Computer Networks + The Internet

Friday, 22 February 2019 5:02 PM

What is the Internet?

A Nut-and-Bolts Description

The Internet is a computer network that interconnected billions of computing devices throughout the world. These devices are called ***hosts*** or ***end systems***.

End systems are connected together by a network of ***communication links*** (including coaxial cable, copper wire, optical fiber, and radio spectrum) and ***packet switches***. Different links can transmit data at different rates, with ***transmission rate*** of a link measure in bits/seconds. When one end system has data to send to another end system, the sending end system segments the data and adds *header* bytes to each segment. These resulting ***packets*** are send through the network to the destination end system, where they are reassembled into the original data.

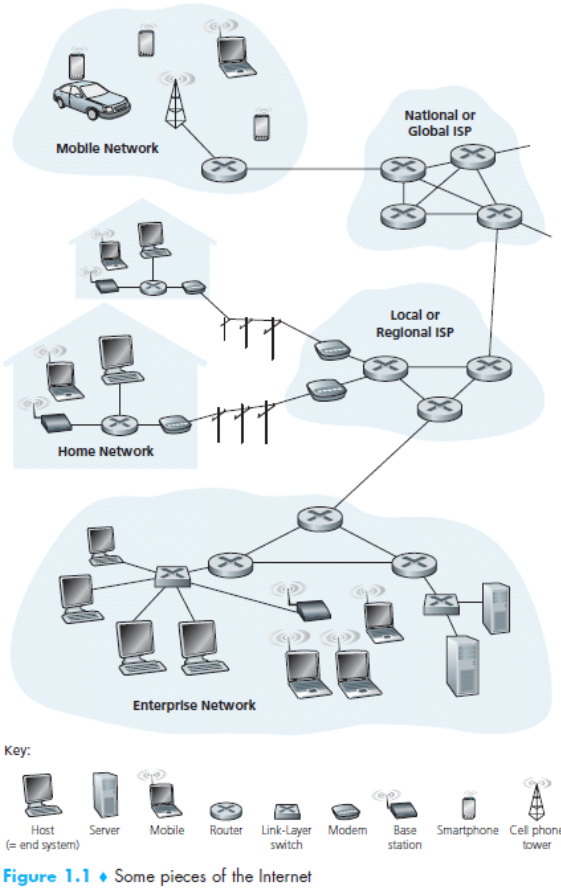


Figure 1.1 ♦ Some pieces of the Internet

A packet switch such as a ***router*** or ***link-layer switch*** takes a packet arriving on one of its incoming communication links and forwards that packet on one of its outgoing communication links. The sequence of communication links and packet switches traversed by a packet from the sending end system to the receiving end system is known as a ***route*** or ***path*** through the network.

End systems access the Internet through ***Internet Service Providers*** (ISPs). Each ISPs is in itself a network of packet switches and communication links. ISPs provide a variety of types of network access to the end systems, including residential broadband access such as cable modem or DSL, high-speed local area network access, and mobile wireless access. ISPs that provide access to end systems must also be interconnected.

End systems, packet switches, and other pieces of the Internet run ***protocols*** that control the sending and receiving of information within the Internet. The ***Transmission Control Protocol (TCP)*** and the ***Internet Protocol (IP)*** are the two of the most important protocol in the Internet.

Internet standards are developed by the Internet Engineering Task Force (IETF). The IETF standards documents are called ***request for comments (RFCs)***. The define protocols such as TCP, IP, HTTP, and SMTP.

A Service View

We can also describe the Internet as ***an infrastructure that provides services to application***. Applications include the Web, VoIP, email, games, e-commerce, music streaming, movie and television streaming, online social networks and others. Internet applications run on end systems - they do not run in the packet switches in the network core. Packet switches merely facilitate the exchange of data among end systems. Thus the Internet is analogous to a postal service that delivers messages to and from the application program

Network Protocols

A ***protocol*** defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.
Different protocol are used to accomplish different communication tasks.

The Network Edge

Recall, computers and other devices connected to the Internet are often called end systems. This is because they sit at the edge of the Internet.

Access Networks

The access network physically connect an end system to the first router (also known as the "edge router") on a path from the end system to any other distant end system.

The two most prevalent types of broadband residential access are ***digital subscriber line (DSL)*** and ***cable***. DSL uses existing telephone lines to exchange data with a digital subscriber line access multiplexer (DSLAM) located in the telco's local central office (CO). The residential telephone line carries both data and traditional telephone signals simultaneously, which are encoded at different frequencies:

- A high-speed downstream channel, in the 50 kHz to 1 MHz band
- A medium-speed upstream channel, in the 4 kHz to 50 kHz band
- An ordinary two-way telephone channel, in the 0 to 4 kHz band

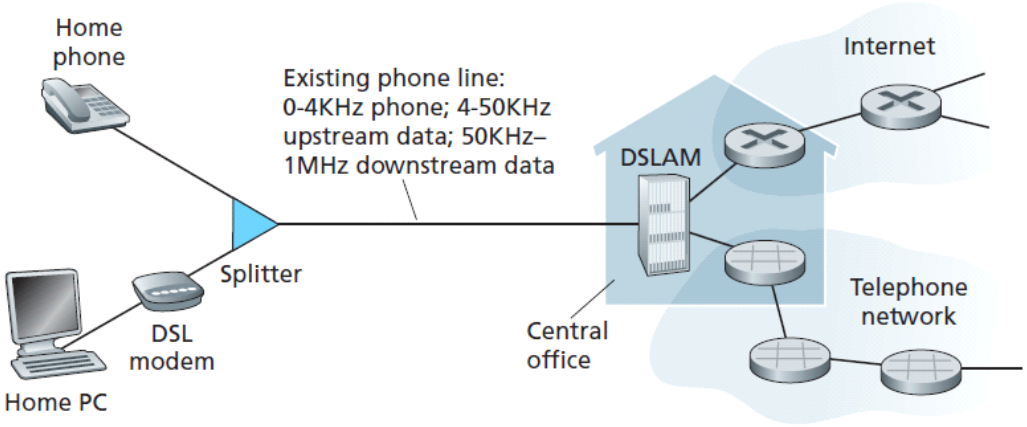


Figure 1.5 ♦ DSL Internet access

DSL typically provide:

- < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
- < 24 Mbps downstream transmission rate (typically < 10 Mbps)

Cable Internet access makes use of the cable television company's existing cable television infrastructure. A residence obtains cable Internet access from the same company that provides its cable television. Fiber optics connect the cable head end to neighbourhood-level junctions, from which traditional coaxial cable is then used to reach individual houses and apartments. Because both fiber and coaxial cable are employed in this system, it is often referred to as ***hybrid fiber coax (HFC)***.

Cable modems divide the HFC network into two channels, a downstream and an upstream channel.
This provides:

- up to 30Mbps downstream transmission rate
- 2 Mbps upstream transmission rate

It should be noted that cable Internet access is a shared broadcast medium unlike DSL, which has dedicated access to the CO.

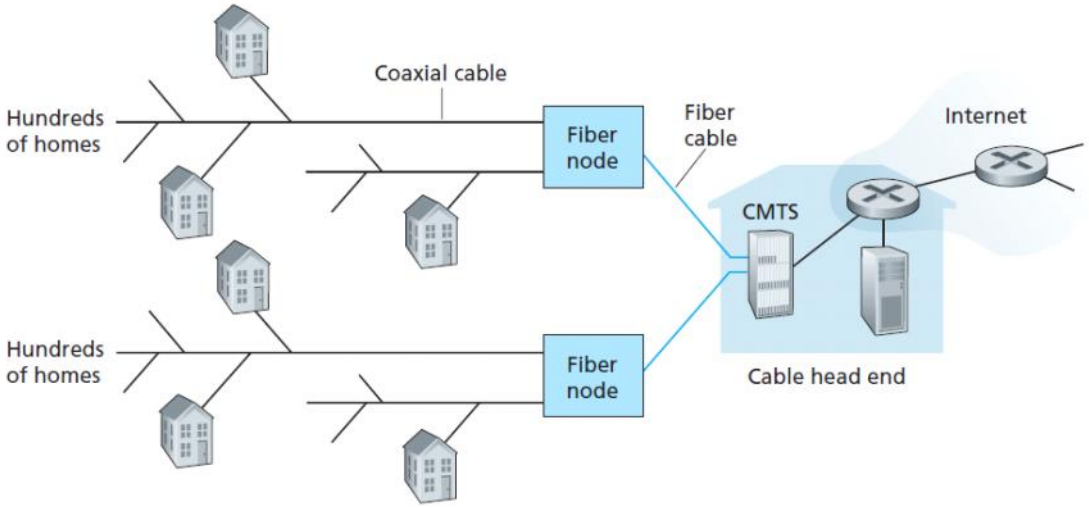


Figure 1.6 ♦ A hybrid fiber-coaxial access network

Fiber to the home (FTTH) provides a fully optical fiber path from the CO directly to the home. Fibers leaving the CO are commonly shared by many homes; it isn't until the fiber gets relatively close to the homes that it is split into individual customer-specific fibers. There are two competing optical-distribution network architectures that perform this splitting: ***active optical networks (AONs)*** and ***passive optical networks (PONs)***. AON is essentially switched Ethernet.

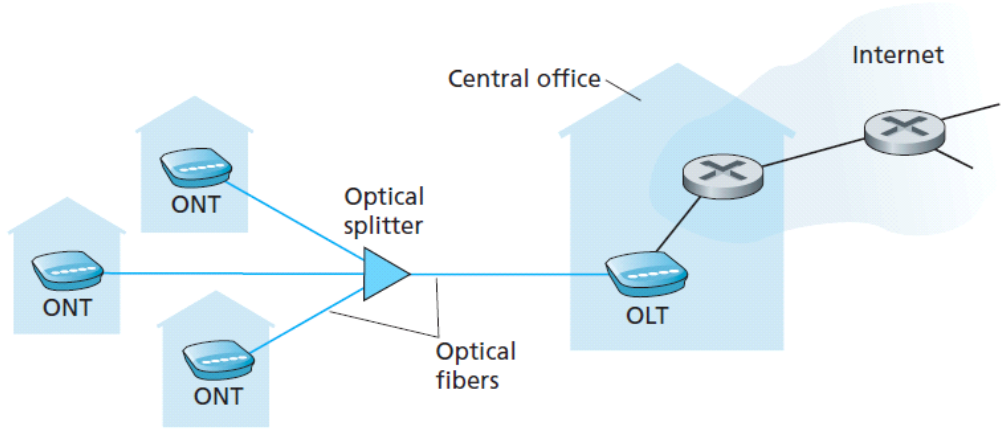


Figure 1.7 ♦ FTTH Internet access

On corporate and university campuses, and increasingly in home settings, a **local area network (LAN)** is used to connect an end system to the edge router. Although there are many types of LAN technologies, ***Ethernet*** is by far the most prevalent access technology in corporate, university, and home networks.
The Ethernet switch, or a network of such interconnected switches, is then in turn connected into the larger Internet. With Ethernet access, users typically have 100 Mbps access to the Ethernet switch, whereas servers may have 10 Mbps or even 10 Gbps access.

In a ***wireless LAN setting***, wireless users transmit/receive packets to/from an access point that is connected into the enterprise's network (most likely including wired Ethernet), which in turn is connected to the wired Internet. A wireless LAN user must typically be within a few tens of meters of the access point.

Wide-Area Wireless Access is provided through a base station that is operated by the cellular network provider. A user needs to eb within a few tens of kilometers of a base station to use the Internet. They typically have 1 to 10 Mbps access.

Physical Media

A bit, when traveling from source to destination, passes through a series of transmitter-receiver pairs. A ***physical medium*** is what lies between a transmitter-receiver pair. The physical media can be ***guided media*** or ***unguided media***. With guided media, the waves are guided along a solid medium, such as a fiber-optic cable, a twisted-pair copper wire, or a coaxial cable. With unguided media, the waves propagate in the atmosphere and in outer space,

such as in a wireless LAN or a digital satellite channel.

Twisted-Pair Copper Wire
Twisted pair consists of two insulated copper wires, each about 1 mm thick, arranged in a regular spiral pattern. The wires are twisted together to reduce the electrical interference from similar pairs close by. Typically, a number of pairs are bundled together in a cable by wrapping the pairs in a protective shield.
Category 5 cables can have 100Mbps to 1Gbps Ethernet data rates, while category 6 can have 10Gbps .

Coaxial Cable
Coaxial cable consists of two copper conductors, but the two conductors are concentric rather than parallel. They can provide users with Internet access at rates of tens of Mbps. Coaxial cable can be used as a guided ***shared medium***. Specifically, a number of end systems can be connected directly to the cable, with each of the end systems receiving whatever is sent by the other end systems.

Fiber Optics
An optical fiber is a thin, flexible glass medium that conducts pulses of light, with each pulse representing a bit. A single optical fiber can support tremendous bit rates, up to tens or even hundreds of gigabits per second. They are also immune to electromagnetic interference.

Radio
Radio channels carry signals in the electromagnetic spectrum. The characteristics of a radio channel depend significantly on the propagation environment and the distance over which a signal is to be carried. Reflection, obstruction by objects and electromagnetic interference (other transmission) can affect the signal.

Types of radio links:

- Terrestrial microwave - very short distances (1 or 2 meters). Have up to 45Mbps data rates
- LAN - ten to a few hundred meters, such as WiFi. Have 11, 54, 450Mbps data rates
- Wide-area - spanning tens of kilometers. Have approx. 10Mbps data rates
- Satellite - links two or more Earth-based microwave transmitter/receiver. The satellite receives transmissions on one frequency band, regenerates the signal using a repeater, and transmits the signal on another frequency. Two types of satellites are used in communications: ***geostationary*** satellites and ***low-earth orbiting (LEO)*** satellites.
  Geostationary satellites permanently remain above the same spot on Earth approximately 36,000 kilometers above Earth's surface. This introduces a substantial signal propagation delay of 280 milliseconds. LEO satellites are placed much closer to Earth and do not remain permanently above one spot on Earth. They rotate around Earth (just as the Moon does) and may communicate with each other, as well as with ground stations. The satellites typically have Kbps to 45Mbps channels (or multiple smaller channels)

The Network Core
The network core is a mesh of packet switches and links that interconnect the Internet's end systems.
There are two types of switched networks: packet switching and circuit switching.

Packet Switching
To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as ***packets***. Packets consist of a ***header*** and ***payload***. The payload is the data being carried, while the header hold instructions to the network for how to handle the packet such as the Internet Address, time to life (TTL) and checksum to protect the header. Between source and destination, each packet travels through communication links and ***packet switches*** (for which there are two predominant types, routers and link-layer switches).

Store-and-Forward Transmission
Most packet switches use store-and-forward transmission at the inputs to the links. Store-and-forward transmission means that the packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link.

It is possible to start transmitting a packet as soon as the header has been processed. This is known as ***cut-through***

switching.

Queuing Delays and Packet Loss

For each attached link in a packet switch, the switch has an *output buffer* (or an *output queue*), which stores packets that the router is about to send into that link. If an arriving packet needs to be transmitted onto a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer *queuing delays*. The buffer space is finite, so if an arriving packet finds that the buffer is completely full, *packet loss* will occur—either the arriving packet or one of the already-queued packets will be dropped.

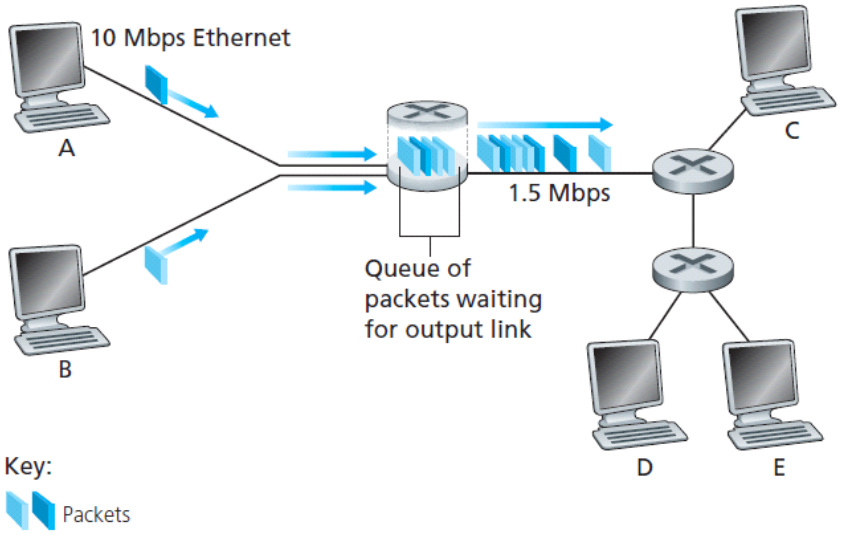


Figure 1.12 ♦ Packet switching

Forwarding Tables and Routing Protocols

When a packet arrives at a router in the network, the router examines the address, searches its forwarding table using the destination address to find the appropriate outbound link. The router then directs the packet to this outbound link. Each router has a *forwarding table* that maps destination addresses (or portions of the destination addresses) to that router's outbound links. The Internet has a number of special *routing protocols* that are used to automatically set the forwarding tables.

Circuit Switching

In circuit-switched networks, the resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are reserved for the duration of the communication session between the end systems. In packet-switched networks, these resources are not reserved; a session's messages use the resources on demand, and as a consequence, may have to wait (that is, queue) for access to a communication link.

When two hosts want to communicate, the network establishes a dedicated *end-to-end connection* between the two hosts. Thus, in order for Host A to communicate with Host B, the network must first reserve one circuit on each of two links.

Multiplexing

A circuit in a link is implemented with either *frequency-division multiplexing (FDM)* or *time-division multiplexing (TDM)*.

With FDM, the frequency spectrum of a link is divided up among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection. The width of the band is called *bandwidth*.

For a TDM link, time is divided into frames of fixed duration, and each frame is divided into a fixed number of time slots. When the network establishes a connection across a link, the network dedicates one time slot in every frame to this connection. These slots are dedicated for the sole use of that connection, with one time slot available for use (in every frame) to transmit the connection's data.

Packet switching is implemented using ***statistical-time-division multiplexing (STDM)***, which allows the bandwidth to be divided arbitrarily among a number of channels. This is different to TDM, where the number of channels and the channel data rates are fixed. STDM analyses statistics and determines on-the-fly how much time each device should be allocated for data transmission on the cable or line. It relies on the assumption that not all channels will be used at the same time, and instead allows devices to use all channels. If the link capacity does overload, then packets will merely queue into a buffer and wait their turn to be used. If there are persistent overloads, packets will be dropped.

Packet Switching vs Circuit Switching

Packet Switching	Circuit Switching
Pros: • Better sharing of transmission capacity • Simpler, more efficient, and less costly to implement Cons: • Not suitable for real-time services due to variable end-to-end delays	Pros: • Entire bandwidth reserved for driver Cons: • Bandwidth wastable • Can support a limited no. of users

A Network of Networks

End systems connect to the Internet via access ISPs (residential, company and university ISPs). The access ISPs must be interconnected so that any two hosts can send packets to each other. This is done by creating a ***network of networks***. Other the years, the network of networks that form the Internet has become a complex structure. The evolution was driven by economics and national policy.

This brings the question, how can we connect each access ISP to every other access ISP?
We can connect each access ISP to a global transit ISP. The access ISP is said to be the ***customer***, while the global transit ISP is said to be a provider. But if one global ISP is a viable business, there will be competitors. These competitors also need to be connect to each other. Regional ISP networks may also arise, in which the access ISPs in regions connect. Content providers such as Google, Microsoft may also run their own networks to bring services and content to close end users.

At the centre of the network of networks there is a small no. of well-connected large networks. They are known as ***Tier-1 ISPs*** and they provide national coverage. ***Content provider networks*** are private networks that connect its data centres to the Internet, often bypassing tier-1, regional ISPs.

Delay, Loss, and Throughput in Packet-Switched Networks

Delay in Packet-Switched Networks

There are 4 types of delay:

- ***Processing*** - the time required to examine the packet's header and determine where to direct the packet. Also includes time to check for bit-level errors.
  Usually takes *microseconds or less*.
- ***Queuing*** - when the packet needs to wait to be transmitted onto the outgoing link. Time depends on the no. of earlier-arriving packets that are queued and waiting for transmission onto the link.
  Can be on *microseconds to milliseconds* in practice.
- ***Transmission*** - time required for the router to "push" out the packet. Packet can only transmitted when all the packets that came before it have been sent.
  Let ***L*** be the length of the packets by ***L*** bits.
  Let ***R bits/sec*** be the transmission rate.
  Transmission delay = ***L/R***
- ***Propagation*** - the time required to propagate from the beginning of link to the next router.
  Let ***d*** be distance between router A and router B.
  Let ***s*** be the propagation speed of the link.
  Propagation delay = ***d/s***

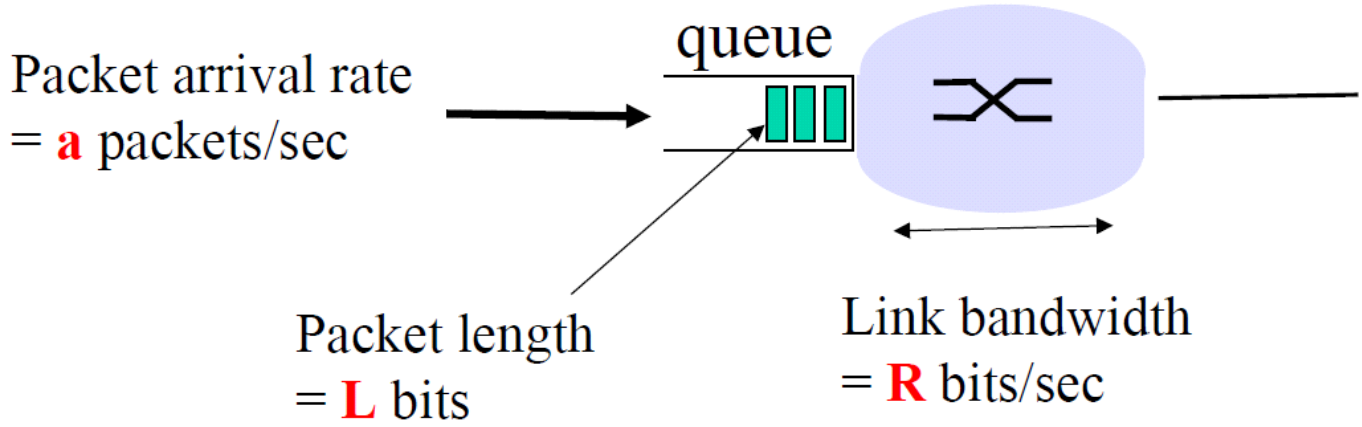
If we let d_{proc}, d_{queue}, d_{trans}, and d_{prop} denote the processing, queuing, transmission,and propagation delays, then the total nodal delay is given by

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

The contribution of these delay components can vary significantly.

Queuing Delay and Packet Loss

Queuing delay can vary from packet to packet. Whether queueing delay is large or insignificant depends on the rate at which traffic arrived at the queue, the transmission rate of the link, and the nature of the arriving traffic (i.e. whether traffic arrives periodically or in bursts).



Let ***a*** (packets/sec) be the average arrival rate of packets in the queue.
Recall ***R*** (bits/sec) is the transmission rate of the link.
Suppose all packets consist of ***L*** bits.
Then the average rate at which the bits arrive at the queue is ***La*** bits/sec.
The ***traffic intensity*** is ***La/R.***

If:

- ***La/R*** ~ 0, the average queuing delay will be small
- ***La/R*** -> 1, the delays become large
- ***La/R*** > 1, the average delay will approach infinity

Packet Loss

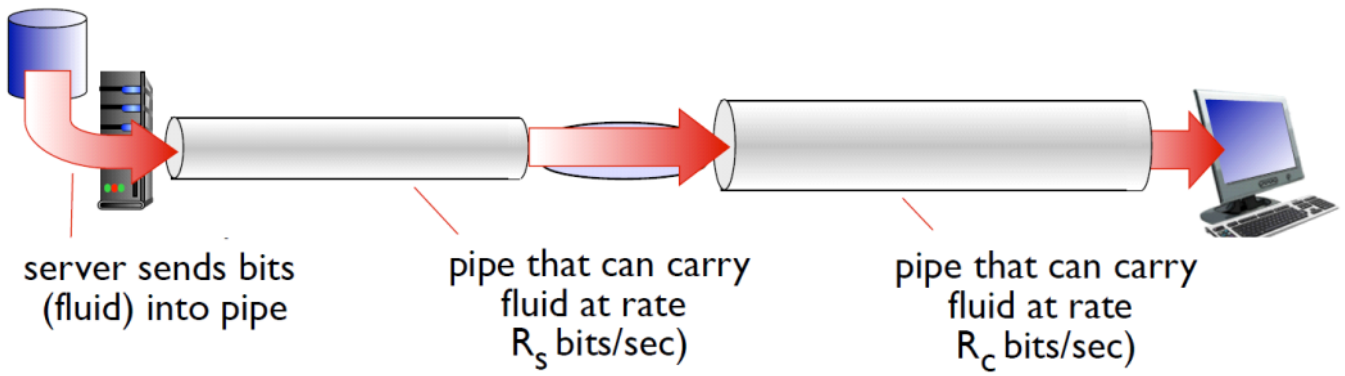
Queues have finite capacity, so when a packet arrives at a full queue, a router will **drop** that packet; so the packet will be lost. The lost packet may be retransmitted by the previous node, by the source end system, or it might not even be retransmitted.

