new address rather than your old address, the next immediately apparent problems relate to security.

⇨ The IP addresses actually serve two tasks.

1) They are used as an **identifier** of an endpoint. We should think of it as a long-lived name for the endpoint.
2) They are used to **locate** the endpoint. This is more temporary information about how to rout e packets to the endpoint.

As long as devices do not move, or do not move often, using a single address for both jobs seem pretty reasonable. But once devices start to move, you would rather like to have an identifier that does not change as you move.
= endpoint identifier or host identifier + separate locator
The separating locators from identifiers has been around for a long time, and most of the approaches to handling mobility described below provide such a separation in some form.

While we are all familiar with endpoints that move, it is worth noting that routers can also move. This is certainly less common today than endpoint mobility, but there are plenty of environments where a mobile router might make sense.

As with many technologies, support for mobility raises issues of incremental deployment. Given that, for its first couple of decades, the Internet consisted entirely of nodes that didn't move, it's fair to assume that there will be a lot of routers and hosts around for the foreseeable future that make that assumption.

⇨ Mobility solutions need to deal with **incremental deployment**. Conversely, IP version 6 had the ability to make mobility part of its design from the outset, which provides it with some advantages.

5.2 Routing to Mobile Hosts (Mobile IP)

The mobile host is assumed to have a permanent IP address, called its home address, which has a network prefix equal to that of its home network. This is the address that will be used

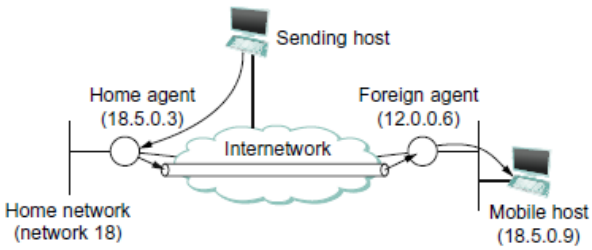
by other hots when they initially send packets to the mobile host. Because it does not change, it can be used by long-lived applications as the host roams. We can think of this as the long-lived identifier of the host.

When the host moves to a new foreign network away from its home network, it typically acquires a new address on that network using some means such as DHCP. This address is going to change every time the host roams to a new network, so we can think of this as being more like the locator for the host, but it is important to note that the host does not lose its permanent home address when it acquires anew address on the foreign network. This home address is critical to its ability to sustain communications as it moves.

While the majority of routers remain unchanged, mobility support does require some new functionality in at least one router, known as the home agent of the mobile node. This router is located on the home network of the mobile host. In some cases, a second router with enhanced functionality, the foreign agent, is also required. This router is located on a network to which the mobile node attaches itself when it is away from its home network.

Both home and foreign agents periodically announce their presence on the networks to which they are attached using agent advertisement messages. A mobile host may also solicit an advertisement when it attaches to a new network. The advertisement by the home agent enables a mobile host to learn the address of its home agent before it leaves its home network. When the mobile host attaches to a foreign network, it hears an advertisement from a foreign agent and registers with the agent, providing the address of its home agent. The foreign agent then contacts the home agent, providing a care-of address. This is usually the IP address of the foreign agent.
At this point, we can see that any host that tries to send a packet to the mobile host will send it with a destination address equal to the home address of that node. Normal IP forwarding will cause that packet to arrive on the home network of the mobile node on which the home agent is sitting. Thus, we can divide the problem of delivering the packet to the mobile node into three part

1) How does the home agent intercept a packet that is destined for the mobile node?
2) How does the home agent then deliver the packet to the foreign agent?
3) How does the foreign agent deliver the packet to the mobile node?



ROUTE OPTIMIZATION IN MOBILE IP

⇨ Drawback to the above approach. The route from the correspondent node to the mobile node can be significantly suboptimal. One of the most extreme examples is when a mobile node and the correspondent node are on the same network, but the home network for the mobile node is on the far side of the internet. The sending correspondent node addresses all packets to the home network. They traverse the Internet to reach the home agent. It would clearly be nice if the correspondent node could find out that the mobile node is actually on the same network and deliver the packet directly.
In the more general case, the goal is to deliver packets as directly as possible from correspondent node to mobile node without passing through a home agent.
= **triangle routing problem.**

Basic idea: Let the correspondent node know the care-of-address of the mobile node. The correspondent node can then create its own tunnel to the foreign agent. This is treated as an optimization of the process just described.

Problem: the binding cache may become out-of-date if the mobile host moves to a new network. If an out-of-date cache entry is used, the foreign agent will receive tunneled packets for a mobile node that is no long registered on its network. In this case, it sends a binding warning message back to the send to tell it to stop using this cache entry.

MOBILITY IN IPv6

- Since all IPv6-capable hosts can acquire an address whenever they are attached to a foreign network, Mobile IPv6 does away with the foreign agent and includes the necessary capabilities to act as a foreign agent in every host.
- Inclusion of a flexible set of extension headers. Rather than tunneling a packet to the mobile node at its care-of address, an IPv6 node can send an IP packet to the care-of address with the home address contained in a routing header.
- Many open issues remain in mobile networking! Managing the power consumption of mobile devices is increasingly important, so that smaller devices with limited battery power can be built.

CHAPTER 5: END-TO-END PROTOCOLS

⇨ Turn host-to-host packet delivery service into a **process-to-process communication channel**. This is the role played by the transport level of the network architecture, which, because it supports **communication** between application programs running in end nodes, is sometimes called the **end-to-end protocol**.

Common end-to-end services

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

1) Message delivery
2) Delivers messages in the same order they are sent
3) Delivers at most one copy of each message
4) Supports arbitrarily large messages
5) Supports synchronization between the sender and the receiver
6) Allows the receiver to apply flow control to the sender
7) Supports multiple application processes on each host.

Underlying best-effort network

8) Drop messages
9) Re-orders messages
10) Delivers duplicate copies of a given message
11) Limits messages to some finite size
12) Delivers messages after an arbitrarily long delay.

1. Simple demultiplexer (UDP)

The simples possible transport protocol is one that **extends the host-to-host delivery service** of the underlying network into a process-to-process communication service.

13) There are likely to be many processes running on any given host, so the protocol needs to add a level of demultiplexing, thereby allowing multiple application processes on each host to share the network.

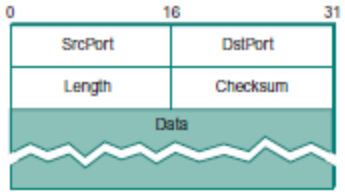
14) The transport protocol adds no other functionality to the best-effort service provided by the underlying network.
Ex: Internet's User Datagram Protocol

Issue: form of the address used to identify the target process. Although it is possible for processes to directly identify each other with an OS-assigned process id, such an approach is only practical in a closed distributed system in which a single OS runs on all hosts and assigns each process a unique ID.

A more common approach, and the one used by UDP, is for processes to indirectly identify each other using an abstract locater, usually called a port. The basic idea is for a source process to send a message to a port and for the destination process to receive the message from a port.

The header for an end-to-end protocol that implements this demultiplexing function typically contains an identifier (port) for both the sender and the receiver (source and destination) of the message.

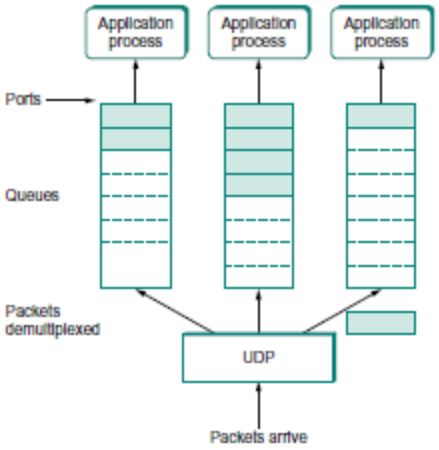
Ex: the UDP header is given in the above figure.



15) No flow control to telle the sender to slow donw.
    - If queue full: discard message
    - If queue empty: receiving process blocks
16) Checksum: UDP ensures the correctness of the message by the use of a checksum.
    - Optional in IPv4 (mandatory in IPv6)
    - Computed on UDP header + data + pseudo header
      Pseudo header = IP protocol number + source IP address + destination IP address + UDP length.

Issue: how does a process learn the port for the process to which it wants to send a message. Typically, a client process initiates a message exchange with a server process. Once a client has contacted a server, the server knows the client's port and can reply to it. The real problem, therefore is how the client learns the serve's port in the first place. A common approach is for the server to

accept messages at a well-known port. That is, each serve receives its messages at some fixed port that is widely published.



2. Reliable byte stream (TCP)

- ⇨ A more sophisticated transport protocol is one that offers a **reliable, connection-oriented byte-stream** service.
- ⇨ The **Internet's Transmission Control Protocol** is probably the most widely used protocol of this type. It is also the most carefully tuned.

TCP is a **full-duplex protocol**, meaning that each TCP connection supports a pair of by streams, one flowing in each direction. It also includes a **flow-control mechanism** for each of this byte streams that allows the receiver to **limit how much data** the sender can transmit at a given time.

Like UDP, TCP **supports a demultiplexing mechanism** that allows multiple application programs on any given host to simultaneously carry on a conversation with their peers.

TCP also implements a highly tuned **congestion-control mechanism**. The idea of this mechanism is to throttle how fast TCP sends data, not for the sake of keeping the sender from over-running the receiver, but so as to keep the sender from overloading the network.

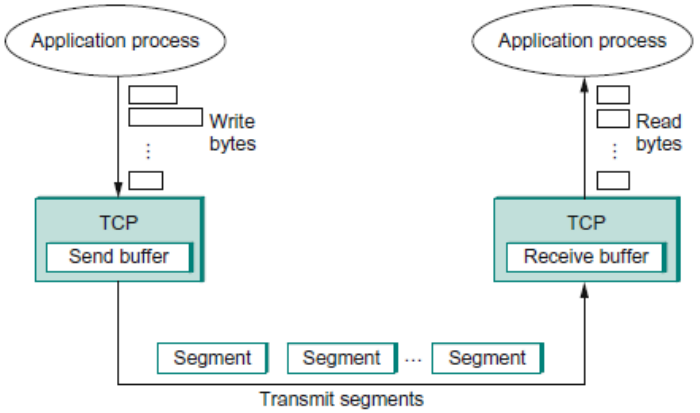
2.1 End-to-End issues

- ⇨ At the heart of TCP is the sliding window algorithm. Even though this is the same basic algorithm as the one we discussed in the

second chapter, there are many important differences since TCP runs over the Internet rather than a point-to-point link.

- TCP supports logical connections between processes that are running on any two computers in the Internet. This means that TCP needs an explicit **connection establishment phase** during which the two sides of the connection agree to exchange data with each other.
  >< the sliding window algorithm seen earlier runs over a single physical link.
- Whereas a single physical link that always connects the same two computers has a fixed round-trip time (RTT), TCP connections are likely to have widely different round-trip times.
- Packets may be reordered as they cross the Internet, but this is not possible on a point-to-point link where the first packet put into one end of the link must be the first to appear at the other end.
  How far out can order packets get. How late can a packet arrive at the destination?
- The computers connected to a point-to-point kink are generally engineered to support the link.
- Because the transmitting side of a directly connected link cannot send any faster than the bandwidth of the link allows, and only one host is pumping data into the link, it is not possible to unknowingly congest the link.
- The load on the link is visible in the form of a queue of packets at the sender.

2.2 Segment Format



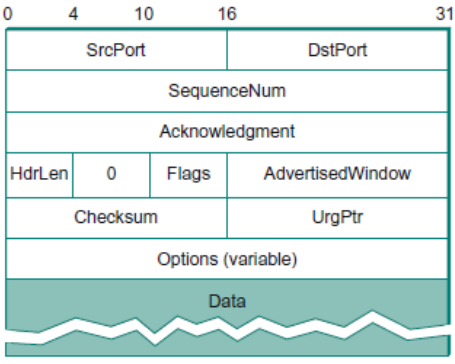
TCP is a byte-oriented protocol, which measns that the sender wirtes bytes into a TCP connection and the receiver reads bytes out of the TCP connection.
Although "byte stream" describes the service TCP offers to application processes, TCP does not, itself, transmit individual bytes over the Internet.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

The packets exchanged between TCP peers in the given figure are called segments, since each one carries a segment of the byte stream.
Tuple <SrcPort, SrcIPAddr, DstPort, DstIPAddr> identifies connection.

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 values flowing in the opposite direction.
- **Flags**
  - SYN: establish connection
  - FIN: terminate connection
  - ACK: ack field is valid
  - URG: segment starts with UrgPtr bytes of urgent data
  - PUSH: data must be pushed to receiving process
  - RESET: sender has become confused and aborts connection
- **Checksum**: computed on TCP header + TCP data + IP pseudo header.

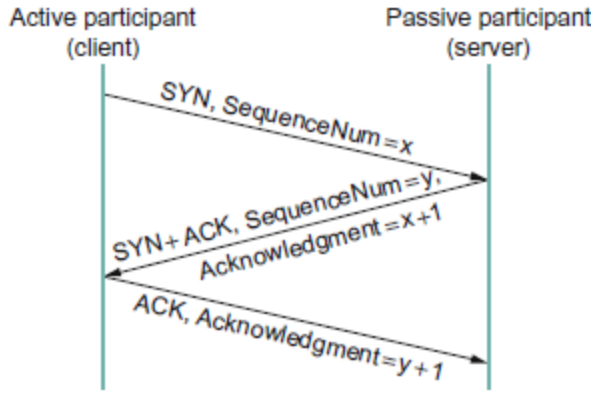


2.3 Connection Establishment and Termination

- A TCP connection begins with a client (**caller**) doing an active o a server (**callee**).
  Assuming that the server had earlier done a passive open, the two sides engage in an exchange of messages to establish the connection.
  A party wanting to initiate a connection performs an active open, while a party willing to accept a connection does a passive open.
- After this connection establishment phase is over, the two sides begin sending data. Likewise, as soon as a participant is done sending data, it closes one direction of the connection, which causes TCP to initiate a round of connection termination messages.

⇨ Connection setup is an asymmetric activity (one side does a passive open and the other side does an active open) >< connection teardown is symmetric (each side has to close the connection independently).

THREE-WAY HANDSHAKE



⇨ 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 stream.

1) The client sends a segment to the server stating the **initial sequence number** it plans to use.
   Flags = SYN, SequenceNum = x
2) The server 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).

3) The client responds with a third segment that acknowledges the server's sequence number (Flags = ACK, Ack = y +1)

⇨ The reason why each side acknowledges a sequence number that is one larger than the one sent is hat the Acknowledgement field actually identifies the "next sequence number expected".

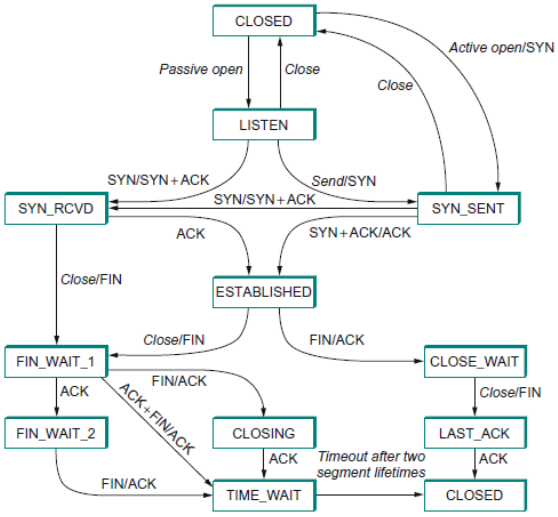
⇨ A timer is scheduled for each of the first two segments, and if the expected response is not received the segment is retransmitted.

CONNECTION TERMINATION

Both sides of the connection must be closed independently. If one side closes the connection, it can still receive data (simple).

STATE-TRANSITION DIAGRAM

⇨ TCP is complex enough that its specification includes a **state-transition diagram**.



This diagram shows only the states involved in **opening a connection** (above ESTABLISHED) and in **closing a connection** (below ESTABLISHED). Everything that goes on while a connection is open, that is, the operation of the sliding window algorithm, is hidden in the established state.
Each circle denotes a stat that one end of a TCP connection can find itself in.
All connections start in the CLOSED state. As the connection progresses, the connection moves from state to state according to the arcs. Each arc is labeled with a tag of the form event/action.

Two kinds of events trigger a state transition:

1) A segment arrives from the peer.
2) The local application process invokes an operation on TCP.

Typical transitions taken through the diagram:

- When opening a connection, the server first invokes a **passive open operation on TCP**, which causes TCP to move to the LISTEN state. At some later time, the client does an active open, which causes its end of the connection to send a SYN segment to the server and to move to the SYN_SENT state.

- When the SYN segment arrives at the server, it moves to the SYN RCVD state and responds with a SYN+ACK segment. The arrival of this segment causes the client to move to the ESTABLISHED state and to send an ACK back to the server.
- When this ACK arrives, the server finally moves to the ESTABLISHED state. In other words, we have just traced the three-way handshake.

⇨ There are three thins to notice about the connection establishment half of the state-transition diagram

1) If the client's ACK to the server is lost, corresponding to the third leg of the three-way handshake, then the connection still functions correctly.
2) There is a funny transition out of the LISTEN state whenever the local process invokes a send operation on TCP.
3) Most of the states that involve sending a segment to the other side also schedule a timeout that eventually causes the segment to be present if the expected response does not happen.

When **terminating a connection**, the important thing to keep in mind is that the application process on both sides of the connection must independently close its half of the connection.

4) If only one side closes the connection, then this means it has no more data to send, but it is still available to receive data from others. → complicates the state-transition diagram because it must account for the possibility that the two sides invoke the close operator at the same time, as well as the possibility that first one side invokes close and then, at some later time, the other side invokes close.

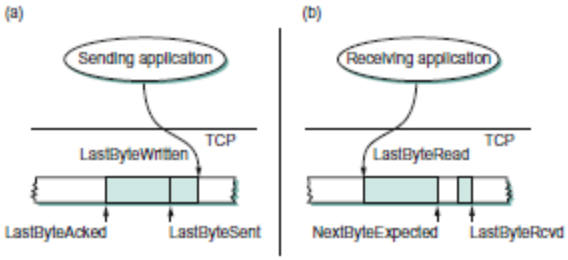
2.4 Sliding Window Revisited

The TCP's variant of the sliding window algorithm serves several purposes.

5) Guarantees the reliable delivery of data
6) Ensures that data is delivered in order

7) Enforces flow control between the sender and the receiver.

RELIABLE AND ORDERED DELIVERY



8) TCP on the **sending side** maintains a send buffer. This buffer is used to store data that has been sent but not yet acknowledged, as well as data that has been written by the sending application but not transmitted.

   Three pointers are maintained into the send buffer, each with an obvious meaning: LastByteAcked, LastByteSent, and LastByteWritten.
   LastByteAcked ≤ LastByteSent
   since the receiver cannot have acknowledged a byte that has not yet been sent, and
   LastByteSent ≤ LastByteWritten
   since TCP cannot send a byte that the application process has not yet written.
   Buffer bytes between LastByteAcked and LastByteWritten.

9) On the **receiving side**, TCP maintains a receive buffer. This buffer holds data that arrives out of order, as well as data that is in the correct order but that the application process has not yet had the chance to read.
   LastByteRead, NextByteExpected, and LastByteRcvd. The inequalities are a little less intuitive, however, because of the problem of out-of-order delivery.
   LastByteRead < NextByteExpected
   is true because a byte cannot be read by the application until it is received

and all preceding bytes have also been received. NextByteExpected points to the byte immediately after the latest byte to meet this criterion. Second,
NextByteExpected ≤ LastByteRcvd+1
since, if data has arrived in order, NextByteExpected points to the byte after LastByteRcvd, whereas if data has arrived out of order, then NextByteExpected points to the start of the first gap in the data.
Buffer bytes between LastByteRead and LastByteRcvd.

FLOW CONTROL

⇨ The sending and receiving application processes are filling and emptying their local buffer, respectively.

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. Recall that in a sliding window protocol, the size of the window sets the amount of data that can be sent without waiting for acknowledgment from the receiver. Thus, the receiver throttles the sender by advertising a window that is no larger than the amount of data that it can buffer.

- TCP on the **receive side** must keep LastByteRcvd−LastByteRead ≤ MaxRcvBuffer to avoid overflowing its buffer. It therefore advertises a window size of AdvertisedWindow = MaxRcvBuffer−((NextByteExpected−1) −LastByteRead) which represents the amount of free space remaining in its buffer.
- As data arrives, the receiver acknowledges it as long as all the preceding bytes have also arrived. In addition, LastByteRcvd moves to the right (is incremented), meaning that the advertised window potentially shrinks. Whether or not it shrinks depends on how fast the local application process is consuming data. If the local process is reading data just as fast as it arrives (causing LastByteRead to be incremented at the same rate as LastByteRcvd), then the advertised window stays open

(i.e., AdvertisedWindow = MaxRcvBuffer). If, however, the receiving process falls behind, perhaps because it performs a very expensive operation on each byte of data that it reads, then the advertised window grows smaller with every segment that arrives, until it eventually goes to 0.

- TCP on the send side must then adhere to the advertised window it gets from the receiver. This means that at any given time, it must ensure that LastByteSent−LastByteAcked ≤ AdvertisedWindow.
  Said another way, the sender computes an effective window that limits how much data it can send:
  EffectiveWindow= AdvertisedWindow−(LastByteSent−LastByteAcked)

PROTECTING AGAINST WRAPAROUND

- TCP's SequenceNum field is 32 bits long
- AdvertisedWindow field is 16 bits long

⇨ TCP has easily satisfied the requirement of the sliding window algorithm that the sequence number space be twice as big as the window size.

⇨ We need to make sure that the sequence number does not wrap around within a 120-second period of time. Whether or not this happens depends on how fast data can be transmitted over the Internet.

⇨ How fast can 32-bit sequence number space be exhausted?

Table 5.1 Time Until 32-Bit Sequence Number Space Wraps Around

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

Solution:

- Store 32-bit timestamp in outgoing segments
- Extend sequence space with timestamp
- Implemented as TCP header option

KEEPING THE PIPE FULL

The relevance of the 16-bit AdvertisedWindow field is that it must be big enough to allow the sender to keep the pipe full.

The receiver is free to not open the window as large as the AdvertisedWindow field allows. In this subsection, we are interested in the situation in which the receiver has enough buffer space to handle as much data as the largest possible AdvertisedWindow allows.

⇨ It is not just the network bandwidth but the delay x bandwidth product that dictates how big the AdvertisedWindow field needs to be.

Table 5.2 Required Window Size for 100-ms RTT

Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

⇨ TCP's AdvertisedWindow field is in even worse shape than its SequenceNum field. It is not big enough to handle even a T3 connection across the continental United States, since a 16-bit field allows us to advertise a window of only 64 KB.

Solution:

- How many bits to shift advertised window?
- Implemented as TCP header option

2.5 Triggering Transmission

⇨ How does TCP decides to transmit a segment?

TCP supports a byte-stream abstraction. In other words, application programs wirte bytes into the stream, and it is up to TCP to decide that it has enough bytes to send a segment.

⇨ What factors govern this decision?

Three mechanisms to trigger the transmission of a segment

- **Maximum segment size** (MSS). It sends a segment as soon as it has collected MSS bytes from the sending process.
  MSS is usually set to the size of the largest segment TCP can send without causing the local IP to fragment.

- The second thing that triggers TCP to transmit a segment is that the sending process has **explicitly asked it to do so**. TCP supports a push operation, and the sending process invokes this operation to effectively flush the buffer of unsent bytes.
- **Timer fires:** the resulting segment contains as many bytes as are currently buffered for transmission.

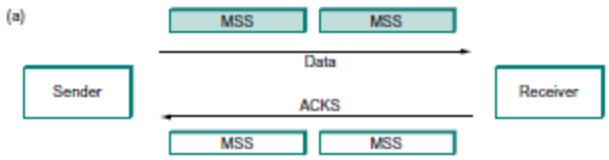
SILLY WINDOW SYNDROME

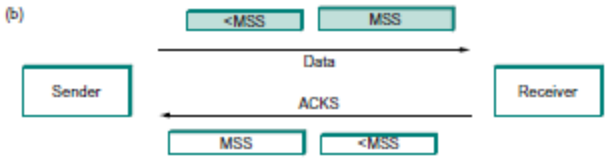
⇨ If the sender has MSS bytes of data to send and the window is open at least that much, then the sender transmits a full segment.
We assume that the sender is accumulating bytes to send, but the window is currently closed.

⇨ We now suppose that an ACK arrives that effectively opens the window enough for the sender to transmit, say MMS/2 bytes. Should the sender transmit a half-full segment or wait for the window to open to a full MSS?

It turns out that the strategy of aggressively taking advantage of any available window leads to a situation known as the silly window syndrome.

If we think of a TCP stream as a conveyer belt with full containers going in one direction and empty containers going in the reverse direction, then MSS-sized segments correspond to large containers and 1-byte segments correspond to very small containers. As long as the sender is sending MSS-sized segments and the receiver ACKs at least one MSS of data at a time, everything is good.