Throughput in Computer Networks

Throughput is the rate (bits/time unit) at which bits are transferred from sender to receiver. ***Instantaneous throughput*** is the rate at a given point in time, which the ***average throughput*** is the rate over a longer period of time.

Consider transferring a large file from Host A to Host B across a computer network. This transfer might be, for example, a large video clip from one peer to another in a P2P file sharing system. The instantaneous throughput at any instant of time is the rate (in bits/sec) at which Host B is receiving the file.
If the file consists of F bits and the transfer takes ***T*** seconds for Host B to receive all ***F*** bits, then the average throughput of the file transfer is ***F/T*** bits/sec.

Consider the throughput for a file transfer from a server to a client.



Let R_s denote the rate of the link between the server and the router;
and R_c denote the rate of the link between the router and the client. Suppose that the only bits being sent in the entire network are those from the server to the client. Clearly, the server cannot pump bits through its link at a rate faster than R_s bps; and the router cannot forward bits at a rate faster than R_c bps.
If $R_s < R_c$, then the throughput will be R_s bps.
If $R_c < R_s$, then the throughput will be R_c bps.

For our simple two-link network, the throughput is $min\{R_c, R_s\}$, that is, it is the transmission rate of the ***bottleneck link***. So for a network with ***N*** links between a server and a client, with transmission rates of the ***N*** links being R_1, R_2, …, R_N. The throughput for the file transfer is $min\{R_1, R_2, …, R_N\}$.

Protocol Layers and Their Service Models

Protocol Layering

To provide structure to the design of network protocols, network designers organize protocols = and the network hardware and software that implement the protocols - in ***layers***. Each protocol belongs to one of the layers. Each layer provides a service by:

1. Performing certain actions within that layer
2. Using the services of the layer directly below it

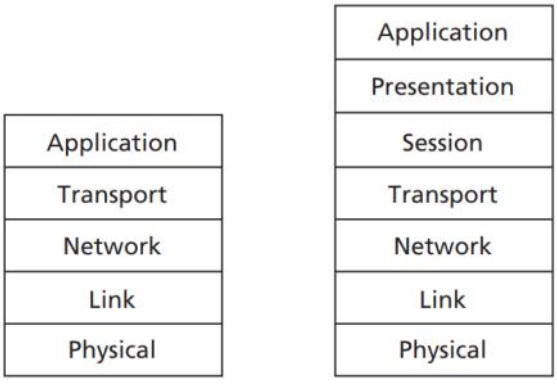
A protocol layer can be implemented in software, in hardware, or in a combination of the two.

Benefits of layering:

- Provides structured way to discuss system components
- Modularity makes it easier to update
- Don't need to worry about other layers e.g. (if you are an application developer you don't have to care about what links we are using
- You don't have to re-implement every network technology (the link mediums, the protocols)

Disadvantage:

- One layer may duplicate lower-layer functionality
- Functionality in one layer may need information that is only present in another layer *(which violates the goal of separation of layers)*
- Information hiding may hurt performance
- Headers start to get really big



a. Five-layer Internet protocol stack

b. Seven-layer ISO OSI reference model

Figure 1.23 ♦ The Internet protocol stack (a) and OSI reference model (b)

The protocols of various layers are called the ***protocol stack***. The Internet protocol stack consists of five layers; the physical, data link, network, transport and application layers.

Application Layer

The application layer is where network applications and their application-layer protocols reside. The Internet's application layer includes many protocols, such as the **HTTP** protocol (which provides for Web document request and transfer), **SMTP** (which provides for the transfer of e-mail messages), and **FTP** (which provides for the transfer of files between two end systems).

An application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system. We'll refer to this packet of information at the application layer as a ***message***.

Transport Layer

The Internet's transport layer transports application-layer messages between application endpoints.
In the Internet there are two transport protocols, **TCP** and **UDP**, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching). TCP also breaks long messages into shorter segments and provides a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested. The UDP protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. In this book, we'll refer to a transport-layer packet as a ***segment***.

Network Layer

The Internet's network layer routes a ***datagram*** through a series of routers between the source and destination. The Internet transport-layer protocol (TCP or UDP) in a source host passes a transport-layer segment and a destination address to the network layer. The network layer then provides the *service of delivering the segment* to the transport layer in the destination host.
The Internet's network layer includes the celebrated IP Protocol, which defines the fields in the datagram as well as how the end systems and routers act on these fields. There is only **one IP protocol**, and all Internet components that have a network layer must run the IP protocol. The Internet's network layer also contains many **routing protocols** that determine the routes that datagrams take between sources and destinations.

Link Layer

The network layer relies on the services of the link layer to make a packet from one node (host or router) to the next node in the route. At each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the datagram up to the network layer.
The services provided by the link layer depend on the specific link-layer protocol that is employed over the link. Examples of link layer protocols include **Ethernet**, **WiFi**, and the cable access networks **DOCSIS** protocol. A datagram may be handled by different link layer protocols at different links along its route.

Physical Layer

While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the *individual bits* within the frame from one node to the next. The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link (for example, twisted-pair copper wire, single-mode fiber optics).

Encapsulation

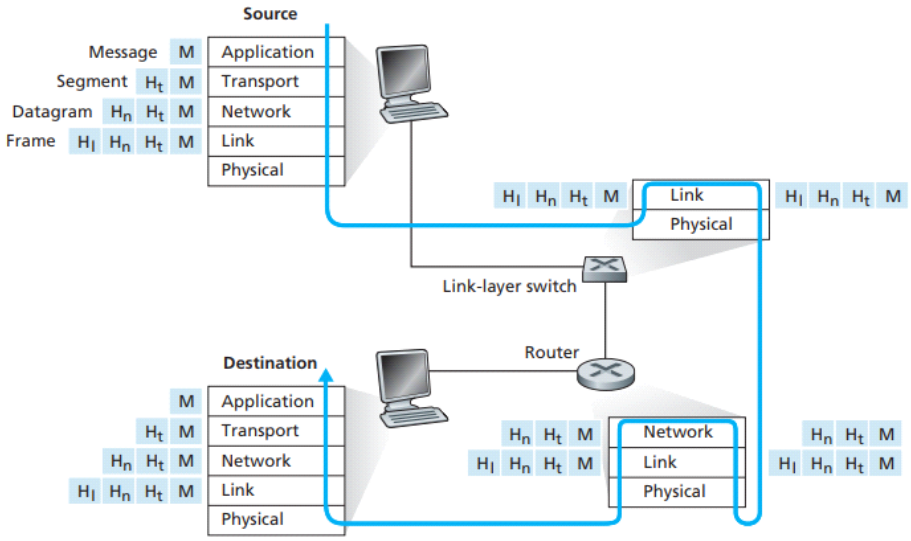


Figure 1.24 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

The figure above shows the physical path that data takes down a sending end system's protocol stack, up and down the protocol stacks of an intervening layer switch and router, and then up the protocol stack at the receiving end system.

At the sending host, an **application-layer message** (M in Figure 1.24) is passed to the transport layer. In the simplest case, the transport layer takes the message and appends additional information (so-called transport-layer header information, H*t*) that will be used by the receiver-side transport layer. The application-layer message and the transport-layer header information together constitute the **transport-layer segment**. The transport-layer segment thus ***encapsulates*** the application-layer message. The added information might include information allowing the receiver-side transport layer to deliver the message up to the appropriate application, and error-detection bits that allow the receiver to determine whether bits in the message have been changed in route. The transport layer then passes the segment to the network layer, which adds network-layer header information (H*n*) such as source and destination end system addresses, creating a **network-layer datagram**. The datagram is then passed to the link layer, which (of course!) will add its own link-layer header information and create a **link-layer frame**. Thus, we see that at each layer, a packet has two types of fields: header fields and a **payload field**. The payload is typically a packet from the layer above.

The process of encapsulation can be more complex than that described above.

Networks Under Attack

History

Applications Layer

Wednesday, 27 February 2019 12:09 PM

Principles of Network Application

At the core of network application development is writing programs that run on different end systems and communicate with each other over the network. E.g. a web server software communicates with browser software.

There are varying degrees of integration;

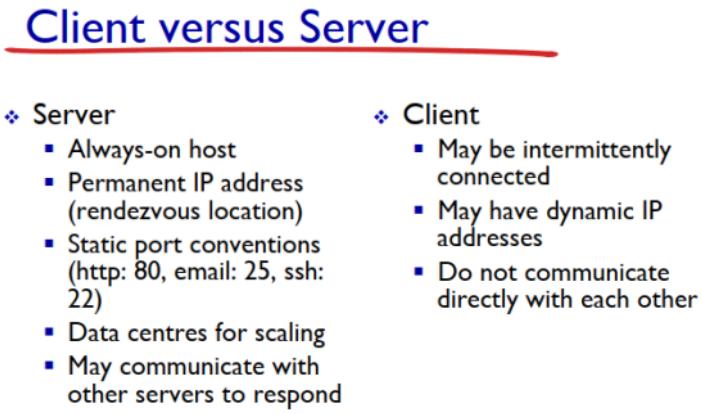
- Loose; email, web browsing
- Medium; chat, skype, remote file systems
- Tight; process migration, distributed file systems

When developing your new application, you need to write software that will run on multiple end systems. Importantly, you do not need to write software that runs on network-core devices, such as routers or link-layer switches.
Even if you wanted to write application software for these network-core devices, you wouldn't be able to do so, since network-core devices do not function at the application layer but instead function at lower layers— specifically at the network layer and below.

Network Application Architectures

Application architecture is designed by the application developer and dictates how the application is structured over the various end systems. There are two predominant architectural paradigms used in modern network applications: the ***client-server architecture*** or the ***peer-to-peer (P2P) architecture***.

In a ***client-server architecture***, there is an always-on host, called the ***server***, which services requests from many other hosts, called ***clients***.
A server exports a well-defined requests/response interface. It is a long-lived process that waits for requests. Upon receiving a request, it carries it out. Servers typically have a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address. If a single-server host is incapable of keeping up with all the requests from clients, they can use ***data centers***. A data center houses a large number of hosts, and is often used to create a powerful virtual server. They can have hundreds of thousands of servers, which must be powered and maintained.
A client is a short-lived process that makes requests. It is often the "user-side" of applications and initiates the communication.



In P2P architecture, there is minimal (or no) reliance on dedicated servers in data centres. Instead the application exploits direct communication between pairs of intermittently connected hosts, called ***peers***.
Benefits of P2P architecture:

- Each peer adds service capacity to the system
- Cost effective since it does not require significant server infrastructure and server bandwidth
- Speed - parallelism, less contention
- Reliability - redundancy, fault tolerance
- Geographic distribution

Disadvantages of P2P architecture:

- Decentralised structure and control:
  - State uncertainty - no shared memory or clock
  - Action uncertainty - mutually conflicting decisions
  - Security
  - Performance
  - Reliability?
- Distributed algorithms are complex

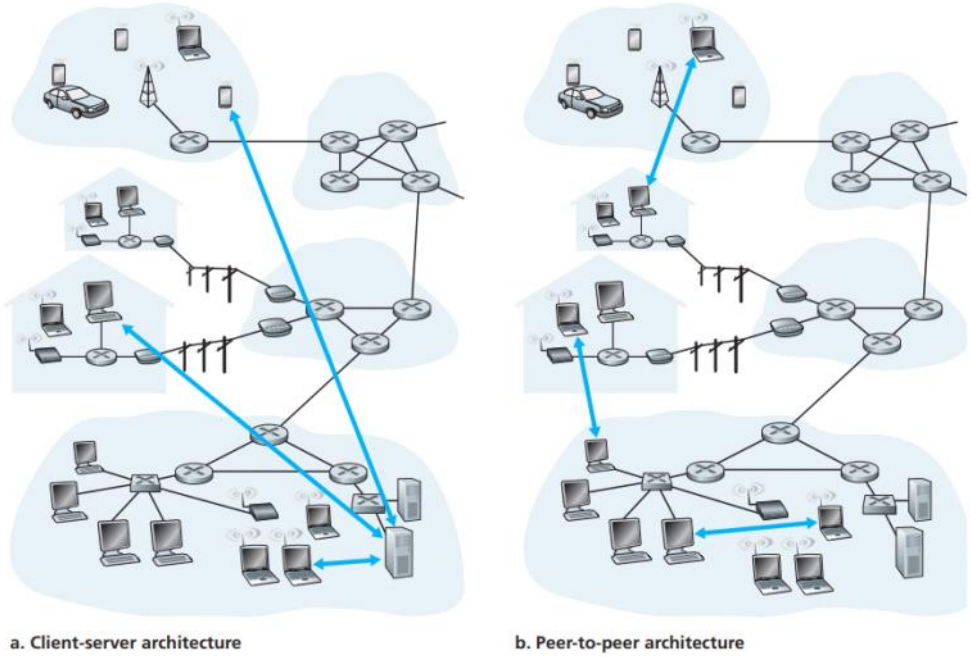


Figure 2.2 ♦ (a) Client-server architecture; (b) P2P architecture

Some applications have hybrid architectures, combining both client-server and P2P elements.

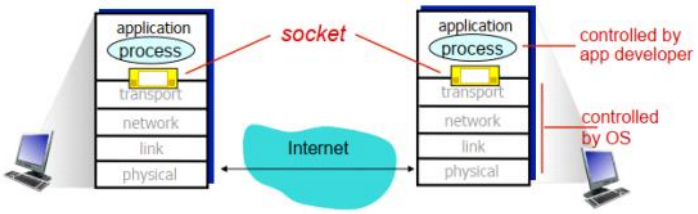
Process Communicating

It's not actually program but ***processes*** that communicate. A process can be thought of an a program that is running within an end system. When processes are running on the same end system, they can communicate with each other with ***interprocess communication (IPC)***, using rules that are governed by the end system's operating system.

Process on across machines communicate with each other by exchanging messages across the computer network; a receiving process receives these messages and possibly responds by sending messages back.

Processes send and receive messages through the network through a software interface called a ***socket***. A socket is the interface between the application layer and the transport layer within a host. It is also referred to as the Application Programming Interface (API) between the application and the network. The application developer has control of everything on the application-layer side of the socket, but has little control of the transport layer side of the socket. The only control they have on the transport-layer side is:

1. The choice of transport protocol (TCP or UDP)
2. Maybe the ability to fix a few transport-layer parameters such as max buffer and max segment sizes.



Addressing Processes

To receive messages, a process must have a ***identifier***. The identifier includes both the ***IP address*** (unique 32-bit address) and ***port numbers*** associated with the process on the host.

Transport Services Available to Applications

Reliable Data Transfer/Data Integrity

Some apps (email, file transfer, web transactions) require 100% ***reliable data transfer***. When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence tha the data will arrive without errors at the receiving process.

Other apps (audio, video streaming) can tolerate some loss. This is known as ***loss-tolerant applications***.

Throughput

Some apps (multimedia) require a minimum amount of throughput to be *effective*. The application could request a guaranteed throughput of ***r*** bits/sec, and the transport protocol would then ensure the available throughput is at least ***r*** bits/sec. Applications with throughput requirements are said to be ***bandwidth-sensitive applications***.

Other apps (email, file transfer) can make use of whatever throughput they get. These applications are known as ***elastic applications***.

Timing
Some apps (Internet telephony, interactive games) require low delay to be *effective*. Timing guarantees can be provided by transport-layer protocols. An example guarantee might be that every bit that the sender pumps into the socket arrives at the receiver's socket no more than 100ms later. For non-real-time applications, lower delay is always preferable to high delay, but no tight constraint is places on the end-to-end delays.

Security
A transport protocol can encrypt all data transmitted by the sending process, and in the receiving host, the transport-layer protocol can decrypt the data before delivering the data to the receiving process. A transport protocol can also provide other security services in addition to confidentiality, including data integrity and end-point authentication

Internet Transport Services
The Internet makes two transport protocols available to applications; ***UDP*** and ***TCP***. Each of these protocols offer a different set of services to the invoking applications:

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 50kbps-1Mbps video:100kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few msecs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
Chat/messaging	no loss	elastic	yes and no

TCP service:
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:
- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, orconnection setup,

Q: why bother? Why is there a UDP?

Notice that TCP and UDP do not provide throughput or timing guarantees of the 4 services mentions above. (TCP itself does not provide security, but can be used with ***Secure Sockets Layer (SSL)*** to provide security services.

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Application Layer Protocols
An ***application-layer protocol*** defines how an application's process, running on different end systems, pass messages to each other. In particular, an application-layer protocol defines:
- The ***types of messages exchanged*** - e.g. request messages and response messages
- The ***syntax*** of various message types - the fields in the message and how the fields are delineated
- The ***semantics*** - the meaning of information in the fields
- ***Rules*** for when and how processes send and respond to messages

Open protocols are defined in RFCs (request for comments). This allows for interoperability and they are available in the public domain. Examples include HTTP, SMTP.
Proprietary protocols are intentionally not available in the public domain. For example Skype uses proprietary application-layer protocols.

The Web and HTTP
Overview of HTTP
The ***HyperText Transfer Protocol (HTTP)***, is the Web's application-layer protocol. It is implemented in a client-server model, where the client is a browser that requests, receives and displays Web objects, and the server sends objects in response to requests.

A ***Web page*** (also called a ***document***) consists of objects. An ***object*** is simply a file - HTML file, JPEG image, Java applet etc. - that is addressable by a single URL. Most web pages consist of a ***base HTML file*** and several referenced objects. Each URL has two components: the ***hostname*** of the server that houses the object and the object's ***path name***. e.g:

www.someschool.edu/someDept/pic.gif

host name | path name

Uniform Resource Locator (URL)

`protocol://host-name[:port]/directory-path/resource`

- *protocol*: http, ftp, https, smtp, rtsp, *etc.*
- *hostname*: DNS name, IP address
- *port*: defaults to protocol's standard port; e.g. http: 80 https: 443
- *directory path*: hierarchical, reflecting file system
- *resource*: Identifies the desired resource

`protocol://host-name[:port]/directory-path/resource`

- Extend the idea of hierarchical hostnames to include anything in a file system
  - http://www.cse.unsw.edu.au/~salilk/papers/journals/TMC2012.pdf
- Extend to program executions as well…
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&Msgld=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
  - Server side processing can be incorporated in the name

HTTP uses TCP as its underlying transport protocol.

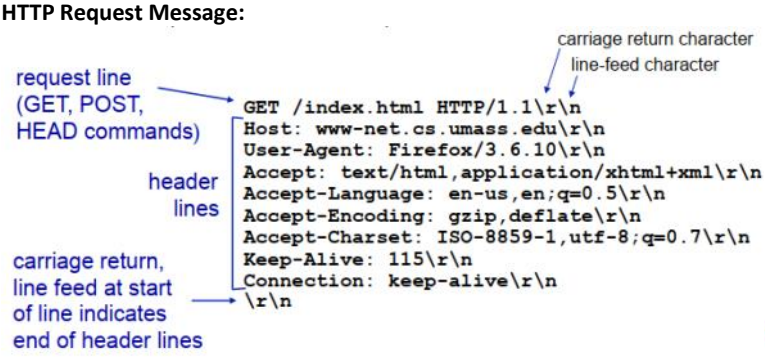
1. The HTTP client first initiates a TCP connection (creates socket) with the server.
2. The server accepts the TCP connection from the client.
3. HTTP messages are exchanged - the client sends HTTP request messages into its socket interface and receives HTTP response message from its socket interface. Similarly the server receives request messages from its socket interface and sends response messages into its socket interface.
4. The TCP connection is closed

HTTP is a ***stateless protocol***, meaning that the server does not maintain information about past client requests. Protocols that maintain state are complex. The past history (state) must be maintained. If a server/client crashes, their views of the state may be inconsistent and must be reconciled.

HTTP Message Format

There are two types of HTTP messages: request messages and response messages.

HTTP Request Message:



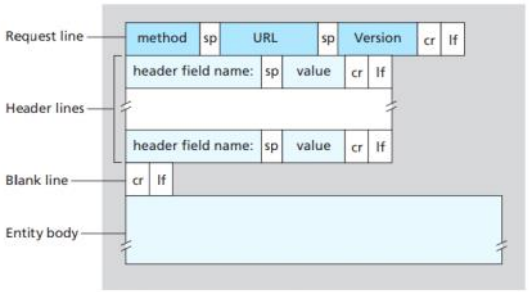


Figure 2.8 ♦ General format of an HTTP request message

HTTP request messages are written in ordinary ASCII text, so that us humans can read it. Each line in the message is ended with a carriage return and line feed (\r\n).

The first line of an HTTP request message is called a ***request line***; the subsequent lines are called ***header lines***. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE. The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field.

Header lines:

- Host: the host on which the object resides
- User-Agent: the browser type that is making the request
- Accept*: content negotiation header
- Connection: tell the server if you want to keep the connection alive after it has sent the requested object

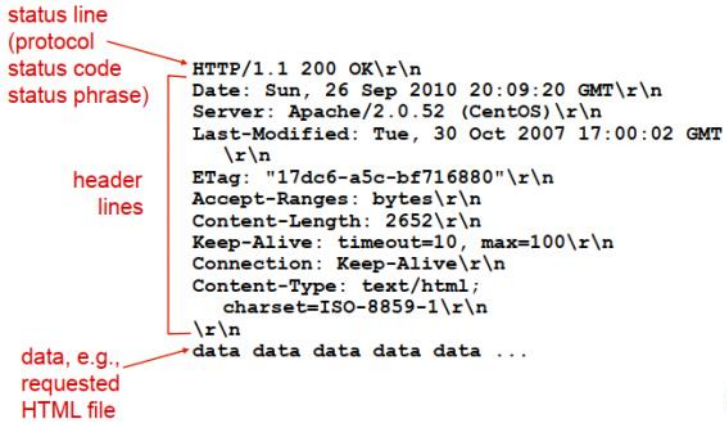
Request Method Types:

- GET - request a page
- POST - upload user's response to a form
- HEAD - ask server to leave request object out of response
- PUT - upload file in entity body to path specified in URL field
- DELETE - delete file specified in URL field
- TRACE, OPTIONS, CONNECT, PATCH - for persistent connections

When uploading form input, for the POST method, input is uploaded to the server in the entity body, while the GET

method uploads input in the URL field of request line; e.g. www.somesite.com/animalsearch?monkeys&banana

HTTP Response Message:



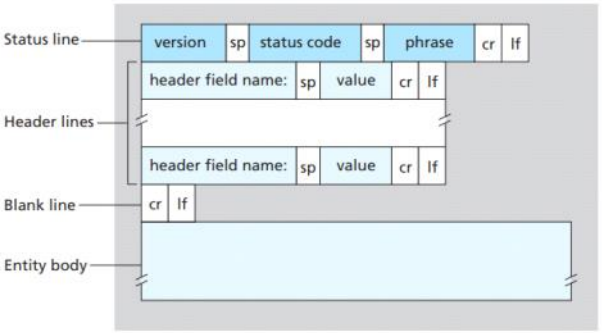


Figure 2.9 ♦ General format of an HTTP response message

It has three sections: an initial ***status line***, six ***header lines***, and then the ***entity body***. The entity body is the meat of the message—it contains the requested object itself (represented by data data data data data ...). The status line has three fields: the protocol version field, a status code, and a corresponding status message.

Some sample status codes:

200 OK
- request succeeded, requested object later in this msg

301 Moved Permanently
- requested object moved, new location specified later in this msg (Location:)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

451 Unavailable for Legal Reasons

429 Too Many Requests

418 I'm a Teapot

Non-Persistent and Persistent Connections

The client and server communicate for an extended period of time. Depending on the application and on how the application is being used, the series of requests may be ***back-to-back***, ***periodical***ly at regular intervals, or ***intermittent***ly. A application developer can decide whether they want the application to use ***non-persistent connections*** (request/response pairs sent over *separate* TCP connections) or ***persistent connections*** (all request and their corresponding responses send over the *same* TCP connection)

HTTP with Non-Persistent Connections

This allows at most one object to be send over a TCP connection, where once the client receives the HTTP response, the connection is closed. This means that downloading multiple objects require multiple connections as a brand new connection must be established and maintained for *each requested object***.**

It is possible to obtain objects through parallel TCP connections, but this does not necessarily maintain the order of the responses.

Calculating estimate response time:
RTT: time it takes for a small packet to travel from client to server and back to client
Response time
It takes:

- One RTT to initiate a TCP connections (client sends TCP segment to server, server acknowledges and sends TCP segment back)
- One RTT for HTTP request to be send and for first few bytes of HTTP response to return
- Then the server sends the file into the TCP connection at rate *file transmission time*

So, ***non-persistent HTTP response time = 2RTT + file transmission time***

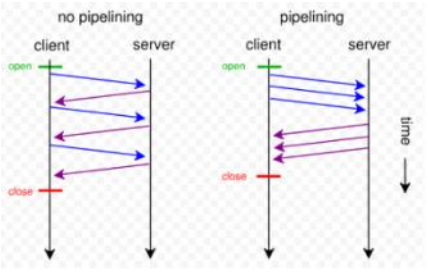
HTTP with Persistent Connections

The server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. This allows TCP to learn a more accurate RTT estimate and TCP congestion window to increase.

The requests for objects can be made back-to-back, without waiting for replies to pending requests (known as ***pipelining***)

and TCP congestion window to increase.

The requests for objects can be made back-to-back, without waiting for replies to pending requests (known as ***pipelining***). This is the default in HTTP/1.1. the client sends requests and as soon as it encounters a referenced objects (even one RTT for all the referenced objects).
Otherwise requests can be made without pipelining, where the client issues a new request only when the previous response has been received.



User-Server Interaction: Cookies

We mentioned that an HTTP server is stateless. HTTP uses ***cookies*** for the Web to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity.

Cookie technology has four components:

1. A cookie header line in the HTTP response message
2. A cookie header line in the HTTP request message
3. A cookie file kept on the user's end system, managed by the user's browser
4. A back-end database at the Web site

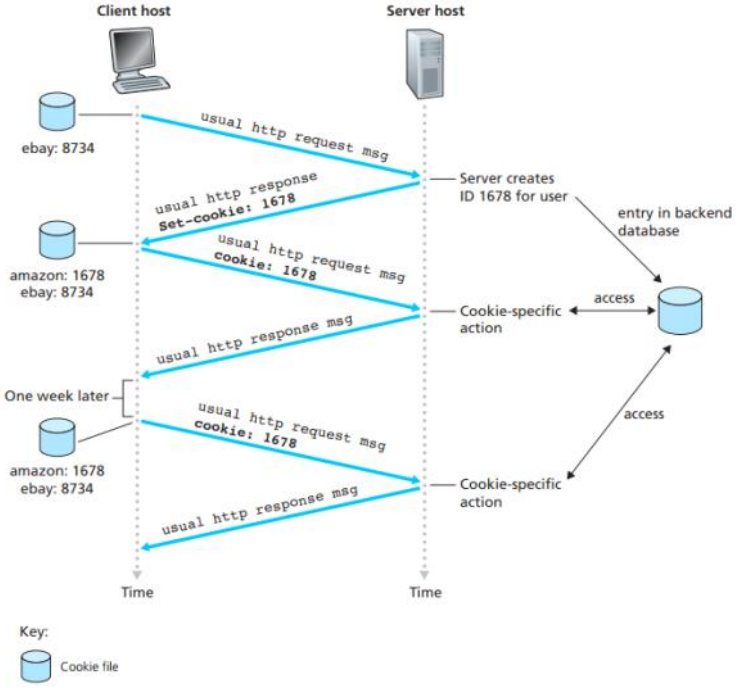


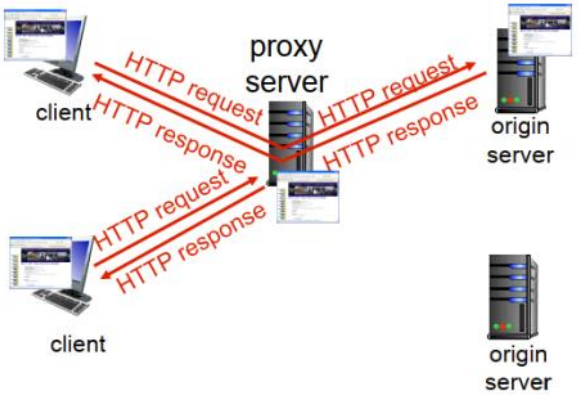
Figure 2.10 ♦ Keeping user state with cookies

Cookies allow sites to lean a lot about you. You can supply your name and email to sites (and more). Third party cookies (from ad networks etc.) can follow you across multiple sites.

To summarise, the first time a user visits a site, the user can provide user identification (possibly his or her name). During the subsequent sessions, the browser passes a cookie header to the server, which lets the server identify the user. Cookies can then be used to create a user session layer on top of stateless HTTP.

Web Caching

A ***Web cache*** - also called a ***proxy server*** - is a network entity that satisfies HTTP requests on behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.



Here's what happens:

1. The browser establishes a TCP connection to the Web cache and sends a HTTP request for the object in the Web cache
2. The Web cache checks to see if it has a copy of the object stored locally. If it does the Web cache returns the object within a HTTP response message to the client browser
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server. The Web cache then sends a HTTP request for the object into the cache-to-server TCP connection. After receiving this request, the origin server sends the object within an HTTP response
4. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within a HTTP response message, to the client browser (over the existing TCP connection b/w the client browser and the Web cache)

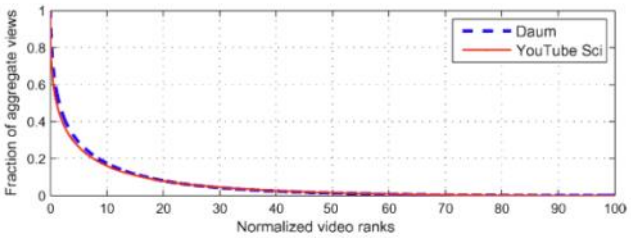
The cache acts as both a server and client at the same time. It is a server for the original requesting client, but is a client to the origin server. Typically a cache is purchased and installed by an ISP.

Web caching has been used in the Internet for two reasons:
1. It reduces response time for a client request
2. It reduces traffic on an institution's access link to the Internet. By reducing traffic, the institution does not have to upgrade bandwidth as quickly (thereby reducing costs)

Through the use of Content Distribution Networks (CDNs), Web caches are increasingly playing an important role in the Internet. A CDN company installs many geographically distributed caches throughout the Internet, which localises much of the traffic.

But what is the likelihood of cache hits?

- Distribution of web object requests generally follows a Zipf-like distribution
- *The probability that a document will be referenced k requests after it was last referenced is roughly proportional to 1/k* . That is, web traces exhibit excellent **temporal locality**.



Video content exhibits similar properties: 10% of the top popular videos account for nearly 80% of views, while the remaining 90% of videos account for total 20% of requests.

Paper – http://yongyeol.com/papers/cha-video-2009.pdf

Paper – "Web Caching and Zipf-like Distributions: Evidence and Implications"
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.8742&rep=rep1&type=pdf

78

The Conditional GET
Caching can reduce a user's perceived response time by having a copy of the object, but what if the object has been modified since the copy was cached.
A HTTP request message is a so-called conditional GET message if:
1. The request messages uses the GET method
2. The request message includes an If-Modified-Since: header line

The cache specifies the last modified date of the cached copy in HTTP request. The server's response contains now object if the cached copy is up-to-date. `HTTP/1.0 304 Not Modified`

E-mail in the Internet

There are three major components in an Internet mail system:
1. ***User agents*** - allows users to read, reply to, forward, save, and compose messages
2. ***Mail servers*** - contains ***mailboxes***, which maintains incoming messages for a user
3. ***Simple mail transfer protocol*** - uses reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. Has two sides: a client side, which executes on the sender's mail server, and a server side, which executes on the recipient's mail server. Both client and server sides of SMTP run on every mail server.

SMTP
SMTP transfers messages from a sender's mail server to the recipient's mail server.

SMTP restricts the body of all mail messages to be in 7-bit ASCII. This means that binary multimedia data is required to be encoded to ASCII before being sent over SMTP, and requires the corresponding ASCII message to be decoded back to binary after SMTP transport. SMTP also uses persistent connections, and servers use CRLF.CRLF to determine the end of a message.

Here is a common scenario. Suppose Alice wants to send Bob a simple ASCII message:

1. Alice uses UA (user agent) to compose message "to" bob@someschool.edu
2. Alice's UA sends message to her mail server; message is placed in message queue
3. Client side of Alice's SMTP mail server opens a TCP connection with Bob's mail server
4. SMTP client sends Alice's message over the TCP connection
5. Bob's mail server places the message in Bob's mailbox
6. Bob invokes his user agent to read message

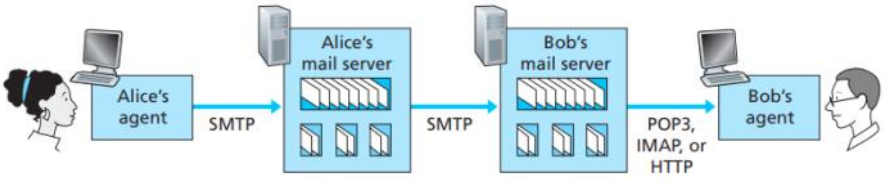


Figure 2.18 ♦ E-mail protocols and their communicating entities

Note: it is important to observe that SMTP does not normally use intermediate mail servers for sending mail, even when two mail servers are located at opposite ends of the world. If the receiving server is down, the message stays in the sending server's mailbox and waits for a new attempt - the message does not get placed in some intermediate mail server.

A closer look at how messages are transferred over SMTP:
The client SMTP (running on the sending mail server host) has TCP establish a connection to port 25 at the server SMTP (running on the receiving mail server host). If the server is down, the client tries again later. Once this connection is established, the server and client perform some application-layer ***handshaking***—SMTP clients and servers introduce themselves before transferring information. During this SMTP handshaking phase, the SMTP client indicates the e-mail address of the sender (the person who generated the message) and the e-mail address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client ***sends the message***. SMTP can count on the reliable data transfer service of TCP to get the message to the server without errors. The client then repeats this process over the same TCP connection if it has other messages to send to the server; otherwise, it instructs TCP to ***close the connection***.

Comparison with HTTP

HTTP	SMTP
• Transfers files from Web server to Web client • Uses persistent connection if using persistent HTTP • Mainly a ***pull*** protocol (for retrieving information) • No limitation on character encoding • Each object encapsulated within its own HTTP response message	• Transfers files from one mail server to another • Uses persistent connections • Primarily a ***push*** protocol (for pushing files to receiving mail server) • Character must be in 7-bit ASCII • Multiple objects are sent in multipart message

Mail Message Formats
SMTP messages contain a header and body. The headers contain peripheral information in a series of header lines, which are defined in RFC 5322. Each header line contains readable text, consisting of a keyword followed by a colon, followed by a value. Every header must have:

```
From:
To:
Subject:
```

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

Note: these header lines are different from SMTP MAIL FROM, RCPT TO commands.

After the header message, a blank line follows; then the message body in ASCII follows

Mail Access Protocols
A mail access protocol transfers messages from recipient's mail server to their local PC. The most popular mail transfer protocols are Post Office Protocol V3 (POP3), Internet Mail Access Protocol, and HTTP.

POP3
POP3 is a simple mail access protocol with limited functionality. POP3 begins when the user agent (the client) opens a TCP connection to their mail server on port 110. with the TCP connection established POP3 progress thrrough 3 phases:

1. ***Authorization***: user agent sends username and password to authenticate user

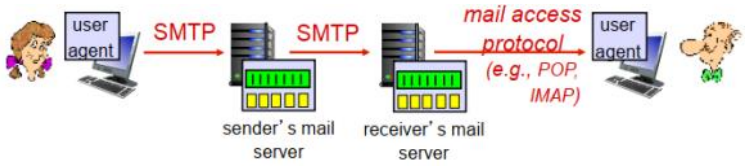
2. ***Transaction***: agent retrieves (downloads) messages to the local machine. The user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics
3. ***Update***: the client issues the quit command, ending the POP3 session; the mail server deletes messages that were marked for deletion.

POP3 does not carry state information across POP3 sessions.

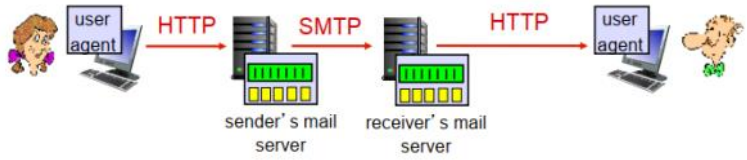
IMAP

IMAP has more features that POP3, and is more complex. An IMAP server will associate each message with a folder; when a message first arrives at the server, it is associated with the recipient's INBOX folder. The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on. IMAP also permits a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message.

An IMAP server maintains user state information across IMAP sessions unlike POP3 - for example, the names of folders and which messages are associated with which folders.



Web-Based Email



In this service, the user agent is an ordinary Web Browser and the user communicates with its remote mailbox via ***HTTP***. When a sender wants to send an e-mail message, the e-mail message is sent from their browser to their mail server over HTTP rather than SMTP . When a recipient wants to access a message in their mailbox, the message is send from their mail server to their browser using the HTTP protocol rather than IMAP or POP3.

DNS

We can identify hosts in two ways - a ***hostname*** and an ***IP address***. A ***Domain Name System (DNS)*** is:

1. A distributed database implemented in a hierarchy of ***DNS servers***
2. An application-layer protocol that allows hosts to query the distributed database.

The DNS protocol runs over UDP and uses port 53.

DNS is commonly used by other application-layer protocols (HTTP, SMTP) to translate user-supplied hostnames to IP address. DNS provides a few other important services:

- ***Host aliasing***: a host with a complicated host name can have one or more alias names. e.g. relay1.west-coast.enterprise.com could have aliases enterprise.com and www.enterprise.com. The original hostname is called the ***canonical hostname***. DNS can be invoked to obtain the canonical hostname as well as the IP address of the supplied alias hostname.
- ***Mail server aliasing***: DNS can be invokes by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- ***Load distribution***: busy sites such as cnn.com are replicated over multiple servers, with each server running on a different end system and each having different IP addresses. For replicated Web servers, a set of IP addresses are associated with one canonical hostname. The DNS database contains this set of IP addresses, and when a client makes DNS query for a name mapped to a set of addresses, the server responds with the entire set of addresses, but rotates the ordering of the addresses within each reply. This is because clients typically send their requests to the first listed IP address. Content Distribution Networks use the IP address of the requesting host to find the best suitable server. e.g. the server closest to the host, or one that is least loaded

A Centralised Database?

A simple design for DNS would be to have a centralised DNS server that contains all the mappings. Clients can simply direct all queries to the single DNS server, and the server responds directly to the querying clients. However, this design can many problems:

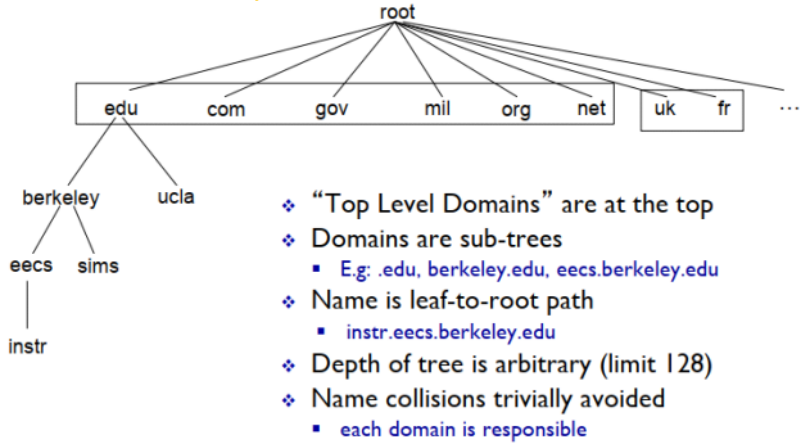
- A ***single point of failure***. If the DNS server crashes, so does the entire Internet.
- ***Traffic volume***. A single DNS server would have to handle all DNS queries (for all HTTP requests and email messages from hundreds of millions of hosts!)
- ***Distant centralised database***. A single DNS server cannot be *close to* all querying clients. This can lead to significant delays
- ***Maintenance***. The single DNS would have to keep records for all Internet hosts, and would have to be updated frequently to account for every new host.

A Distributed, Hierarchical Database

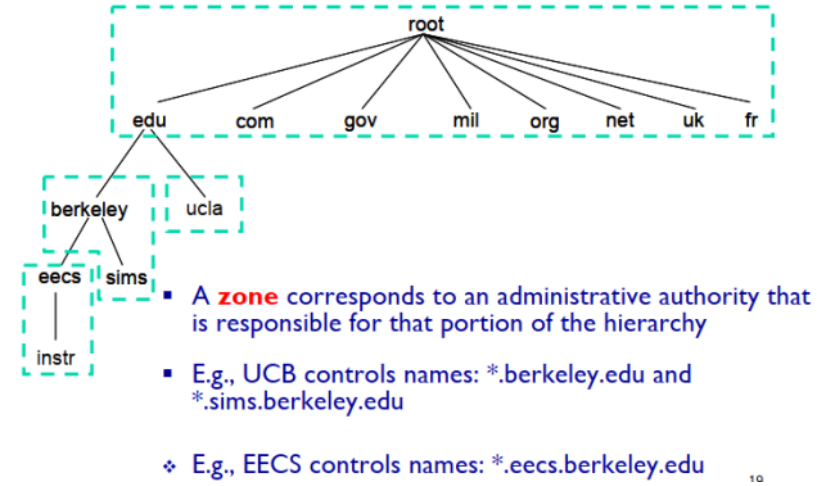
The DNS uses a large number of servers, organised in a hierarchical fashion and distributed around the world. No single DNS has all of the mappings for all the hosts in the Internet. Instead the mappings are distributed across DNS servers.

There are 3 intertwined hierarchies:

1. ***Hierarchical namespace***



2. ***Hierarchically administered***



1. ***(Distributed) hierarchy of servers***

There are 3 classes of DNS servers organised in hierarchy:

- ***root DNS servers***. There are over 400 root name servers scattered all over the world. These root name servers are managed by 13 different organizations. Root name servers provide the IP addresses of the TLD servers. Its location is hardwired into the other servers
- ***top-level domain (TLD) DNS servers***. For each of the top-level domains - such as org, com, net, edu, gov, all country domains - there is a TLD server (or server cluster). TLD servers provide the IP address of authoritative DNS server and are managed professionally
- ***authoritative DNS servers***. Every organisation with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organisation's authoritative DNS server houses these DNS records. A organisation can choose to implement their own authoritative DNS server or pay to have these records stored in an authoritative DNS serve of some service provider.
  The authoritative DNS servers actually store the name-to-address mapping. An organisation's own DNS server(s), provide authoritative hostname to IP mappings for the organisation's named host. They can be maintained by the corresponding administrative authority (the organisation or a server provider).

Each server stores a small subset of the total DNS database. Each server needs to know other servers that are responsible for the other portions of the hierarchy. e.g. every server knows the root, and the root server knows all the top-level domains

Local DNS Name Server

A ***local DNS server*** does not strictly belong to the hierarchy of servers. Each ISP has a DNS server (also called a ***default name server***).

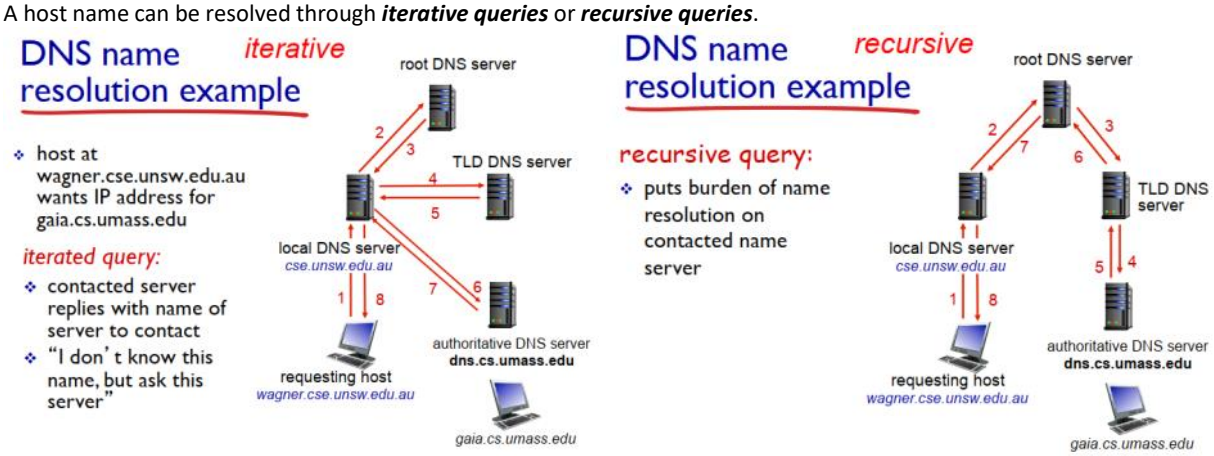
When a host connects to an ISP, the ISP provides the host with the IP addresses of one or more of its local DNS servers (typically through DHCP). When a host makes a DNS query, the query is sent to the local DNS server. The local DNS server has a local cache of recent name-to-address translation pairs (but they may be out of date). So, the local DNS server acts a proxy, forwarding the query into the DNS server hierarchy .

DNS Name Resolution

A host name can be resolved through ***iterative queries*** or ***recursive queries***.

DNS Name Resolution

A host name can be resolved through ***iterative queries*** or ***recursive queries***.



In theory, any DNS query can be iterative or recursive, but in practice, queries are typically iterative to reduce the burden of name resolution on the root server.

DNS Caching

DNS extensively exploits DNS caching to improve the delay performance and to reduce the number of DNS messages going around the Internet.

When a DNS server receives a DNS reply (containing, for example, a mapping from a hostname to an IP address), it can cache the mapping in its local memory. Cache entries timeout after their TTL. TLD servers are typically cached in local name servers, thus root name servers are not visited often.

If a hostname/IP address pair is cached in a DNS server and another query arrives to the DNS server for the same hostname, the DNS server can provide the desired IP address, even if it is not authoritative for the hostname, because the cached information times out eventually.

DNS Records

The DNS servers that implemented the DNS distributed database store ***resource records (RRs)*** that provide hostname - to-IP address mappings. Each DNS reply message carries one or more resource records.

A RR is a four-tuple that contains the following fields:

`(Name, Value, Type, TTL)`

TTL is the time to live of the recorded resource; it determines when the resource should be removed from a cache.

`Name` and `Value` depend on `Type`:

type=A
- **name** is hostname
- **value** is IP address

type=NS
- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

type=CNAME
- **name** is alias name for some "canonical" (the real) name
- **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

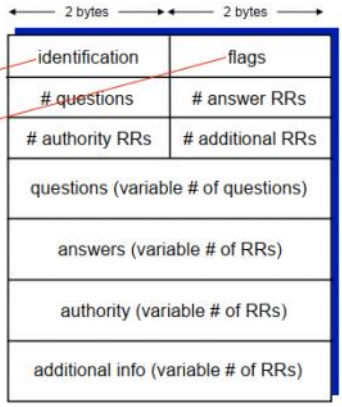
type=MX
- **value** is name of mailserver associated with **name**

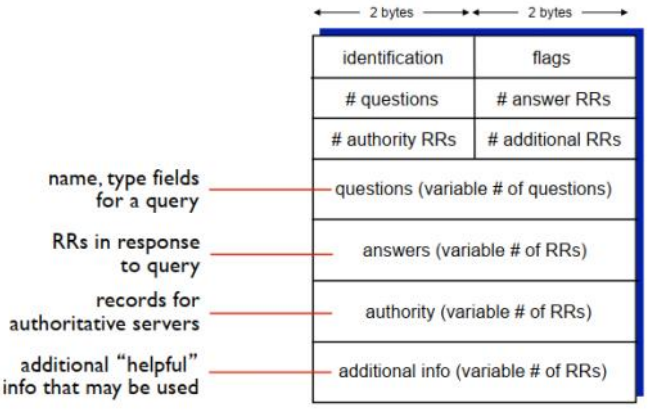
Examples:
- Type A: (relay1.bar.foo.com, 145.37.93.126, A)
- Type NS: (foo.com, dns.foo.com, NS)
- Type CNAME: (foo.com, relay1.bar.foo.com, CNAME)
- Type MX: (foo.com, mail.bar.foo.com, MX)

DNS Messages

DNS query and reply messages have the same format.

msg header
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative





Inserting Records into the DNS Database

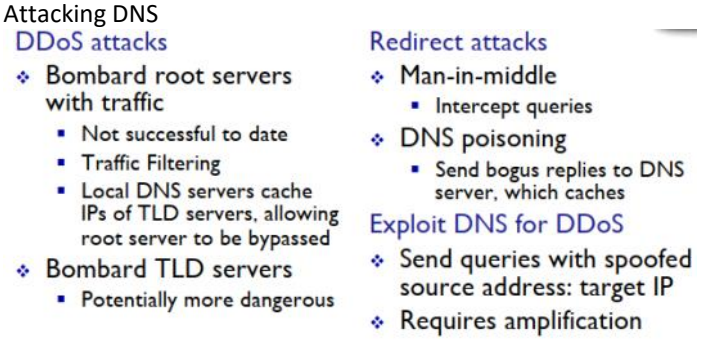
Say you want to register a domain name `networkutopia.com` at a registrar. A ***registrar*** is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database and collects a small fee from you for its services.