What will happen if the receiver has to reduce the window, so that at some time the sender can't send a full MSS of data? If the sender aggressively fills a smaller-than-MSS empty container as soon as it arrives, then the receiver will ACK that smaller number of bytes, and hence the small container introduced into the system remains in the system indefinitely.
That is, it is immediately filled and emptied at each end and is never coalesced with adjacent containers to create larger containers.

⇨ The silly window syndrome is only a problem when either the **sender transmits a small segment** or the **receiver opens the window a small amount**.

Solution: Nagle's self-clocking algorithm

⇨ If there is data to send but the window is open less than MSS, then we may want to wait some amount of time before sending the available data, but the question is how long?

1) If we wait too long, then we hurt interactive applications like Telnet
2) If we don't wait long enough, then we risk sending a bunch of tiny packets and falling into the silly window syndrome.

⇨ Introduce **timer** and transmit when it expires.

Self-clocking solution

As long as TCP has any data in flight, the sender will eventually receive an ACK. This ACK can be treated like a timer firing, triggering the transmission of more data. Nagle's algorithm provides a simple, unified rule for deciding when to transmit.

```
When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment
  else
    if there is unACKed data in flight
      buffer the new data until an ACK arrives
    else
      send all the new data now
```

- It is always OK to send a full segment if the window allows.
- It is all right to immediately send a small amount of data if there are currently no segments in transit, but if there is anything in flight the sender must wait for an ACK before transmitting the next segment.

2.6 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.

ORIGINAL ALGORITHM

Original algorithm (Jacobson): this algorithm was originally described in the TCP specification.

Basic idea: Keep a running average of the RTT and then compute the timeout as a function of this RTT. Every time TCP sends a data segment, it records the time.

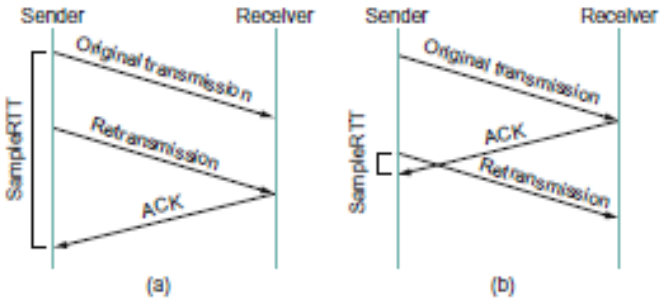
When an ACL for that segment arrives, TCP reads the time again, and then takes the difference between these two times as a SampleRTT. TCP then computes an EstimatedRTT as a weighted average between the previous estimate and this new sample.

$$\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT}$$

The alpha parameter is selected to smoot the EstimatedRTT.

- A small alpha tracks changes in the RTT but is perhaps too heavily influenced by temporary fluctuations.
- A large alpha is more stable but perhaps not quick enough to adapt to real changes.

Problem: ambiguity after retransmission



The problem is that an ACK does not really acknowledge a transmission. It actually acknowledges the receipt of data. In other words, whenever a segment is retransmitted and then an ACK arrives at the sender, it is impossible to determine if this ACK should be associated with the first or the second transmission of the segment for the purpose of measuring the sample RTT.

⇨ It is necessary to know which transmission to associate it with so as to compute an accurate SampleRTT.

1) If the ACK is for the original transmission but it was really for the second, then the SampleRTT is too large. (a)
2) If we assume that the ACK is for the second transmission but it was actually for the first, then the SampleRTT is too small. (b)

KARN/PARTRIDGE ALGORITHM

- The solution proposed is that, whenever TCP retransmits a segment, it stops taking samples of the RTT. It only measures SampleRTT for segments that have been sent only once.
- Second small change: each time TCP retransmits, it sets the next timeout to be twice the last timeout, rather than basing it on the last EstimatedRTT.

Problem: the sample RTT variance is not taken into account. There is no reason to double the timeout if variance is very small.

JACOBSON/KARELS ALGORITHM

In the new approach, the sender measures a new SampleRTT as before. It then folds this new sample into the timeout calculation as follows.

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$
$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference})$$
$$\text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation})$$

where δ is a fraction between 0 and 1. That is, we calculate both the mean RTT and the variation in that mean.

TCP then computes the timeout value as a function of both EstimatedRTT and Deviation as follows:

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

Based on the experience, µ is typically set to 1 and Φ set to 4. So, when the variance is small, Timeout is close to EstimatedRTT. A large variance causes the deviation term to dominate the calculation.

Remark: accurate timeout mechanism is important for congestion control. Congestion may be cause of timeout: fast reaction will aggravate situation.

2.7 Record Boundaries

Since TCP is a byte-stream protocol, the number of bytes written by the sender are not necessarily the same as the number of bytes read by the receiver.
Ex: the application might write 8 bytes, then 2 bytes, then 20 bytes to a TCP connection, while on the receiving side the application reads 5 bytes at a time inside a loop that iterates 6 times.

⇨ TCP does not interject record boundaries between the 8th and 9th bytes, nor between the 10th and 11th bytes.
>< UDP: in which the message that is sent is exactly the same length as the message that is received.

⇨ Inconvenient for applications with discrete messages: they must be able to **add record boundaries**.

- Urgent data feature, as implemented by the URG flag and the UrgPtr field in the TCP header.
  This mechanism was designed to allow the sending application to send out-of-band data to its peer. This means, data that is separate from the normal flow of data. This out-of-band data was identified in the segment using the UrgPtr field and was to be delivered to the receiving process as soon as it arrived, even if that meant delivering it before data with an earlier sequence number.

⇨ This use has developed because, as with the push operation, TCP on the receiving side must inform the application that urgent data has arrived. That is, the urgent data in itself is not important. It is the fact that the sending process can effectively send a signal to the receiver that is important.

- Second mechanism for inserting end-of-record markets into a byte is the **push operation**. This mechanism was designed to allow the sending process to tell TCP that it should send (flush) whatever bytes it had collected to its peer. The push operation can be used to implement record boundaries because the specification says that TCP must send whatever data it has buffered at the source when the application says push, and optionally, TCP at the destination notifies the application whenever an incoming segment has the PUSH flag set.

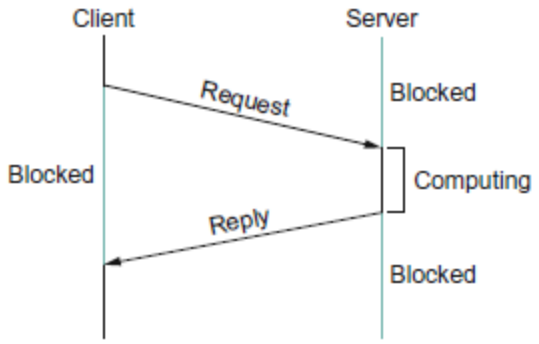
2.8 Alternative Design choices

⇨ The design space for transport protocols is quite large. TCP is by no means the only valid point in that design space.

- **Stream-oriented** protocols like **TCP** and **request/reply protocols** like RPC. We could further divide the stream-oriented protocols into two groups: reliable and unreliable, with the former containing TCP and the latter being more suitable for interactive video applications that would rather drop a frame than incur the delay associated with a retransmission.

TCP	Interactive audio/video?	Database request/reply
Full-duplex	V	Half duplex
Byte-stream	V	Message oriented
Explicit setup/teardown	V	No setup
Window based	Rate based	-
Reliable	Unreliable	-
	RTP/UDP	RPC/UDP

3. Remote Procedure Call



Timeline for RPC
This figure illustrates the basic interaction between the client and the server in message transactions. We have previously seen the request/reply paradigm, also called message-transaction. A client sends a request message to a server, and the server responds with a reply message, with the client blocking (suspending execution) to wait for the reply.

3.1 RPC Fundamentals

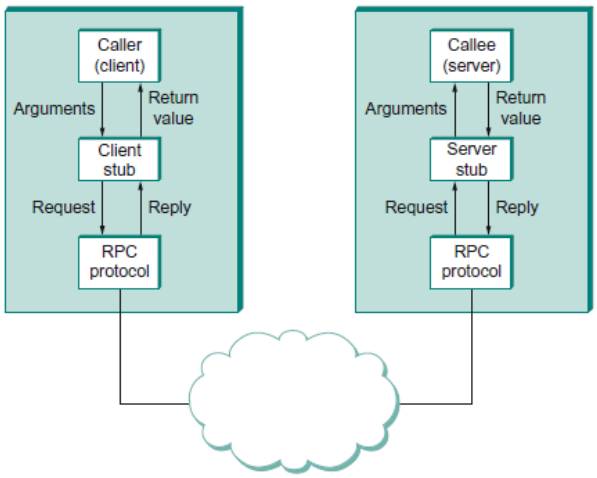
⇨ RPC is a popular mechanism for structuring distributed systems. It is based on the semantics of a local procedure call. The application program makes a call into a procedure without regard for whether it is local or remote and blocks until the call returns. RPC is known as remot method invocation (RMI).

Problems:

- The network between the calling process ant the called process has much more complex properties than the backplane of a computer. For example, it is likely to limit message sizes and has a tendency to lose and reorder messages.
- The computers on which the calling and called processes run may have significantly different architectures and data representation formats.

Two major components:

- A **protocol** that manages the messages sent between the client and the server processes and that deals with the potentially undesirable properties of the underlying network.
- **Programming language** and **compiler support** to package the arguments into a request message on the client machine and then to translated this message back into the arguments on the server machine, and, with the return value.



The client calls a local stub for the procedure, passing it the arguments required by the procedure. This stub hides the fact that the procedure is remote by translating the arguments into a request message and then invoking an RPC protocol to send the request message to the server machine. At the server, the RPC protocol delivers the request message to the server stub (**skeleton**), which translates it into the arguments to the procedure and then calls the local procedure. After the server procedure completes, it returns the answer to the server stub, which packages this return value in a reply message that it hands off to the RPC protocol for transmission back to the client.
The RPC protocol on the client passes this message up to the client stub, which translates it into a return value that it returns to the client program.

IDENTFIIERS IN RPC

- ⇨ Two functions that must be performed by any RPC protocol are
  - Provide a **name space for uniquely identifying** the procedure to be called.
    One of the design choices when identifying things is whether to make this name space **flat** or **hierarchical**. A flat name space would simply assign a unique, unstructured identifier to each procedure, and this number would be carried in a single field in a RPC request message. Plus, the protocol could implement a hierarchical name space, analogous to that used for file pathnames, which requires only that a file's base name be unique within its directory.
  - **Match each reply message** to the **corresponding request message.**
- ⇨ One of the recurrent challenges in RPC is **dealing with unexpected responses**, and we see this with message IDs.
  Ex: a client machine sends a request message with a message ID of 0, then crashes and reboots, and then sends an unrelated request message, also with a message ID of 0. The server may not have been aware that the client crashed and rebooted and, upon seeing a request message with a message ID of 0, acknowledges it and discards it as a duplicate. The client never gets a response to the request.
- ⇨ One way to eliminate this problem is to use a boot ID. A machine's boot ID is a number that is incremented each time the machine reboots. This number is read from nonvolatile storage (e.g., a disk or flash drive), incremented, and written back to the

storage device during the machine's start-up procedure. If a message is received with an old message ID but a new boot Id , it is recognized as a new message. The message ID and boot ID combine to form a unique ID for each transaction.

OVERCOMING NETWORK LIMITATIONS

RPC protocols often perform additional functions.

- **Provide reliable message delivery**. RPC protocol might implement reliability because the underlying protocols do not provide it, or perhaps to recover more quickly or efficiently from failures that otherwise would eventually be repaired by underlying protocols.
  It can implement reliability using acknowledgments and timeouts.

  A complication that RPC must address is that the server may take an arbitrarily long time to produce the result, and worse yet, it may crash before generating the reply. To help the client distinguish between a slow server and a dead server, the RPC's client side can periodically send an "Are you alive?" message to the server, and the server side responses with an ACK. Alternatively, the server could send "I am still alive" messages to the client without the client having first solicited them.

- **Support large message sizes** through **fragmentation** and reassembly
  The two reasons why an RPC protocol might implement message fragmentation and reassembly are that it is not provided by the underlying protocol stack or that it can be implement more efficiently by the RPC protocol.
  Let's consider the case where RPC is implemented on top of UDP/IP and relies on IP for fragmentation and reassembly. If even one fragment of a message fails to arrive within a certain amount of time, IP discards the fragments that did arrive and the message is effectively lost.
    - ⇨ In the case of an RPC protocol that implements its own fragmentation and reassembly and aggressively ACKs or NACKs individual fragments, lost fragments would be more quickly detected and retransmitted, and only the lost fragments would be retransmitted, not the whole message.

SYNCHRONOUS VERSUS ASYNCHRONOUS PROTOCOLS

- At the asynchronous end of the spectrum, the application knows absolutely nothing when send returns. Not only does it not know if the message was received by its peer, but it doesn't even know for sure that the message has successfully left the local machine.
- At the synchronous end of the spectrum, the send operation typically returns a reply message. That is, the application not only knows that the message it sent was received by its peer, but it also knows that the peer has returned an answer.

- ⇨ Synchronous protocols implement the request/reply abstraction, while asynchronous protocols ae used if the sender wants to be able to transmit many messages without having to wait for a response.
- ⇨ RPC protocols are obviously synchronous protocols.

3.2 RPC Implementations

- ⇨ Example implementations of RPC protocols: SunRPC, DCE (= Distributed Computing Environment).

1) **SunRPC**
SunRPC can be implemented over several different transport protocols. The details of SunRPC's semantics depend on the underlying transport protocol. It does not implement its own reliability, so it is only reliable if the underlying transport is reliable.

2) DCE-RPC
This is the protocol at the core of the DCE system and was the basis of the RPC mechanism underlying Microsoft's DCOM and ActiveX.
DCE, like SunRPC, can be implemented on top of several transport protocols including UDP and TCP. It is also similar to SunRPC in that it defines a two-level addressing scheme. The transport protocol demultiplexes to the correct server, DCE-RPC dispatches to a particular procedure exported by that sever, and clients consult an "endpoint mapping service".
Each request/reply transaction in DCE-RPC takes place in the context of an activity. An activity is a logical request/reply channel between a pair of participants. At any given time, there can be only one message transaction active on a given channel.

Another design choice made in DCE that differs from SunRPC is the support of fragmentation and reassembly in the RPC protocol.

4. Transport for real-time applications (RTP)

- ⇨ Real-time applications place some specific demands on the transport protocol that are not well met by the protocols we previously discussed in this chapter.

Ex: Internet telephony, or voice over IP, is a classic example of a real-time application, because you can't easily carry on a conversation with someone if it takes more than a fraction of a second to get a response.

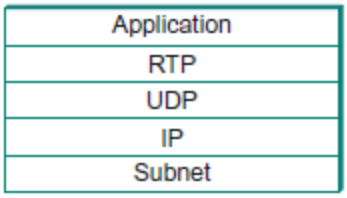
Multimedia applications are sometimes divided into **two classes**

- **Interactive** applications
- **Streaming** applications. Streaming applications lack human-to-human interaction, so they place somewhat less stringent real-time requirements on the underlying protocols. However, timeliness is still important. Ex: if you want a video to start playing so after pushing play. Late packets will either cause it to stall or create some sort of visual degradation.

⇨ While streaming applications are not strictly real time, thy still have enough in common with interactive multimedia applications to warrant consideration of a common protocol for both types of applications.

Much of RTP actually derives from protocol functionality that was originally embedded in the application itself. RTP can run over many lower-layer protocol, but still commonly runs over UDO. That leads to the protocol stack shown in the figure below.

We are running a transport protocol over a transport protocol. There is actually no rule against that, and in fact it makes a lot of sense, since UDP provides such a minimal level of functionality, and the basic demultiplexing based on port numbers happens to be just what RTP needs as a starting point. So, rather than recreate port numbers in RTP, RTP outsources the demultiplexing function to UDP.



- Independent of underlying transport
- Typically UDP
- Supports IP multicasting
- TCP not really useful, unless the behavior of UDP is too bad.

4.1 Requirements

- It should **allow similar applications to interoperate with each other**. Ex: it should be possible for two independently implemented audioconferencing applications to talk to each other. Since there are quite a few different coding schemes for voice, each with its own trade-offs among quality, bandwidth requirements, and

computational cost, it would probably be a bad idea to decree that only one such scheme can be used.
The protocol should provide a way that a sender can tell a receiver which coding scheme it wants to use, and possible negotiate until a scheme that is available to both parties is identified.

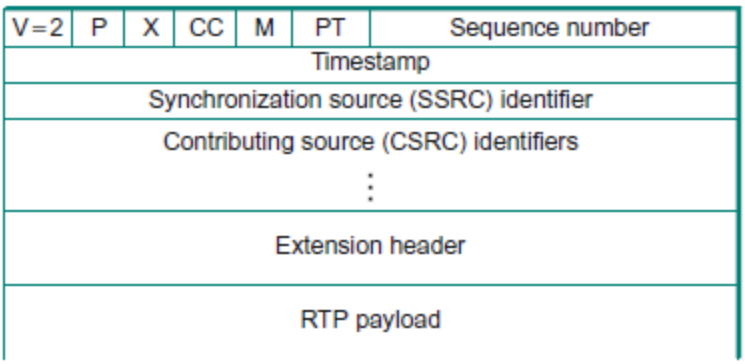
- Enable the recipient of a data stream to **determine the timing relationship** among the received data. Some sort of timestamping of the data will be necessary to enable the receiver to play it back at the appropriate time.
- **Indication of packet loss.** An application with tight latency bounds generally cannot use a reliable transport like TCP because retransmission of data to correct for loss would probably cause the packet to arrive too late to be useful. So, the application must be able to deal with missing packets, and the first step in dealing with them is noticing that they are in face missing.
- **Concept of frame boundary indication.**
- Identifying senders in a more user-friendly way than an IP address.
- **Respond to congestion**
- It should make reasonably **efficient use of bandwidth**. We don't want to introduce a lot of extra bits that need to be sent with every packet in the form of a long header. The reason for this is that audio packets, which are one of the most common types of multimedia data, tend to be small, so as to reduce the time it takes to fill them with samples.

But: no flow control, no error control, no ACK, no retransmit.

4.2 RTP Design

⇨ The RTP standard defines a pair of protocols, **RTP** and the **Real-Time transport control protocol**. The former is used for the exchange of multimedia data, while the latter is used to periodically send control information associated with a certain data flow.

RTP provides a flexible mechanism by which new applications can be developed without repeatedly revising the RP protocol itself. For each class of application, RTP defines a **profile** and one or more **formats**. The profile provides a range of information that ensures a common understanding of the fields in the RTP header for that application class.



- The first 12 bytes are always present, whereas the contributing source identifiers are only used in certain circumstances.
- After this header there may be optional header extensions.
- The header is followed by the RP payload, the format of which is determined by the application.
    - ⇨ The intention of this header is that it contains only the fields that are likely to be used by many different applications, since anything that is very specific to a single application would be more efficiently carried in the RTP payload for that application only.

- The first 2 bits are a **version identifier**, which contains the value 2 in the RTP version.
- The next bit is the **padding** (p) bit, which is set in circumstances in which the RTP payload has been padded for some reason. RTP data might be padded to fill up a block of a certain size as required by an encryption algorithm.
- The extension (x) bit is used to indicate the presence of an extension header, which would be defined for a specific application and follow the main header.
- The X bit is followed by a 4-bit field that counts the number of contributing sources, if any are included in the header.
- The 7-bit **payload type field** indicates what type of multimedia data is carried in this packet. One possible use of this field would be to enable an application to switch from one coding scheme to another based on information about resource availability in the network or feedback on application quality. The payload type is generally not used as a demultiplexing key to direct data to different applications.
- The **sequence number** is used to enable the receiver of an RTP stream to detect missing and disordered packets. The

sender simply increments the value by one for each transmitted packet.

- The function of the timestamp field is to enable the receiver to play back samples at the appropriate intervals and to enable different media streams to be synchronized. Since different applications may require different granularities of timing, RTP itself does not specify the units in which time is measured. Instead, the timestamp is just a counter of ticks, where time between ticks is dependent on the encoding in use. This timestamp allows correct synchronization of each media stream: audio and video streams are synchronized separately.
- The **synchronization source (SSRC)** is a 32-bit number that uniquely identifies a single source of an RTP stream.
- The **contributing source (CSRC)** is used only when a number of RTP streams pass through a mixer. A mixer can be used to reduce the bandwidth requirements for a conference by receiving data from many sources and sending it as a single stream.

⇨ **Data source**: a single node can have multiple sources.

4.3 Control protocol

⇨ Control stream that provides <u>three main functions</u>

1) **Feedback** on the performance of the application and the network. This function may be useful for detecting and responding to congestion. Some applications are able to operate at difference rates and may use performance data to decide to use a more aggressive compression scheme to reduce congestion, or to send a high-quality stream when there is little congestion. Performance feedback can also be useful in diagnosing network problems.
2) A way to **correlate** and **synchronize** different media streams that have come from the same sender
3) A way to convey the **identity of a sender** for display on a user interface.

RTCP defines a number of different packet types, including:

4) Sender reports, which enable active senders to a session to report transmission and reception statistics.
5) Receiver reports, which receivers who are not senders use to report reception statistics.
6) Source descriptions, which carry CNAMEs and other sender description information
7) Application-specific control packets.
8) RTCP traffic must be limited: <5% of RTP traffic. To accomplish this goal, the participants should know how much data bandwidth is likely to be in use and the number of participants.

⇨ The challenge in designing a transport protocol is to make it general enough to meet the widely varying needs of many different applications without making the protocol itself impossible to implement.
⇨ RTP has proven very successful in this regard, forming the basis for the majority of real-time multimedia communications over the Internet today.

CHAPTER 6: CONGESTION CONTROL AND RESOURCE ALLOCATION

1. Introduction

⇨ How to **effectively and fairly allocate resources** among a collection of competing users.
The resources being shared include the **bandwidth** of the links and the **buffers** on the **routers** or **switches** where packets are queued awaiting transmission.
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.**

Congestion control and resource allocation are two sides of the same coin.

1) If the network takes an active role in allocating resources, for instance, scheduling which virtual circuit gets to use a given physical link during a certain period of time, then congestion may be avoided, thereby making congestion control unnecessary.
2) you can always let packet sources send as much data as they want and then recover from congestion should it occur. This is an easier approach, but it can be disruptive because many packets may be discarded by the network before congestion can be controlled.

2. Issues in resource allocation

Congestion control and resource allocation are not isolated to one single level of a protocol hierarchy, therefore, it makes these issues quite complex.

Resource allocation is partially implemented in the routers, switches, and links inside the network and partially in the transport protocol running on the end hosts.

- **Resource allocation:** process by which network elements try to meet the competing demands that applications have for network resources, primarily link bandwidth and buffer space in routers or switches.

⇨ It will often not be possible to meet all the demands, meaning some users or applications may receive fewer network resources than they want.

- **Congestion control:** efforts made by network nodes to prevent or respond to overload conditions. Since congestion is generally bad for everyone, the first order of business is making congestion subside, or preventing it in the first place.
  Briefly said, too many packets are contending for the same link. The queue overflows and packets get dropped.

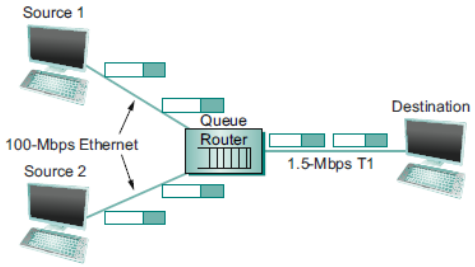
  /!\ flow control ≠ congestion control

- **Flow control** involves keeping a fast sender from overrunning a slow receiver.
- **Congestion control** is intended to keep a set of senders from sending too much data into the network because of lack of resources at some point.

2.1 Network Model

PACKET-SWITCHED NETWORK

⇨ We consider resource allocation in a packet-switched network, consisting of multiple links and switches.
In such an environment, a given source may have more than enough capacity on the immediate outgoing link to send a packet, but somewhere in the middle of a network its packets encounter a link that is being used by many different traffic sources.



Two high-speed links are feeding a low-speed link. This is in contrast to shared-access networks like Ethernet and wireless networks, where the source can directly observe the traffic on the network and decide accordingly whether or not to spend a packet.

CONNECTIONLESS FLOWS

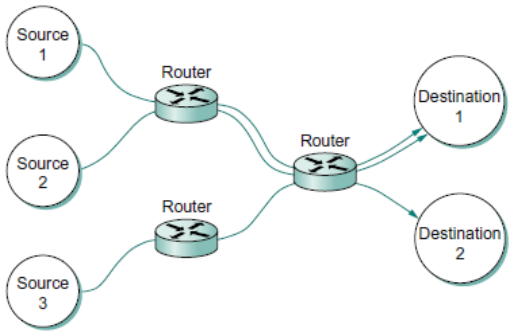
⇨ We assume that most of the networks are essentially connectionless, with any connection-oriented service implemented in the transport protocol that is running on the end hosts. This is the model of the Internet, where IP provides a connectionless datagram delivery service and TCP implements an end-to-end connection abstraction. In this case, all packets are routed independently. TCP implements a connection abstraction, but only in the end hosts.