When you register the domain name `networkutopia.com` with some registrar, you also need to provide the registrar with the names and IP addresses of your primary and secondary authoritative DNS servers.
Suppose the names and IP addresses are `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1`, and `212.212.212.2`. For each of these two authoritative DNS servers, the registrar would then make sure that a Type NS and a Type A record are entered into the TLD com servers. Specifically, for the primary authoritative server for networkutopia.com, the registrar would insert the following two resource records into the DNS system:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

You'll also have to make sure that the Type A resource record for your Web server `www.networkutopia.com` and the Type MX resource record for your mail server `mail.networkutopia.com` are entered into your authoritative DNS servers.

Attacking DNS



Video Streaming and Content Distribution Networks

Internet Video

Prerecorded videos are placed on servers, and users send requests to the server to view the video *on demand*.

A video is a sequence of images displayed at a constant rate (e.g. 24 frames per sec). An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color. An important characteristic of video is that it can be compressed, which trades off video quality with bit rate. The higher the bit rate, the better the image quality and the better the overall user viewing experience.

The most important performance measure for streaming video is average end-to-end throughput. In order to provide continuous playout, the network must provide an average throughput to the streaming application that is at least as large as the bit rate of the compressed video.

We can also use compression to create multiple versions of the same video, each at a different quality level.

HTTP Streaming and DASH

In HTTP streaming, the video is simply stored at an HTTP server as an ordinary file with a specific URL. When a user wants to see the video, the client establishes a TCP connection with the server and issues a HTTP GET request for that URL. The server then sends the video file, within an HTTP response message, as quickly as the underlying network protocols and traffic conditions will allow. On the clients side, the bytes are collected in a client application buffer. Once the number of bytes in this buffer exceeds a predetermined threshold, the client application begins playback (the streaming video periodically grabs video frames, decompresses them and displays them on the user's screen).

In ***Dynamic Adaptive Streaming over HTTP (DASH)***, the video is encoded in several different versions, with each version having a different bit rate and a different quality level. The client dynamically requests chucks of video segments of a few seconds in length. When the amount of the available bandwidth is high, the client naturally selects chunks from a high-rate version; and when the bandwidth is low, it selects chunks from a low-rate version. The client selects different chunks one at a time with HTTP GET.

With DASH, each video version is stored in the HTTP server, each with a different URL. The HTTP server also has a ***manifest file***, which provides a URL for each version along with its bitrate. The client first requests the manifest file and learns about the various versions. The client then selects one chunk at a time by specifying a URL and a byte range in a HTTP GET request message for each chunk. While downloading chunks, the client also measures the received bandwidth and runs a rate determination algorithm to select the chunk to request next.

Content Distribution Networks

A ***Content Distribution Network (CDN)*** manages servers in multiple geographically distributed location, stores copies of the video (and other types of Web content, including documents, images, and audio) in its servers, and attempts to direct each user request to a CDN location that will provide the best user experience.

A CDN may be a ***private CDN*** that is owned by the content provider itself (e.g. Google's CDN distributes YouTube videos and other types of content). The CDN may alternatively be a ***third-party CDN*** that distributes content on behalf of

multiple content providers (e.g. Akamai and Limelight operate on third-part CDNs).

CDNs typically adopt one of two different server placement philosophies:

- ***Enter Deep***. Push CDN servers deep into many access networks (close to users)
- ***Bring Home***. Smaller number (10s) of larger clusters in POPs near (but not within) access networks. Typically results in lower maintenance and management at the expense of higher delay and lower throughput to users.

Once its clusters are in place, the CDN replicates content across its clusters. But the CDN may not want to place a copy of every video in each cluster. Many CDNs do not *push* videos to their cluster but instead use a *pull* strategy. If a client requests a video from a cluster that it is not storing, then the cluster retrieves the video (from another cluster or repository) and stores a copy locally while streaming the video to the client at the same time.

Socket Programming

Passing Messages Locally

The OS has a mechanism for inter-process communication (***IPC***).

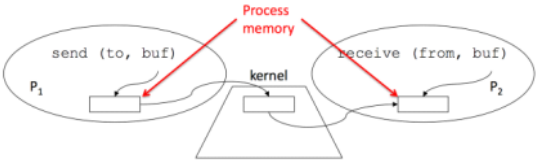
- `send(destination, message_buffer)`
- `receive(source, message_buffer)`

Here, data is transferred into and out of the kernel message buffer.

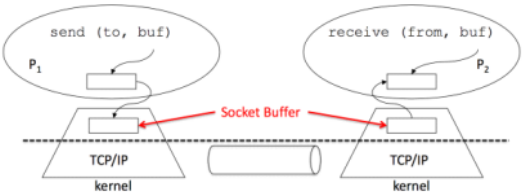
Passing Message via Network

Data transfer occurs by copying to and from the OS socket buffer.
The extra steps that happen across the network are hidden from applications.

IPC:



Sockets:



Recall, that a socket is a door between application processes and the transport-layer side. Sockets define the interfaces between an application and the transport layer. Applications choose the type of transport layer by choosing the type of socket.

- UDP Sockets - called `DatagramSocket` in Java, `SOCK_DGRAM` in C
- TCP Sockets - called `Socket/ServerSocket` in Java, `SOCK_STREAM` in C

The client and server agree on the type of socket, the server port number and the protocol.

An application programming interface (API) defines the interface between the application and transport layer. Sockets are Internet APIs; two processes communicate by sending data into a socket, and reading data out of a socket.

Processes identify each other by using IP addresses of the host running the other process and a port number, which allows the receiving host to determine which local process the message should be delivered to.

Basic steps of socket programming:

1. Create a socket (like opening a file)
2. Read from it and write from it

Some system calls for a socket include:

```
socket()  read()
bind()    write()
connect() close()
listen()
accept()
```

Before you write socket code, you must decide

- Do you want a TCP-style reliable, full-duplex, connection oriented channel?
  Or do you want a UDP-style, unreliable, message oriented channel?
- Will the code you are writing be the client or the server?
  - Client: assume that there is a process already running on another machine that you need to connect to
  - Server: you will just start up and wait to be contacted

TCP service:

- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing or minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Socket Programming with TCP

Pseudocode for TCP Server

```
Create socket (doorbellSocket)
Bind socket to a specific port where clients can contact you
Register with the kernel your willingness to listen that on socket for client to contact you
    Loop
        Listen to doorbell Socket for an incoming connection, get a connectSocket
```

```
        Read and Write Data Into connectSocket to communicate with client
        Close connectSocket
    End Loop
Close doorbellSocket
```

Pseudocode for TCP client

```
Create socket, connectSocket
Do an active connect specifying the IP address and port number of server
Read and Write Data Into connectSocket to communicate with server
Close connectSocket
```

Concurrent/Multithreaded TCP Servers

We can make concurrent TCP servers by handing off processing to another process or thread.

- The parent process creates the “door bell” or “welcome” socket on well-known port and waits for clients to request connection
- When a client does connect, fork off a child process or pass to another thread to handle that connection so that parent can return to waiting for connections as soon as possible

Pseudocode for concurrent TCP Server

```
Create socket doorbellSocket
Bind
Listen
Loop
    Accept the connection, connectSocket
    Fork
    If I am the child
        Read/Write connectSocket
        Close connectSocket
        exit
EndLoop
Close doorbellSocket
```

Socket Programming with UDP

Pseudocode for UDP server

```
Create socket
Bind socket to a specific port where clients can contact you
Loop
    (Receive UDP Message from client x)
    (Send UDP Reply to client x)*
Close Socket
```

Pseudocode for TCP client

```
Create socket
Loop
    (Send Message To Well-known port of server)
    (Receive Message From Server)
Close Socket
```

P2P Applications

In a P2P architecture, there is minimal (or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called ***peers***, communicate directly with each other. The peers are not owned by service providers, but are instead desktops and laptops controlled be users. This means they can change IP addresses.

Examples of P2P services include file distribution (BitTorrent), streaming (KanKan) and VoIP (Skype_..

In a client-server file distribution, the server must send a copy of the file to each of the peers - placing an enormous burden on the server and consuming a large amount of the server bandwidth. In P2P file distribution, each peer can redistribute any portion of the file it has received to any other peers, thereby assisting the server in the distribution process.

Scalability of P2P Architectures: File distribution time - client-server vs. P2P

Client-server architecture

Denote the upload rate of the server's access link by u_s, the upload rate of *the* ***i***th peer's access link by u_i, and the download rate of the ***i***th peer's access link by d_i. Also denote the size of the file to be distributed in bit by ***F*** and the number of peers that want to obtain a copy by ***N***. The distribution time is the time it takes to get a copy of the file to all ***N*** peers. Let us denote the distribution time for client-server architecture with D_{cs}.

- The server must transmit one copy of the file to each of the ***N*** peers. Thus the server must transmit ***NF*** bits. Since the server's upload rate is u_s, the time to distribute the file must be NF/u_s.
- Let d_{min} denote the download rate of the peer with the lowest download rate ($d_{min} = \min\{d_1, d_2, \ldots, d_n\}$). The peer with the lowest download rate cannot obtain all the ***F*** bits of the file in less than F/d_{min} seconds. Thus the minimum distribution time is $\geq F/d_{min}$

Putting the two together, we obtain $D_{cs} \geq \max\{NF/u_s, F/d_{min}\}$

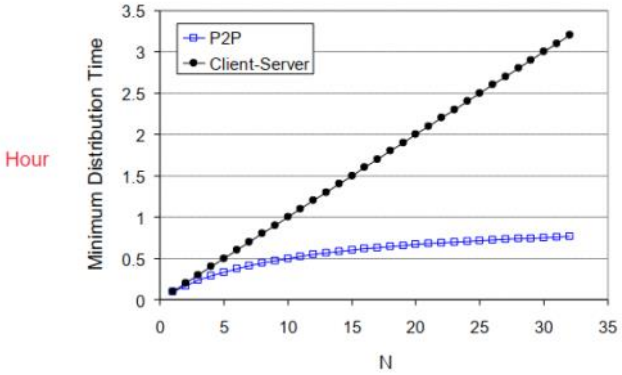
P2P architecture

When a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers. Here are some observations in P2P file distribution:

- At the beginning of distribution, only the server has the file. The file must sent by the server to at least one peer. Thus the minimum distribution time is F/u_s.
- The peer with the lowest download rate cannot obtain all bits of the file in less that F/d_{min} seconds. So this remains the same
- The total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers, that is, $u_{total} = u_s + u_1 + \ldots + u_N$. The system must deliver ***F*** bit to each of the ***N*** peers, thus delivering a total of ***NF*** bits. This cannot be done at a rate faster than u_{total}. Thus, the minimum distribution time is also $\geq NF/u_{total}$

Putting these three together, we obtain $D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/u_{total}\}$

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$ (for simplicity)



BitTorrent

BitTorrent is a popular P2P protocol for file distribution. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a ***torrent***. Peers in a torrent download equal-size ***chunks*** of the file from one another, with a typical chunk size of 256 kbytes.

When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later re-join the torrent.

Each torrent has a ***tracker***. When a peer joins a torrent it, registers itself with the tracker and periodically informs the tracker that it is still in the torrent. Through this the tracker keeps track of the peers that are participating in the torrent.

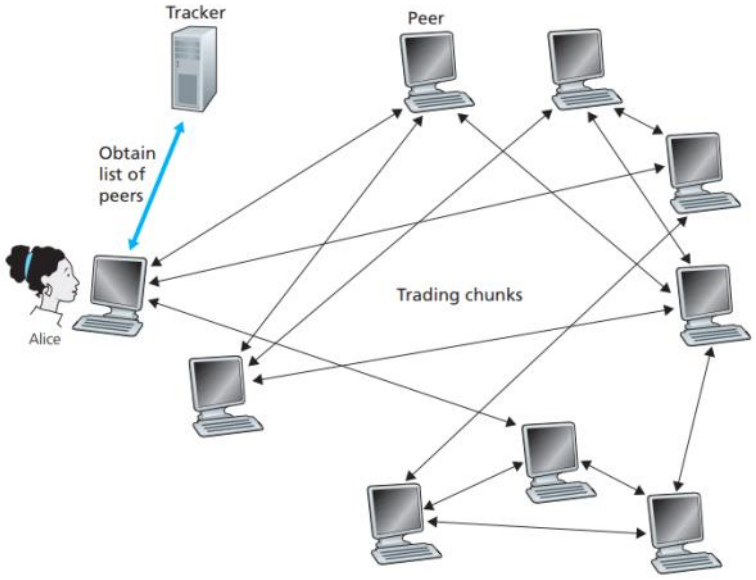


Figure 2.26 ♦ File distribution with BitTorrent

In the above diagram, when Alice first joins a torrent, the tracker randomly selects a subset of peers from the set of

participating peers, and sends the IP addresses of these 50 peers to Alice. The peers, which Alice successfully connects with are called ***neighbouring peers***. The neighbouring peers fluctuate since peers come and go.

At any given time, each peer will have a subset of chunks from a file, with different peers having different subsets. Periodically, Alice will ask each of her neighbouring peers (over the TCP connections) for the list of the chunks they have. If Alice has ***L*** different neighbours, she will obtain ***L*** lists of chunks. With this knowledge, Alice will issue requests (again over the TCP connections) for chunks she currently does not have. So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbours have.

To decide which chunk to request, Alice uses a technique called ***rarest first***. The idea is to determine, from among the chunks, which she does not have, which is the rarest (least repeated copy) among her neighbours. In this manner, the rarest chunks get more quickly redistributed, aiming to *roughly* equalise the number of copies of each chunk in the torrent.

To determine which request she responds to, BitTorrent uses a ***tit-for-tat*** method.
The basic idea is that Alice gives priority to the neighbours that are currently supplying her data *at the highest rate*. Specifically, for each of her neighbours, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers. Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers. In BitTorrent lingo, these four peers are said to be ***unchoked***. Every 30 seconds, she also picks one additional neighbour at random and sends it chunks. This peer is said to be ***optimistically unchoked***.

The effect is that peers capable of uploading at compatible rates tend to find each other. The random neighbour selection also allows new peers to get chunks, so that they can have something to trade. All other neighbouring peers besides these five peers (four "top" peers and one probing peer) are "choked," that is, they do not receive any chunks from Alice.

Free riding: is it possible for a peer to join torrent and receive of complete copy of a file without distributing any chunks?
Answer: It depends.
Free-riding is not possible because if you aren't helping anyone everyone will choke you. Hence, you will get no chunks. Alternatively, people may optimistically unchoke you, and you get a bit of something every time and eventually, you can get everything.

Distributed Hash Tables: Getting Rid of the Server/Tracker

A ***distributed hash table*** **(DHT)** is simply a database, with the database records (tracker information) being distributed over the peers in a P2P system. A DHT is a lookup structure. It maps keys to an arbitrary value and works a lot like a hash table; so it will simply contain *(key, value)* pairs. We query the database with a key. If there are one or more key-value pairs in the database, the database returns the corresponding values.

In a P2P system, each peer will only hold a small subset (section) of all the (key, value) pairs.
We'll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (key, value) pairs and return the key-value pairs to the querying peer. Any peer will also be allowed to insert new key-value pairs into the database.

The general process of a DHT query goes like this:

1. Peer A queries the DHT for a copy of a file, with the file as a key
2. The DHT finds which peer B is responsible for the key
3. The DHT contacts peer B and obtains the key-value pairs
4. The DHT passes the key-value pairs to peer A
5. Peer A can now download the file

Designing a DHT
Let's first assign an identifier to each peer, where each identifier is an integer in the range $[0, 2^n-1]$ for some fixed n. Note that each such identifier can be expressed by an n-bit representation. Let's also require each key to be an integer in the same range. Keys are usually not integers, so to create integers out of such keys, we will use a hash function that maps each key (e.g., social security number) to an integer in the range [0, 2^n-1]. A hash function is a many-to-one function for which two different inputs can have the same output (same integer), but the likelihood of the having the same output is extremely small. The hash function is assumed to be available to all peers in the system. Henceforth, when we refer to the "key," we are referring to the hash of the original key.

Our next problem is assigning a key to a peer. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natural approach is to assign each (key, value) pair to the peer whose identifier is the *closest* to the key. The closest peer is the ***closest successor of the key***. If the key is exactly equal to one of the peer identifiers, we store the (key, value) pair in that matching peer; and if the key is larger than all the peer identifiers, we use a modulo-2n convention, storing the (key, value) pair in the peer with the smallest identifier.

To **insert a (key, value) pair** into the DHT, consider this example.
Alice, wants to insert a (key, value) pair into the DHT. She first determines the peer whose identifier is closest to the key;

she then sends a message to that peer, instructing it to store the (key, value) pair. To determine the closest peer, we can use a circular DHT.

Circular DHTs

We organise the peers into a circle. In this arrangement, each peer only keeps track of its immediate successor and immediate predecessor (modulo 2^n).

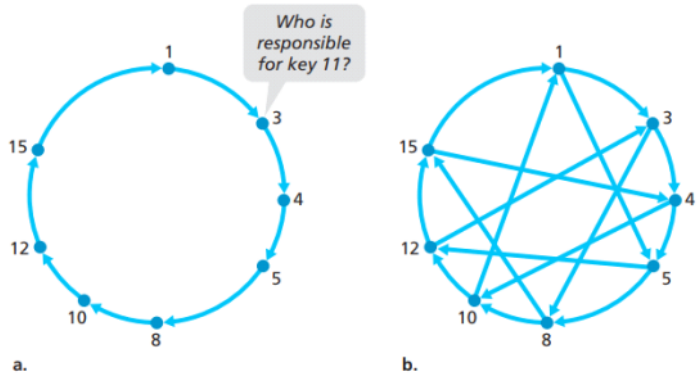


Figure 2.27 ♦ (a) A circular DHT. Peer 3 wants to determine who is responsible for key 11. (b) A circular DHT with shortcuts

This circular arrangement of the peers is a special case of an ***overlay network***. In an overlay network, the peers form an abstract logical network which resides above the "underlay" computer network consisting of physical links, routers, and hosts. The links in an overlay network are not physical links, but are simply virtual liaisons between pairs of peers.

In *Figure 2.27(a)* using the circular overlay, the origin peer (peer 3) creates a message saying "Who is responsible for key 11?" and sends this message clockwise around the circle. Whenever a peer receives such a message, because it knows the identifier of its successor and predecessor, it can determine whether it is responsible for (that is, closest to) the key in question. If a peer is not responsible for the key, it simply sends the message to its successor. If it is responsible for the key it sends a message back to the querying peer.

The circular DHT provides a very elegant solution for reducing the amount of overlay information each peer must manage. Although each peer is only aware of two neighbouring peers, to find the node responsible for a key (in the worst case), all ***N*** nodes in the DHT will have to forward a message around the circle; ***N/2*** messages are sent on average.

There is trade-off between the number of neighbours each peer has to track and the number of messages that the DHT needs to send to resolve a single query. We can refine our DHT design by adding ***shortcuts*** to our circular overlay network. This means each peer keeps track of its immediate successor, predecessor and a relatively small number of shortcut peers scattered about the circle as seen in *Figure2.27(b)*. Shortcuts are used to accelerate the routing of query messages. When a peer receives a message that is querying for a key, it forwards the message to the neighbour (successor neighbour or one of the shortcut neighbours) which is the closet to the key.

It has been shown that it is possible to design shortcuts so that the number of neighbours per peer and the number of messages per query is *O(logN)*, where *N* is the number of peers.

Peer Churn

In P2P systems, a peer can come or go without warning. Thus, when designing a DHT, we also must be concerned about maintaining the DHT overlay in the presence of such ***peer churn***. Let us consider *Figure 2.27(a)*. To handle peer churn, we will now require each peer to track (know the IP addresses of) its first and second successors. We require each peer to periodically verify that its two successors are alive (e.g. by periodically sending ping messages to them and asking for responses). If a peer leaves, the peers preceding the departed peer will need to update their successor state information.

Now let's consider a peer wanting to join a DHT. Say a peer with identifier 13 wants to join the DHT, and at the time of joining, it only knows about peer 1's existence in the DHT. Peer 13 would first send peer 1 a message, saying "what will be 13's predecessor and successor?" This message gets forwarded through the DHT until it reaches peer 12, who realizes that it will be 13's predecessor and that its current successor, peer 15, will become 13's successor. Next, peer 12 sends this predecessor and successor information to peer 13. Peer 13 can now join the DHT by making peer 15 its successor and by notifying peer 12 that it should change its immediate successor to 13.

Transport Layer

Wednesday, 13 March 2019 12:54 PM

Transport Layer Services

A transport-layer protocol provides for ***logical communication*** between application processes running on different hosts. By *logical communication*, we mean that from an application's perspective, it is as if the hosts running the processes were directly connected; in reality, the hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types. Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages.

Transport-layer protocol are implemented in the end systems but not in network routers.
On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer ***segments*** in Internet terminology. This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create the transport-layer segment.
The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a datagram) and sent to the destination.
On the receiving side, the network layer extracts the transport-layer segment from the datagram and passes the segment up to the transport layer. The transport layer then processes the received segment, making the data in the segment available to the receiving application.

Relationship b/w Transport and Network Layers

While a transport-layer protocol provides logical communication between *processes* running on different end hosts, a network-layer protocol provides logical communication between hosts.

The services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network layer protocol cannot provide delay or bandwidth guarantees for transport layer segments between hosts, then the transport layer-protocol cannot provide delay or bandwidth for application messages sent between processes.

Even so, certain services *can* be offered by a transport protocol even when the underlying network protocol doesn't offer the corresponding service at the network layer. Examples include reliable data transfer and encryption.

Overview of the Transport Layer

Recall that the Internet, and more generally a TCP/IP network, makes two distinct transport-layer protocols available to the application layer; UDP and TCP.

- ***UDP*** (User Datagram Protocol) provides an unreliable, connectionless service to the invoking application.
- ***TCP*** (Transmission Control Protocol) provides a reliable, connection-oriented service to the invoking application.

When designing a network application, the application developer must specify one of these two transport protocols.

In the network layer, the IP service model is a ***best-effort delivery service***. This means that IP makes no guarantees that it will deliver segments between communicating hosts. It does not guarantee segment delivery, orderly delivery of segments and the integrity of the data in segments. Hence, IP is said to be an unreliable service

The most fundamental responsibility of UDP and TCP is to *extend* IP's delivery service between two end systems to a delivery service between two processes running on different end systems.

Multiplexing and Demultiplexing

At the destination host, the transport layer receives segments from the network layer just below. The transport layer has the responsibility of delivering the data in these segments to the appropriate application process running in the host.

Recall that a process (as part of a network application) can have one or more ***sockets***, which data passes from the network to the process and through which data passes from the process to the network. Thus the transport layer in the receiving host does not actually deliver data directly to a process, but instead to an intermediary socket. Because at any given time there can be more than one socket in the receiving host, each socket has a unique identifier.

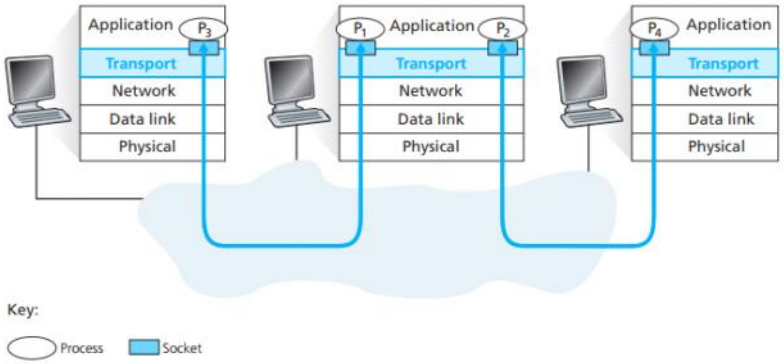


Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

Each transport-layer segment has a set of fields in the segment. At the receiving end of the host, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called ***demultiplexing***.
The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called ***multiplexing***

The special fields in the header of a transport-layer segment are the ***source port number field*** and the ***destination port number field***. Each port number is a 16-bit number, ranging from 0 to 65535. The port numbers ranging from 0 to 1023 are called well-known port numbers and are restricted, which means they are reserved for use by well-known application protocols such as HTTP (port number 80) and FTP (port number 21). When we develop an new application, we must assign the application a port number.

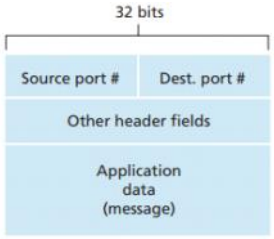


Figure 3.3 ♦ Source and destination port-number fields in a transport-layer segment

To find the list of well-known port numbers, go to: http://www.iana.org

Connectionless Multiplexing/Demultiplexing

When a host sends a UDP segment, its header includes the source IP address and port number and destination IP address and port number. The transport layer then passes the resulting segment to the network layer.
When a host receives a UDP segment, its transport layer checks the destination port number in the segment and directs the UDP segment to the socket with that port number.

It is important to note that at UDP socket is fully identified by a ***two-tuple*** consisting of a destination IP address and a destination port number. As a result, if two UDP segments have different source IP addresses and/or source port numbers, but the same *destination* IP address and *destination* port number, then the two segments will be directed to the same destination process via the same destination socket.

Connection-Oriented Multiplexing/Demultiplexing

One subtle difference between a TCP socket and a UDP socket is that a TCP socket is identified by a ***four-tuple***: (source IP address, source port number, destination IP address, destination port number). Thus, when a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket.
Two arriving TCP segments with different source IP addresses or source port numbers will (with the exception of a TCP segment carrying the original connection-establishment request) be directed to two different sockets.

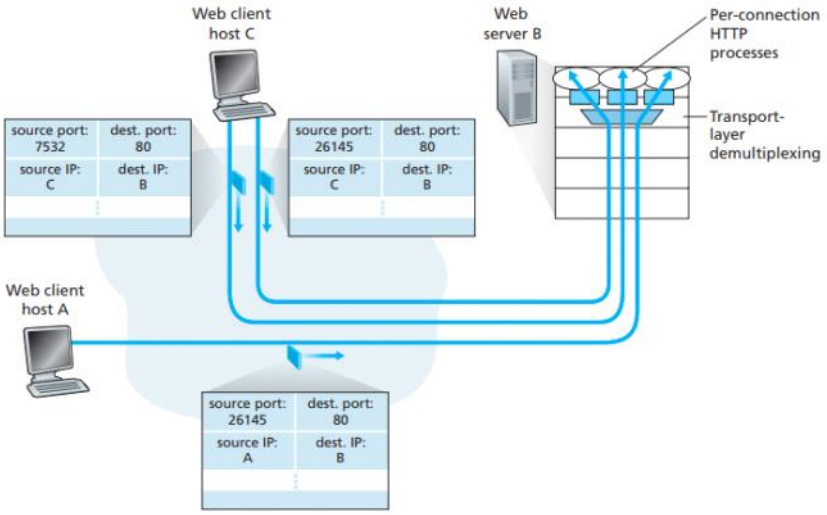


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

Port Scanning

We've seen that the server waits at open ports for client requests. Hackers often perform ***port scans*** to determine open, closed and unreachable ports on candidate victims. Several ports are well-known; ports 0 to 1023 are reserved for well-known apps, some other apps use known ports (MS SQL server port uses 1434 UDP, Sun Network File System (NFS) uses 2049 TCP/UDP). Hackers can exploit well known flaws with these known apps. For example MS SQL server was exploit by a Slammer worm and was subject to a buffer overflow, allowing a remote user to execute arbitrary code on the vulnerable host.

There are a number of public domain programs, called port scanners, that let you scan ports. http://nmap.org is a popular example.

Connectionless Transport: UDP

UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer. The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.
Note that with UDP there is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be ***connectionless***.

Some applications are better suited for UDP for the following reasons:

- ***Finer application-level control over what data is sent, and when***. Data segments are packaged and immediately sent to the network layer. This is useful for real-time applications, and any application that requires latency sensitive and time critical message distribution.
- ***No connection establishment***. UDP does not introduce any delay to establish connection. This is a principal reason as to why DNS runs over UDP rather than TCP (it would be too slow)
- ***No connection state***. UDP does not maintain connection state in end systems and does not track any parameters (receive and send buffers, congestion-control parameters, and sequence and acknowledgement number parameters). This means a server can support *more active clients* when running over UDP.
- ***Small packet overhead***. UDP segments only has 8 bytes of overhead, while TCP has 20 bytes

It *is* possible for an application to have reliable data transfer when using UDP. This can be done if reliability is built into the application itself (for example, by adding acknowledgment and retransmission mechanisms)

UDP Segment Structure

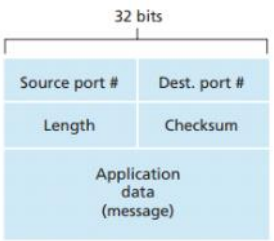


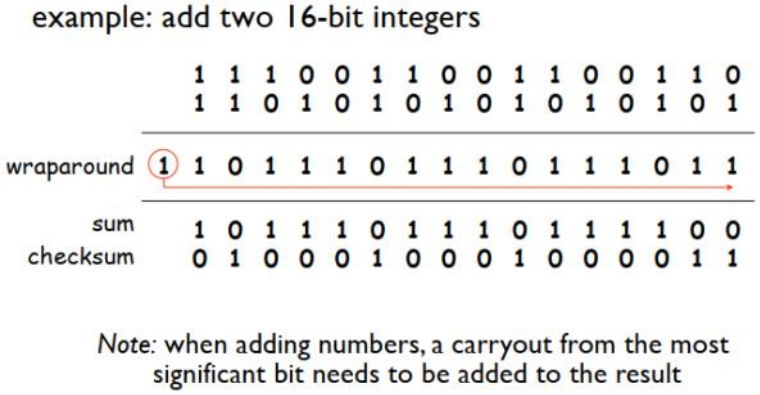
Figure 3.7 ♦ UDP segment structure

The application data occupies the data field of the UDP segment. The UDP header has only 4 fields, each consisting of two bytes. The port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function). The length field specifies the number of bytes in the UDP segment (header plus data). An explicit length value is needed since the size of the data field may differ from one UDP segment to the next. The checksum is used by the receiving host to check whether errors have been introduced into the segment.

UDP Checksum

The UDP checksum provides error detection by determining whether the bits within the UDP segment have been altered as it moved from source to destination. If UDP detects an error, it does not do anything to recover from it.

UDP treats the segment contents (including header fields) as a sequence of 16-bit integers. At the sender side, UDP performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around. This result is put in the checksum field of the UDP segment. The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s.



Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

At the receiver, all 16-bit sequences are added, including the checksum. If no errors are introduced into the packet, then the sum at the receiver will be `1111 1111 1111 1111`. If one bit is a `0`, then we know that errors have been introduced.

> You may wonder why UDP provides a checksum in the first place, as many link-layer protocols (including the popular Ethernet protocol) also provide error checking. The reason is that there is no guarantee that all the links between source and destination provide error checking; that is, one of the links may use a link-layer protocol that does not provide error checking. Furthermore, even if segments are correctly transferred across a link, it's possible that bit errors could be introduced when a segment is stored in a router's memory. Given that neither link-by-link reliability nor in-memory error detection is guaranteed, UDP must provide error detection at the transport layer, *on an end-end basis*, if the end-end data transfer service is to provide error detection. This is an example of the celebrated **end-end principle** in system design [Saltzer 1984], which states that since certain functionality (error detection, in this case) must be implemented on an end-end basis: "functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the higher level."

Principles of Reliable Data Transfer

In a perfect world, reliable transport is easy, but here are all the bad things that can happen in *best-effort* delivery:

- A packet is corrupted
- A packet is lost
- A packet is delayed
- Packets are reordered
- A packet is duplicated

Here are our problems in data transfer:

- Message corruption
- Message duplication
- Message loss
- Message reordering
- Performance

Our toolbox

- Checksums
- Timeouts
- Ack and Nacks
- Sequence numbering
- Pipelining

In a computer network setting, reliable data transfer protocols are based on retransmission known as ***Automatic Repeat Response (ARQ)***. We will use our toolbox to build the following ***ARQ*** protocols:

- Stop-and-wait
- Pipelining

- Go-Back-N
- Selective Repeat

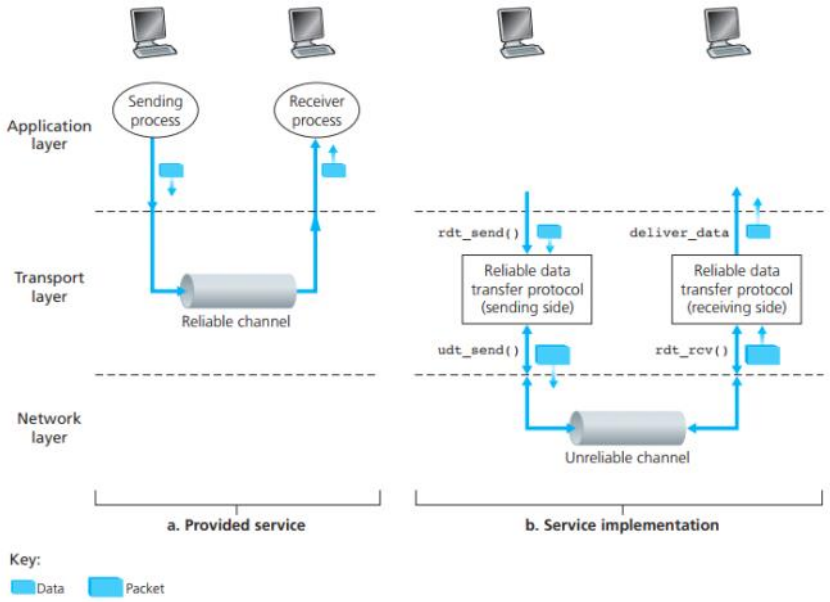


Figure 3.8 ♦ Reliable data transfer: Service model and service implementation

Reliable data transfer is important in application, transport and link layers (probably one of the top 10). The characteristics of an unreliable channel will determine the complexity of a reliable data transfer protocol (***rdt***).

rdt1.0: reliable transfer over a reliable channel

Our underlying channel is perfectly reliable; there are no bit errors, no loss of packets. In this scenario, the transport layer does not have to do anything

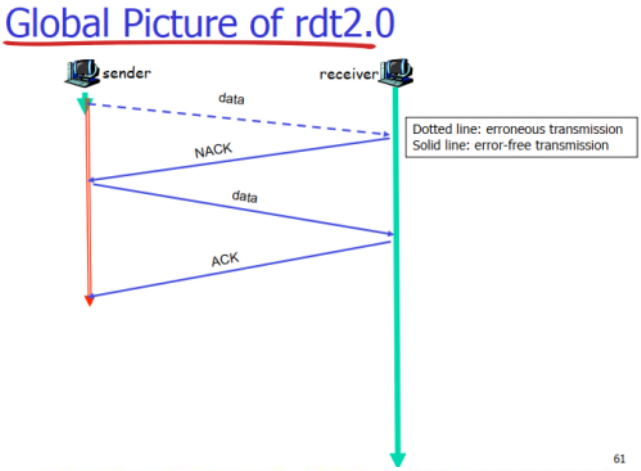
rdt2.0: a channel with bit errors

The underlying channel may flip bits in a packet. We can use checksums to detect the bit errors. The problem in this scenario, is how do we recover from these errors?

We can use:

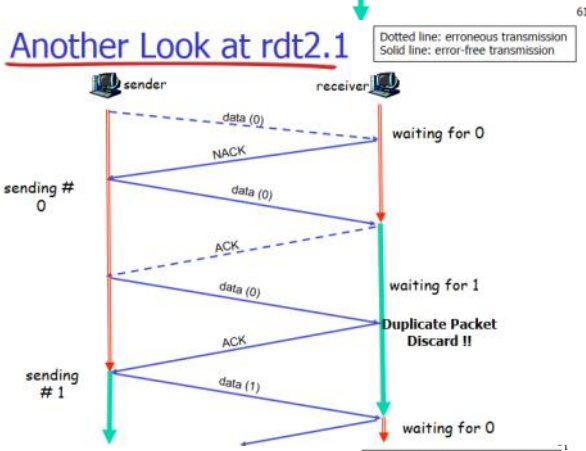
- ***Acknowledgements (ACKs)***: the receiver explicitly tells the sender that the packet was received OK
- ***Negative acknowledgements (NAKs)***: the receiver explicitly tells the sender that the packet had errors

These control messages allow the receiver let the sender know what has been received correct, and what has been received in error and requires repeating.



In this rdt2.0 channel we have introduced two new mechanisms:

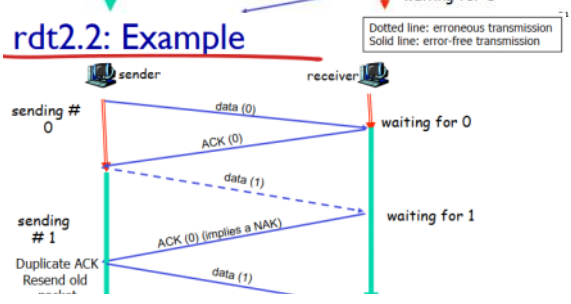
1. Error detection
2. Feedback: control messages (ACK, NAK) from receiver to sender



rdt2.1: corrupted ACKs and NAKs

But out rdt2.0 has a flaw. Our ACK and NAK messages can be corrupted. In this case, the sender does not know what happened at the receiver. If it retransmits the packet it may create a possible duplicate, which we do not want.

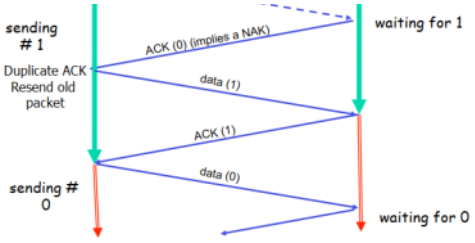
To deal with this, we make the sender add ***sequence numbers*** `(0, 1)` to each packet. The sender checks if it received a corrupted ACK/NAK, and if it did, it retransmits the packet. If the receiver receives a duplicate packet, it discards it but still sends acknowledgement that it has correctly received the packet. This is known as ***stop and wait***, where a sender sends one packet, then waits for the receivers response.



rdt2.2: a NAK-free protocol

Here we want to implement the same functionality as rdt2.1 but using ACKs only. Instead of sending NAK messages, the sender sends *ACK for the last packet it received OK*. That is the receiver sends the sequence number of the packet it last acknowledged.

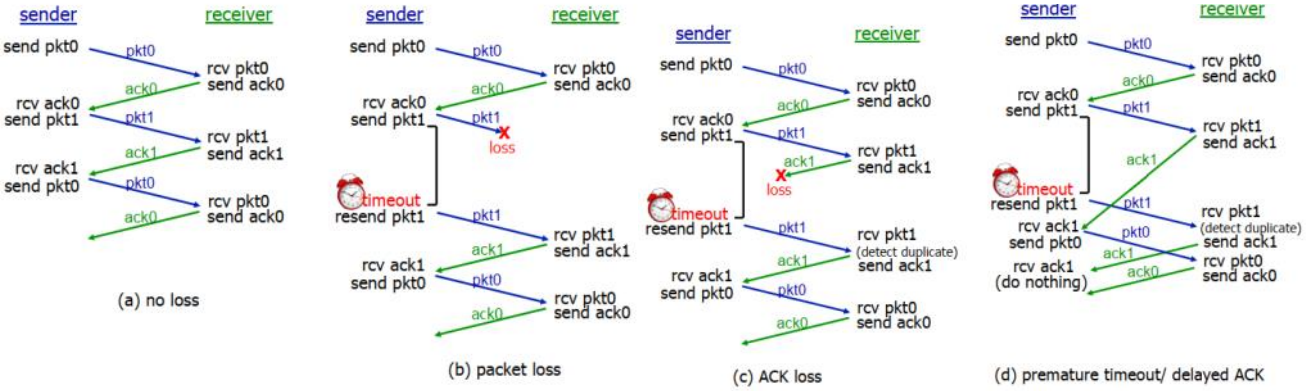
ACKs only. Instead of sending NAK messages, the sender sends *ACK for the last packet it received OK*. That is the receiver sends the sequence number of the packet it last acknowledged.
A duplicate ACK received by the sender results in the same action as receiving a NAK; retransmit the packet.



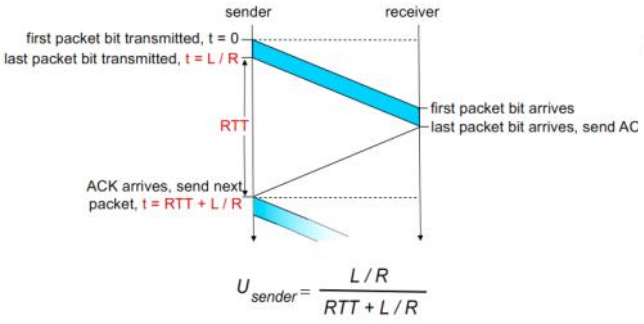
66

rdt3.0: a channel with errors and loss
Now assume that our underlying channel also loses packets. Our checksums, sequence numbers, ACKs and retransmissions will be useful but not enough to deal with the situation. To solve this, we have a countdown timer, where we make our sender wait a *reasonable* amount of time for an ACK.
If there is no ACK received within this time, the sender retransmits the message.
If the packet (or ACK) is just delayed (but not lost), retransmission will be duplicated, but sequence numbers already handles this.
The receiver must specify the sequence number of the packet being acknowledged.

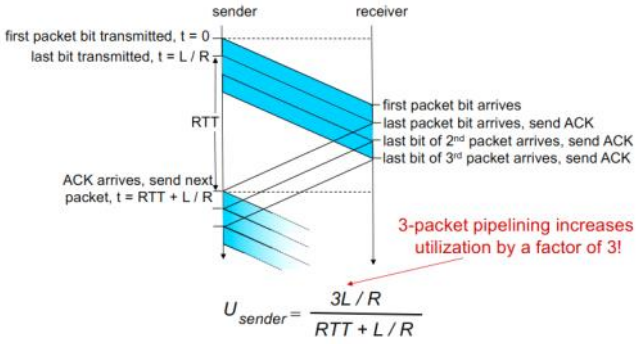
rdt3.0 in action



rdt3.0: stop-and-wait operation



Pipelining: increased utilization



We define a sender's ***utilisation of a link*** as the fraction of time the sender is actually busy sending bits into the channel. ***Stop-and-wait*** is a functionally correct protocol, but because it transmits one packet and wait for response. Its utilisation of the link in not particularly high.

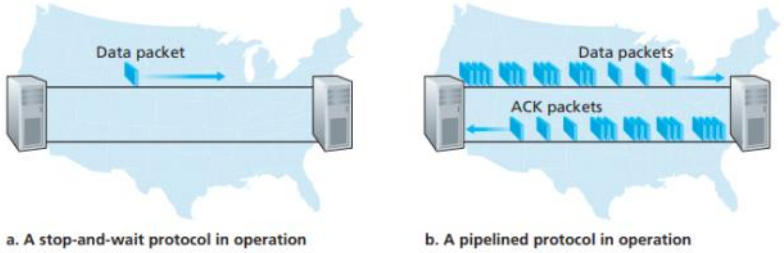


Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

The solution to this performance problem is to allow the sender to send multiple packets without waiting for acknowledgement. This is known as ***pipelining***. Pipelining has the following consequences for reliable data transfer protocols:

- The range of sequence numbers must be increased
- The sender and receiver sides of the protocols may have to buffer more than one packet. The sender will at least have to buffer packets that have been in transit, but are not yet acknowledged

The range of sequence numbers needed and buffering requirements will depend on the data transfer protocol. There

are two generic forms of pipelined (***sliding window***) protocols; ***Go-Back-N*** and ***selective repeat***.

Go-Back-N (GBN)

In a GBN protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is restricted to have no more than some maximum allowable number, *N,* of unacknowledged packets in the pipeline.

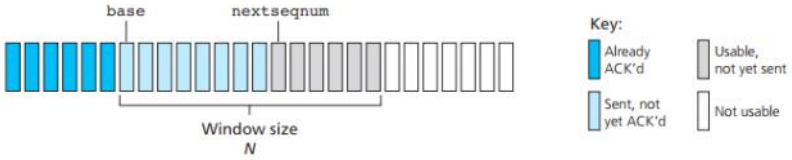


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

Let

`base` be the sequence number of the oldest unacknowledged packet

`nextseqnum` be the smallest unused sequence number (i.e. the next packet to be sent)

Sequence numbers in interval

`[0, base-1]` have already been transmitted and acknowledged

`[base, nextseqnum-1]` have been sent but not yet acknowledged

`[seqnumber, base+N-1]` can be sent immediatbly should data arrive from the upper layer

`[base+N, onwards]` cannot be used until an unacknowledged packet in the pipeline has been acknowledged.

N is often referred to as the ***window size***.

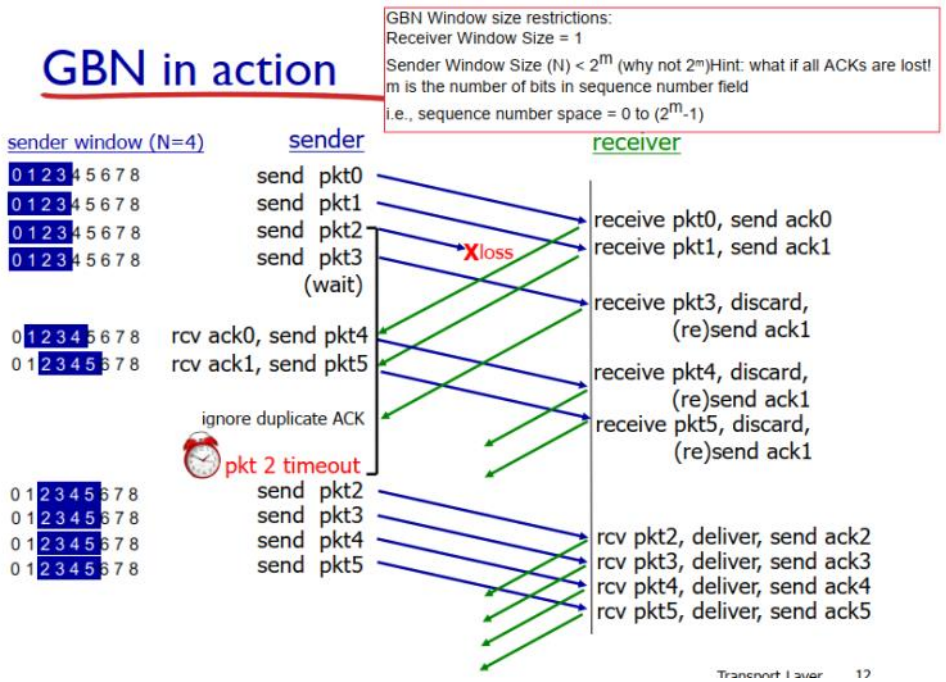
A packet's sequence number is carried in as fixed-length field in the packet header, if ***k*** is the number of bits in the packet sequence number field, the range of sequence numbers is $[0, 2^k-1]$

When an ACK(n) is received/returned it acknowledges all packets up to and including sequence number n. For this reason, a duplicate ACK may be generated but this means the receiver will only need to remember the expected sequence number.

The receiver has a window size of one and **must** receive packets in sequence. If an out-of-order packet is received, the receiver discards (does not buffer) the packet. Instead it will just re-ACK the packet with the highest sequence number. If there is a loss in the sequence GBN is not efficient.

In summary there are 3 types of events a GBN sender must respond to:

1. Sending all available packets in the window size ***N***
2. Receiving acknowledgement for a packet with sequence number ***n*** will be taken as a ***culminative acknowledgement*** of all packet up to an including ***n***
3. If a timeout occurs, the sender resends *all* packets that have been previously sent but that have not yet been acknowledged. Our sender in uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet. If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted. If there are no outstanding, unacknowledged packets, the timer is stopped.



Selective repeat

In selective repeat, the receiver individually acknowledges all correctly received packets (regardless of order), and buffers the packets, as needed, for eventual in-order delivery to the upper layer. The sender only resends packets for which an ACK has not been received. For this the sender will have a timer for each packet sent.