⇨ How to allocate resources if there is no connection setup?

Solution: connectionless flows

The assumption that all datagrams are completely independent in a connectionless network is too strong. The datagrams are certainly switched independently, but it is usually the case that a stream of datagrams between a particular pair of hosts flows through a particular set of routers. This idea of a **flow**, a sequence of packets sent between a source/destination pair and following the same route through the network, is an important abstraction in the context of resource allocation.
Flows can be defined at different granularities:

- **Host-to-host**
- **Process-to-process:** in this case, a flow is essentially the same as a channel.



This figure illustrates several flows passing through a series of routers.
Because multiple related packets flow through each router, it sometimes makes sense to maintain some state information for each flow, information that can be used to make resource allocation decisions about the packets that belong to the flow. **= soft state**

- No explicit signaling required
- Correct operation of network does not depend on soft state.

SERVICE MODEL

With best-effort service, all packets are given essentially equal treatment, with end hosts given no opportunity to ask the network that some packest or flows be given certain gurantees or preferential service.

Defining a service model that supports some kind of preferred service or guarantee, for instance, guaranteeing the bandwidth needed for a video stream, relates to qualities of service.

2.2 Taxonomy

Three dimensions along which resource allocation mechanisms can be characterized.

ROUTER-CENTRIC VERSIS HOST-CENTRIC

Resource allocation mechanisms can be classified into **two broad groups**.

- Those that address the problem from inside the network.
- Those that address the problem from the edges of the network.

⇨ Since it is the case that both the routers inside the network and the hosts at the edges of the network participate in resource allocation, the real issue is where the majority of the burden falls.

- In a **router-centric** design, each router takes responsibility for deciding when packets are forwarded and selecting which packets are to be dropped, as well as for informing the hosts that are generating the network traffic how many packets they are allowed to send.
- In a **host-centric design**, the end hosts observe the network conditions (e.g., how many packets they are successfully getting through the network) and adjust their behavior accordingly.

⇨ Router-centric and host-centric are not mutually exclusive.

RESERVATION-BASED VERSUS FEEDBACK-BASED

Resource allocation mechanisms may also be sometimes classified according to whether they use **reservations** or **feedback**.

- **Reservation-based system**: some entity asks the network for a certain amount of capacity to be allocated for a flow. Each router then allocates enough resources to satisfy this request. If the request cannot be satisfied at some router, because doings so would overcommit its resources, then the router rejects the reservation.
  /!\ This system always implies a router-centric resource allocation mechanism. This is because each router is responsible for keeping track of how much of its capacity is currently available and deciding whether new reservations can be admitted. Routers may also have to make sure each host lives within the reservation it made.
- **Feedback**-based approach: the end hosts begin sending data without first reserving any capacity and then adjust

their sending rate according to the feedback they receive. This feedback can be either:

- **Explicit**: a congested router sends a "please slow down" message to the host
- **Implicit**: the end host adjusts its sending rate according to the externally observable behavior of the network, such as packet losses.

WINDOW BASED VERSUS RATE BASED

⇨ Both flow-control and resource allocation mechanisms need a way to express, to the sender, how much data is allowed to transmit. There are two general ways of doing this.

- With a **window**: the window corresponds to how much buffer space the receiver has, and it limits how much data the sender can transmit. That is, it supports flow control. Such a mechanism, window advertisement, can be used within the network to reserve buffer space. (**bytes**)
- With a **rate**: how many bits per second the receiver or network is able to absorb. Rate-based control makes sense for many multimedia applications, which tend to generate data at some average rate and which need at least some minimum throughput to be useful. **(bps)**

SUMMARY OF RESOURCE ALLOCATION TAXONOMY

- **TCP**: A best-effort service model usually implies that **feedback** is being used, since such a model does not allow users to reserve network capacity. This in turn means that most of the responsibility for congestion control falls to the end **hosts**, perhaps with some assistance from the routers. In practice, such networks use **window-based information**.
- **QoS**: this service model probably implies some form of **reservation**. Support for these reservations is likely to requires significant **router involvement**, such as queuing packets differently depending on the level of reserved resources they require. Besides, it is natural to express such reservations in terms of **rate**, since windows ore only indirectly related to how much bandwidth a user needs from the network.

2.3 Evaluation Criteria

- Effectiveness
- Fairness

EFFECTIVE RESOURCE ALLOCATION

We want:

- **As much throughput as possible**. One sure way for a resource allocation algorithm to increase throughput is to allow as many packets into the network as possible. → drives the utilization of all the links up to 100%. We would do this to avoid the possibility of a link becoming idle because an idle link necessarily hurts throughput.
- **As little delay as** possible

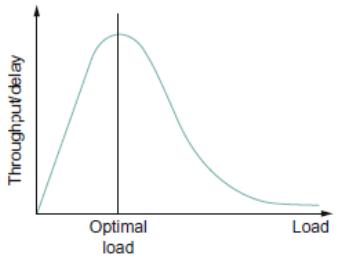
But: increasing the number of packets in the network also increases the length of the queues at each router. Longer ques, in turn, means packets are delayed longer in the network.

⇨ **Conflict**

Solution: ratio of throughput to delay

Power = Throughput/Delay

The objective is to maximize this ratio.



Ideally, the resource allocation mechanism would operate at the peak of this curve. To the left peak, the mechanism is being too conservative. That is, it is not allowing enough packets to be sent to keep the links busy.
To the right of the peak, so many packets are being allowed into the network that increases in delay due to queuing are starting to dominate any small gains in throughput.

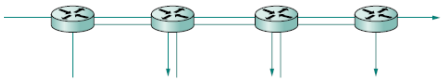
⇨ Many congestion-control schemes are able to control load in only very crude ways. That is, it is simply not possible to turn the "know" a little and allow only a small number of additional packets into the network.
⇨ We should try to tend to a **stable** mechanism >< if a mechanism is not stable, the network may experience **congestion collapse**.

FAIR RESOURCE ALLOCATION

⇨ What does fairness mean? What is fair resource allocation?

- Reservation-based resource allocation leads to unfairness. Indeed, with such a scheme, we might use reservations to enable a video stream to receive 1Mbpcs across some link while a file transfer receives only 10kbps over the same link.

- When several flows share a particular link, we would like for each flow to receive an equal share of the bandwidth. This presumes that a fair share of bandwidth means an equal share of bandwidth.
- Equal shares are not necessarily fair shares. Should we also consider the length of the paths being compared?



Assuming that fair implies equal and that all paths are of equal length, networking research Raj Jain proposed a metric that can be used to quantify the fairness of a congestion control mechanisms.

- Given set of flow throughputs ($x_1, x_2, x_3, \ldots, x_n$)
- The fairness index always results in a number between 0 and 1, with 1 representing greatest fairness.

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$$

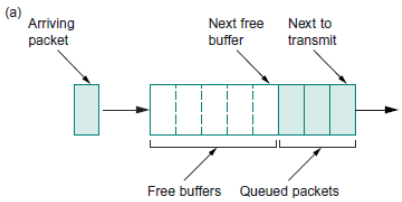
3. Queuing disciplines

⇨ Each router must implement some queuing discipline that governs how packets are buffered while waiting to be transmitted.

3.1 FIFO

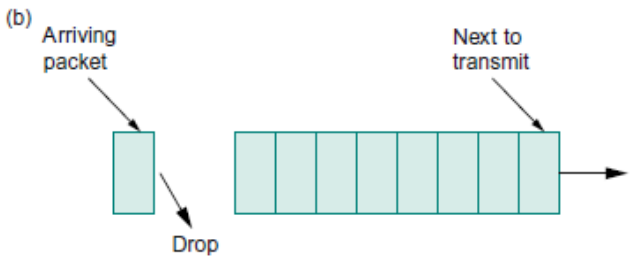
= first-come, first-served (FCFS).

⇨ The first packet that arrives at a router is the first packet to be transmitted.



This figure shows a FIFO with "slots" to hold up the eight packets. Given that the amount of buffer space at each router is finite, if a packet arrives and the queue (buffer space) is full, then the router discards that packet.

The second figure illustrates a **tail drop mechanism**. The buffer is full and therefore the router discards that packet. This is done without regard to which flow the packet belongs to or how important the packet is. Hence the name: tail drop, since packets that arrive at the tail end of the FIFO are dropped.



- **FIFO** is a **scheduling discipline**. It determines the order in which packets are transmitted.
- **Tail drop** is a **drop policy**. It determines which packets get dropped.
  - ⇨ FIFO and tail drop are considered as a bundle: the vanilla queuing implementation. Unfortunately, the bundle is often referred to simply as **FIFO queuing**.
  - ⇨ **Simple**: FIFO with tail drop is the most widely used in Internet routers. This approach pushes all responsibility for congestion control and resource allocation out to the edges of the network. So, the prevalent form of congestion control in the Internet currently assumes no help from the routers. Indeed, TCP takes responsibility for detecting and responding to congestion.
  - ⇨ Although it tends to be quite simple, this system is **unfair**. It is an ill-behaving source that may consume a lot of resources. For instance, in case a sender has a bad behavior, leading to a bad service.

Variation: priority queuing. The idea is to mark each packet with a priority. The mark could for example be carried in the IP header. The routers then implement multiple FIFO queues, one for each priority class. The router always transmits packets out of the highest-priority queue if that queue is nonempty before moving onto the next priority queue. Within each priority, packets are still managed in a FIFO manner.
Priority queuing is used in the Internet to protect the most important packets. Typically, the routing updates that are necessary to stabilize the routing tables after a topology change. Often there is a special queue for such packets, which can be identified by the Differentiated Services Code Point in the IP header.

Problem:

- **Unfair**: The high-priority queue can starve out all the other queues. That is, as long as there is at least one high-priority packet in the high-priority queue, lower–priority queues do not get served.
- For this to be viable, there need to be hard limits on how much high-priority traffic is inserted in the queue. We can't allow users to set their own packets to high priority in an uncontrolled way.

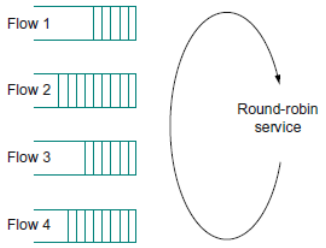
3.2 Fair queuing

⇨ The main problem with FIFO queuing is that it **does not discriminate** between different traffic sources, or, it does not separate packets according to the flow to which they belong. This is a problem at two different levels.

- At one level, it is not clear that any congestion-control algorithm implemented entirely at the source will be able to adequately control congestion with so little help from the routers.
- Since the entire congestion-control mechanism is implemented at the sources and FIFO queuing does not provide a means to police how well the sources adhere to this mechanism, it is possible for an ill-behaved source to capture an arbitrarily large fraction of the network capacity.

⇨ **Fair queuing (FQ) has** been proposed to address this problem.

The idea of FQ is to maintain a separate queue for each flow currently being handled by the router. The router then services these queues in a sort of round-robin.



When a flow sends packets too quickly, then its queue fills up. When a queue reaches a particular length, additional packets belonging to that flows queue are discarded.

⇨ In this way, a given source cannot arbitrarily increase its share of the network's capacity at the expense of the other flows.

Problem: packets are not of the same length. The flow with the large packets would be favored. Therefore, the simple round robin service is not fair.

Solution: take packet length into account. → bit-by-bit round-robin algorithm. What we actually want is bit-by-bit round-robin, where the router transmits a bit from flow 1, then a bit from flow 2, etc. but this is clearly not feasible to interleave the bits from different packets. There, the FQ mechanism simulates this behavior by first determining when a given packet would finish

being transmitted if it were being send using bit-by-bit round-robin and then using this finish time to sequence the packets for transmission.

- Work conserving
- Starvation-free
- Each flow gets guaranteed minimum share 1/n, perhaps more.
- On fully loaded link with n flows sending data, no flow captures more than its fair share (1/n).

ALGORITHM

⇨ Let's consider the behavior of a single flow and imagine a clock that ticks once each time one bit is transmitted from all of the active flows.

- P_i: length of packet i.
- S_i: time when the router starts to transmit packet i.
- F_i: time when the router finishes transmitting packet i.
- A_i: arrival time of packet i.

⇨ $F_i = S_i + P_i$.

⇨ When do we start transmitting packet i? the answer depends on whether packet I arrived before or after the router finished transmitting packet i-1 from this flow.

- If it was before, then logically the first bit of packet I is transmitted immediately after the last bit of packet i-1.
- It is also possible that the router finished transmitting packet i-1 long before I arrived, meaning that there was a period of time during which the queue for this flow was empty, so the round-robin mechanism could not transmit any packets from this flow.

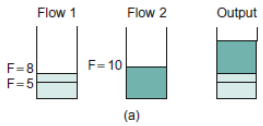
$$S_i = \max(F_{i-1}, A_i).$$

$$F_i = \max(F_{i-1}, A_i) + P_i$$

⇨ New situation, in which there is more than one flow, and we find that there is a catch to determining A_i. We can't just read the wall clock when the packet arrives. As stated before, we want time to advance by one tick each time all the active flows get one bit of service under bit-by-bit round-robin, so we need a clock that advances more slowly when there are more flows. The clock

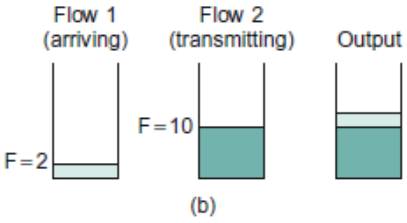
must advance by one tick when n bits are transmitted if there are n active flows. → This clock will be used to calculate A_i.

Then, for every flow, we calculate Fi for each packet that arrives using the previous formula. We then treat all the Fi as timestamps, and the next packet to transmit is always thhe packet that has the lowest timestamp, the packet that, based on the above reasoning, should finish transmission before all others.



In the second figure, the router has already begun to send a packet from flow 2 when the packet from flow 1 arrives. Though the packet arriving on flow 1 would have finished before flow 2, if we had been using perfect bit-by-bit faire queuing, the implementation does not preempt the flow 2 packet.

This figure shows the queues for two flows. The algorithm selects both packets from flow 1 to be transmitted before the packet in the flow 2 queue, because of their earlier finishing times.



- The link is never idle as long as there is at least one packet in the queue. Any queuing scheme with this characteristic is said to be **work conserving**. One effect of being work conserving is that if I am sharing a link with a lot of flows that are not sending any data, then I can use the full link capacity for my flow. As soon as the other flows start sending, they will start to use their share and the capacity available to my flow will drop.
  - ⇨ Because FQ is work conserving, any bandwidth that is not used by one flows is automatically available to other flows.
- If the link is fully loaded and there are n flows sending data, I cannot use more than 1/nth of the link bandwidth. If I try to send more than that, my packets will be assigned increasingly large timestamps, causing them to sit in the queue longer awaiting transmission.

<u>Variation</u>: weighted fair queuing (WFQ)

This variation allows a weight to be assigned to each flow (queue). This weight logically specifies how many bits to transmit each time the router services that queue, which effectively controls the percentage of the link's bandwidth that that flow will get. Simple FQ gives each queue a weight of 1, which means that logically only 1 bit is transmitted from each queue each time around.

- Assign specific weights
  - Flow 1 gets weight 2
  - Flow 2 gets weight 1
  - Flow 3 gets 3
- Assuming that each queue always contains a packet waiting to be transmitted

- Flow 1 will get 1/3 of the available bandwidth
- Flow 2 will get 1/6
- Flow 3 will get ½

Class-based weighted Fair queuing

- Implements FQ in terms of classes of traffic (not individual flows)
- Assign weights to classes of traffic

Low latency queuing (LLQ)

- Add one priority queue with limited bandwidth to CBWFQ
- Ex: voice gets strict priority but cannot exceed bandwidth limits

4. TCP Congestion control

Originally, the Internet didn't have congestion control. Hosts could send packets as fast as advertised window would allow. As mentioned previously, congestion results in dropped packets, causing timeouts at sender. As for retransmission after timeout, it results in even more congestion.

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.

By using ACKs to pace the transmission of packets, TCP is said to be **self-clocking**.

Challenge

- Determining the available capacity in the first place (can be anything between 10 Kbps and 10Gbps)
- Adjusting to changes in the available capacity (capacity fluctuates over time

4.1 Additive increase/Multiplicative Decrease (AIMD)

TCP maintains a new state variable for each connection: **congestionWindow**. This variable 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 maxium number of bytes of unacknowledged dat alled is now the minimum of the congestion window and the advertised window.

$$\text{MaxWindow} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$

$$\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked}).$$

⇨ How does TCP come to lean an appropriate value for CongestionWindow?
Unlike the AdvertisedWindow, which is sent by the receiving side for the connection, there is no one to send a suitable CongestionWindow to the sending side of TCP. 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.

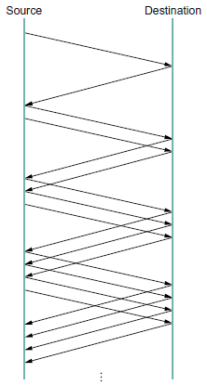
⇨ Mechanism of **additive increase/multiplicative decrease (AMD**).

⇨ How does the source determine that the network is congested and that I should decrease the congestion window?

The main reason why packets are not delivered, and a timeout results, is that a packet was dropped due to congestion. It is rare that a packet is dropped because of an error during transmission. Therefore, TCP interprets **timeouts** as a sign of congestion and reduces the rate at which it is transmitting. Specifically, each time a timeout occurs, the source sets CongestionWindow to half of its previous value.

Although CongestionWindow is defined in terms of bytes, it is easier to understand multiplicative decrease if we think in terms of whole packets.

We also need to be able to increase the congestion window to take advantage of newly available capacity in the network. This is the “additive increase” part of AIMD, and it works as follows. Every time the source successfully sends a congestion Window’s worth of packets that is, each packet sent out during the last round-trip time has been ACKed, it adds the equivalent of 1 packet to CongestionWindow.



The congestion window is incremented as follows each time an ACK arrives:

$$\text{Increment} = \text{MSS} \times (\text{MSS}/\text{CongestionWindow})$$
$$\text{CongestionWindow} += \text{Increment}$$

Rather than increment CongestionWindow by an entire MSS bytes each RTT, we increment it by a fraction of MSS every time an ACK is received.

This pattern of continually increasing and decreasing the congestion window continues throughout the lifetime of the connection. If we plot the current value of CongestionWindow as a function of time, we get a saw tooth pattern.

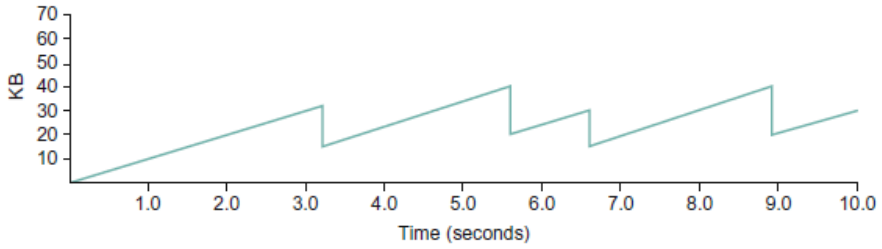
The source is willing to reduce its congestion window at a much faster rate than it is willing to increase its congestion window. In contrast to an additive increase/additive decrease strategy in which the window would be increase by 1 packet when an ACK arrives and decreased by 1 when a timeout occurs.

One reason to decrease the window aggressively and increase it conservatively is that the consequences of having too large a window are much worse than those of it being too small. For instance, when the window is too large, packets that are dropped will be retransmitted, making congestion even worse. Thus, it is important to get out of this state quickly.

Finally, since a timeout is an indication of congestion that triggers multiplicative decrease, TCP needs the most accurate timeout mechanism it can afford.

Two things to remember

- Timeouts are set as a function of both the average RTT and the standard deviation in that average
- Due to the cost of measuring each transmission with an accurate clock, TCP only samples the round-trip time once per RTT using a coarse-grained clock.



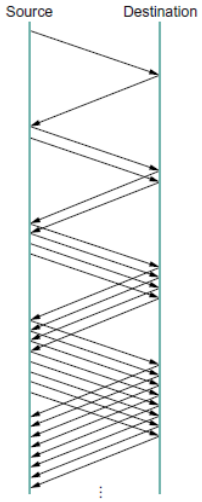
4.2 Slow start

⇨ Additive increase mechanism is the right approach to use when the soruce is operatingclosetotheavailable capacity of the

network, but it takes too long to ramp up a connection when it is starting from scratch.

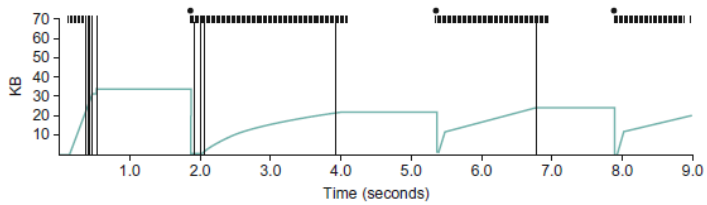
Slow start is used to increase the congestion window rapidly from a cold start. It increases the congestion window exponentially, rather than linearly.
The source starts out by setting CongestionWindow to one packet. When the ACK for this packet arrives, TCP adds 1 to CongestionWindow and then sends two packets. Upon receiving the corresponding two ACKs, TCO increments CongestionWindow by 2, one for each ACK, and next sends four packets. The end result is that TCP effectively doubles the number of packets it has in transit every RTT.

⇨ **Exponential growth**, but slower than all at once



There are two different situations in which slow start runs.

- At the very beginning of a connection, at which time the source has no idea how many packets it is going to be able to have in transit at a given time.
  In this situation, slow start continues to double CongestionWindow each RTT until there is a loss, at which time a timeout causes multiplicative decrease to divide CongestionWindow by 2.
- When the connection goes dead while waiting for a timeout to occur. Recall how TCP's sliding window algorithm works, when a packet is lost, the source eventually reaches a point where it has sent as much data as the advertised window allows, and so it blocks while waiting for an ACK that will not arrive. A timeout may eventually happen but by this time there are no packets in transit, meaning that the source will receive n ACKs to clock the transmission of new packets.



How does TCP's congestionWindow increase and decrease over time? This figure serves to illustrate the interplay of slow start and additive increase/multiplicative decrease.

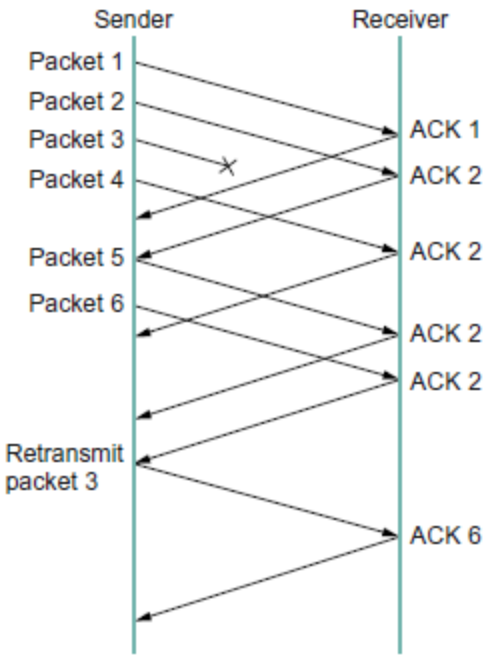
- We notice a rapid increase in the congestion window at the beginning of the connection. This corresponds to the initial slow start phase. The slow start phase continues until several packets are lost at about 0.4 seconds into the connection, at which time CongestionWindow flattens out at about 34 kB.
- The reason why the congestion window flattens is that there are no ACKs arriving, due to the fact that several packets were lost. In fact, no new packets are sent during this time, as denoted by the lack of hash marks at the top of the graph.
- A timeout eventually happens at approximately 2 seconds, at which time the congestion window is divided by 2, and CongestionThreshold is set to this value.
- Between 2 and 4 seconds, additive increase.
- At about 4 seconds, CongestionWindow flattens out, again due to a lost packet.
- At about 5.5 seconds
    - A timeout happens, causing the congestion window to be divided by 2, dropping it from approximately 22KB to approximately 11KB, and CongestionThreshold is set to this amount.
    - CongestionWindow is reset to one packet, as the sender enters slow start.
    - Slow start causes CongestionWindow to grow exponentially until it reaches CongestionThreshold
    - CongestionWindow then grows linearly.
- The same pattern is repeated at around 8 seconds, when another timeout occurs.