The sender has a window size N for consecutive sequence numbers and is used to limit the number of outstanding, unacknowledged packets in the pipeline. The receiver has the same window size of the transmitter.

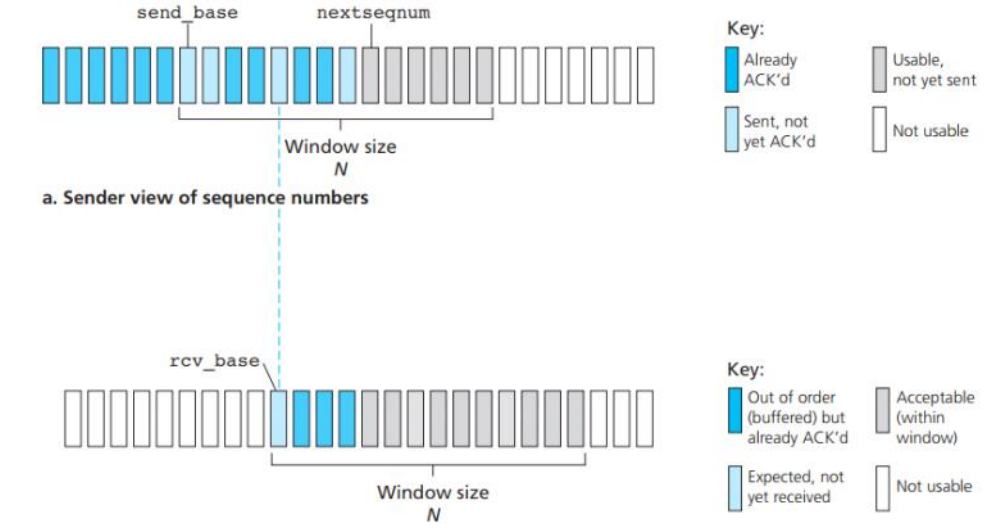


Figure 3.23 ♦ Selective-repeat (SR) sender and receiver views of sequence-number space

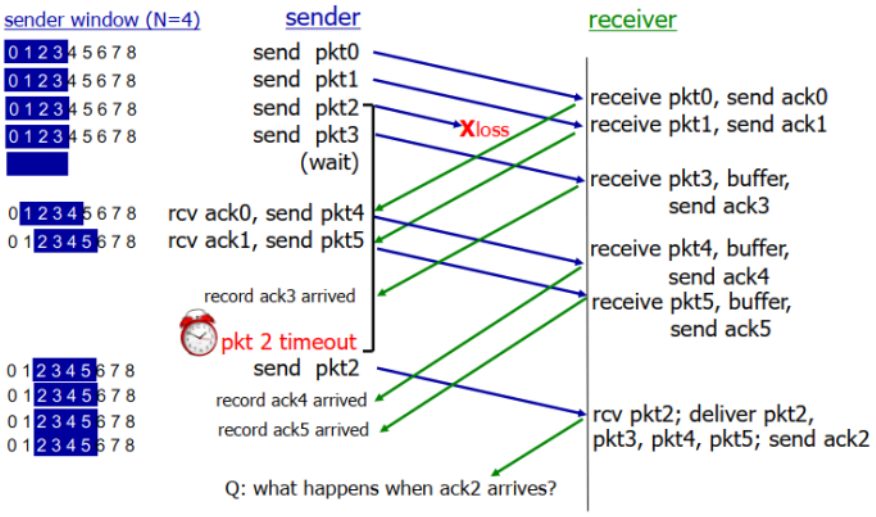
1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Figure 3.24 ♦ SR sender events and actions

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

Figure 3.25 ♦ SR receiver events and actions

Selective repeat in action



The relationship between window size and sequence number

The window size must be less than or equal two half the size of the sequence number space;

`SR sender window size = receiver window size <= 2^(m-1)`

Where `m` is the number of bit required to make the max sequence number.

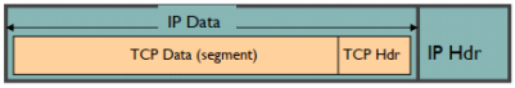
Connection-Oriented Transport: TCP

TCP is said to be ***connection-oriented*** because before one application process can begin to send data to another, the two processes must first *handshake* with each other - i.e. they must send preliminary segments to each other to establish the parameters of the ensuing data transfer.

A TCP connection provides a ***full-duplex service***: if there is a TCP connection process between Process A on one host and Process B on another host, then application-layer data can flow between process A and B at the same time. A TCP connection is also always ***point-to-point***; i.e. between **one** sender and **one** receiver.

How a TCP connection is established
Suppose a process running in one host wants to initiate a connection with another process in another host. Recall that the process that is initiating the connection is called the *client process*, while the other process is called the *server process*. The client application process first informs the client transport layer that it wants to establish a connection to a process in the server. The client first sends a special TCP segment; the server responds with a second special TCP segment; and finally the client responds again with a third special segment. The first two segments carry no payload, that is, no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection-establishment procedure is often referred to as a ***three-way handshake***.

From time to time, TCP will grab chunks of data from the send buffer and pass the data to the network layer. The maximum amount of data that can be grabbed and placed in a segment is limited by the ***maximum segment size (MSS)***. The MSS is typically set by first determining the length of the largest link-layer frame that can be sent by the local sending host (the so-called ***maximum transmission unit (MTU)***) and then setting the MSS to ensure that a TCP segment (when encapsulated in an IP datagram) plus the TCP/IP header length (typically 40 bytes) will fit into a single link-layer frame.



The above IP packet is no bigger than MTU.
The TCP packet is the IP packet (data) with a TCP header and data inside.
The TCP segment is the TCP data. It is no more than MSS
`MSS = MTU - IP Header Length - TCP Header Length`

TCP Segment Structure

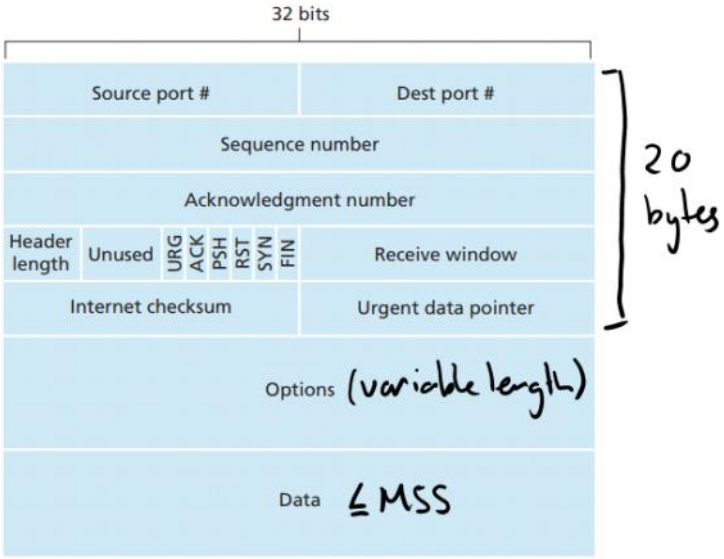


Figure 3.29 ♦ TCP segment structure

Sequence Numbers and Acknowledgements
TCP views data as an unstructured, but ordered, stream of bytes. The ***sequence number for a segment*** is the byte-stream number for the first byte in the segment. Each sequence number is inserted in the sequence number field in the header of the appropriate TCP segment.

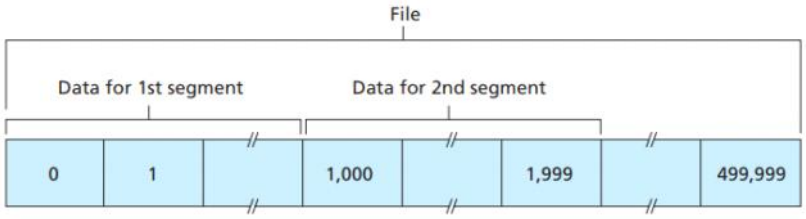


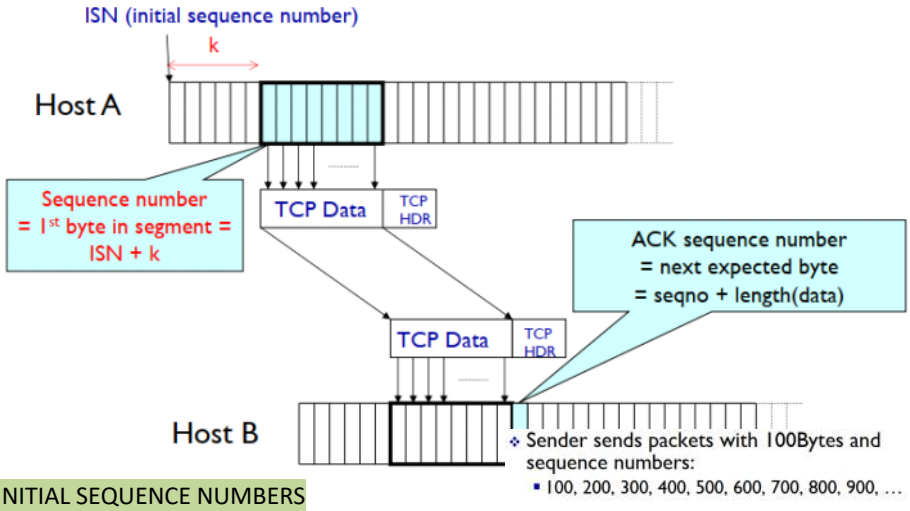
Figure 3.30 ♦ Dividing file data into TCP segments

Each of the segments that arrive from Host B has a sequence number for the data flowing from B to A. The ***acknowlegement number*** that Host B put in its segment is the sequence number of the next byte Host B is expecting from Host A.

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B
- Sender: seqno=X+2B, length=B

- Seqno of next packet is same as last ACK field

Because TCP only acknowledges bytes up to the first missing byte in the stream, TCP is said to provide ***culminative acknowledgements***.



INITIAL SEQUENCE NUMBERS

Both sides of a TCP connection randomly choose e to minimize the possibility that a segment that is still present in the networ onnection between two hosts is mistaken for a valid segment in a later connec

Round-Trip Time Estimation and Timeout

The `sampleRTT` for a segment is the amount of time between when the segment is sent (that is, passed to IP) and when an acknowledgement for the segment is received.

Instead of measuring a `SampleRTT` for every transmitted segment, most TCP implementations take only one `SampleRTT` measurement at a time. That is, at any point in time, the `SampleRTT` is being estimated for only one of the transmitted but currently unacknowledged segments, leading to a new value of `SampleRTT` approximately once every RTT. Also, *TCP never computes a `SampleRTT` for a segment that has been retransmitted*; it only measures `SampleRTT` for segments that have been transmitted once.

Obviously, the `sampleRTT` values will fluctuate from segment to segment due to congestion in routers and to the varying loads on the end systems. Because of this fluctuation, any given `SampleRTT` value may be atypical. In order to estimate a typical RTT, it is therefore natural to take some sort of average of the `SampleRTT` values. TCP maintains an average, called `EstimatedRTT`, of the `SampleRTT` values. Upon obtaining a new `SampleRTT`, TCP updates `EstimatedRTT` according to the following formula:

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

The recommended value of $\alpha = 0.125$ (i.e. 1/8)

Note that `EstimatedRTT` is a weighted average of the `SampleRTT` values.

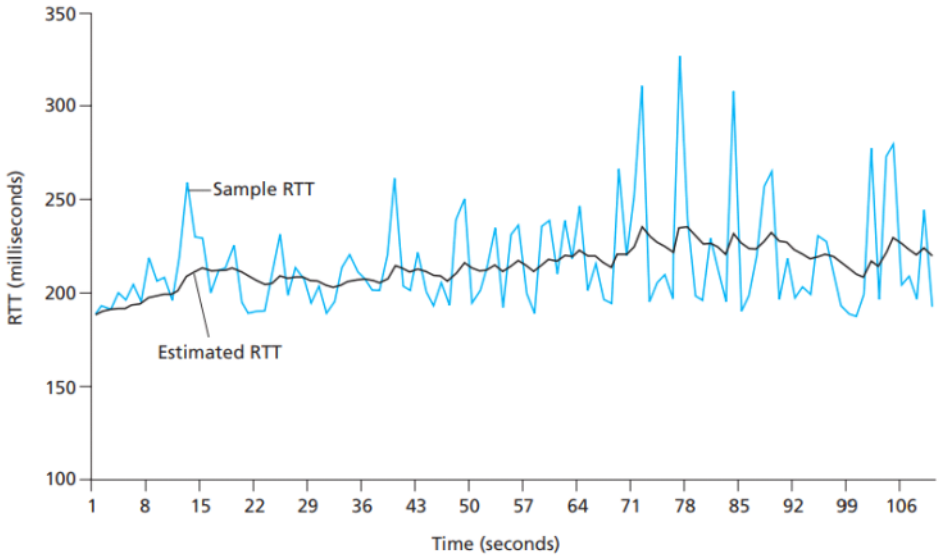


Figure 3.32 ♦ RTT samples and RTT estimates

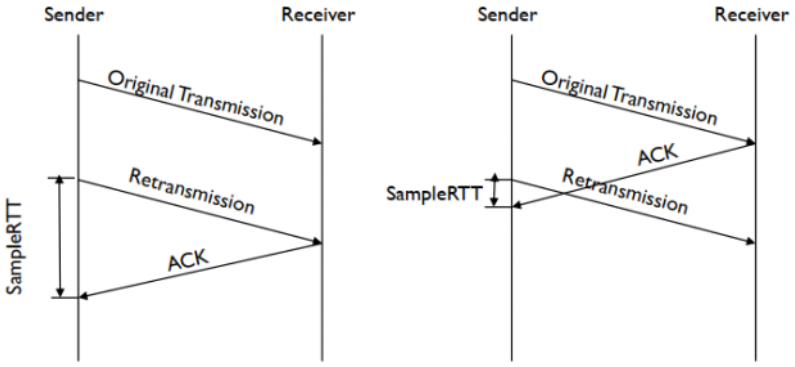
The TimeoutInterval is the EstimatedRTT + 'safety margin'.
The estimated SampleRTT deviation from EstimatedRTT is

$$DevRTT = (1 - \beta) \times DevRTT + \beta \times | SampleRTT - EstimatedRTT |$$

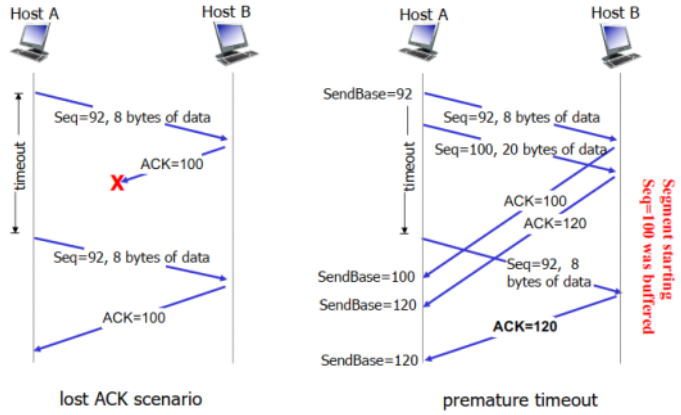
The recommended value of β is 0.25.

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

We exclude retransmissions in RTT, because it is difficult to differentiate between a real ACK and an ACK of a retransmitted packet

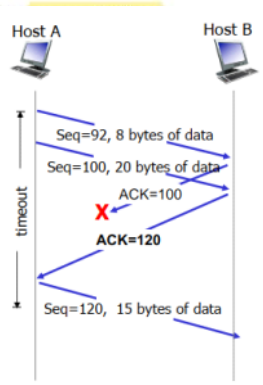


TCP: retransmission scenarios



TCP can deliberately delay packet acknowledgement, by waiting for the next packet to arrive and acknowledging that packet instead. The main reason for this is to reduce traffic.

event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send sing ACK, ACKing both in-
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send dup indicating seq. # of ne but buffer auto
arrival of segment that partially or completely fills gap	immediate send ACK, segment starts at low



cumulative ACK

Reliable Data Transfer

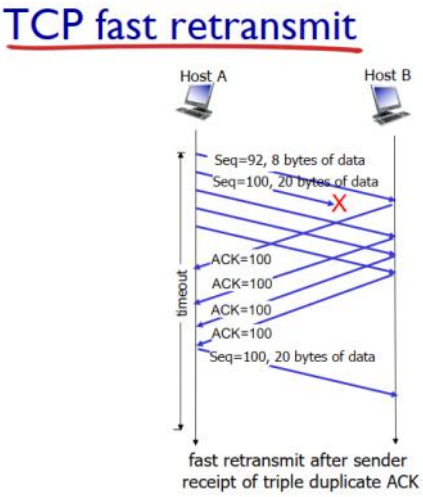
TCP creates a **reliable data transfer service** on top of IP's unreliable l eliable data transfer service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence; that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection.

In our earlier development of reliable data transfer techniques, it was conceptually easiest to assume that an individual timer is associated with each transmitted but not yet acknowledged segment. While this is great in theory, timer management can require considerable overhead. Thus, the recommended TCP timer management procedures [RFC 6298] use only a *single* retransmission timer, even if there are multiple transmitted but not yet acknowledged segments.

Fast Retransmit

One of the problems with timeout-triggered retransmissions is that the timeout period can be relatively long. When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end-to-end delay. A **duplicate ACK** is an ACK that reacknowledges a segment for which the sender has already received an earlier acknowledgment. The lack of ACK progress means that the packet has not been delivered. A stream of ACKs means that some packets are being delivered.

In TCP, if a sender receives 3 duplicate ACKs (i.e. a total of 4 ACKs) for the same data, it will resend the un-ACK-ed segment with the smallest sequence number. It is likely that that un-ACK-ed segment is lost, so we will not wait for it to timeout.



Flow Control

Flow control is about the receiver being able to dynamically control the window size

TCP provides a ***flow-control service*** to its applications to eliminate the possibility of the sender overflowing the receiver's buffer. Flow control is essentially matching the rate at which the sender is sending against the rate at which the receiving application is reading.

TCP provides flow control by having the *sender* maintain a variable called ***receive window***. Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver. Because TCP is full-duplex, both hosts on the connection maintain a discrete receive window.

A host allocates receive buffer for its TCP connection; we will denote its size with `RcvBuffer`. `RcvBuffer`'s size is set via socket options (typically 4096 bytes) and many operating systems auto-adjust the `RcvBuffer`.

The receive window, denoted by `rwnd` is the set amount of spare room in the buffer.

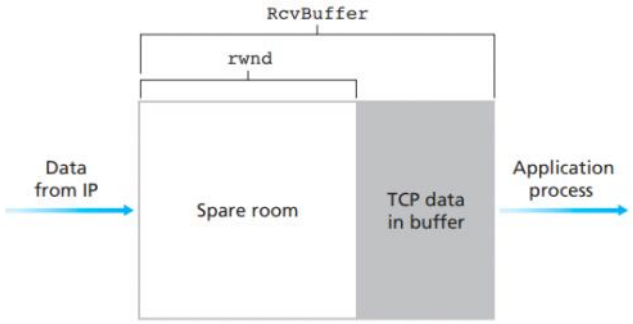


Figure 3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

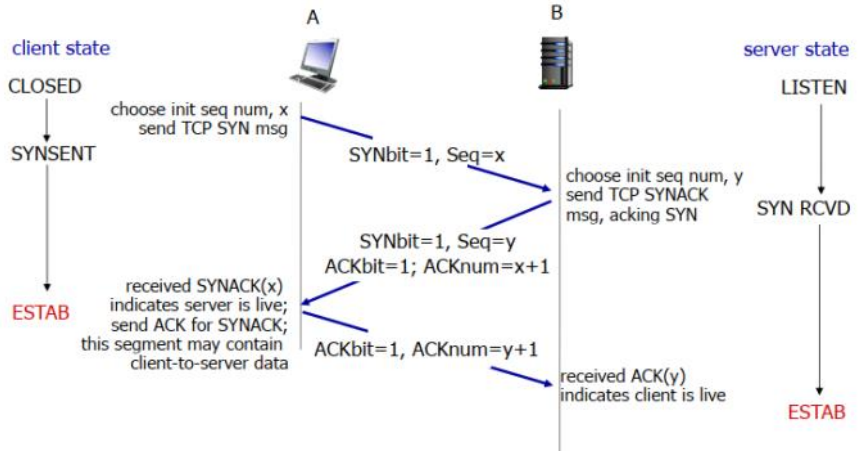
The receiver *advertises* free buffer space including `rwnd` value in the TCP header of the receiver-to-sender segments. The sender limits the amount of un-ACK-ed data to the receiver's `rwnd` value. This guarantees that the receive buffer does not overflow.

Connection Management

Before exchanging data, a sender and receiver perform a ***three-way handshake*** to agree to establish a connection and to agree on connection parameters.

1. The client-side TCP sends a special TCP segment to the server-side TCP. This segment contains no application-layer data, but contains the flag SYN bit. This is often referred to as the SYN segment. The client also chooses an initial sequence number and puts this number in the sequence number field of the TCP SYN segment.
2. The server-side TCP receives the TCP SYN segment , allocates TCP buffers and variables to the connection and sends a connection-granted segment to the client TCP. This segment also contains no application-layer data, but has:
   - The SYN bit set to 1
   - The acknowledgement field of the TCP segment header set to client_isn+1
   - Its own sequence number (server_isn) in the sequence number field of the TCP segment header

   This connection-granted segment is saying, in effect, *"I received your SYN packet to start a connection with your initial sequence number, client_isn. I agree to establish this connection. My own initial sequence number is server_isn."* The connection granted segment is referred to as a ***SYNACK segment***
3. The client also allocates buffers and variables to the connection. The client host then sends the server yet another segment; this last segment acknowledges the server's connection-granted segment (the client does so by putting the value server_isn+1 in the acknowledgment field of the TCP segment header). The SYN bit is set to zero, since the connection is established. This third stage of the three-way handshake may carry client-to-server data in the segment payload.



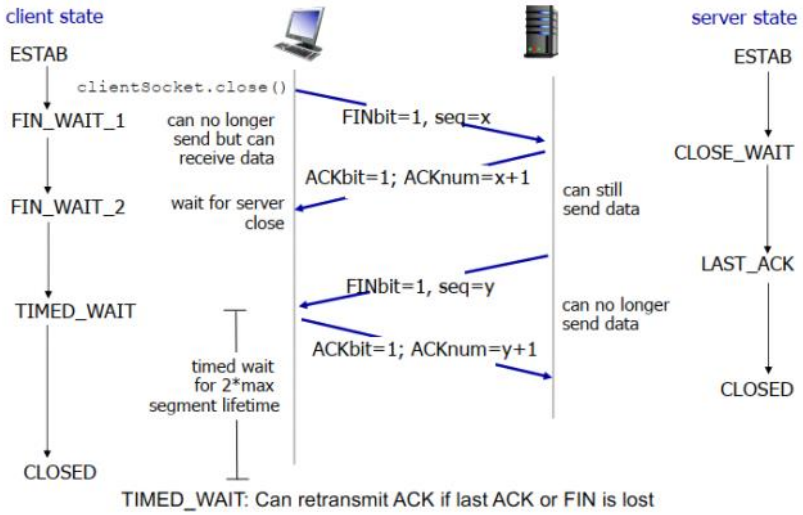
Once these three steps have been completed, the client and server hosts can send segments containing data to each other. In each of these future segments, the SYN bit will be set to zero.

Suppose the SYN packet gets lost because the packet got lost inside the network or the server discarded it because it was too busy. Eventually no SYNACK packet arrives. The sender's timer for the SYNACK packet will time out and they will retransmit the SYN packet if needed. The sender has no idea how far away the receiver is, so it is hard to guess a reasonable length of time to wait. TCP by default **should** use 3 seconds, although some implementations use 6 seconds.

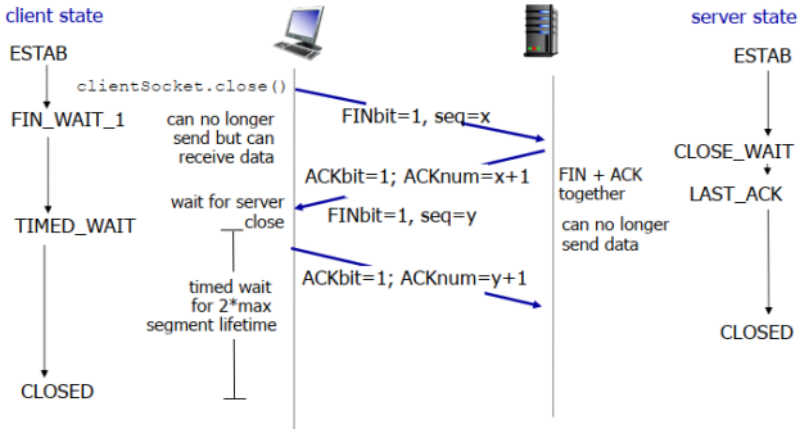
Closing a TCP connection

The client application process issues a close command. This causes the client TCP to send a special TCP segment to the server process. This special segment has a flag bit in the segment's header, the FIN bit , set to 1. When the server

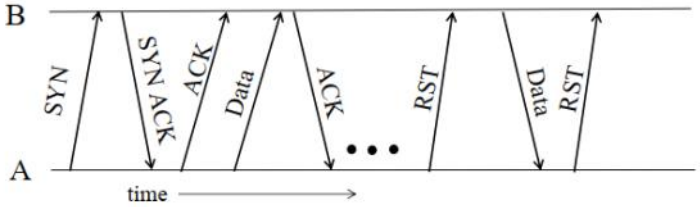
receives this segment, it sends the client an acknowledgment segment in return. The server then sends its own shutdown segment, which has the FIN bit set to 1. Finally, the client acknowledges the server's shutdown segment. At this point, all the resources in the two hosts are now deallocated.
The server and client can terminate one at a time.



Both client and server can close their connections at the same time

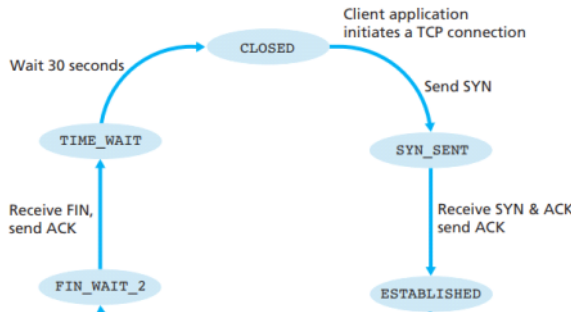


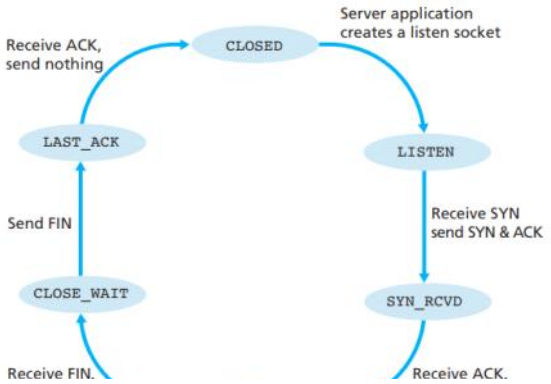
Consider a scenario where a TCP connection is abruptly terminated between host A and host B



- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

During the life of a TCP connection, the TCP protocol run ***states***.





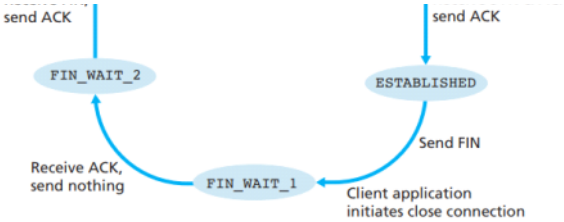


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

TCP SYN Attack (SYN flooding) (self-study)

- Miscreant creates a fake SYN packet
  - Destination is IP address of victim host (usually some server)
  - Source is some spoofed IP address
- Victim host on receiving creates a TCP connection state i.e allocate buffers, creates variables, etc and sends SYN ACK to the spoofed address (half-open connection)
- ACK never comes back
- After a timeout connection state is freed
- However for this duration the connection state is unnecessarily created
- Further miscreant sends large number of fake SYNs
  - Can easily overwhelm the victim
- Solutions:
  - Increase size of connection queue
  - Decrease timeout wait for the 3-way handshake
  - Firewalls: list of known bad source IP addresses
  - TCP SYN Cookies (explained on next slide)

TCP SYN Cookie (self-study)

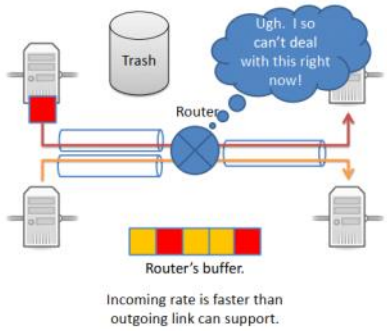
- On receipt of SYN, server does not create connection state
- It creates an initial sequence number (*init_seq*) that is a hash of source & dest IP address and port number of SYN packet (secret key used for hash)
  - Replies back with SYN ACK containing *init_seq*
  - Server does not need to store this sequence number
- If original SYN is genuine, an ACK will come back
  - Same hash function run on the same header fields to get the initial sequence number (*init_seq*)
  - Checks if the ACK is equal to (*init_seq*+1)
  - Only create connection state if above is true
- If fake SYN, no harm done since **no state was created**

http://etherealmind.com/tcp-syn-cookies-ddos-defence/

Principles of Congestion Control

Informally congestion is when too many sources send too much data at a rate which is too fast for the network to handle. This is different from flow control, which is about not overflowing the receiver's buffer. Congestion occurs when packets are lost due to buffer overflows at routers, or when there a long delays because of queueing in router buffers.

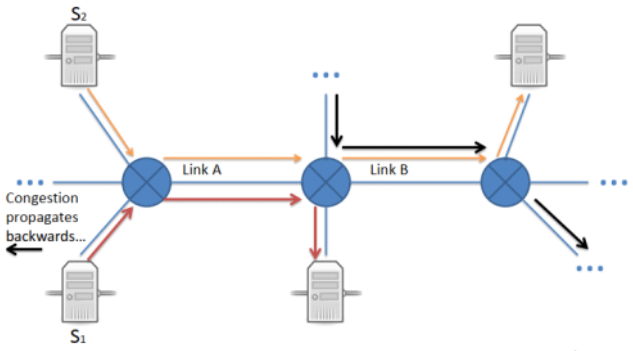
Congestion



Without congestion control, congestion:

- Increases delivery latency
- Increases loss rate
- Increases (many unnecessary) retransmissions
- Wastes capacity of traffic that is never delivered
- Increases congestion, and the cycle continues…

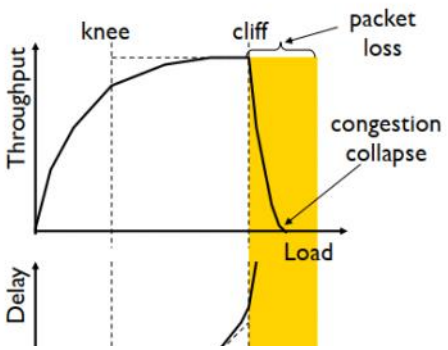
Congestion Collapse



Unrelated traffic passes through and congests link B. This causes S_2's traffic to be dropped at link B, so it starts retransmitting on top of what it was sending.
Now S_2 is sending lots of traffic over link A, that has no hope of crossing link B.
Increased traffic from S_2 causes link A to become congested and now S_1 starts retransmitting.

Cost of Congestion

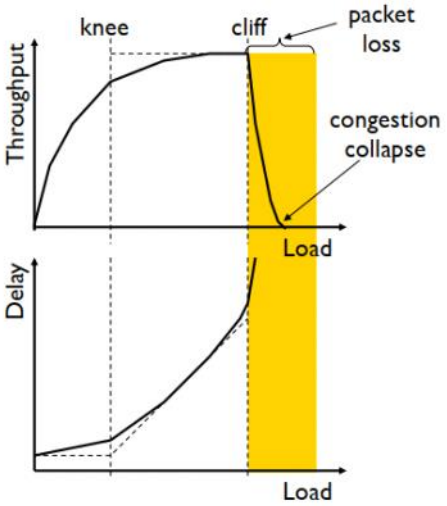
- Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
- Cliff – point after which



and now S_1 starts retransmitting.

Cost of Congestion

- Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to drop to zero (congestion collapse)
  - Delay approaches infinity



There are two broad approaches to congestion control:

- ***End-to-end congestion control*** - the network layer proves no explicit support to the transport layer for congestion control purposes. End systems infer congestions based on observed loss and delay. TCP takes this approach towards congestion control. TCP segment loss (indicated by a timeout or three duplicate ACKs) is taken as an indication of network congestion
- ***Network-assisted congestion control*** - routers provide explicit feedback to the sender and/or receiver refarding the congestion state of the network. This feedback may be a single bit indicating congestion at a link (used by IBM SNA, DECbit, TCP/IP ENC, ATM) or it can be an explicit rate for the sender to send at

TCP Congestion Control

TCP's approach is to have each sender limit the rate at which it sends traffic into its connections as a function of perceived network congestion. If a TCP sender perceives that there is little congestion on the path between itself and the destination, then the TCP sender increases its send rate; if the sender perceives that there is congestion along the path, then the sender reduces its send rate.

TCP has a congestion window, cwnd, which limits the rate at which a TCP sender can send traffic into the network. The TCP sending rate is generally goes like this: TCP sends cwnd bytes, wait RTT for ACKs, then send more bytes. So, $sending\ rate = \frac{cwnd}{RTT}\ bytes/sec$. By adjusting the value of cwnd, the sender can adjust the rate at which it sends data into its connection.

TCP windows summary

Congestion Window	cwnd	Limits how many bytes can be sent without overflowing routers Computed by the sender using a congestion control algorithm
Flow Control Window	AdvertisedWindow/rwnd	Specifies how many bytes can be sent without overflowing the receiver's buffer Determined by the receiver and reported to the sender in TCP segments

The sender's $window = min\{cwnd, rwnd\}$.
We will assume that rwnd is larger than cwnd, so that the amount of unacknowledged data at the sender is solely limited by cwnd.

Let us define a ***loss event*** at a TCP sender as the occurrence of either a timeout or the receipt of three duplicate ACKs from the receiver.
Duplicate ACKs indicate that the network is capable of delivering some segments, while timeouts indicate that several losses may have occurred.

Rate Adjustment

The basic logic of rate adjustment is:

- If acknowledgements of previously unacknowledged segments are received, we increase the sending rate
  - If acknowledgements arrive at a slow rate, the congestion window will be increased at a slow rate
  - If acknowledgements arrive at a high rate, the congestion window will be increased more quickly

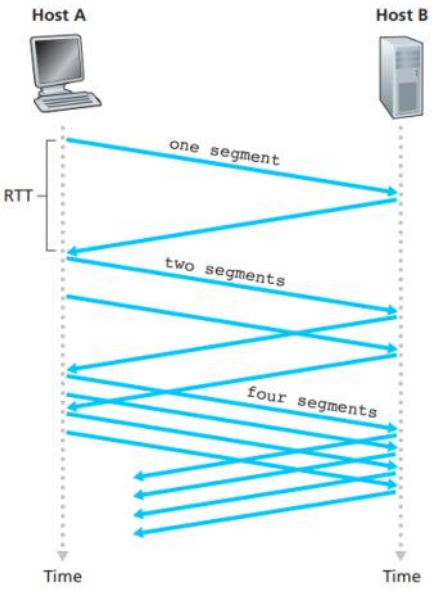
  Because TCP uses acknowledgements to trigger (or clock) its increase in congestion window size, it is said to be ***self-clocking***

- If loss is detected, we decrease the sending rate.

How we increase and decrease the rate depends on the phase of congestion control we are in:
- Discovering available bottleneck bandwidth vs.
- Adjusting to bandwidth variations

TCP Congestion-Control Algorithm

TCP's congestion control algorithm has three major components:
1. Slow start
2. Congestion avoidance
3. Fast recovery (not covered)



Slow Start

Slow start has the goal of estimating available bandwidth. It starts off with a slow sending rate (for safety) then increases the rate exponentially (for efficiency) until the first loss event.

When a TCP connection begins, the value of `cwnd` is typically initialised to a small value of 1 MSS, resulting in an initial sending rate of roughly $\frac{MSS}{RTT}$. In slow start, the value of cwnd begins at 1 MSS and increased by 1 MSS every time a transmitted segment is first acknowledged. Essentially cwnd is doubled every RTT.

Slow start's exponential growth can end in three ways, but we will focus on the second way. `ssthresh` (slow start threshold) is half the value of `cwnd` when congestion was last detected. So upon detection of a timeout or loss, $ssthresh = \frac{cwnd}{2}$. When `cwnd` reaches `ssthresh`, slow start ends and TCP transitions into congestion avoidance mode.

Congestion Avoidance (AIMD)

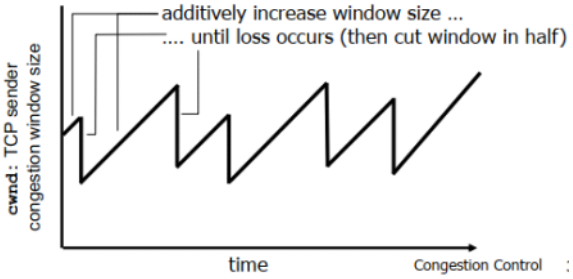
Slow start gave us an estimate of the available bandwidth. Now we want to track variations in this available bandwidth by oscillating around its current value. We can do this by repeated probing (increasing sending rate) and backing off (decreasing sending rate). TCP used *Additive Increase Multiplicative Decrease* (AIMD) to do this.

The approach of AIMD is for the sender to increase transmission rate (window size), by probing for usable bandwidth, until loss occurs.

Additive increase is when we increase cwnd by 1 MSS every RTT until loss is detected. So for each successful RTT, $cwnd = cwnd + 1$ and for each arriving ACK, $cwnd = cwnd + \frac{1}{cwnd}$

Multiplicative decrease is when we cut cwnd by half after we detect loss.

AIMD saw tooth behavior: probing for bandwidth

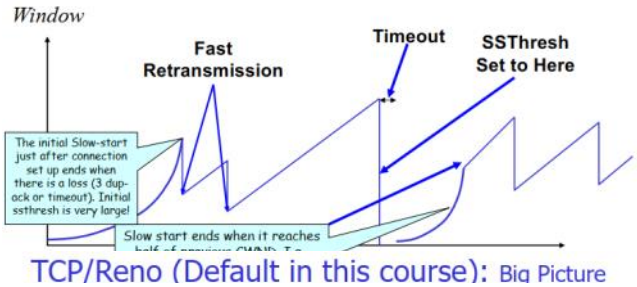


Putting it together

At the sender,
- `cwnd` is initialised to small constant (usually 1 MSS)
- `ssthresh` is initialised to a large constant (so that initial slow start can learn the network condition fast)
- We have a `dupACKcount` and `timer`

When we receive acknowledgement for new data and
```
if cwnd < ssthresh (slow start phase)
    cwnd += 1
else (congestion avoidance phase/additive increase)
    cwnd = cwnd + 1/cwnd
```

When we receive a duplicate acknowledgement
```
dupACKcount++
if dupACKcount == 3
    fast retransmit
    ssthresh = cwnd/2
```



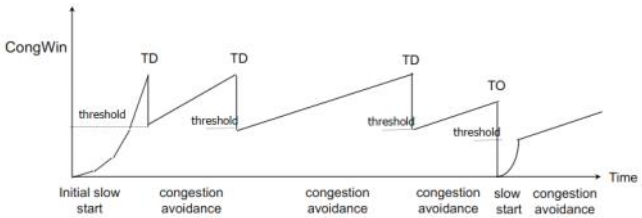
TCP/Reno (Default in this course): Big Picture

```
dupACKcount++
if dupACKcount == 3
     fast retransmit
     ssthresh = cwnd/2
     cwnd = cwnd/2
```

When we have a timeout

```
ssthresh = cwnd/2
cwnd = 1
```

TCP/Reno (Default in this course): Big Picture



TD: Triple duplicate acknowledgements
TO: Timeout

TCP Flavours

TCP has different flavours:

- TCP-Reno (assumed default in this course)
    - On timeout, `cwnd = 1`
    - On triple dup ACK, `cwnd = cwnd/2`
- TCP-Tahoe (old/original version)
    - On timeout and triple dup ACK, `cwnd = 1`

Network Layer: Data Plane

Saturday, 30 March 2019 3:52 PM

Overview of the Network Layer

The primary role of the network layer is to move packets from a sending host to a receiving host. All packets use a common ***Internet Protocol***. When sending a transport segment from the sending host to the receiving host, the sending side encapsulates segments into datagrams, while the receiving side delivers segments to the transport layer. There is a network layer protocol in **every** host and route. The router examines the header fields in all IP datagrams passing through it.

There are two important network-layer functions:

- ***Forwarding*** - when a packet arrives at a router's input link, the router must move the packet to the appropriate output link
- ***Routing*** - the network layer determines the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as ***routing algorithms***.

A router routes in advance, then forwards when a packet arrives. It needs to know where the packet needs to go before it forwards.

Every router has a **forwarding table**. A router forwards a packet by examining the value of a field in the arriving packet's header, and then using this header value to index into the router's forwarding table. The value stored in the forwarding table entry for that header indicates the router's outgoing link interface to which that packet is to be forwarded. Depending on the network-layer protocol, the header value could be the destination address of the packet or an indication of the connection to which the packet belongs. Routing algorithms determine the contents of the router's forwarding table.

Data plane

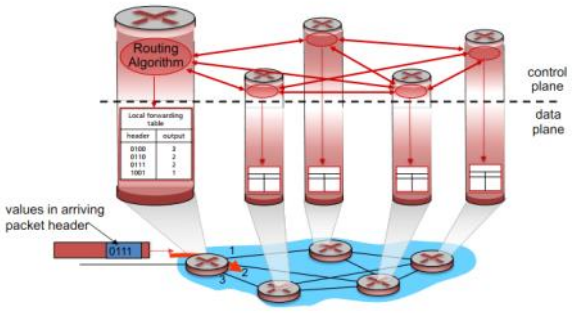
- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port
- forwarding function

Control plane

- network-wide logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: centralised (remote) servers

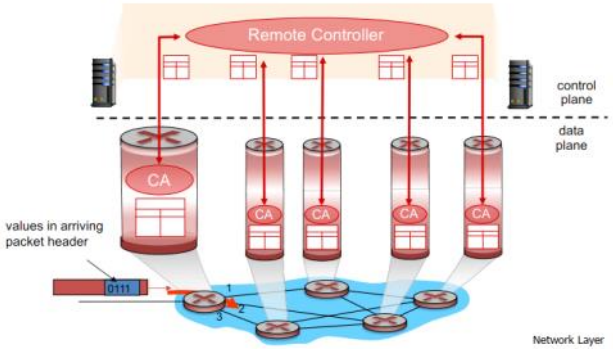
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



Logically centralized control plane (SDN)

A distinct (typically remote) controller interacts with local control agents (CAs)



In SDN (software defined networking); routers do not need to create the routing table, it is computed and distributed to them, by a remote controller. Hence, the routing device only needs to perform forwarding.

Network Service Model

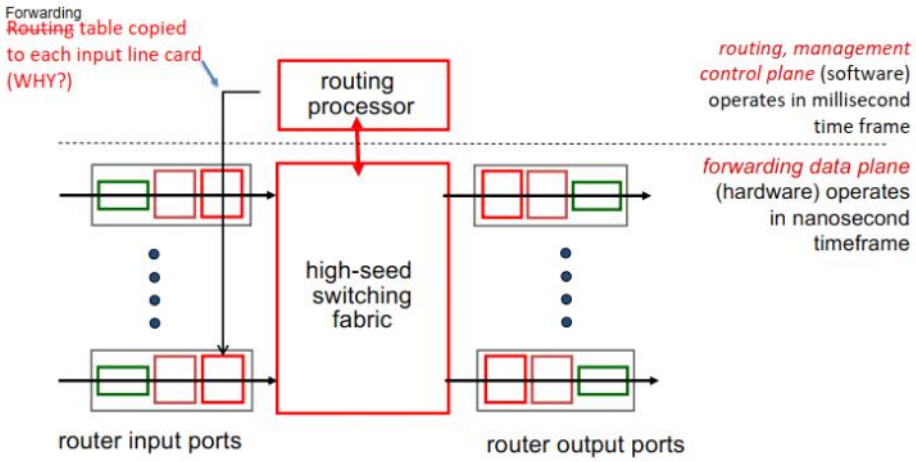
The ***network service model*** defines the characteristics of end-to-end delivery of packets between sending and receiving

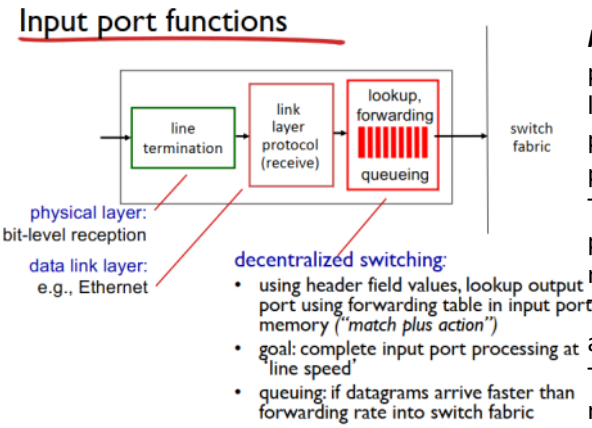
hosts. The possible services that the network layer could provide include:

- Guaranteed delivery - a packet sent by a source host will *eventually* arrive at the destination host
- Guaranteed delivery with bounded delay - a packet sent by a source host will *eventually* arrive at the destination host within a specified host-to-host bound (e.g. 100 msec)
- In-order packet delivery - packet arrive at the destination in the order that they were sent
- Guaranteed minimal bandwidth - as long as the sending host transmits bits (as part of packets) at a rate below the specified bit rate, then all packets are eventually delivered to the destination host
- Security - the network layer could encrypt all datagrams at the source and decrypt them at the destination

What's inside a router?

A high-level view of a generic router architecture:





Input ports terminate the incoming physical link at a router and performs link-layer functions needed to interoperate with the link layer at the other side the incoming link. A lookup function is also performed at the rightmost box to determine the router output port.

The ***switching fabric*** connects the router's input ports to its output ports. This switching fabric is completely contained within the router.

The ***output port*** stores packets received from the switching fabric and transmits these packets on the outgoing link.

The ***routing processor*** performs control-plane functions. It executes routing protocols, maintains and computes forwarding tables and attached link state information.

Input Port Processing and Destination-Based Forwarding

The lookup performed in the input port is central to the router's operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric. The forwarding table is computed and updated by the routing processor or is received from a remote SDN controller, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards.

Longest prefix matching is when the router matches the longest prefix of the packet's destination address with the entries in a forwarding table, to determine which output link it should go through. If there is a match, the router forwards the packet to a link associated with the match, otherwise it forwards the packet to a default interface.

Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

11001000 00010111 00010110 10100001 is forwarded to link interface 0.
11001000 00010111 00011000 10101010 is forwarded to link interface 1.

Longest prefix matching is often performed using Ternary Content Addressable Memories (TCAMs). With a TCAM, a 32-

bit IP address is presented to memory, which returns the content of the forwarding table entry for that entry in one clock cycle, regardless of table size.

Switching (Fabrics)

The ***switching fabric*** transfers the packet from the input buffer to the appropriate output buffer. ***Switching rate*** is the rate at which packets can be transferred from inputs to outputs.

Switching can be accomplished in a number of ways:

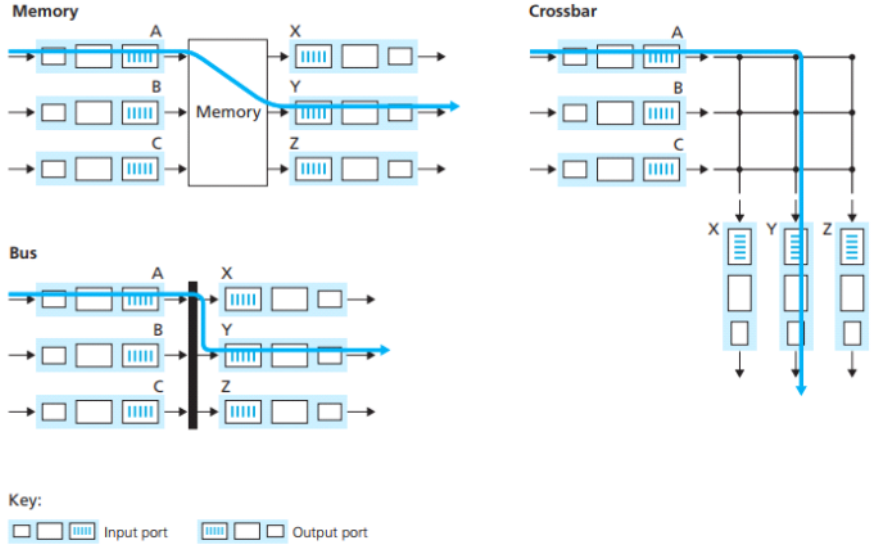


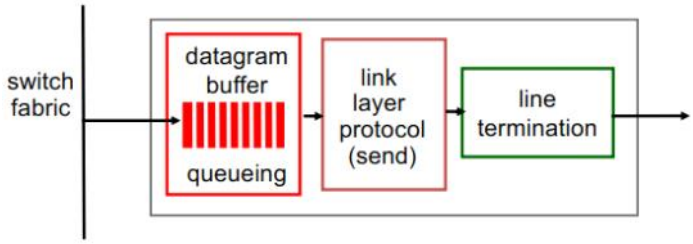
Figure 4.8 ♦ Three switching techniques

- ***Switching via memory*** - which is implemented by first generation routers. Switching was done under direct control of the CPU (routing processor). The arriving packet signalled the routing processed via interrupt, and it was then copied from the input port into processor memory. The destination address is extracted from the header, the appropriate output link is found and the packet was copied to the output port's buffer. The speed of this method is limited by memory bandwidth.
- ***Switching via a bus*** - an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor. The input port pre-pends a switch-internal label (header) to the packet indicating the local port output to transfer the packet to. When it arrives at the output port, the label is removed. The switching speed is limited by the bus speed.
- ***Switching via an interconnection network (crossbar)*** - a crossbar switch is an interconnection network consisting of 2N buses that connect N input ports to N output ports. Each vertical bus intersects a horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric controller. This method is capable of forwarding multiple packets in parallel. It is also non-blocking (a packet being forwarded to an output port will not be blocked from reaching that output port as long as no other packet is currently being forwarded to that output port).