New problems:

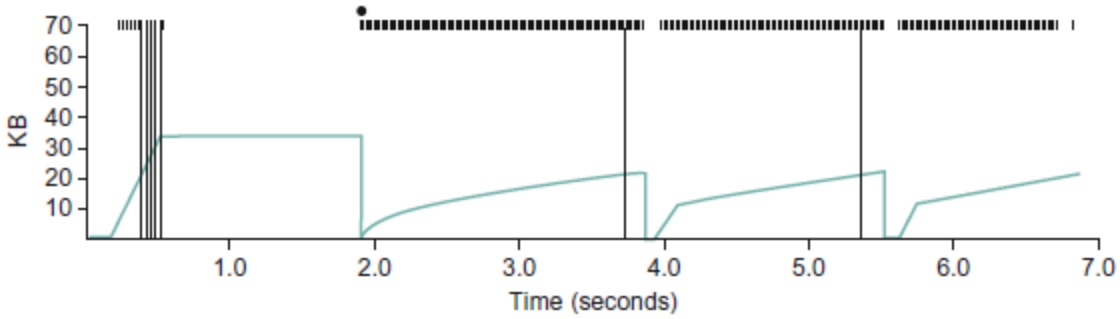
- Lose up to half a CongestionWindow's worth of data
- Long idle period.

4.3 Fast Retransmit and Fast recovery

Problem: The coarse-grained implementation of TCP timeouts led to long periods of time during which the connection went dead while waiting for at time to expire. For this reason, a new mechanism called **fast retransmit** was added to TCP. 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.



Idea: every time a data packet arrives at the receiving side, the receiver responds with an acknowledgement, even if this sequence number has already been acknowledged. Thus, when a packet arrives out of order, when TCP cannot yet acknowledge the data the packet contains because earlier data has not yet arrived, TCP resends the same acknowledgment it sent the last time. This second transmission of the same acknowledgment is called a **duplicate ACK**. When the sending side sees a duplicate ACK, it knows that the other side must have received a packet out of order, which suggest that an earlier packet might have been lost. Since it is also possible that the earlier packet has only been delayed rather than lost, the sender waits until it sees some number of duplicate ACKs and then retransmits the missing packet.
In practice TCP waits until it has seen three duplicate ACKs before retransmitting the packet.
The figure above illustrates how duplicate ACKs lead to a fast retransmit. We hereby see that the destination receives packets 1 and 2, but packet 3 is lost in the network. Thus, the destination will send a duplicate ACK for packet 2 when packet 4 arrives, again when packet 5 arrives, and so on. When the sender sees he third duplicate ACK for packet 2, the one sent because the receiver had gotten packet 6, it retransmits packet 3.



The long periods during which the congestion window stays flat and no packets are sent has been eliminated. In general, this technique is able to eliminate about half of the coarse-grained timeouts on a typical TCP connection, resulting in roughly a 20% improvement in the throughput over what could otherwise have been achieved.

The fast retransmit strategy does not eliminate all coarse-grained timeouts. This is because for a small window size, there will not be enough packets in transit to cause enough duplicate ACKs to be delivered.

There is one last improvement we can make. When the fast retransmit mechanism signals congestion, rather than drop the congestion window all the way back to one packet and run slow start, it is possible to use the ACKs that are still in the pipe to clock the sending of packets. **= fast recovery**

5. Congestion-avoidance mechanisms

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

⇨ TCP needs to create losses to find the available bandwidth of the connection.
= congestion control

>< predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded.
= congestion avoidance

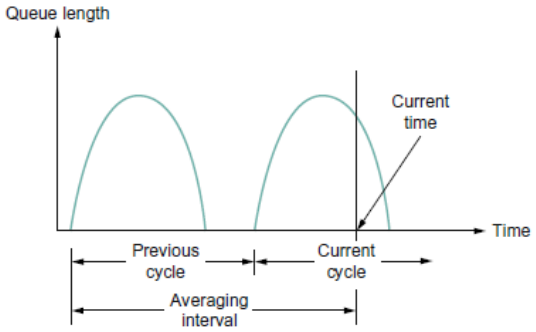
⇨ Three mechanisms

Two possibilities

- **Router-centric**
  - **Explicit** feedback: DECbit
  - **Implicit** feedback: RED
- **Host-centric**: TCP Vegas

5.1 DECbit

Idea: 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. 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.

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

Turning our attention to the host half of the mechanism, the source records how many of its packets resulted in some router setting the congestion bit. In particular, the source maintains a congestion wind, 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.

The value of 50% was chose as the threshold based on analysis that showed it to correspond to the peak of the power curve.

5.2 Random Early Detection (RED)

⇨ 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 differs from DECbit scheme in **two major ways**.

- 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. Therefore, the source is 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.
  In a nutshell, 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.

  RED could easily be adapted to work with an explicit feedback scheme simply by marking a packet instead of dropping it.

- In the details of how RED decides when to drop a packet and what packet it decides to drop, RED and DECbit also differ.
  Ex : consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we could decide to drop each arriving packet with some drop probability whenever the queue length exceeds some drop level. = **early random drop**

Algorithm:

- First, RED computes an **average queue length** using a weighted running average similar to the one used in the original TCP timeout computation.

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

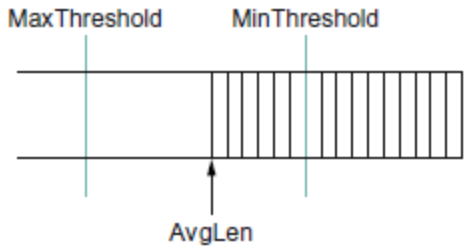
Where 0 < weight < 1 and SimpleLen is the length of the queue when a sample measurement is made.
The reason for using an average queue length rather than an instantaneous one is that it more accurately captures the notion of congestion.

- Second, RED has two queue length thresholds that trigger certain activity: MinThreshold and MaxThreshold. When a packet arrives at the gateway, RED compares the current AvgLen with these wo thresholds according to the following rules.

if AvgLen $\leq$ MinThreshold
$\longrightarrow$ queue the packet

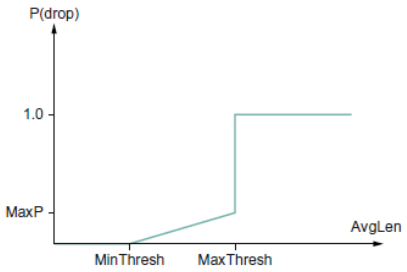
if MinThreshold $<$ AvgLen $<$ MaxThreshold
$\longrightarrow$ calculate probability P
$\longrightarrow$ drop the arriving packet with probability P

if MaxThreshold $\leq$ AvgLen
$\longrightarrow$ drop the arriving packet



If the average queue length is smaller than the lower threshold, no action is taken, and if the average queue length is larger than the upper threshold, then the packet is always dropped. If the average queue length is between the two thresholds, then the newly arriving packet is dropped with some probability p.

The approximate relationship between P and AvgLen is shown in the figure below. The probability of drop increases slowly when AvgLen is between the two thresholds, reaching MaxP at the upper threshold, at which point it jumps to unity.



5.3 Source-Based Congestion Avoidance

⇨ Mechanism for detecting the incipient stages of congestion, before losses occur, from the end hosts.

Basic idea: watch for some sign from the network that some router's queue is building up and that congestion will happen soon if nothing is done about it.
Ex: the source might notice that as packet queues build up in the network's routers, there is a measurable increase in the RTT for each successive packet it sends.

⇨ The congestion window normally increases as in TCP, but every two round-trip delays the algorithm checks to see if the current RTT is greater than the average of the minimum and maximum RTTs seen so far. If it is, then the algorithm decreases the congestion window by one-eight.

⇨ Another algorithm does something similar. The decisions as to whether or not to change the current window size is based on changes to both the RTT and the window size. The window is adjusted once every two round-trip delays based on the product (CurrentWindow - OldWindow) x (Current RTT – OldRTT)

- If the result is positive, the source decreases the window size by one-eight.
- If the result is negative or 0, the source increases the window by one maximum packet size.
  NB: the window changes during every adjustment. It oscillates around its optimal point.

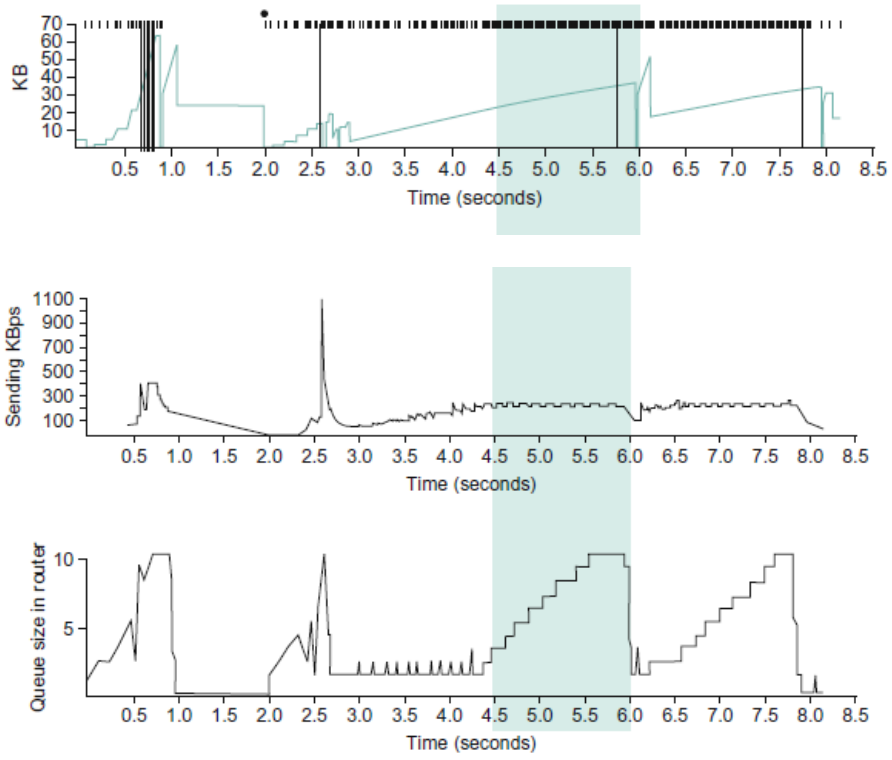
⇨ Third algorithm is based on the flattening of the sending rate. Every RTT, it increases the window size by one packet and compares the throughput achieved to the throughput when the window was one packet smaller. If the difference is less than one-half the throughput achieved when only one packet was in transit, as was the case at the beginning of the connection, the algorithm decreases the window by one packet.

This scheme calculates the throughput by dividing the number of bytes outstanding in the network by the RTT.

⇨ Fourth algorithm: this one looks at changes in the throughput rate or more specifically, changes in the sending rate. Instead of looking for a change in the slop of the throughput it compares the measured throughput rate with an expected throughput rate.
= TCP Vegas

TCP VEGAS

Idea: the source watches for some sign that some router's queue is building up and congestion will happen soon. Ex: RTT grows, sending rate flattens.



The first graph traces the **connection's congestion window**. The middle and bottom graphs depict new information. The middle graph shows the average sending rate as measured at the source, and the bottom graph shows the average queue length as measured at the bottleneck router. All three graphs are synchronized in time.

In the period between 4.5 and 6.0 seconds, the congestion window increases. We expect the observed throughput to also increase, but instead it stays flat (cfr. Middle graph). This is because the throughput cannot increase beyond the available bandwidth. Beyond this point, any increase in the window size only results in packets taking up buffer space at the bottleneck router.

TCP Vegas measures and controls the amount of extra data this connection has in transit. By extra data we mean data that the source would not have transmitted had it been trying to match exactly the available bandwidth of the network.
Goal of TCP: maintain the right amount of extra data in the network.

- If a source is sending too much extra data, it will cause long delays and possibly lead to congestion.
- If a connection is sending too little extra data, it cannot respond rapidly enough to transient increase in the available network bandwidth.

⇨ TCP Vegas's congestion-avoidance actions are based on changes in the estimated amount of extra data in the network, not only on dropped packets.

ALGORITHM

- First define a given flow's **BaseRTT** to be the RTT of a packet when the flow is not congested. In practice, BaseRTT is set to the minimum of all measured RTT's. it is commonly the RTT of the first packet sent by the connection, before the router queues increase due to traffic generated by this flow.

$$\text{ExpectedRate} = \text{CongestionWindow}/\text{BaseRTT}$$

- Second, TCP Vegas calculates the **current sending rate**, ActualRate. This is done by recording the sending time for a distinguished packet, recording how many bytes are transmitted between the time that packet is sent and when its acknowledgement is received, computing the sample RTT for the distinguished packet when its acknowledgement arrives, and dividing the number of bytes transmitted by the sample RTT.

- Third, TCP Vegas compares ActualRate to ExpectedRate and adjusts the window accordingly.

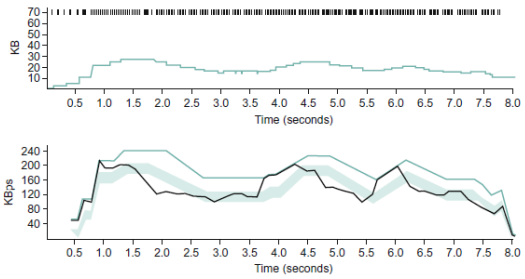
  - Let Diff = ExpectedRate – ActualRate.
    Difff is positive or 0 by definition, since ActualRate> ExpectedRate. Therefore, we need to change BaseRTT to the latest sampled RTT.
  - We define two thresholds, $\alpha<\beta$, corresponding to having too little and too much extra data in the network respectively.

- When Diff < α, TCP Vegas increase the congestion window linearly during the next RTT
- When Diff > β, TCP Vegas decreases the congestion window linearly during the next RTT.
- When α< Diff < β, TCP Vegas leaves the window unchanged.

⇨ The farther away the actual throughput gets from the expected throughput, the more congestion there is in the network. This implies that the sending rate should be reduced. **β triggers this decrease.**

⇨ When the actual throughput rate gets too close to the expected throughput, the connection is in danger or not utilizing the available. **α triggers this increase.**

⇨ The **overall goal** is to keep **between α and β** extra bytes in the network.



- The black line tracks the Actual Rate
- Region between α and β
- The blue line tracks the ExpectedRate

The top graph traces the congestion window. The bottom graph traces the expected and actual throughput rates that govern how the congestion window is set.
The goal is to keep the ActualRate between these two thresholds, within the shaded region.

- Whenever ActualRate falls below the shaded region, TCP Vegas decrease the congestion window because it fears the too many packets are being buffered in the network.
- Whenever ActualRate goes above the shaded region, TCP Vegas increases the congestion because it fears that it is underutilizing the network.

6. Quality of Service

For many years, packet-switched networks have offered the promise of supporting multimedia applications that combine audio, video, and data.

Actually, once digitized, audio and video information becomes like any other form of data, a stream of bits to be transmitted. One obstacle to the fulfillment of this promise has been the need for higher-bandwidth links.

Actually, there is more to transmitting audio and video over a network than just providing sufficient bandwidth.
Ex: participants in a telephone conversion 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.

- ⇨ **Timeliness** of delivery can be very important. We refer to applications that are sensitive to the timeliness of data as **real-time applications.**
  Ex**:** in industrial control, you would like a command send to a robot arm to reach it before the arm crashes into something. 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.
- ⇨ These applications need some sort of assurance from the network, that data is likely to arrive on time.
- ⇨ The best-effort model, in which the network tries to deliver data but makes no promises and leaves the cleanup operation to the edges, is not sufficient for real-time applications.

6.1 Application Requirements

We can divide applications into two types

- **Real-time**
- **Non real-time**. They are sometimes called traditional data applications since they have traditionally been the major applications found on data networks. They can work without guarantees of timely delivery of data. They are able to stretch gracefully in the face of increased delay.

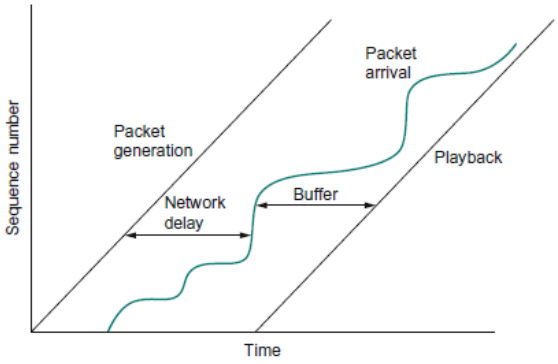
REAL-TIME AUDIO EXAMPLE

Data is generated by collecting samples from a microphone and digitizing them using an analog-to-digital (A → Converter). The digital samples are placed in packets, which are transmitted across the network and received at the other end. At the receiving host, the data must be played back at some appropriate rate.

We can think of each simple as having a particular play back time: the point in time at which it is needed in the receiving host. 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.

⇨ 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.



The left-hand diagonal line shows packets being generated at a steady rate. The wavy line shows when the packets arrive, some variable amount of time after they were sent, depending on what they encountered in the network. The right-hand diagonal line shows the packets being played back at a steady rate, after sitting in the playback buffer for some period of time. As long as the playback line is far enough to the right in time, the variation in network delay is never noticed by the application.
However, if we move the playback line a little to the left, then some packets will begin to arrive too late to be useful.

⇨ For an audio application, there are limits to how far we can delay playing back data. It is hard to carry on a conversion if the time between when you speak and when your listener hears you is more than 300ms. So, what we want from the network in this case is a guarantee that all our data will arrive within 300ms. The bigger the delay, the more useless an application is. A 300ms delay is rather big and a consequence thereof will be a decrease in the quality perceived.

- If data arrives early, we buffer it until its correct playback time.
- If it arrives late, we have no use for it and must discard it.

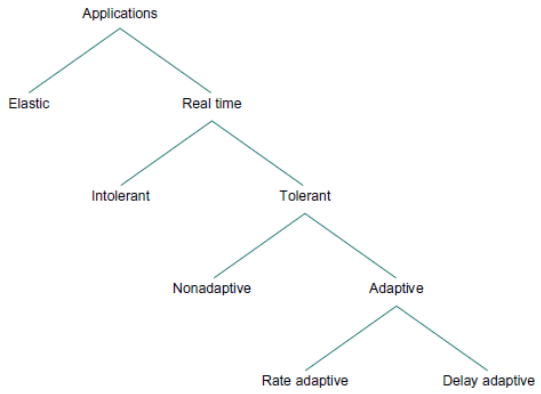
VoIP: QoS REQUIREMENTS

- **Availability**: fraction of time that connectivity is available. High availability means 99,99% (5 min downtime/year)
- **Packet loss:** percentage of packets dropped. Occasional loss of a single packet (20 msec) can be tolerated. More than 1% loss is not acceptable.
- **Delay**: time it takes to reach the endpoint. 150msec is acceptable. More than 200msec causes unacceptable degradation.
- **Jitter**: delay variation. More than 30msec causes significant degradation.
- **Throughput**: available bandwidth.

- From 64Kbps (no compression) down to 5.3 Kbps (with compression -> quality decrease), depending on CODEC.
- Including packet headers (50 bytes): from 84 Kbps to 28 Kbps.

⇨ As delay increases, the user satisfaction tend to decrease.

TAXONOMY OF REAL-TIME APPLICATIONS



- **Tolerance of loss of data**. Losses might occur because a packet arrived too late to be played back as well as arising from the usual causes in the network. One lost audio sample could be considered as acceptable. It is only as more and more samples are lost that quality declines to the point that the speech becomes incomprehensible. On the other hand, some applications may be **intolerant** to losses!
  Ex: a robot control program cannot tolerate loss. Losing the packet that contains the command instructing the robot arm to sop is unacceptable.
  ⇨ Real-time applications are more tolerant of occasional loss than non-real time applications.
- Applications could also be characterized by their **adaptability**. For instance, an audio application might be able to adapt to the amount of delay that packets experience as they traverse the network.
  Playback point adjustment may be fairly easy and it could be implemented for several voice applications such as the audio teleconferencing program known as vat. Playback point adjustment can happen in either direction, but doing so actually involves distorting the payed-back signal during the period of adjustment and that the effects of this distortion will very much depend on how the end user uses the data.
  We would advance the playback point only when it provides a perceptible advantage and only when we have some evidence that the number of late packets will be acceptably small.

⇨ Applications that can adjust their playback point are **delay-adaptive applications**

- Another class of adaptive applications is **rate adaptive**.

APPROACHES TO QoS SUPPORT

⇨ We need a service model with not just one class but with several classes, each available to meet the needs of some set of applications.

Two broad categories

- **Fine-grained approaches:** provide QoS to **individual** applications or flows
- **Coarse-grained approaches**: provide QoS to **large classes** of data or aggregated traffic.

6.2 Integrated Services (RSVP)

SERVICE CLASSES

- **Guaranteed service:** the network should guarantee that the maximum delay that any packet will experience has some specified value. The application can then set its playback point so that no packet will ever arrive after its playback time.

⇨ Intended for **intolerant applications**

- Controlled load: it was motivated by the observation that existing applications run quite well on networks that are not heavily loaded.
  The aim of controlled load service is to emulate a lightly loaded network for those applications that request the service, even though the network as a whole may in fact be heavily loaded.

⇨ Useful for **tolerant/adaptive applications**

OVERVIEW OF MECHANISMS

Whereas with a best-effort service we can just tell the network where we want our packets to go and leave it like that, a real-time service involves telling the network something more about the type of service we require.
We may give it qualitative or quantitative information.

- **FlowSpec**: in addition to describing what we want, we need to tell the network something about what we are going to inject into it.
- **Admission control**: networks need to decide if service can be provided without impacting existing flow. Ex: if 10 users ask for a service in which each will consistently use 2 Mbps of link capacity, and they all share a link with 10 Mbps capacity, the network will have to say no to some of them.
- **Resource reservation:** mechanism to exchange service request, FlowSpecs and admission control decisions.
- **Packet classifying and scheduling:** network nodes have to queue and schedule packets in the appropriate way in order to meet the requirements of the flows.

FLOWSPECS

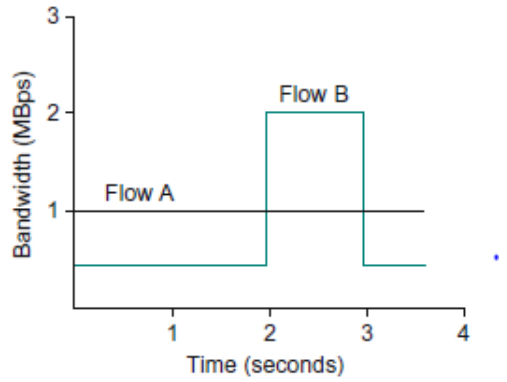
- **RSpec**: part that describes the service **requested from the network**. This is very service specific and relatively easy to describe.

- **TSpec**: this part describes the flow's traffic characteristic. We need to give the network enough information about the bandwidth used by the flow to allow intelligent admission control decisions to be made. The bandwidth used by the flow might vary constantly. It is not a single number.

  One way to describe the bandwidth characteristics of sources is called a **token bucket filter**. It has two parameters:
  - A token rate r
  - A bucket depth B

⇨ To be able to send a byte, I must have a toke. To send a packet of length n, I need n tokens. I start with no tokens and I accumulate them at a rate of r per second. I can accumulate no more than B tokens. So, I can send a burst of as many as B bytes into the network as fast as I want, but over a sufficiently long interval I can't send more than r byte per second.

⇨ This information is very helpful to the admission control algorithm when it tries to figure out whether it can accommodate a new request for service.



Assume that each flow can send data as individual bytes rather than as packets. Flow A generate data at a steady rate of 1MBps, so it can be described by a token bucket filter with a rate r = 1 MBps and a bucket depth of 1 byte. This means that it receives tokens at a rate of 1 MBps but that it cannot store more than 1 token. It spends them immediately.
Flow B also sends at a rate that averages out to 1 MBps over the long term, but does so by sending at 0.5 MBps for 2 seconds and then 2 MBps for 1 second.
Since the token bucket rate r is, in a sense, a long-term average rate, flow A can be described by a token bucket with a rate of 1 MBps. Unlike flow A, however, flow B needs a bucket depth B of at least 1 MB, so that it can store up tokens while it sends at less than 1 MBps to be used when it sends at 2 MBps. For the first 2 seconds in this example, it receives tokens at a rate of 1 MBps but spends them at only 0.5 MBps, so it can save up 2×0.5 = 1 MB of tokens, which it then spends in the third second (along with the new tokens that continue to accrue in that second) to send data at 2 MBps.

⇨ In general, it is good to be as **explicit** about the bandwidth needs of an application as possible to avoid over-allocation or resources in the network.

ADMISSION CONTROL

When some new flow wants to receive a particular level of service, admission control looks at the TSpec and RSpec of the flow and tries to decide if the desired service can be provided to hat amount of traffic, given the currently available resources, without causing any previously admitted flow to receive worse service than it had requested. If it can provide the service, the flow is admitted. If not, then it is denied.

⇨ The hard part is figuring out when to say yes and when to say no.

Admission control is very dependent on the type of request service and on the queuing discipline.

/!\ Admission control should not be confused with policing.

- **Admission control**: **per-flow decision** to admit a new flow or not.
- **Policing:** function applied on a **per-packet basis** to make sure that a flow conforms to the TSpec that was used to make the reservation. If a flow does not conform to its TSpec, for instance, because it is sending twice as many bytes per second as it said it would, then it is likely to interfere with the service provide to other flows, and some corrective action must be taken.

RESERVATION PROTOCOL

IETF standard: Resource Reservation Protocol (RSVP)

One of the key assumptions underlying RSVP is that it should not detract from the robustness that we find in today's connectionless networks. Because connectionless networks rely on little or no state being stored in the network itself, it is possible for routers to crash and reboot and for links to go up and down while end-to-end connectivity is still maintained. RSVP tries to maintain this robustness by using the idea of soft state in the routers.

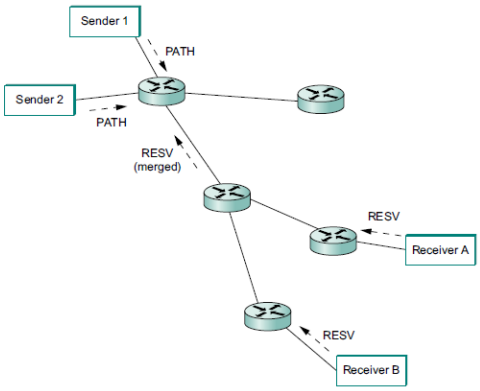
- RSVP tries to maintain this robustness by using **soft state** in the routers. In contrast to hard state found in connection-oriented networks, soft state does not need to be explicitly deleted when it is no longer needed. It times out after some fairly short period if it is not periodically refreshed.
- RSVP aims to **support multicast flows** just as effectively as unicast flows.
- **Receiver-oriented approach**: One of the insights of RSVP's designer's is that most multicast applications have many more receivers than senders, as typified by the large audience and one speaker for a lecture. Also, receivers may have different requirements.
    - The receiver needs to know what traffic will be received (TSpec) so that it can make an appropriate reservation.
    - Path along which traffic will be received must be known so that the receiver can establish a resource reservation at each router on the path.

Two messages: PATH and RESV

- **PATH**: the source transmits a PATH message with TSpec to the receiver. Every router can figure out from which upstream router this message came.
- **RESV**: the receiver responds with RESV request containing RSpec + TSpec. Each router tries to allocate necessary resources and passes RESV to the next upstream router.
    - ⇨ As long as the receiver wants to retain the reservation, it sends the same RESV message about once every 30 seconds.

➔ Now we can see **what happens** when a router or link **fails**.

Routing protocols will adapt to the failure and create a new path from send to receiver. If all goes well, it will establish a new reservation on the new path. Meanwhile, the routers that are no longer on the path will stop getting RESV messages, and these reservations will time out and be released. RSVP deals quite well with changes in topology, as long as routing changes are not excessively frequent.



➔ How to cope with multicast, where there may be multiple senders to a group and multiple receivers.

First, let's deal with multiple receivers for a single sender. As a RESV message travels up the multicast tree, it is likely to hit a piece of the tree where some other receiver's reservation has already been established. It may be the case that the resources reserved upstream of this point are adequate to serve both receivers. In general, reservations can be merged to meet the needs of all receivers downstream of the merge point.

If there are also multiple senders in the tree, receivers need to collect the TSpecs from all senders and make a reservation that is large enough to accommodate the traffic from all senders.

RSVP has **different reservation styles** to deal with such options as:

- Reserve resources for all speakers
- Reserve resources for any n speakers
- Reserve resources for speakers A and B only.

PACKET CLASSIFYING AND SCHEDULING

⇨ Deliver the requested service to the data packets.

Two things that need to be done

- **Classifying packets**: Associate each packet with the appropriate reservation so that it can be handled correctly. This is done by examining up to five fields in the packet: the source address, the destination address, the protocol number, the source port, and the destination port.
- **Packet scheduling:** Manage the packets in the queues so that they receive the service that has been requested. The FIFO queuing is no longer sufficient. Hence, more sophisticated queue management is required.

- **Separate queue** for each flow with Guaranteed service: que gets certain share of bandwidth and can guarantee delay bound.
- Only **one global queue** for aggregated Controlled Load might be sufficient.

Policing and shaping

- Make sure flow conforms specification
- Token bucket can also be used for this purpose
- Corrective action: drop or mark packet

6.3 Differentiated Services (EF, AF)

⇨ The Differentiated Services model allocates resources to a small number of classes of traffic. Some proposed approaches to DiffServ simply **divide traffic into two classes** (best-effort and premium).

We will need some way to figure out which packets are premium and which are regular old best effort. Rather than using a protocol like RSVP to tell all the routers that some flow is sending premium packets, it would be much easier if the packets could just identify themselves to the router when they arrive. For two classes: one bit is sufficient. Indeed, this could be done by using a bit in the packet header, if that bit is a 1, the packet is a premium packet, if it is a 0, the packet is best effort.

⇨ Who sets the premium bit and under what circumstances?
⇨ What does a router do differently when it sees a packet with the bit set?

No reservation protocol is required.

- Packets identify themselves to a router when they arrive.
- Individual router takes appropriate action depending on marking
- Per Hop Behavior (PHB) instead of end-to-end reservation. In other words, they define the behavior of individual routers rather than end-to-end services. Because there is more than one new behavior, there is also a need for more than 1 bit in the packet header to tell the routers which behavior to apply.

THE EXPEDITED FORWARDING (EF) PHB

⇨ Packets marked for EF treatment should be forwarded by the router with minimal delay and loss.

The only way that a router can guarantee this to all EF packets is if the arrival rate of EF packets at the router is strictly limited to

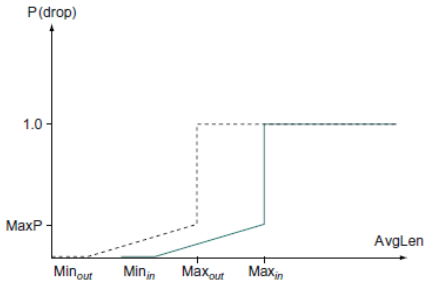
be less than the rate at which the router can forward EF packets. The rate limiting of EF packets is achieved by configuring the routers at the edge of an administrative domain to allow a certain maximum rate of EF packet arrivals into the domain.
A simple approach would be to ensure that the sum of the rates of all EF packets entering the domain is less than the bandwidth of the slowest link in the domain.

Use priority queuing (PQ) for this class. An alternative is WFQ (weighted fair queuing) between EF packets and other packets, with the weight of EF set sufficiently high that all EF packets can be delivered quickly.

AQM (Active Queue Management) should not be applied since the traffic does not respond dynamically to packet drops.

THE ASSURED FORWARDING (AF) PHB

⇨ The AF PHB has its roots in an approach known as RED with In and Out (RIO) or Weighted RED, both of which are enhancements to the basic RED algorithm.



We see drop probability on the y-axis increasing as average queue length increases along the x-axis. We now have two separate drop probability curves corresponding to the two classes of traffic. RIO calls the two classes "in" and "out".
Because the "out" curve has a lower MinThreshold than the "in" curve, it is clear that under low levels of congestion, only packets marked "out" will be discarded by the RED algorithm.
If the congestion becomes more serious, a high percentage of "out" packets are dropped, and then if the average queue length exceeds Min_{in}, RED starts to drop in packets as well.

The **effectiveness** of a mechanism like RIO depends to some extent on correct parameter choices, and there are considerably more parameters to set for RIO.

One interesting property of RIO is that it does not change the order of "in" and "out" packets. If a TCP connection is sending packets through a profile meter, and some packets are being marked "in" while others are marked "out", those packets will receive different drop probabilities in the router queues, but they will be delivered to the receiver in the same order in which they were sent.
This is important for most TCP implementations, which perform much better when packets arrive in order, even if they are designed to cope with misordering.

A third way to provide Differentiated Services is to use the **DSCP value** to determine which queue to put a packet into in a weighted fair queuing scheduler. We might use one code point to indicate the best-effort queue and a second code point to select the premium queue. We then need to choose a weight for the premium queue that makes the premium packets get better service than the best-effort packets.

DSCP: 000000

MPLS and QoS

Problems

- IntServ: limited scalability (state per flow)
- DiffServ: only aggregates classes, no end-to-end guarantee per flow.

MPLS combines datagram and VC approach

- IP routing + label switching
- Label stacking allows aggregation

Traffic Engineering (TE) and Explicit Routing

- Possible to create LSP (Label Switch Path) with bandwidth reservation end-to-end
- CSPF: select path that guarantees QoS requirements
- Makes use of RSVP for signaling

MPLS incorporates features from both IntServ and DiffServ

- IntServ-like bandwidth reservation per flow over LSP
- DiffServ-like aggregates can be mapped onto LSPs
- Forwarding Equivalence Class (FEC) based on one or more parameters: IP addresses, port numbers, protocol number, network prefix, DSCP, …
- PHB based on label: queue selection and discard policy.

6.4 Equation-Based Congestion Control

⇨ **TCP** is **not appropriate for real-time applications** as it appears to be too slow. One reason is that TCP is a reliable protocol, and real-time applications often cannot afford the delays introduced by retransmission.

⇨ What if we were to decouple TCP from its congestion control mechanism, to add TCP-like congestion control to an unreliable protocol like UDP? Could real-time applications make use of such a protocol?

It is possible to add TCP-friendly congestion-control algorithms. These algorithms have two main goals:

- Slowly adapt the congestion window. This is done by adapting over relatively longer time periods rather than one per-packet basis. This smooths out the transmission rate.
- To be TCP friendly in the sense of being fair to competing TCP flows. This is often enforced by ensuring that the flow's behavior adheres to an equation that models TCP's behavior.
  = equation-based congestion control

$$Rate \propto \left(\frac{1}{RTT \times \sqrt{\rho}}\right)$$

  - The receiver must periodically report the loss rate ρ. Ex: it might report that it failed to receive 10% of the last 100 packets.
    The larger the loss rate, the lower the throughput.
  - The sender adjusts the sending rate. An adaptive application reacts to rate changes.

CHAPTER 7: END-TO-END DATA

1. Introduction

- From a network's perspective, application programs send messages to each other. Each of these messages is just an uninterrupted string of bytes.
- Form the application's perspective, these messages contain various kinds of data: arrays of inters, video frames, lines of text, digital images, etc. in other words, these bytes have a meaning.

Different concerns

- **Framing problem:** The receiver must be able to extract the same message from the signal as the transmitter sent.
- We should make the encoding as efficient as possible.
- Video-capable consumer devices are proliferating
- Is there enough bandwidth
- What with incompatible video formats?

2. Presentation Formatting

⇨ One of the most common transformations of networks data is from the representation used by the application program into a form that is suitable for transmission over a network and vice versa.
= presentation

The sending program translates the data it wants to transmit from the representation it uses internally into a message that can be transmitted over the network. That is, the data is **encoded** in a message (= unmarshalling).
On the receiving side, the application translates this arriving message into a representation that it can then process. The message is **decoded**. (= marshalling).

Computers represent data in different ways depending on the architecture.

- Floating point numbers
- Own nonstandard format
- Different sizes: 16-bit, 32-bit, 64-bit
- Two's complement notation >< binary coded decimal. In a two's complement notation, each number is represented by a block of bits.
- Signed >< unsigned

- Big-endian form >< little-endian (Intel x86)

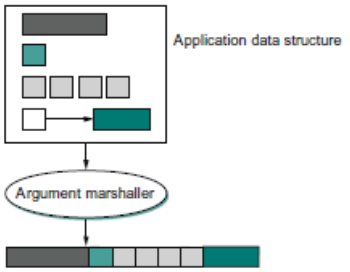
Application programs are written in different languages, which makes encoding more difficult.

2.1 Taxonomy

DATA TYPES

⇨ What data types the system is going to support?

- Base types: integers, floating-point numbers, and characters. + ordinal types and Booleans.
- Flat types: structures and arrays.
- Complex types: built using points. The data structure that one program wants to send to another might not be contained in a single structure, but might instead involve points from one structure to another.



Depending on how complicated the type system is, the task of argument marshalling usually involves converting the base types, packing the structures, and linearizing the complex data structures, all to form a contiguous message that can be transmitted over the network.

CONVERSION STRATEGY

⇨ Once the type system is established, the next issue is what conversion strategy the argument marshaller will use.

- **Canonical intermediate form**: settle on an external representation for each type. The sending host translates from its internal representation to this external representation, before sending data, and the receiver translates from this external representation into its local representation when receiving data.
  Preferred if many different architectures exist.
- **Receiver-makes-right:** the sender transmits data in its own internal format. The sender does not convert the base types, but usually has to pack and flatten more complex data structures. The receiver is then responsible for translating the data from the sender's format into its own local format.
  Problem: every host must be prepared to convert data from all other machine architectures.
  This strategy is preferred if most hosts are using similar architecture.

TAGS

⇨ How does the receiver know what kind of data is contained in the message it receives?

- **Tagged data**

A **tag** is any additional information included in a message, beyond the concrete representation of the base types, that helps the receiver decoding the message.

Several possible tags that might be included in a message:

- **Type tag:** indicates that the value that follows is an integer, floating-point number, …
- **Length tag**: indicate the number of elements in an array or the size of an integer.
- **Architecture tag**: might be used in conjunction with the receiver-makes-right strategy to specify the architecture on which the data contained in the message was generated.

- **Untagged data**

⇨ How does the receiver then know how to decode the data in this case?

It knows because it was programmed to know.
Although it works for most cases, the one place it breaks down is when sending variable-length arrays. In such a case, a length tag is commonly used to indicate how long the array is.

STUBS

Stub: piece of code that implements argument marshalling. Stubs are typically used to support RPC.

- On the client side, the stub marshals the procedure arguments into a message that can be transmitted by means of the RPC protocol.
- On the server side, the stub converts the message back into a set of variables that can be used as arguments to call the remote procedure.

2.2 Examples (XDR, ASN.1, NDR)

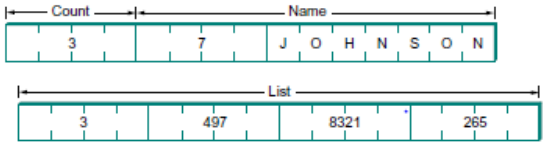
XDR

External Data Representation is the network format used with SunRPC.

- Supports the entire C-type system with the exception of function pointers
- Defines a canonical intermediate form
- Does not use tags (except to indicate array lengths)
- Use compiled stubs

Integer: An XDR integer is **a 32-bit data** item that encodes a C integer. It is represented in **two's complement notation**, with the most significant byte of the C integer in the first byte of the XDR integer. That is, XDR uses **big-endian format** for integers. It supports both **signed** and **unsigned** integers.

Array: XDR represents variable-length arrays, by first specifying an unsigned integer (4 bytes) that gives the number of elements in the array, followed by that many elements of the appropriate type.



Example of a C structure and the XDR routine that encodes/decodes this structure. It depicts XDR's on-the-wire representation of this structure when the field name is seven characters long and the array list has three values in it.

ASN.1

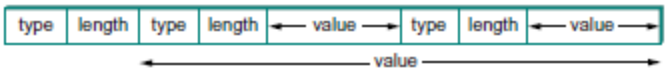
Abstract Syntax Notation One is an ISO standard that defines, among other things, a representation for data sent over a network.

The representation-specific part of ASN.1 is called the Basic encoding rules.

- It supports the C-type system without function pointers
- Defines a canonical intermediate form
- Uses type tags
- Its stubs can be either interpreted or compiled.

ASN.1 represents each data item with a triple of the form <tag, length, value>.

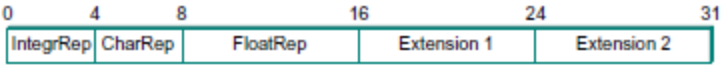
- This tag is typically an 8-bit field, although ASN.1 allows for the definition of mutibyte ttags.
- The length field specifies how man bytes make up the value
- Compound data types such as structures, can be constructed by nesting primitive types.



NDR

Network Data Representation (NDR) is the data-encoding standard used in the Distributed Computing Environement.

- Uses receiver-makes-right. It does this by inserting an architecture tag at the front of each message. Individual data items are untagged.



The above figure illustrates the 4-byte architecture definition tag that is included at the front of each NDR-encode message.

- The first byte contains two 4-bit fields.
- The first field, IntegrRep, defines the format for all integers contained in the message. A 0 in this field indicates big-endian integers, and a 1 indicates little-endian integers.
- The CharRep field indicates what character format is used. 0 means ASII and 1 means EBCDIC.
- The FloatRep byte defines which floating-point representation is being used.
- The final 2 bytes are reserved for future use.

2.3 Markup languages (XML)

⇨ How does a web server describe a Web page so that any number of different browsers know what to display on the screen?
In that specific case, we use HTML. This language indicates that certain character strings should be displayed in bold or italics, what font type and size should be used, and where images should be positioned.

Besides, the popularity of the Web and availability of all sorts of applications and data on it have also created a situation in which different Web applications need to communicate with each other and understand each other's data.

⇨ Approach taken in the Web to enable such communication: **Extensible Markup Language (XML).** XML is a framework for defining different markup languages for different kinds of data.
Such languages, HTML and XML, take the tagged data approach to the extreme. Data is represented as text, and text tags known as markup are intermingled with the data text to express information about the data.

- In the case of HTML, markup merely indicates how the text should be displayed
- XML expresses the type and structure of the data.

```
<?xml version="1.0"?>
<employee>
   <name>John Doe</name>
   <title>Head Bottle Washer</title>
   <id>123456789</id>
   <hiredate>
      <day>5</day>
      <month>June</month>
      <year>1986</year>
   </hiredate>
</employee>
```

XML Namespaces

⇨ XML has to solve a common problem, that of name clashes. The problem arises because schema languages such as XML Schema support modularity in the sense that a schema can be reused as part of another schema.
Ex: Suppose two XSDs are defined independently, and both happen to define the markup name idNumber. Perhaps one XSD uses that name to identify employees of a company, and the other XSD uses it to identify laptop computers owned by the company. We might like to reuse those two XSDs in a third XSD for describing which assets are associated with which employees, but to do that we need some mechanism for distinguishing employees' idNumbers from laptop idNumbers.

A possible solution to this problem is XML namespaces. A namespace is a collection of names. Each XML namespace is identified by a Uniform Resource Identifier.

```
targetNamespace="http://www.example.com/employee"
```

3. Multimedia Data

⇨ Multimedia data comprises audio, video, and still images. It now makes up the majority of traffic on the Internet by many estimates.
Part of what has made the widespread transmission of multimedia across networks possible is advances in **compression technology**.
Challenges: you want to try to keep the information that is most important to a human, while getting rid of anything that does not improve the human's perception of the visual or auditory experience.

The uses of compression are not limited to multimedia data. Ex: Zip or compress files before sending them over a network, or uncompress data file after downloading.

Lossless compression >< lossy compression

⇨ Compression used for multimedia data, does not promise that the data received is exactly the same as the data sent.

3.1 Lossless Compression Techniques

In many ways, compression is inseparable from data encoding. When thinking about how to encode a piece of data in a set of bits, we might just as well think about how to encode the data in the smallest set of bits possible. For example, if you have a block of data that is made up of the 26 symbols A through Z, and if all of these symbols have an equal chance of occurring in the data block you are encoding, then encoding each symbol in 5 bits is the best you can do (since 25 = 32 is the lowest power of 2 above 26). If, however, the symbol R occurs 50% of the time, then it would be a good idea to use fewer bits to encode the R than any of the other symbols. In general, if you know the relative probability that each symbol will occur in the data, then you can assign a different number of bits to each possible symbol in a way that minimizes the number of bits it takes to encode a given block of data. This is the essential idea of **Huffman codes**, one of the important early developments in data compression.

WHEN TO COMPRESS

It might seem that compression data before sending it would always be a good idea, since the network would be able to deliver compressed data in less time than uncompressed data. But this is not necessarily the case.

⇨ Compression and decompression algorithms often involve time-consuming computations. Is it worth it?

For many compression algorithms we may not need to compress the whole data set before beginning transmission (videoconferencing would be impossible if we did), but rather we need to collect some amount of data first.

RUN LENGTH ENCODING

⇨ Simple compression technique.
Idea: replace consecutive occurrences of a given symbol with only one copy of the symbol, plus a count of how many times that symbol occurs.
AAABBCDDDD would be encoded as 3A2B1C4D.

RLE turns out to be useful for compressing some classes of images. It can be used in this context by comparing adjacent pixel values and then encoding only the changes. For images that have large homogeneous regions, this technique is quite effective.

DIFFERENTIAL PULSE CODE MODULATION (DPCM)

Idea: first output a reference symbol and then, for each symbol in the data, output the difference between that symbol and the reference symbol.

Ex: using A as the reference symbol, the string AAABBCDDDD would be encoded as A0001123333. Because A is the same as the reference symbol, B has a difference of 1 from the reference symbol, etc. As soon as the difference becomes too large, a new reference symbol is selected.

Benefit: when the differences are small, they can be encoded with fewer bits than the symbol itself.

DPCM works better than RLE for most digital imagery, since it takes advantage of the fact that adjacent pixels are usually similar. Due to this correlation, the dynamic range of the difference between the adjacent pixel values can be significantly less than the dynamic range of the original image. Therefore, this range can be represented using fewer bits.

DELTA ENCODING

Idea: it encodes a symbol as the difference from the previous one.

Ex: AAABBCDDDD would be represented as A001011000.

This encoding is likely to work well for encoding images where adjacent pixels are similar. It is also possible to perform RLE after delta encoding, since we might find long strings of 0s if there are many similar symbols next to each other.

DICTIONARY-BASED METHODS

Idea: build a dictionary (table) of variable-length strings that you expect to find in the data and then to replace each of these strings when it appears in the data with the corresponding index to the dictionary.

Ex: instead of working with individual characters in text data, we could treat each word as a string and output the index in the dictionary for that word.

To give an example, the word compression has the index 4978 in one particular dictionary. To compress a body of text, each time the string "compression" appears, it would be replaced by 4978.

- ⇨ Where does the dictionary come from? One option is to define a static dictionary, preferably one that is tailored for the data being compressed. A more general solution is to adaptively define the dictionary based on the contents of the data being compressed.

3.2 Image representation and compression (GIF, JPEG)

- ⇨ The ISO defined a digital image format known as JPEG. JPEG is the most widely used format for still images in use today.
- ⇨ Many techniques used in JPEG also appear in MPEG, the set of standards for video compression and transmission.

Digital images are made up of pixels. Each pixel represents one location in the two-dimensional grid that makes up the image, and for color images each pixel has some numerical value representing a color. There are lots of ways to represent colors, referred to as color spaces. The most commonly used is RGB (Red, Green, Blue).

The encoding and transmission of color images requires agreement between the two ends on the color space. Otherwise, you would end up with different colors being displayed by the receiver than were captured by the sender.

GRAPHICAL INTERCHANGE FORMAT (GIF)

GIF uses the RGB color space and starts out with 8 bits to represent each of the three dimensions of color for a total of 24 bits.

Rather than sending those 24 bits per pixel, GIF first reduces 24-bit color images to 8-bit color images. This is done by identifying the colors used in the picture, and then picking the 256 color that must closely approximate the colors used in the picture. There might be more than 256 colors, however, so the trick is to try not to distort the color too much by picking 256 colors such that no pixel has its color changed too much.

- ⇨ This is a lossy compression for any picture with more than 256 colors.
- ⇨ Using this approach, GIF is sometimes able to achieve compression ratios on the order of 10:1. But only when the

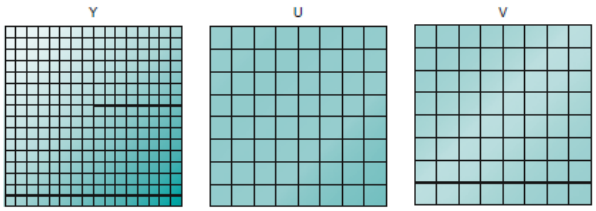
image consists of a relatively small number of discrete colors. Ex: Graphical logos.

The JPEG format is considerably more well suited to photographic images. It does not reduce the number of colors like GIF. Instead, it starts off by transforming the RGB colors to the YUV space.

YUV and RGB are alternative ways to describe a point in a 3-dimensional space. And it is possible to convert from one color space to another.

Once the image has been transformed into YUV space, we can thing about compression each of the three components separately. We want to be more aggressive in compressing the U and V components, to which human eyes are less sensitive. One way to compress the U and V components is to subsample them.

Idea: take a number of adjacent pixels, calculate the average U or V value for that group of pixels, and transmit that, rather than sending the value for every pixel.



The luminance (Y) component is not subsampled, so the Y value of all the pixels will be transmitted as indicated by the 16x16 gird of pixels on the left. In the case of U and V, we treat each group of four adjacent pixels as a group, calculate the average of the U or V value for that group, and transmit that.
Hence, we end up with an 8x8 grid for U and V values to transmit.

In this example, for every four pixels, we transmit six values (4 Y and each of U and V), rather than the original 12 values (4 each for all three components), for a 50% reduction in information.

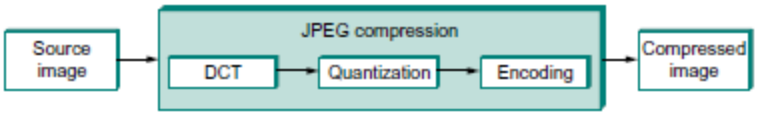
You could be either more or less aggressive in the subsampling, with corresponding increase in compression and decrease in quality.

Once subsampling is done, we now have three grids of pixels to deal with, and each one is dealt with separately.

JPEG compression of each component takes place in **three phases.** On the compression side, the image is fed through these three phases one 8x8 block at a time.

- **1st phase:** applies the discrete cosine transform (DCT) to the block. If we think of the image as a signal in the spatial domain, then DCT transforms this signal into an equivalent signal in the spatial frequency domain. This is a lossless operation but a necessary precursor to the next lossy step.

- **2nd phase:** applies a quantization to the resulting signal and, in so doing, loses the least significant information contained in that signal.
- **3rd phase:** encodes the final result, but in so doing, also adds an element of lossless compression to the lossy compression achieved by the first two phases.
    - ⇨ Decompression follows these same three phases, but in a reverse order.



DCT PHASE

= Discrete Cosine Transform

This transformation is closely related to the fast Fourier Transform (FFT). It takes an 8x8 matrix of pixel values as input and outputs an 8x8 matrix of frequency coefficients. So, you can think of the input matrix as a 64-point signal that is defined in two spatial dimensions. DCT breaks this signal into 64 spatial frequencies.

- ⇨ The **idea** behind the DCT is to separate the gross features, which are essential to viewing the image from the fine detail, which is less essential and, in some cases, might be barely perceived by the eye.
- ⇨ DCT itself does not lose information. It just transforms the image into a form that makes it easier to know what information to remove.

QUANTIZATION PHASE

- ⇨ Compression becomes lossy.

Idea: quantization is a matter of dropping the insignificant bits of the frequency coefficients.

Ex: imagine you want to compress some whole numbers less than 100, such as 45, 98, 23, 66 and 7. If you decided that knowing these numbers truncated to the nearest multiple of 10 is sufficient for you pruposes, then you could divide each number by the quantum 10 using interger arithmetic, yealding 4, 9, 2, 6 and 0. This numbers can all be encoded in 4 bites rather than the 7bits needed to encode the original numbers.

JPEG uses a quantization table that gives the quantum to use for each of the coefficients. We can think of this table as a parameter

that can be set to control how much information is lost and, how much compression is achieved.

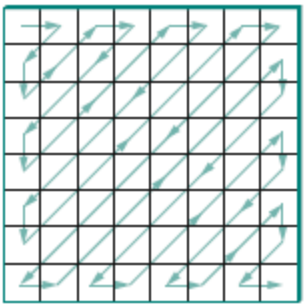
Table 7.1 Example JPEG Quantization Table

$$\text{Quantum} = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}$$

ENCODING PHASE

The final phase of JPEG encodes the quantized frequency coefficients in a compact form. This results in additional compression, but this compression is lossless.

Starting with the DC coefficient in position (0,0), the coefficients are processes in the zigzag sequence. Along this zigzag, a form of run length encoding is used. RLE is applied to only the 0 coefficients, which is significant because many of the later coefficients are 0. The individual coefficients are then encoded using a Huffman code.



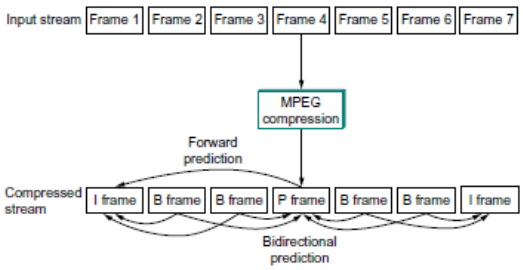
⇨ JPEG includes a number of variations that control how much compression you achieve versus the fidelity of the image.

3.3 Video Compression (MPEG)

A moving picture (video) is simply a succession of still images, also called frames or pictures, displayed as some video rate.
Each of these frames can be compressed using the same DCT-based technique used in JPEG.
Stopping at this point would be a mistake, however, because it fails to remove the interframe redundancy present in a video sequence.
Ex: two successive frames of video will contain almost identical information if there is not much motion in the scene, so it would be unnecessary to send the same information twice.

Besides, MPEG also defines a mechanism for encoding an audio signal with the video, but we consider only the video aspect of MPEG in this section.

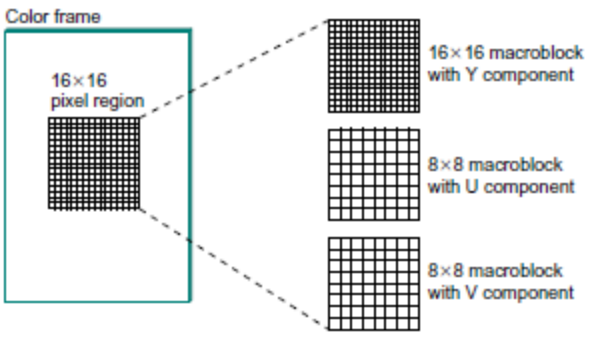
FRAME TYPES



MPEG takes a sequence of video frames as input and compresses them into three types of frames: **I frame** (intrapicture), **P frames** (predicted picture), **B frames** (bidirectional predicted picture). Each frame of input is compressed into one of these three frame types. I frames can be thought of as reference frames. They are self-contained, depending on neither earlier frames nor later frames. An I frame is simply the JPEG compressed version of the corresponding frame in the video source.
P and B frames are not self-contained. They specify relative differences from some reference frame. A P frame specifies the differences from the previous I frame, while a B frame gives an interpolation between the previous and subsequent I of P frames.

The upper figure illustrates a sequence of seven video frames that, after being compressed by MPEG, result in a sequence of I, P, and B frames. The two I frames stand alone. Each can be decompressed at the receiver independently of any other frames. The P frame depends on the preceding I frame.it can be decompressed at the receiver only if the preceding I frame also arrives. Each of the B frames depends on both the preceding I or P frame and the subsequent I of P frame. Both of these reference frames must arrive at the receiver before MPEG can decompress the B frame to reproduce the original video frame.

- **I frames** are approximately equal to the JPEG compressed version of the source frame. The main difference is that MPEG works in units of 16x16 macroblocks. For a color video represented in YUV, the U and V components in each macroblock are subsampled into an 8x8 block. Each 2x2 sub block in the macroblock is given by one U value and one V value, corresponding to the average of the four pixel values. The sub block still has four Y values.



- The P and B frames are also processed in units of macroblocks. The information they carry for each macroblock captures the motion in the video. That is, it shows in what direction and how far the macroblock moved relative to the reference frame(s).

⇨ When generating a B or P frame during compression, MPEG must decide where to place the macroblocks. Recall that each macroblock in a P frame, for example, is defined relative to a macroblock in an I frame, but that the macroblock in the P frame need not be in the same part of the frame as the corresponding macroblock in the I frame—the difference in position is given by the **motion vector**.

EFFECTIVENESS AND PERFORMANCE

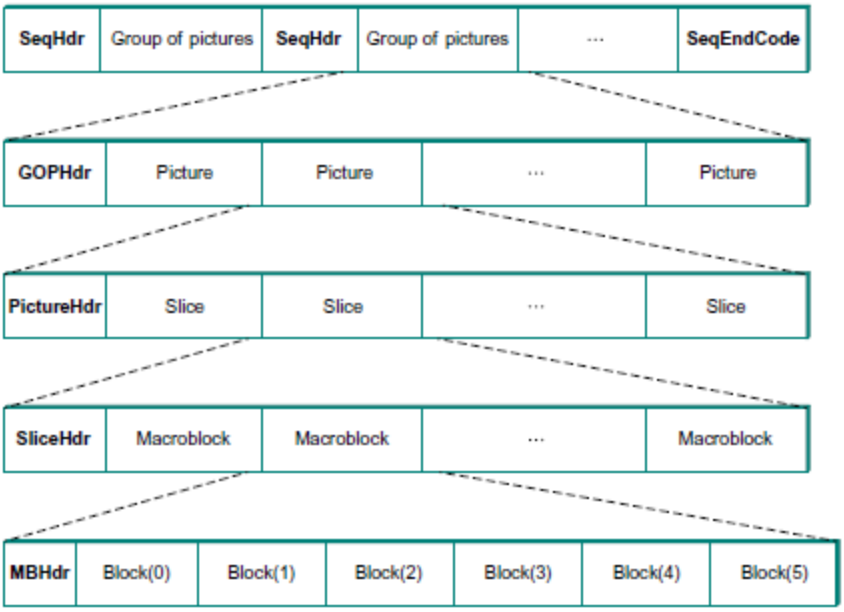
MPEG involves an expensive computation.

- On the **compression side**, it is typically done offline, which is not a problem for preparing movies for a video-on-demand service. Video can be compressed in real time using hardware today, but software implementations are quickly closing the gap.
- On the decompression side, low-cots MPEG video boards are available, but they do little more than YUV color lookup, which fortunately is the most expensive step.

3.4 Transmitting MPEG over a network

⇨ MPEG and JPEG are not just compression standards but also definitions of the format of video and images. Focusing on MPEG, the first thing to keep in mind is that it defines the format of a video stream. It does not specify how this stream is broken into network packets. Thus, MPEG can be used for videos stored on disk, as well as videos transmitted over a stream-oriented network connection, like that provided by TCP.

MAIN PROFILE OF AN MPEG-2



A main profile MPEG-2 stream has a nested structure. At the outermost level, the video contains a sequence of groups of pictures (GOP) separated by a SeqHdr. The sequence is terminated by a SeqEndCode. The SeqHdr that precedes every GOP specifies, among oter things, the size of each picture (frame) in the GOP and two quantization matrices for the macroblocks within this GOP.

- Each GOP is given by a **GOPHdr**, followed by the set of pictures that make up the GOP. The GOPHdr specifies the number of pictures in the GOP, as well as synchronization information for the GOP.
- Each picture is given by a **PictureHdr** and a set of slices that make up the picture.
- The **SliceHdr** gives the vertical position of the slice, plus another opportunity to change the quantization table. Next, the SliceHdr is followed by a sequence of macroblocks.
- Each macroblock includes a header that specifies the block address within the picture, along with data for the six blocks within the macroblock: one for the U component, one for the V component, and four for the Y component.

⇨ One of the powers of the MPEG formats is that it gives the encoder an **opportunity to change the encoding over time**. It can change the frame rate, the resolution, the mix of frame types that define a GOP, the quantization table, and the encoding used for individual macroblocks. Consequently, it is possible to adapt the rate at which a video is transmitted over a network by trading picture quality for network bandwidth.

⇨ Another interesting aspect is how the stream is **broken into packets**. If sent over a TCP connection, packetization is not an issue. TCP decides when it has enough bytes to send the next IP datagram.
When using video interactively, however, it is rare to transmit it over TCP, since TCP has several features that are ill suited to highly latency-sensitive applications such as abrupt rate changes after a packet loss and retransmission of lost packets.

Packetizing the stream is only the first problem in sending MPEG-compressed video over a network. The next complication is dealing with packet loss.

- If a B frame is dropped by the network, then it is possible to simply replay the previous frame without seriously compromising the video.
- A lost I frame has serious consequences. None of the subsequent B and P frames can be processed without it. Losing an I frame would result in losing multiple frames of the video.
  While you can retransmit the missing I frame, the resulting delay would probably not be acceptable in a real-time videoconference.
  Solution: use the Differentiated Service techniques to mark the packets containing I frames with a lower drop probability than other packets.

3.5 Audio Compression (MP3)

⇨ MPEG also defines a standard for compressing audio. This standard can be used to compress the audio portion of a movie or it can be used to compress stand-alone audio.

- CD-quality audio is sampled at a rate of 44.1 KHz. Each sample is 16 bits, which means that a stereo (2-channel) audio stream results in a bit rate of 2 x 44.1 x 1000 x 16 = 1.41 Mbps.
- Telephone-quality voice is sampled at a rate of 8 KHz, with 8-bit samples, resulting in a bit rate of 64kbps, which is not coincidentally the speed of an ISDN data/voice line pair.

Three levels of compression

Table 7.2 MP3 Compression Rates		
Coding	Bit Rates	Compression Factor
Layer I	384 kbps	4
Layer II	192 kbps	8
Layer III	128 kbps	12

⇨ Layer III, widely known as MP3, is the most commonly used.

To achieve these compression ratios, MP3 uses techniques that are similar to those used by MPEG to compress video.

- First, it **splits the audio stream** into some number of frequency **sub bands**, loosely analogous to the way MPEG processes they Y, U and V components of a video stream separately.
- Second, **each sub-band is broken** into a sequence of blocks, which are similar to MPEG's macroblocks except they can vary in length from 64 to 1024 samples.
- Finally, **each block is transformed** using a modified DCT algorithm, quantized, and Huffman encoded.

⇨ The trick to MP3 is how many sub-bands it elects to use and how many bits it allocates to each sub-band, keeping in mind that it is trying to produce the highest-quality audio possible for the target bit rate.

⇨ Once compressed, the sub-bands are packaged into fixed-size frames, and a header is attached. This header includes synchronization information, as well as the bit allocation information needed by the decoder to determine how many bits are used to encode each sub-band.

These audio frames can then be interleaved with video frames to form a complete MPEG stream. One interesting side not is that, while it might work to drop B frames in the network should congestion occur, experience teaches us that it is not a good idea to drop audio frames since users are better able to tolerate bad video than bad audio.

CHAPTER 8 : NETWORK SECURITY

1. Introduction

Computers networks are typically a shared resources used by many applications representing different interests. The Internet is particularly widely shared, being used by competing businesses, mutually antagonistic governments, and opportunistic criminals.

- ⇨ Unless security measures are taken, a network conversation or a distributed application may be **compromised by an adversary**.
- ⇨ How might eavesdropping be accomplished? It is trivial on a broadcast network such as an Ethernet, where any node can be configured to receive all the message traffic on that network.
- ⇨ It is possible and practical to encrypt messages so as to prevent an adversary from understanding the message contents. A protocol that does so is said to provide **confidentiality**.

  >< even with confidentiality there still remains threats for the website customer.

  - **Integrity**: an adversary who can't read the contents of an encrypted message might still be able to change a few bits in it, resulting in a valid order for a different item. Integrity consists in preventing such tampering.
    Integrity refers to the fact that I receive exactly the same content that the one sent.
  - **Replay attack**: when the adversary transmits an extra copy of your message.
  - **Originality** is provided by a protocol that detects replays.
  - **Timeliness:** Receive access to the actual information.
  - **Authentication**: ensures that you are really talking to whom you think you are talking.
  - **Access control**: enforcing the rules regarding who is allowed to do what.
  - **Availability**: ensuring a degree of access
- ⇨ Security doesn't refer ton only one specific layer. Indeed, security solution seem to appear on many different levels.

Another threat to the customer is unknowingly being directed to a false website. This can result from a **Domain Name System (DNS) attack**, in which false information is entered in a DNS server or the name service cache of the customer's computer.

Websites have also been subject to **denial of service (DoS**) attacks, during which would-be customers are unable to access the website because it is being overwhelmed by bogus requests.

Worms, pieces of self-replicating code that spread over networks, have been known for several decades and continue to cause problems, as do their relatives, viruses, which are spread by the transmission of infected files.

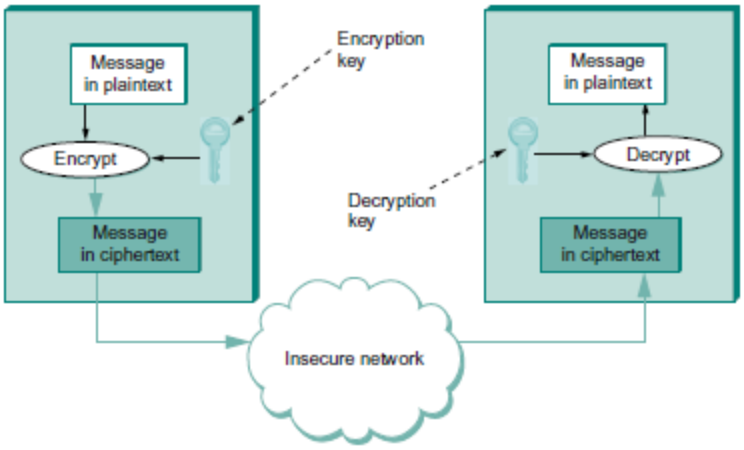
2. Cryptographic Building Blocks

⇨ Cryptographic algorithms are parameterized by **keys**.

2.1 Principles of Ciphers

Encryption transforms a message in such a way that it becomes unintelligible to any party that does not have the secret of how to reverse the transformation.

- The **sender** applies an **encryption function** to the original plaintext message, resulting in a ciphertext message that is sent over the network.
- The **receiver** applies a **secret decryption function**, the inverse of the encryption function, to recover the original plaintext.
  - ⇨ The ciphertext transmitted across the network is unintelligible to any eavesdropper, assuming the eaves dropper doesn't know the description function.



Encryption and decryption functions should be parameterized by a key, and these functions should be considered public knowledge, only the key need to be secret.

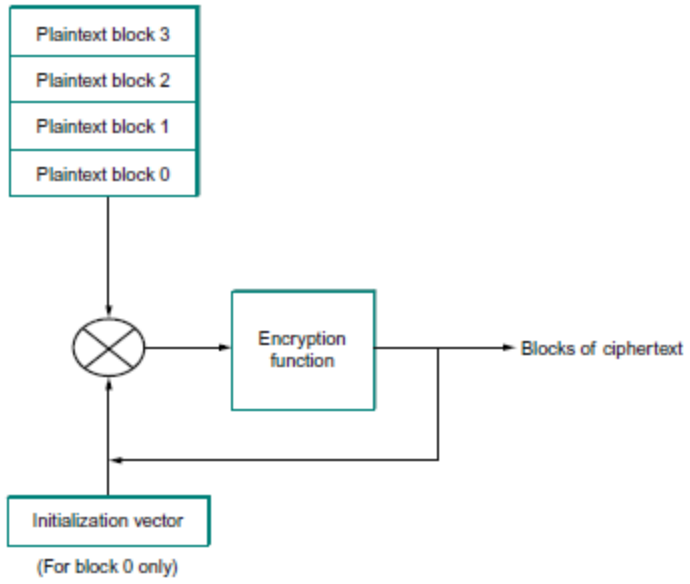
The ciphertext produced for a given plaintext message depends on both the encryption function and the key. One reason for this principle is that if you depend on the cipher being kept secret, then you have to retire the cipher when you believe it is no longer secret. This means potentially frequent changes of cipher, which is problematic since it takes a lot of work to develop a new cipher. There is cost + risk in developing a new cipher.

Basic requirement for an encryption algorithm is that it turns plaintext into ciphertext in such a way that only the intended recipient, the holder of the decryption key, can recover the plain text. → means that the encrypted messages cannot be read by people who do not hold the key.

⇨ The best cryptographic algorithms can prevent the attacker from deducing the key even when the individual knows both the plaintext and the ciphertext.

Most ciphers are **block ciphers**. They are defined to take as input a plaintext block of a certain fixed size, typically 24 to 128 bits. Using a block cipher to encrypt each block independently has the weakness that a given plaintext block value will always result in the same ciphertext block. Hence, recurring block values in the plaintext are recognizable as such in the ciphertext, making it much easier for a cryptanalyst to break the cipher.

To prevent this, block ciphers are always augmented to make the ciphertext for a block vary depending on context. Ways in which a block cipher may be augmented are called modes of operation. A common mode of operation is **cipher block chaining**, in which each plaintext block is XORed with the previous block's ciphertext before being encrypted. The result is that each block's ciphertext depends in part on the preceding blocks.



2.2 Symmetric-key Ciphers

In a symmetric-key cipher, both participants share the same key. If a message is encrypted using a particular key, the same key is required for decrypting the message.

⇨ Encryption and decryption keys are identical.

Symmetric-key ciphers are also known as **secret-key ciphers** since the shared key must be known only to the participants.

DES

= Data Encryption Standard

⇨ Standard issued by the US National Institute.

Encryption = decryption (only key ordering is reversed)

There is no real proff that DES is secure (based on confusion and diffusion)

Problem: it is not possible to try all 2^{56} keys.

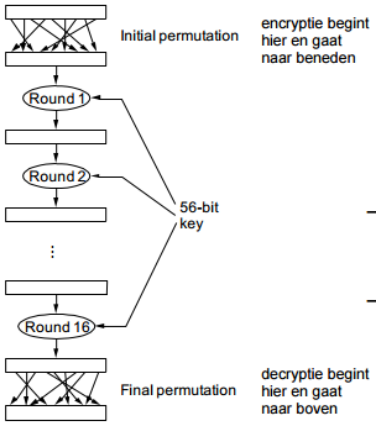
Improvement: Cipher Triple DES (3DES) leverages the cryptanalysis resistance of DES while in effect increasing the key size. A 3DES key has 168 (= 3x56) independent bits, and is used as three DES keys.

It is used as three DES-keys: DES-key1, DES-key2, DES-key3.

- 3DES **encryption** of a block is performed by first DES encrypting the block using DES key1, then DES decrypting the result using DES-key2, and finally DES encrypting that result using DES-key3.
- **Decryption** involves decrypting using DES-key3, then encrypting using DES-key2, then decrypting using DES-key1.

Shortcomings: Software implementations of DES/3DES are slow because it was originally designed for implementation in hardware.

Rounds: 16



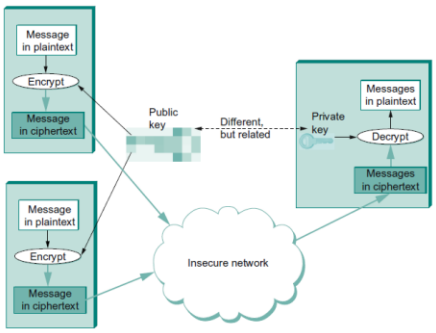
2.3 Public-key Ciphers

Symmetric-key ciphers >< Asymmetric key ciphers (public-key)

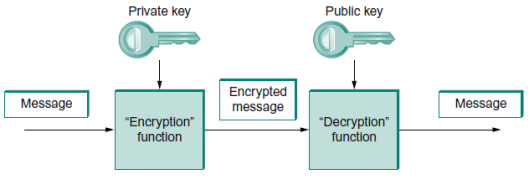
Instead of a single key shared by two participants, a public-key cipher uses a pair of related keys, one for encryption and a different one for decryption. The pair of keys is "owned" by just one participant. The owner keeps the decryption key secret so that only the owner can decrypt messages. That key is called the **private key**.
The owner makes the encryption key public, so that anyone can encrypt messages for the owner **(= public-key).**

⇨ For such a scheme to work, it must not be possible to deduce the private key from the public key. Consequently, any participant

can get the public key and send an encrypted message to the owner of the keys, and only the owner has the private key necessary to decrypt it.



- A public/private key pair provides a channel that is one way and many to one from everyone who has the public key to the owner of the private key.
- The private "decryption" key can be used with the encryption algorithm to encrypt messages so that they can only be decrypted using the public encryption key. However, this property wouldn't be useful for confidentiality since anyone with the public key could decrypt such a message.
  This property is however useful for authentication since it tells the receiver of such a message that it could only have been created by the owner of the keys.



Anyone with the public key can decrypt the encrypted message, and, assuming that the result of the decryption matches the expected result, it can be concluded that the private key must have been used to perform the encryption.

The best-known public-key cipher is RSA. RSA relies on the high computational cost of factoring large numbers.

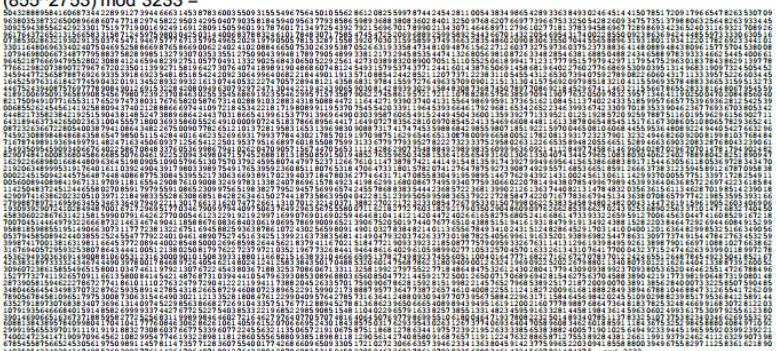
RSA needs relatively large keys, at least 1024 bits, to be secure. This is larger than keys for symmetric-key ciphers because it is faster to break an RSA private key by factoring the large number on which the pair of keys is based than by exhaustively searching the key space.

RSA: example

Key generation
p = 61, q = 53 (only 6 bits), n = 3233 (12 bits), e = 17, d = 2753,

Encryption
plaintext = 123
(123^17) mod 3233 = 337587917446653715596592958817679803 mod 3233 = 855

Decryption (naive)
(855^2753) mod 3233 =



= 123

⇨ Public-key ciphers are several orders of magnitude slower than symmetric-ley ciphers. Consequently, symmetric-key ciphers are used for the vast majority of encryption, while public-key ciphers are reserved for use in authentication and session key establishment.

2.4 Authentication

⇨ Integrity and authentication are fundamentally inseparable

Encryption alone does not provide data integrity. For example, just randomly modifying a ciphertext message could turn it into something that decrypts into valid-looking plaintext, in which case the tampering would be undetectable by the receiver. Nor does encryption alone provide authentication.

An **authenticator** is a value, to be included in a transmitted message, that can be used to verify simultaneously the authenticity and the data integrity of a message.

As we have seen before, checksum and CRCs are ways of detecting when a message has been inadvertently modified by bit errors. A similar concept applies to authenticators, with the added challenge that the corruption of the message is likely to be deliberately performed by someone who wants the corruption to go undetected.

An authenticator includes some proof that whoever created the authenticator knows a secret that is known only to the alleged sender of the message.

One kind of authenticator combines encryption and a cryptographic hash function. Cryptographic hash algorithms are treated as public knowledge, as with cipher algorithms. It is a function that outputs sufficient redundant information about a message to expose any tampering. A cryptographic checksum is designed to expose deliberate corruption of

messages by an adversary. The value it outputs is called a **message digest** and, like an ordinary checksum, is appended to the message.

An authenticator can be created by encrypting the message digest. The receiver computes a digest of the plaintext part of the message and compares that to the decrypted message digest. If they are equal, then the receiver would conclude that the message is indeed from its alleged sender and has not been tampered with. No adversary could get away with sending a bogus message with a matching bogus digest because she would not have the key to encrypt the bogus digest correctly.
An adversary could obtain the plaintext original message and its encrypted digest by **eavesdropping**. The adversary could then compute the digest of the original message and generate alternative messages looking for one with the same message digest. If she finds one, she could undetectably send the new message with the old authenticator.

- ⇨ Security requires that the hash function has the **one-way property**. In other words, it must be computationally infeasible for an adversary to find any plaintext message that has the same digest as the original.
  How to ensure this: its outputs must be fairly randomly distributed. Otherwise, if the outputs are not randomly distributed, that is, if some outputs are much more likely than others, then for some messages you could find another message with the same digest much more easily than this, which would reduce the security of the algorithm.

- ⇨ The algorithm itself is considered public knowledge which means that anyone could adapt this algorithm.

  Examples:

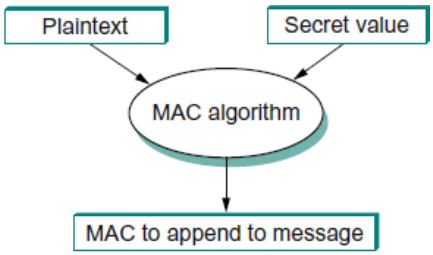
  - Message Digest 5 (MD5): 128-bit checksum. Now obsolete.
  - Secure Hash Algorithm 1 (SHA-1): 160-bit digest.

When generating an encrypted message digest, the digest encryption could use either a symmetric-key cipher or a public-key cipher. If a public-key cipher is used, the digest would be encrypted using the sender's key private key, and the receiver, or anyone else, could decrypt the digest using the sender's public key.

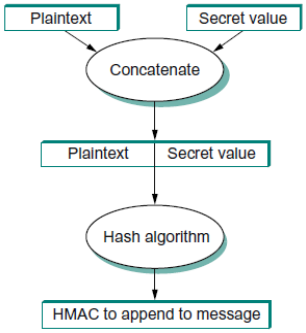
A digest encrypted with a public key algorithm but using the private key is called a **digital signature** since it provides nonrepudiation like a written signature.

Any public-key cipher can be used for digital signatures. Digital Signature Standard is a digital signature format that has been standardized by NIST. DSS signatures may use any one of three public-key ciphers, one based on RSA, another on ElGamal, and a third called the Elliptic Curve Digital Signature Algorithm.

Another kind of authenticator is similar, but instead of encrypting a hash it uses a hash-like function that takes a secret value (known to only the sender and the receiver) as a parameter. Such a function outputs an authenticator called a **message authentication code** (**MAC**). The sender appends the MAC to her plaintext message. The receiver recomputes the MAC using the plaintext and the secret value and compares that recomputed MAC to the received MAC.



A common variation on MACs is to apply a cryptographic hash to the concatenation of the plaintext message and the secret value. The resulting digest is called a **hashed message authentication code** (**HMAC**) since it is essentially a MAC. The HMAC, but not the secret value, is appended to the plaintext message. Only a receiver who knows the secret value can compute the correct HMAC to compare with the received HMAC.



3. Key predistribution

⇨ **Problem**: To use ciphers and authenticators, the communicating participants need to **know what keys to use**. In the case of a symmetric-key cipher, how does pair of participants obtain the key they share?
In the case of public-key cipher, how do participants know what public key belongs to a certain participant?

⇨ A and B need to know what keys to use
- How do A and B obtain the same secret key?
- How can A be sure that $Public_B$ really belongs to B?

⇨ The answer differs depending on whether the keys are **short-lived session keys** or **longer-lived predistributed keys**.

A session key is a key used to secure a single, relatively short episode of communication: a session. Each distinct session between a pair of participants uses a new session key, which is always a symmetric key for speed.

⇨ Participants determine what session key to use by means of a protocol: a **session key establishment protocol**. This protocol needs its own security. That security is based on the longer-lived predistributed keys.

⇨ There are several motivations for this division of labor between session keys and predistributed keys:

- **Limiting the amount of time** a key is used results in less time for computationally intensive attacks, less ciphertext for cryptanalysis, and less information exposed should the key be broken.
- Predistribution of symmetric keys is problematic.
- Public key ciphers are generally superior for authentication and session key establishment but too slow to use for encrypting entire messages for confidentiality.

3.1 Predistribution of Public Keys

Problem:

- How can A be sure that $Public_B$ is really the public key of B?
- If A knows B, they could get together to exchange $Public_B$.

Solution:

⇨ Use of certificates

One of the major standards for certificates is known as **X.509**. This standard leaves at lot of details open, but specifies a basic structure. A certificate clearly must include:

- The identity of the entity being certified
- The public key of the entity being certified (explicitly specified)
- The identity of the signer
- The digital signature
- A digital signature algorithm identifier

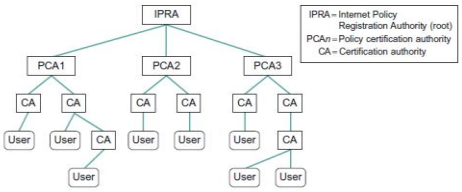
⇨ An optional component is an expiration time for the certificate.

CERTIFCATION AUTHORITIES

A Certification Authority (CA) is an administrative entity that issues certificates. It is useful only to someone that already holds the CA's public key. Much of today's Internet security depends on CAs.
A certification authority or certificate authority is an entity claimed to be trustworthy for verifying identities and issuing public key certificates. There are commercial CAs, governmental CAs, and even free CAs. To use a CA, you must know its own key. You can learn that CA's key, however, if you can obtain a chain of CA-signed certificates that starts with a CA whose key you already know. Then you can believe any certificate signed by that new CA.

⇨ How can A be sure that $Public_{Ca}$ is really the public key of CA?

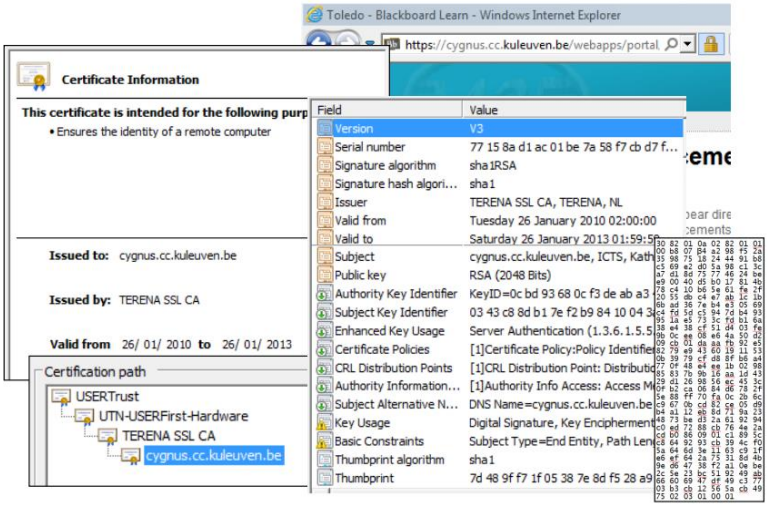
We enter in a kind of vicious circle. Indeed, a certification CA does also need a certification signed by someone. We enter in a **chain of trust**.
All you need is a chain of certificates, all signed by entities you trust, as long as it leads back to an entity whose key you already know.
Ex: if X certifies that a certain public key belongs to Y, and then Y goes on to certify that another public key belongs to Z, then there exists a chain of certificates from X to Z, even though X and Z may have never met. If you know X's key, and you trust X and Y, then you can believe the certificate that gives Z's key.



Tree-structured hierarchy. If everyone has the public key of the root CA, then any participant can provide a chain of certificates to another participant and know that it will be sufficient to build a chain of trust for that participant.
A complete scheme for certifying bindings between public keys and identities, what key belongs to who, is called a **Public Key Infrastructure (PKI)**. A PKI starts with the ability to verify identities and bind them to keys out of band.

Looking at the tree, we observe that IPRA is the root certification authority. We trust him for sure. There can be more than one root to a certification tree. This is common in securing Web transactions today. Browsers are pre-equipped with certificates of multiple trusted CAs.

There are some significant issues with building chains of trust. Even if you are certain that you have the public key of the root CA, you need to be sure that every CA from the root on down is doing its job properly. If just one CA in the chain is willing to issue certificates to entities without verifying their identities, then what looks like a valid chain of certificates becomes meaningless. For instance, a root CA might issue a certificate to a second-tier CA and thoroughly verify that the name on the certificate matches the business name of the CA, but that second-tier CA might be willing to sell certificates to anyone who asks, without verifying their identity.



WEB OF TRUST

An alternative model of trust is the **web of trust**. Ex: Pretty Good Privacy (PGP). PGP is a security system for email, so email addresses are the identities to which keys are bound and by which certificates are signed. In keeping with PGP's roots as protection against government intrusion, there are no CAs. Instead, every individual decides whom they trust and how much they trust them.

⇨ PGP recognizes that the problem of establishing trust is quite a personal matter and gives users the raw material to make their own decisions, rather than assuming that they are all willing to trust in a single hierarchical structure of CAs.

CERTIFICATE CLASSES

- Class 1: individuals → free, no guarantee. Anyone can buy this kin of certificates.
- Class 2: organizations → with identity proof
- Class 3: servers → identity and authority verified.
- Class 4: …

CERTIFICATE REVOCATION

Issue: how to revoke, or undo certificates?

⇨ Why is this important? If you suspect that someone has discovered your private key, there may be any number of certificates in the universe that assert that you are the owner of the public key corresponding to that private key. The person who discovered your private key has everything he needs to impersonate you: valid certificates and your private key.

Solution: it would be nice to be **able to revoke** the certificates that bind your old, compromised key to your identity, so that the impersonator will no longer be able to persuade other people that he is you.
Each CA can issue a **certificate revocation list** (CRL), which is a digitally signed list of certificates that have been revoked. The RL is periodically update and made publicly available. Since it is digitally signed, it can just be posted on a website.
Ex: when A receives a certificate for B that she wants to verify, she will first consult the latest CRL issued by the CA. as long as the certificate has not been revoked, it is valid.

3.2 Predistribution of Symmetric Keys

If A wants to use a secret-key cipher to communicate with B, she can't just pick a key and send it to him because, without already having a key, they can't encrypt this key to keep it confidential and they can't authenticate each other.

⇨ Predistribution is harder for symmetric keys than for public keys for **two reasons**:
- While only one public key per entity is sufficient for authentication and confidentiality, there must be a symmetric key for each pair of entities who shish to communicate. If there are N entities, that means N(N-1)/2 keys.
- Unlike public keys, secret keys must be kept secret.

4. Authentication protocols

⇨ Recall: it might seem as if all we have to do to make a protocol secure is append an authenticator to every message and, if we want confidentiality, encrypt the message.

Problems:

- **Replay attack:** an adversary retransmitting a copy of a message that was previously sent.
  Ex: if the message was an order you had placed on a website, then the replayed message would appear to the website as though you had ordered more of the same. Even though it wasn't the original incarnation of the message, its authenticator would still be valid. After all, this message was created by you and wasn't modified.
- **Suppress-replay attack**: an adversary might merely delay your message, so that it is received at a time when it is no longer appropriate.

Ex: an adversary could delay your order to buy stock from an auspicious time to a time when you would not have wanted to buy. Although this message would in a sense be the original, it wouldn't be timely.

- ⇨ **Originality** and **timeliness** may be considered aspects of integrity.
- ⇨ **Authentication problem**: if a message is not original and timely, then from a practical standpoint we want to consider it as not being authentic, not being from whom it claims to be. When you are arranging to share a new session key with someone, you want to know you are sharing it with the right person.

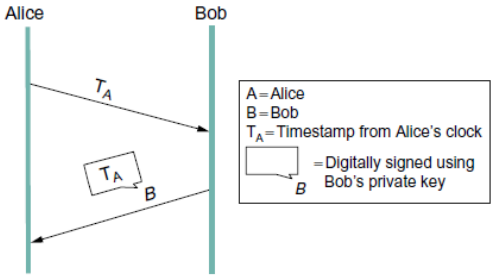
4.1 Originality and Timeliness Techniques

- ⇨ Authenticators alone do not enable us to detect messages that are not original or timely.
- ⇨ Challenge-response protocol

- One approach is to include a **timestamp** in the message. The timestamp itself must be tamperproof, so it must be covered by the authenticators.
  Drawback: they require **distributed clock synchronization**. The clock synchronization itself would need to be defended against security threats, in addition to the usual challenges of clock synchronization.
  Another issue is that distributed clocks are synchronized to only a certain degree, a certain margin of error. Thus, the timing integrity provided by timestamps is only as good as the degree of synchronization.
- Another approach is to include a **nonce**, a random number used only once, in the message. Participants can then detect replay attacks by checking whether a nonce has been used previously.
  Drawback: this requires keeping track of past nonces, of which a great many could accumulate.

Solutions

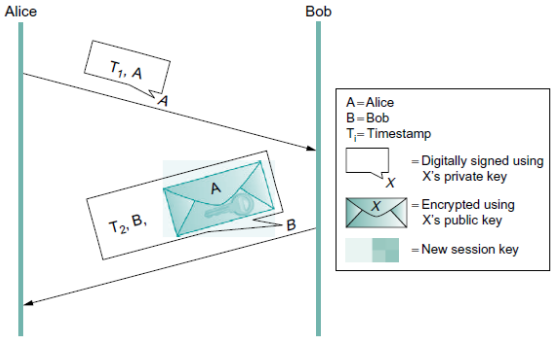
- Combine the use of timestamps and nonces, so that nonces are required to be unique only within a certain span of time.
  → makes ensuring uniqueness of nonces manageable while requiring only loose synchronization of clocks.
- Use one or both of them (timestamps and nonces) in a challenge-response protocol.



Alice sends Bob a timestamp, challenging Bob to encrypt it in a response message (if they share a symmetric key) or digitally sign it in a response message (if Bob has a public key). The encrypted timestamp is like an authenticator that additionally proves timeliness.
The challenge-response protocol combines timeliness and authentication.

4.2 Public-key authentication Protocols

The first protocol relies on Alice and Bob's clocks being synchronized. Alice sends Bob a message with a timestamp and her identity in plaintext plus her digital signature. Bob uses the digital signature to authenticate the message and the timestamp to verify its freshness. Bob sends back a message with a timestamp and his identity in plaintext, as well as a new session key encrypted using Alice's Public key, all digitally signed.

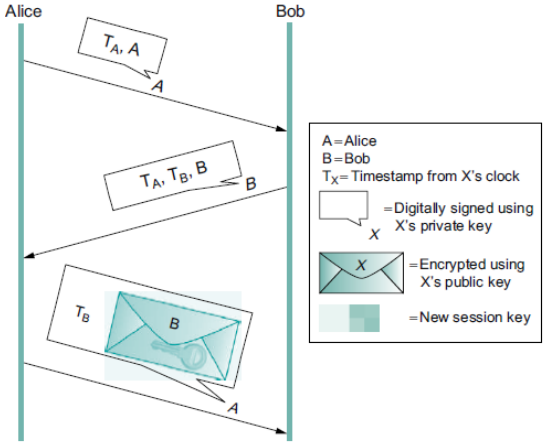


A public-key authentication protocol that depends on synchronization

Alice can verify the authenticity and freshness of the message, so she knows she can trust the new session key.

⇨ Based on timestamps from **synchronized clocks.**

In this protocol, Alice again sends Bob a digitally signed message with a timestamp and her identity. Because their clocks aren't synchronized, Bob cannot be sure that the message is fresh. Bob sends back a digitally signed message with Alice's original timestamp, his own new timestamp, and his identity. Alice can verify the freshness of Bob's reply by comparing her current time against the timestamp that originated with her. She then sends Bob a digitally signed message with his original timestamp and a new session key encrypted using Bob's public key.



A public-key authentication protocol that does not depend on synchronization

Bob can verify the freshness of the message because the timestamp came from his clock, so he knows he can trust the new session key.

⇨ Based on timestamps or nonces, but **no synchronized clocks**.

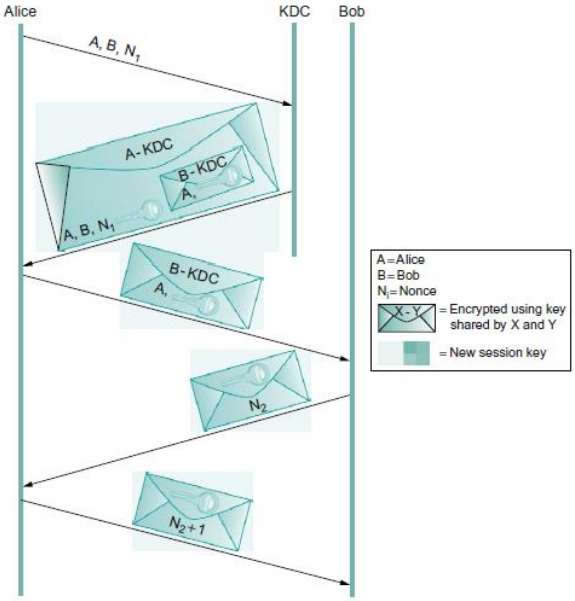
Problem: it doesn't work if the third party has no key with a certificate.

4.3 Symmetric-Key Authentication Protocols

⇨ Only in fairly small systems it is practical to predistribute symmetric keys to every pair of entities.

⇨ Now, focus on larger system, where each entity would have its own **master key** shared only with **a Key Distribution Center (KDC)**.

NEEDHAM-SCHROEDER AUTHENTICATION PROTOCOL

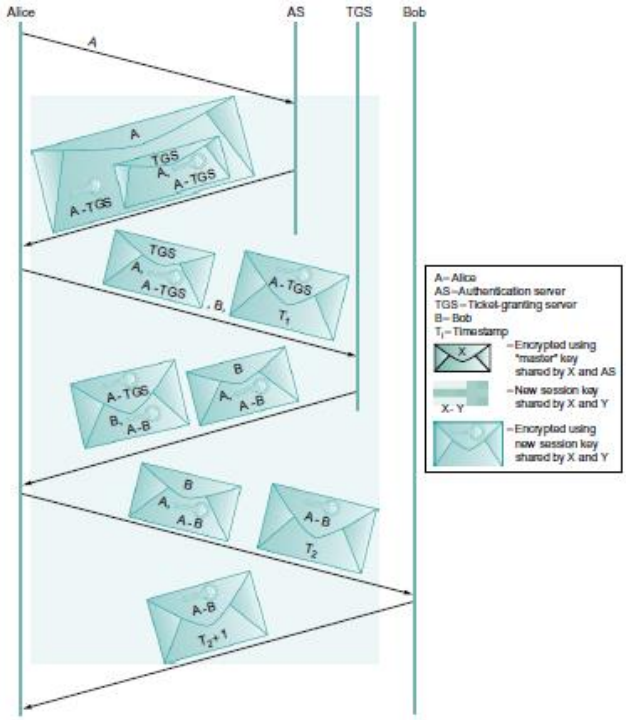


The KDC uses its knowledge of Alice's and Bob's master keys to construct a reply that would be useless to anyone other than Alice, because only Alice can decrypt it. This reply contains the necessary ingredients for Alice and Bob to perform the rest of the authentication protocol themselves.

- The nonce in the first two messages is to assure Alice that the KDC's reply is fresh.
- One key per person, only known by the person itself and the trusted third party.
- The second and third messages include the new session key and Alice's identifier, encrypted tighter using Bob's master key. It is a sort of symmetric-key version of a public-key certificate.
- Although the nonce in the last two messages is intended to assure Bob that the third message was fresh, there is a flaw in the reasoning.

KERBOS AUTHENTICATION

⇨ Specialized for client/server environments. This IETF standard is based on Needham-Schroeder.



Alice's master key, shared with the KDC, is derived from her password. If you know the password, you can compute the key.
Therefore, it is important to minimize the exposure of Alice's password or master key not just in the network but also on any machine where she logs in.
In Needham-Schroeder, the only time Alice needs to use her password is when decrypting the reply from the KDC. Kerberos client-side software waits until the KDC's reply arrives, prompts Alice to enter her password, computes the master key and decrypts the KDC's reply, and then erases all information about the password and master key to minimize its exposer.

The only sign a user sees of Kerberos is when the user is prompted for a password.

4.4 Diffie-Hellman Key Agreement

- ⇨ Establishes a session key without using any predistributed keys. The messages exchanged between A and B can be read by anyone able to eavesdrop, and yet the eavesdropper won't know the session key that A and B end up with.
- ⇨ One of the main uses of Diffie-Hellman is the Internet Key Exchange (IKE) protocol.

The Diffie-Hellman protocol has two parameters, p and g, both of which are public and may be used by all the users in a particular system.

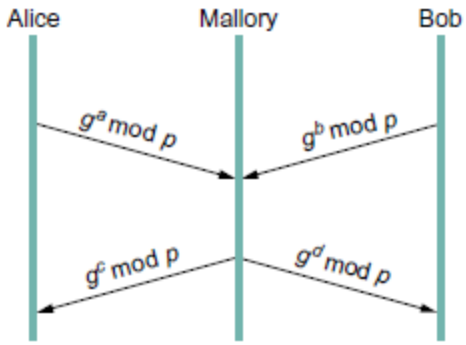
- P must be a prime number. The integers mod p is 0 through p-1, since x mod p is the remainder after x is divided by p.
- G must be a primitive root of p: for every number n from 1 through p-1 there must be some value k such that $n = g^k \bmod p$.

- ⇨ Suppose Alice and Bob want to **agree on a shared symmetric key**. Alice and Bob, and everyone else, already know the values of p and g.
  - Alice generates a random private value a and Bob generates a random private value b. Both a and b are drawn from the set of integers {1, . . . ,p−1}.

- Alice and Bob derive their corresponding public values, the values they will send to each other unencrypted, as follows. Alice's public value is $g^a \bmod p$ and Bob's public value is $g^b \bmod p$.
- They then exchange their public values. Finally, Alice computes $g^{ab} \bmod p = (g^b \bmod p)^a \bmod p$ and Bob computes $g^{ba} \bmod p = (g^a \bmod p)^b \bmod p$.
- Alice and Bob now have $g^{ab} \bmod p$ (which is equal to $g^{ba} \bmod p$) as their shared symmetric key. Any eavesdropper would know p, g, and the two public values $g^a \bmod p$ and $g^b \bmod p$. If only the eavesdropper could determine a or b, she could easily compute the resulting key. Determining a or b from that information is, however, computationally infeasible for suitably large p, a, and b; it is known as the **discrete logarithm problem**.

Problem: It doesn't authenticate the participants, since it is rarely useful to communicate securely without being sure whom you are communicating with.
One attack that can take advantage of this is the man-in-middle attack.

Suppose Mallory is an adversary with the ability to intercept messages. Mallory already knows p and g since they are public and she generates random private values c and d to use with Alice and B, respectively. When Alice and Bob sends their public values to each other, Mallory intercepts them and sends her own public values. The result is that Alice and Bob each end up unknowingly sharing a key with Mallory instead of each other.



Variant: **fixed Diffie-Hellman**

⇨ Supports authentication of one or both participants.
It relies on certificates that are similar to public key certificates but instead certify the Diffie-Hellman public parameters of an entity. Such a certificate would assure B that the other participant in Diffie-Hellman is A. or else the other participant won't be able to compute the secret key, because she won't know the parameter a.
If both participants have certificates for their Diffie-Hellman parameters, they can authenticate each other. If just one has a certificate, then just that one can be authenticated.

Used in several security systems

- IPsec Internet Key Exchange (IKE)
- SSL/TLS

5. Example systems

⇨ Security has to be provided at many different layers. One reason is that different threats require different defensive measures, and this often translates into securing a different protocol layer.
Ex: if your main concern is with a person in the building next door snooping on you traffic as it flows between your laptop and your 802.11 access point, then you probably want security at the link layer.

- Application layer: Pretty Good Privacy (**PGP**)
  Provides security for electronic mail
- Transport layer: Secure Socket Layer (**SSL**)
  Providing a secure channel. This is the most important one. It also appears as quite transparant.
- Network layer: IP Security (**IPsec**)
  Optional in IPv4; mandatory in IPv6
- Datalink layer: IEEE 802.11i
  Securing a wireless link

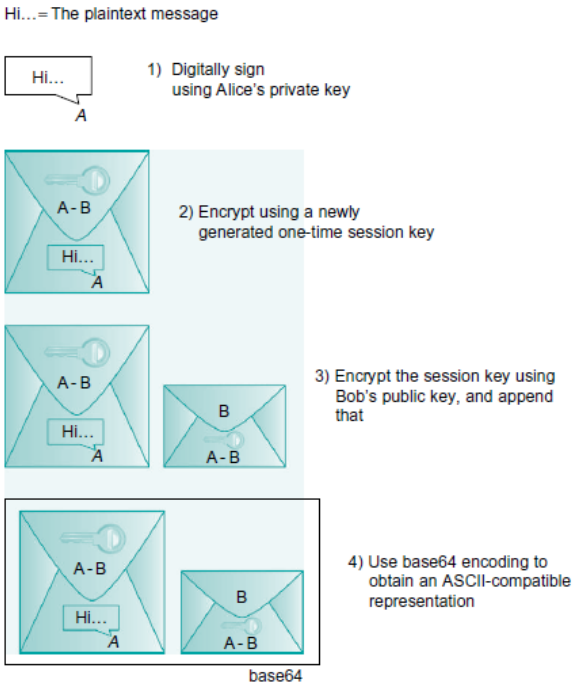
5.1 Pretty Good Privacy (PGP)

⇨ Widely used approach to provide security for electronic mail. It provides **authentication**, **confidentiality**, **data integrity**, and **nonrepudiation**. But no timeliness, no originality and no guaranteed delivery.
⇨ IETF standard
⇨ It combines multiple cryptographic algorithms: symmetric and asymmetric ciphers, cryptographic hashes.

PGP's confidentiality and receiver authentication depend on the receiver of an email message having a public key that is known to the sender. To provide sender authentication and nonrepudiation, the sender must have a public key that is known

by the receiver. These public keys are predistributed using certificates and a web-of-trust PKI

This is an example of PGP being used to provide both sender authentication and confidentiality. Suppose Alice has a message to email to Bob. Alice's PGP application goes through different steps.



1) The message is digitally signed by Alice. MD5, SHA-1 and SHA-2 are examples that may be used in the digital signature.
2) Her PGP application then generates a new session key for just this one message.
3) The digital signed message is encrypted using the session key, then the session key itself is encrypted using Bob's public key and appended to the message.
4) Alice's PGP application reminds her of the level of trust she had previously assigned to Bob's public key, based on the number of certificates she has for Bob and the trustworthiness of the individuals who signed the certificates.
5) A base64 encoding is applied to the message to convert it to an ASCII-compatible representation.

Upon receiving the PGP message in an email, Bob's PGP application reverses this process step-by-step to obtain the original plaintext message and confirm Alice's digital signature and reminds Bob of the level of trust he has in Alice's public key.

5.2 Secure Shell (SSH)

⇨ Used to provide a remote login service and is intended to replace the less secure Telnet and rlogin programs used in the early days of the Internet.
⇨ SSH is most often used to provide strong client/server authentication/message integrity, where the SSH client runs on the user's desktop machine and the SSH server runs on some remote machine that the user wants to log into, but it also supports confidentiality.

Ex: Telecommuters often subscribe to ISPs that offer high-speed cable modem or DSL service, and they use these ISPs, and some chain of other ISPs as well, to reach machines operated by their employer. This means that wen a telecommuter logs into a machine inside his employer's data center, both the passwords and all the data sent or received potentially passes through any number of untrusted networks.

SSH-TRANS

⇨ Provides an encrypted channel between the client and the server machines. It runs on top of a TCP connection. Any time a user uses an SSH application to log into a remote machine, the first step is to set up an SSH-TRANS channel between those two machines. The two machines establish this secure channel by first having the client authenticate the server using RSA.

⇨ Once authenticated, the client and server establish a session key that they will use to encrypt any data sent over the channel.

⇨ How does the client come to possess the server's public key that it needs to authenticate the server?
The server tells the client its public key at connection time. The first time a client connects to a particular server, the SSH application warns the user that it has never talked to this machine before and asks if the user wants to continue.
The SSH application then remembers the server's public key, and the next time the user connects to that same machine it compares this saved key with the one the server responds with.

- If they are the **same**, SSH authenticates the server.
- If they are **different**, SSH application again warns the user that something is amiss, and the user is then given an opportunity to abort the connection.

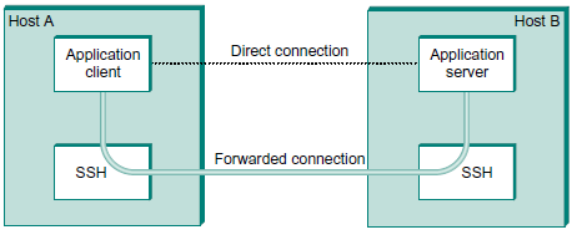
Once the SSH-Trans channel exists, the next step is for the user to actually log into the machine, or authenticate himself or herself to the server.

Three different mechanisms:

- Since the two machines are communicating over a secure channel, it is OK for the user to **simply send his or her password to the server**.
  This is not a safe thing to do when using Telnet since the password would be sent in the clear, but in the case of SSH the password is encrypted in the SSH-TRANS channel.
- Use of public-key encryption. This requires that the user has already placed his or her public key on the server.
- Host-based authentication. It says that any user claiming to be so-and-so from a certain set of trusted hosts is

automatically believed to be that same user on the server.

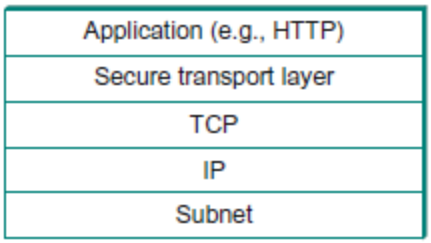
SSH has proven so useful as a system for securing remote login, it has been extended to also support other applications, such as sending and receiving email.
The idea is to run these applications over a secure "SSH tunnel".
= port forwarding



A client on host A indirectly communicates with a server on host B by forwarding its traffic through an SSH connection. This is called port forwarding because when messages arrive at the well-known SSH port on the server, SSH first decrypts the contents and then "forwards" the data to the actual port at which the server is listening.

5.3 Transport Layer Security (TLS, SSL, HTTPS)

- **Origin**: Web transactions needed some level of security. As the WWW became popular and commercial enterprises began to take an interest in it, it became clear that some level of security would be necessary for transactions on the Web.
  Ex: making purchases by credit card. There are several issues of concern when sending your credit card information to a computer on the Web. You might worry that the information would be intercepted in transit and used to make unauthorized purchases. You might also worry about the details of a transaction being modified.
- **Solution**: as this problem is not specific to Web transaction, we should build a general-purpose protocol that sits between an application protocol such as HTTP and a transport protocol such as TCP.



TLS looks like a normal transport protocol except for the fact that it is secure. That is, the sender can open connections and deliver bytes for transmission, and the secure transport layer will get them to the receiver with the necessary confidentiality, integrity, and authentication. By running the secure transport layer on top

of TCP, all of the normal features of TCP are also provided to the application.

When HTTP is used in this way, it is known as HTTPS (Secure HTTP).

THE HANDSHAKE PROTOCOL

⇨ The participants may negotiate the use of a compression algorithm because it is easy to do when you are negotiating all this other stuff and you are already decided to do some expensive per-byte operations on the data.

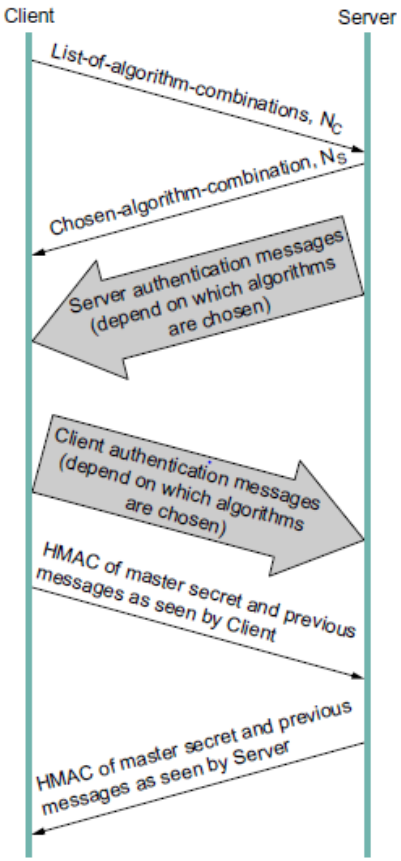
In TLS, the confidentiality cipher uses two keys, one for each direction, and similarly two initialization vectors.

Negotiation of parameters at runtime:

- Version? (SSL, TLS, 1.0, 2.0, …)
- Cryptographic algorithms
- Key establishment approach?
- Compression algorithm?

A TLS session requires six keys. TLS derives all of them from a single shared master secret. The master secret is a 384-bit value that in turn is derived in part from the "session key" that results from TLS's session key establishment protocol.
The part of TLS that negotiates the choices and establishes the shared master secret is called the **handshake protocol**. It is at heart of a session key protocol, with a master secret instead of a session key. Since TLS supports a choice of approaches to session key establishment, these call for correspondingly different protocol variants. Furthermore, the handshake protocol supports a choice between mutual authentication of both participants, authentication of just one participant, or no authentication at all.

⇨ The handshake protocol knits together several session key establishment protocols into a single protocol.



This figure shows the handshake protocol at a high level.

- The client initially sends a list of the combinations of cryptographic algorithms that it supports, in decreasing order of preference.
- The server responds, giving the single combination of cryptographic algorithms it selected from those listed by the client.
  - ➔ These messages also contain a **client nonce** and a **server nonce**, that will be incorporated in generating the master secret later.
- At this point, the negotiation phase is complete. The sever now sends additional messages based on the negotiated session key establishment protocol.
- Now the client and server each have the information necessary to generate the master secret. The "session key" that they exchanged is not in face a key, but instead a pre-master secret.

Problem: The negotiation itself is not protected which means that a man-in-the-middle attack is possible.

THE RECORD PROTOCOL

⇨ TLS's record protocol adds **confidentiality** and **integrity** to the underlying transport service.

- The record protocol uses an HMAC as an authenticator. The HMAC uses whichever hash algorithm was negotiated by the participants. The client and server have different keys to use when computing HMACs, making them even harder to break. Moreover, each record protocol message is assigned a **sequence number**, which is included when the HMAC is computed, even though the sequence number is never explicit in the message.
- Another interesting feature of the TLS protocol is the **ability to resume a session**. Each HTTP operation, such as getting a page of text or an image from a server, requires a

new TCP connection to be opened. Retrieving a single page with a number of embedded graphical objects might take many TCP connections. Once the TCP connection is ready to accept data, the client would then need to start the TLS handshake protocol, taking at least another two RTT before actual application data could be sent.

Reuse previously negotiated parameters: Session resumption is an optimization of the handshake that can be used in those cases where the client and the server have already established some shared state in the past. The client simply includes the session ID from a previously established session it its initial handshake message. If the server finds that it still has state for that session, and the resumption option was negotiated when that session was originally created, then the server can reply to the client with an indication of success, and data transmission can begin using the algorithms and parameters previously negotiated.
If the session ID does not match any session state cached at the server, of if resumption was not allowed for the session, then the server will fall back to the normal handshake process.

5.4 IP Security (IPsec)

⇨ Probably, the most ambitious of all the efforts to integrate security into the Internet happens at the IP layer.
⇨ Support for IPsec is optional in IPv4 but mandatory in IPv6.

IPsec provides <u>three degrees of freedom</u>

- Modular framework of all possible algorithms and protocols.
- Security services: allows users to select from a large menu of security properties: access control, integrity, authentication, originality, confidentiality, …

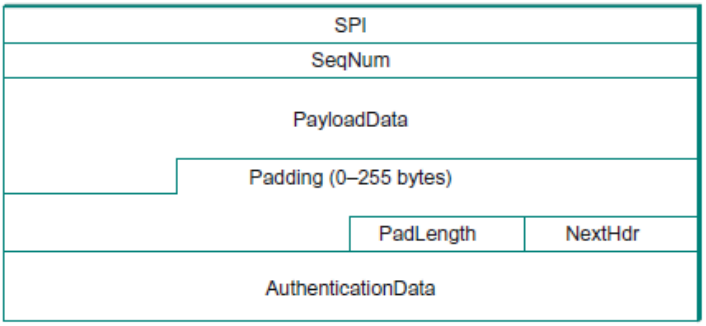
- Can be used to protect **narrow streams** (packets belonging to a particular TCP connection being sent between a pair of hosts) or **wide streams** (all packets flowing between a pair of routers).

IPsec consists of **<u>two parts</u>**

- Pair of protocols that implement the available security service
  - **AH** (Authentication Header), which provides access control, connectionless message integrity, authentication, and antireplay protection.
  - **Encapsulating Security Payload** (ESP), which supports these same services, plus confidentiality.
- Support for key management, which fits un an umbrella protocol: the Internet Security Association and Key Management Protocol (**ISAKMP**).

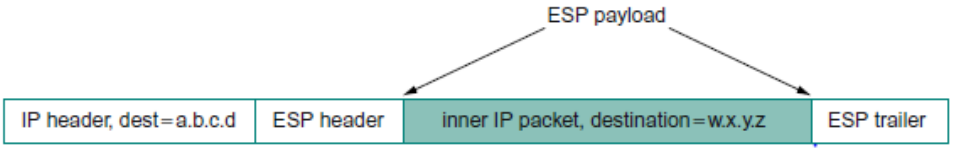
⇨ The abstraction that binds these two pieces together is the **security association (SA).**

- Simplex connection with one or more of the available security properties. Securing a bidirectional communication between a pair of hosts, e.g. corresponding to a TCP connection, requires two SAs, one in each direction.
- When created, an SA is assigned an ID number: security parameters index (SPI) by the receiving machine.
- SAs are established, negotiated, modified, and deleted using ISAKMP. It defines packet formats for exchanging key generation and authentication data.



⇨ In IPv4, the ESO header follows the IP header, in IPv6, it is an extension header. Its format uses both a header and a trailer.

- **SPI**: lets the receiving host identify the security association to which the packet belongs.
- **SeqNum**: protects against replay attacks.
- **PayloadData** contains the data described by the **NextHdr** field. If confidentiality is selected, then the data is encrypted using whatever cipher was associate with the SA.
- **PadLenght**: records how much padding was added to the data.
- **Trailer: AuthenticationData**: carries the authenticator. The Authenticator (HMAC) computed over previous part of ESP after encryption is performed. Decryption and integrity check can be done in parallel.



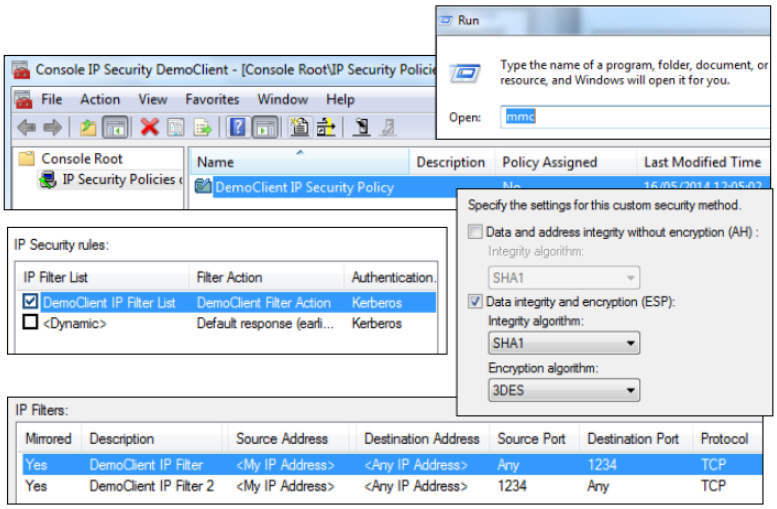
MODE

- **Tunnel mode:** ESP's payload data is itself an IP packet. The source and destination of this inner IP packet may be different from those of the outer IP packet. When an ESP message is received, its payload is forwarded on as a normal IP packet.
  - NextHeader = 4 (IPv4) or 41 (IPv6)
  - Typically used to build IPsec tunnel between routers (VPN)
  - Complete IP packet is encrypted (confidentiality) and authenticated.
- **Transport mode:** ESP's payload data is simply a message for a higher layer such as UDP or TCP. In this mode, IPsec acts as an intermediate protocol layer, much like SSL/TLS does between TCP and a higher

layer. When an ESP message is received, its payload is passed to the higher level protocol.

- NextHeader = 6 (TCP) or 17 (UDP)
- Cannot provide integrity, authenticity, confidentiality of entire packet (IP header not included).

Configuring IPsec



Selecting an IP VPN architecture

MPLS	IPsec	SSL
mesh-style site-to-site connectivity	site-to-site & remote-access supports multicast	typically remote-access (but OpenVPN claims site-to-site)
managed service offered by SP	originates/terminates at customer equipment	used for unmanaged computers
one-time provisioning of edge devices no VPN client required	special VPN client required for remote-access	only web browser required (but OpenVPN requires VPN client)
needs (expensive) MPLS-capable network	can be deployed across any IP network (although restricted by some SPs)	low cost of deployment
transparent to application	transparent to application	many standard applications not supported
traffic separation augmented with IPsec for additional security	strong security to all traffic: encryption, integrity, authentication, antireplay, ... DoS resilience	authentication & encryption of specific sessions DoS attack still possible
- mobility needs extensions	restricted by firewall/NAT mobility support (IKEv2)	easy NAT/firewall traversal session cannot survive roaming (?)
high scalability	acceptable scalability	server can become bottleneck
QoS: guaranteed bandwidth & traffic engineering	does not address QoS	does not provide QoS

this table is evolving continuously

5.5 Wireless Security (802.11i)

Problem: Wireless links are particularly exposed to security threats due to the lack of any physical security on the medium.

- There is no inherent physical protection
- Everyone in the neighborhood has "access" to the medium

    - Transmission can be overheard by anyone in range
    - Anyone can generate transmission
  - Replaying previously recorded messages is easy
  - Denial of service is easily achieved by jamming
  - Frame is carrying many details

⇨ There has been considerable work on securing WI-FI links. One of the early security techniques developed for 802.11 turned out to be seriously flawed and quite easily breakable.

The **IEEE 802.11i standard** provides authentication, message integrity, and confidentiality to 802.11 at the link layer. 802.11in includes definitions of first-generation security algorithms, including WEP, that are now known to have major security flaws.
802.11i supports two modes.

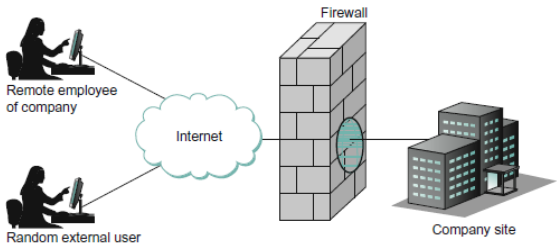
- **Weak authentication mode:** In either mode, the end result of successful authentication is a shared PairWise Master Key. Personal mode, also known as **Pre-Shared Key** (PSH) mode, provides weaker security bit is more convenient and economical for situations like a home 802.11 network.
  The wireless device and the Access Point (AP) are preconfigured with a shared passphrase.
- Stronger authentication mode: based on the IEEE 802.1X framework for controlling access to a LAN.
  AS and AP must be connected by a secure channel and could even be implemented as a single box, but they are logically separate.
  The protocol used for authentication is called the Extensible Authentication Protocol (EAP). It is designed to support multiple authentication methods.

⇨ 802.11i does not place any restrictions on what the EAP method can use as a basis for authentication. It does however require an EAP method that performs mutual authentication, because not only do we want to prevent an adversary from accessing the network via our APA, we also want to prevent an adversary from fooling our wireless devices with a bogus malicious AP.

One of the main differences between the stronger mode and the weaker personal mode is that the former readily supports a unique key per client. This in turn makes it easier to change the

set of clients that can authenticate themselves without needing to change the secret stored in every client.

6. Firewalls



⇨ System that sits at some point of connectivity between a site it protects and the rest of the network. It is usually implemented as an appliance or part of a router, although a "personal firewall" may be implemented on and end-user machine. Firewall-based security depends on the firewall being the only connectivity to the site from outside. There should be no way to bypass the firewall via other gateways, wireless connections, or dial-up connections.

⇨ By default, it blocks traffic unless that traffic is specifically allowed to pass through.

Why are firewalls needed?

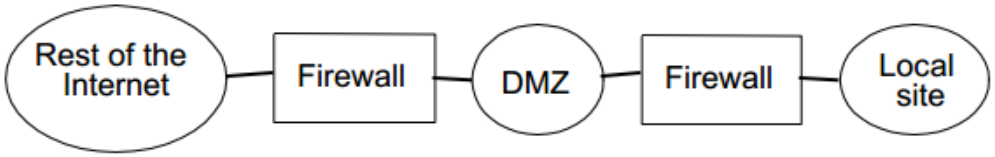
- Previous security mechanisms are not universally deployed
- Previous security mechanisms cannot be unilaterally enforced
- Single point of control for network administrators.

- A firewall **divides a network** into a **more-trusted zone** internal to the firewall and a **less-trusted zone** external to the firewall. This is useful if you do not want external users to access a particular host or service within your site.
- The complexity comes from the fact that you want to allow different kinds of access to different external users, ranging from the general public, to business partners, to remotely located members of your organization.
- A firewall may also impose restrictions on outgoing traffic to prevent certain attacks and to limit losses if an adversary succeeds in getting access inside the firewall.

Firewalls may be used to create multiple zones of trust, such as a hierarchy of increasingly trusted zones.

Three zones of trust:

- The internal network (more-trusted)
- The demilitarized zone (DMZ zone) with publicly accessible servers
- The rest of the Internet (less-trusted)



1st generation

- ⇨ Firewalls filter based on IP, TCP, and UDP information, among other things. They are configured with a table of addresses that characterize the packets they will, and will not forward.
- ⇨ Also called **level 4 switch**
- ⇨ Default for maximum security: do not forward unless it is **explicitly** allowed.

Ex: a firewall might be configured to filter out all packets that match the following description: < 192.12.13.14, 1234, 128.7.6.5, 80 >. This pattern says to:

- Discard all packets from part 1234 on host 192.12.13.14 addressed to port 80 on host 128.7.6.5

It is often not practical to name every source host whose packets you want to filer, therefore, patterns can include wildcards

Ex: < *, *, 128.7.6.5, 80 >

Problem: the firewall is **stateless**.

- Incoming packet won't be forwarded to the internal host (unless explicitly allowed).
- The response from the external server must be forwarded to internal client with dynamically assigned port.

2nd generation

A **stateful firewall** keeps track of the state of each connection. An incoming packet addressed to a dynamically assigned port would then be allowed only if it is a valid response in the current state of a connection on that part.

- External host cannot initiate connection setup, but it can respond to it.
- Incoming packet will not be forwarded, unless the connection is established.

It can help to prevent certain DoS attacks (SYN flood attack)

Problem: complex policy cannot be expressed as a filter. Ex: corporate web server is accessible to all external users, but some pages are restricted to remote corporate users.

6.1 Strengths and weaknesses of Firewalls

⇨ At best, a firewall protects a network from undesired access from the rest of the Internet.
⇨ Why are firewalls so comoon?

- Firewalls can be deployed unilaterally, using mature commercial products >< cryptography-based security requires support at both endpoints of the communication.
- Firewalls encapsulate security in a centralized place, factoring security out of the rest of the network.

Limitations:

- Since it does not restrict communication between hosts that are inside the firewall, the adversary who does manage to run code internal to a site can access all local hosts.
- Any parties granted access through your firewall, such as business partners or externally located employees, become a security vulnerability. If their security is not as good as yours, then an adversary could penetrate your security by penetrating their security.
- Vulnerability to the exploitation of bugs in machines inside the firewall.

Malware (Malicious software): software that is designed to act on a computer in ways concealed from and unwanted by the computer's

user. Ex: **viruses**, **worms**, and **spyware.** Viruses and worms are characterized by the ability to make and spread copies of themselves.

- **Worm**: complete program that replicates itself
- **Virus**: bit of code that is inserted into another piece of software or a file, so that it is executed as part of the execution of that piece of software or as a result of opening the file.

  Viruses and worms cause problems such as consuming network bandwidth as mere side effects of attempting to spread copies of themselves.

- Spyware: software that without authorization collecst and transmits private information about a computer system or its users. Usually it is secretly embedded in an otherwise useful program and is spread by users deliberately installing copies.

3rd generation: Application layer firewall

Related to firewalls are systems known as intrusion detection systems (IDS) and intrusion prevention systems (IPS). These systems try to look for anomalous activity, such as an unusually large amount of traffic targeting a given host or port number, and generate alarms for network managers or perhaps even take direct action to limit a possible attack.