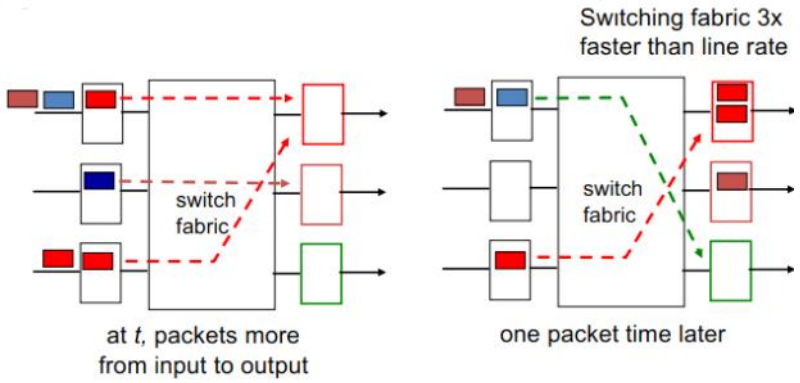
Outport Port Processing

Output port processing takes packets that have been stored in the output port's memory and transmits them over the output link. This includes selecting and de-queuing packets for transmission, and performing the needed link-layer and physical-layer transmission functions.

Output Queueing

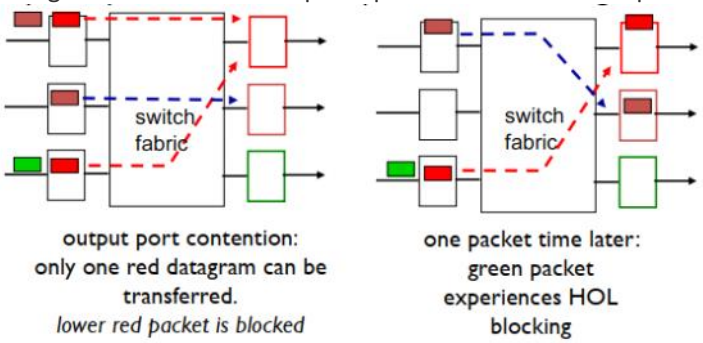


Buffering is required when datagrams arrive from the switching fabric faster than the transmission rate. As the queues grow large, the router's memory can eventually be exhausted and ***packet loss*** will occur when no memory is available to store arriving packets. A rule of thumb is that the average buffering size if equal to the *typical* RTT (e.g. 250ms) time link capacity *C.* A recent recommendation is that with ***N*** flows, buffering is equal to $\frac{RRT \times C}{\sqrt{N}}$



Input Queueing

If the switch fabric is slower than the input ports combined, queueing may occur at the input ports. Once again, queueing delays and loss can occur because of input buffer overflows. Head-of-the-line blocking is when a queued datagram at the front of a queue prevents others in the queue from moving forward.



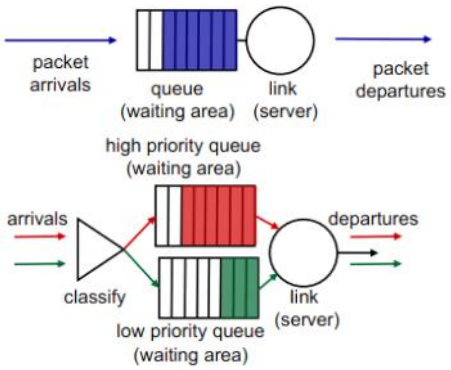
Packet Scheduling

Scheduling is determining the order in which queued packets are transmitted over an outgoing link.

First-In-First-Out (FIFO)

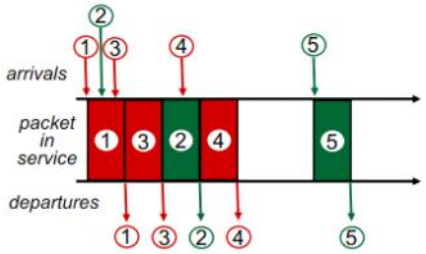
The FIFO scheduling disciple selects packets for link transmission in the same order in which they arrived at the output link queue. When a packet arrives at a buffer with insufficient space to hold the arriving packet, the queue's packet-discarding policy determines which packet will be dropped:

- Tail drop - drop the arriving packet
- Priority drop - drop the on priority basis
- Random - drop a random packet



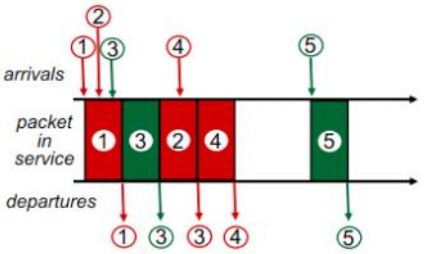
Priority Queueing

In priority queuing, packets arriving at the output link are classified into priority classes (network management info, real-time, SMTP, IMAP email packets). There multiple priority classes and each typically has their own queue. When choosing a packet to transmit, the priority queuing discipline will transmit a packet from the highest priority class that has a non-empty queue. The choice among packet in the same priority queue is typically done in a FIFO manner.



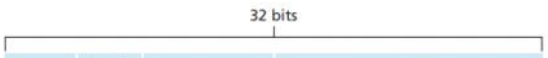
Round Robin (RR) Scheduling

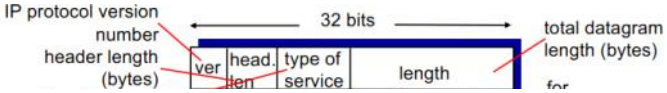
Packets are sorted into classes as with priority queuing, however, instead of having a strict service priority among classes, a round robin scheduler alternates services among the classes. This method cyclically scans queues and sends one complete packet from each class if the queues in the class are non-empty



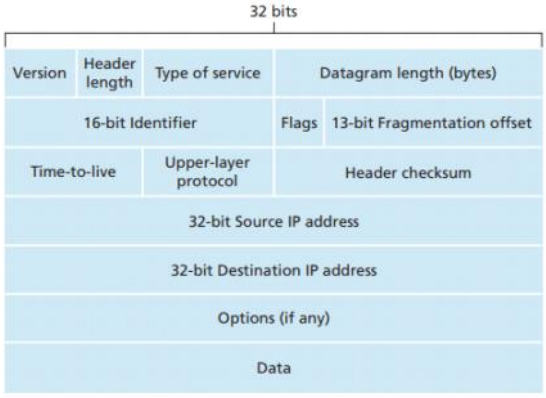
The Internet Protocol (IP): IPv4, Addressing, and more

IPv4 Datagram Format





IPv4 Datagram Format



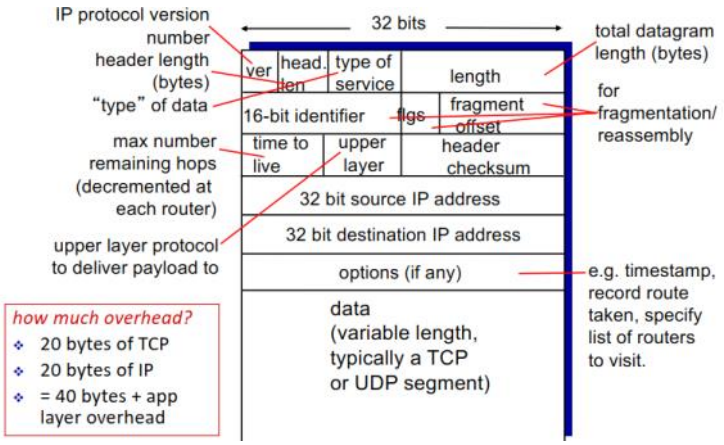


Figure 4.13 ♦ IPv4 datagram format

A typical IP datagram has a 20 byte header. Some things to note:

Version (4-bits) - indicated the version of the IP protocol. Necessary to determine how to interpret the remainder of the IP datagram. Usually "4" for IPv4 but sometimes "6" for IPv6

- ***Header length*** (4-bits) - is the number of 32-bit words in the header. Typically "5" for a 20 byte IPv4 header.
- ***Type of service*** (8-bits) - allows pacekts to be treated differently based on needs. For example, it might be useful to distinguish real-time datagrams (IP telephony) and non-real-time traffic (FTP).
- ***Datagram length*** (16-bits) is the total length of the IP datagram (including the header and payload) measured in bytes. The maximum size if 65535 ($2^{16} - 1$)
- ***Time-to-live*** (8-bits) - the number of times a datagram can arrive at a router. The field is decremented each time the datagram is processed at a router. If the TTL value reaches 0, the router drops the packet.
- ***Upper-layer protocol*** (8-bits) - identifies the higher-level protocol. e.g. "6" for TCP and "17" for UDP. For a list of all possible values: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml. This is important for demultiplexing at the receiving host
- ***Checksum*** (16-bits) - aids the router in detecting bit errors in each received IP datagram. Datagrams with errors are usually discarded. The checksum is recalculated at every router. Note that only the IP header is checksummed at the IP layer, while the TCP and UDP checksum is computed over the entire TCP/UDP segment.

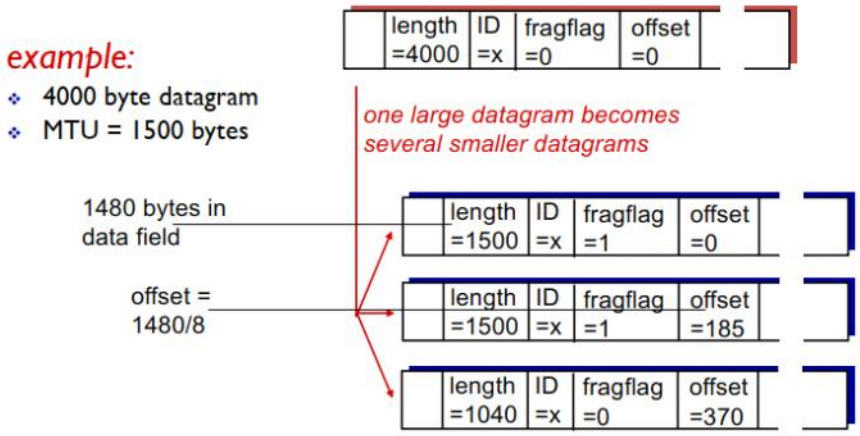
Datagram Fragmentation

Some protocols can carry big datagrams, while other protocols can only carry little datagrams. The maximum amount of data that a link-layer frame can carry is called the ***maximum transmission unit (MTU)***.

Suppose you have an IP datagram and the MTU of the outgoing link is smaller than the length of your datagram. The solution to this is to fragment the payload in the IP datagram into two or more smaller IP datagrams, encapsulate each of these smaller IP datagrams in a separate link-layer frame and send these frames over the outgoing link. Each of the smaller datagrams are called ***fragments***. The fragments need to be reassembled before they reach the transport layer at the destination. The job of datagram reassembly is done at the end systems rather than in network routers.

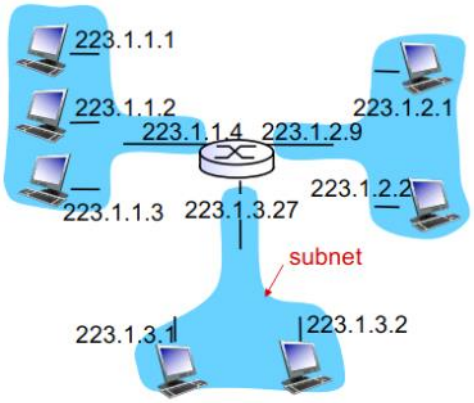
IP header bits are used to identify and order related fragments.

When a router needs to fragment a datagram, each resulting datagram (that is, fragment) is stamped with the source address, destination address, and identification number of the original datagram. When the destination receives a series of datagrams from the same sending host, it can examine the identification numbers of the datagrams to determine which of the datagrams are actually fragments of the same larger datagram. Because IP is an unreliable service, one or more of the fragments may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, **the last fragment has a flag bit set to 0**, **whereas all the other fragments have this flag bit set to 1**. Also, in order for the destination host to determine whether a fragment is missing (and also to be able to reassemble the fragments in their proper order), the offset field is used to specify where the fragment fits within the original IP datagram.



IPv4 Addressing

An IP address is a 32-bit identifier for a host and a router interface. An ***interface*** is the connection between the host/router and a physical link. Router's typically have multiple interfaces, while hosts typically have one or two. Each interface is associated with an IP address. A portion of an interface's IP address will be determined by the subnet it is connected to.



network consisting of 3 subnets

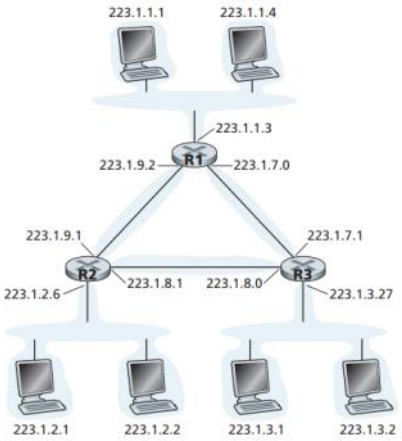
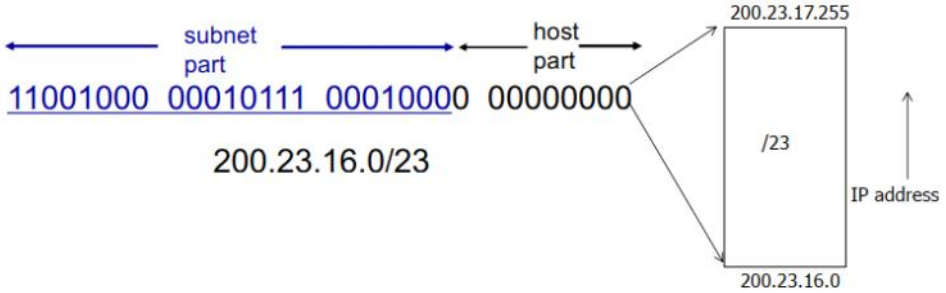


Figure 4.17 ♦ Three routers interconnecting six subnets

A ***subnet*** is simply an IP network. Devices in a subnet can physically reach each other without an intervening router. To determine a subnet, detach each interface from its host or router, reaching an island of isolated networks; each isolated network is a subnet. IP addressing assigns an address to each subnet (e.g. 223.1.1.0/24, where the /24 notation is sometimes known as a ***subnet mask***; it indicates that the first 24 bits define the subnet address).

The Internet has an address assignment strategy known as ***Classes Interdomain Routing*** (***CIDR*** - pronounced *cider*). CIDR generalises the notion of subnet addressing. As with subnet addressing, the 32-bit IP address is divided in to two parts and has the dotted decimal form *a.b.c.d/x*, where *x* indicates the number of bits in the subnet part of the address AKA the ***prefix.*** The IP addresses of devices in an subnet share the common prefix.



A ***subnet mask*** is used in conjunction with the network address to indicate how many higher order bits are used for the network part of the address. e.g 223.1.1.0/24 is equivalent to 223.1.1.0 with subnet mask 255.255.255.0

Host B	Dot-decimal address	Binary
IP address	223.1.1.2	11111101.00000001.00000001.00000010
Subnet Mask	255.255.255.0	11111111.11111111.11111111.00000000
Network Part	223.1.1.0	11111101.00000001.00000001.00000000
Host Part	0.0.0.2	00000000.00000000.00000000.00000010

There is another type of IP address; the IP broadcast address 255.255.255.255. When as host sends a datagram with this destination, the message is delivered to all hosts on the same subnet.

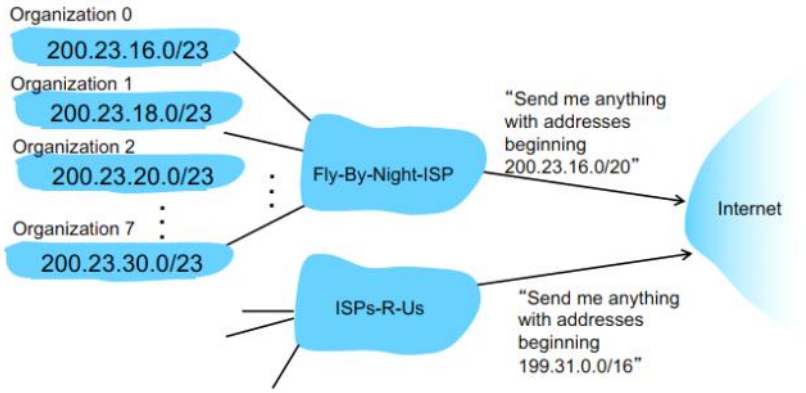
Obtaining a Block of Address

In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP. E.g if the ISP was allocated the address block 200.23.16.0/20 it could divide its address block into eight equal-sized contiguous address blocks and give one of these address blocks out to each of up to eight organizations that are supported by this ISP

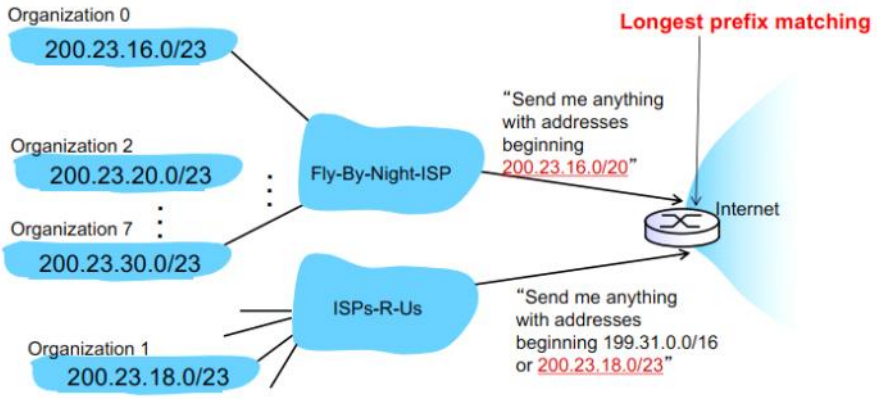
ISP's block	200.23.16.0/20	11001000 00010111 00010000 00000000
Organization 0	200.23.16.0/23	11001000 00010111 00010000 00000000
Organization 1	200.23.18.0/23	11001000 00010111 00010010 00000000
Organization 2	200.23.20.0/23	11001000 00010111 00010100 00000000
...	...	...
Organization 7	200.23.30.0/23	11001000 00010111 00011110 00000000

Who allocated the address space to the ISP and other organisations? The Internet Corporation for Assigned Names and Numbers (ICANN). They are responsible for managing IP addresses and DNS root servers.

Hierarchical addressing allows efficient advertisement of routing information:



When an organisation decides to switch ISPs, we can make more specific route for the moving organisation. For example if organisation 1 were to move:



Routers in the Internet will have two entries in their table:

200.23.16.0./20	Fly-by-Night-ISP
200.23.18.0/23	ISPs-R-Us

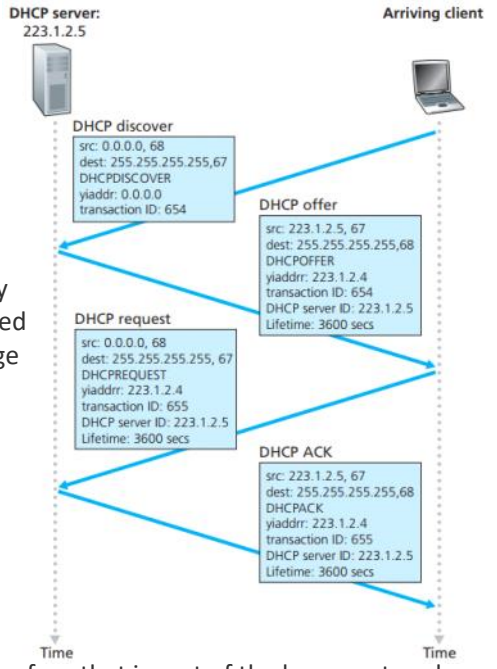
The longest prefix match will cause datagrams addressed to the subnet 200.23.18.0/23 to be sent to ISPs-R-Us.

IP addresses are allocated as blocks and have geographical significant. It is possible to determine the geographical location of an IP address.

Obtaining a Host Address: the Dynamic Host Configuration Protocol (DHCP)

Once an organisation has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organisation. A system administrator will typically manually configure the IP addresses into the router. Host addresses can also be configured manually but are typically done using **Dynamic Host Configuration Protocol (DHCP).** DHCP allows a host to obtain (be allocated) and IP address automatically. A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network, or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network.

DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the ***default gateway***), and the address of its local DNS server.



DHCP overview:

1. The host broadcasts a *DHCP discover* message
2. The DHCP server responds with a *DCHP offer* message
3. The host requests an IP address with a *DHCP request* message
4. The DHCP server sends the request through and *DCHP ack* message

DHCP uses UDP and port number 67 (server side) and 68 (client side). Usually the MAC address is used to identify clients. The DHCP server can be configured with a "registered list" of acceptable MAC addresses. The DHCP offer message includes the IP address, length of lease, subnet mask, DNS servers, default gateway etc.

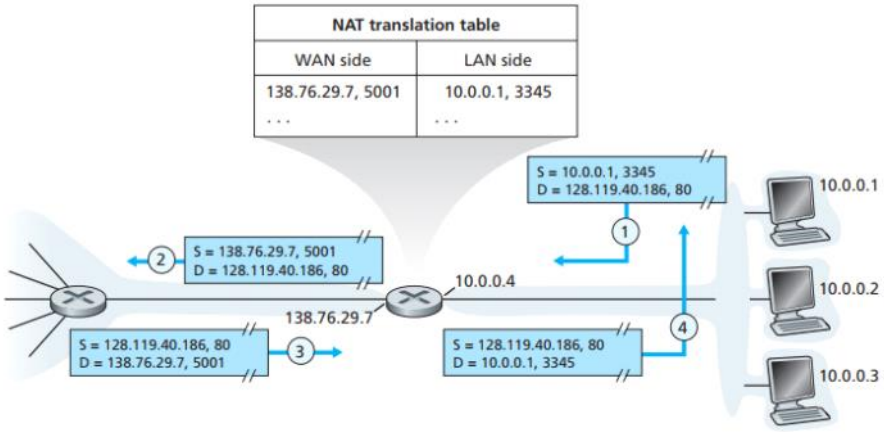
Security holes in DHCP include:

- DoS attack by exhausting the pool of IP addresses
- Masquerading as a DHCP server
- Authentication for DHCP -RFC3118

Network Address Translation

In our figure below, the NAT-enabled router, residing in the home, has an interface that is part of the home network on the right. Addressing within the home network is exactly as we have seen in IPv4 addressing - all four interfaces in the home network have the same subnet address 10.0.0/24. The address space 10.0.0.0/8 is one of the three portions of the IP address space that is reserved for a private network.

A ***private network*** or ***realm with private addresses*** refers to a network, whose addresses only have meaning to devices within that network. Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing. However, packets forwarded *beyond* the home network into the larger global Internet clearly cannot use these addresses (as either a source or a destination address) because there are hundreds of thousands of networks using this block of addresses. That is, the 10.0.0.0/24 addresses can only have meaning within the given home network.



The NAT-enabled router does not *look* like a router to the outside world. Instead the NAT router behaves to the outside world as a *single* device with a *single* IP address. Hence a local network uses just one IP address as far as the world is concerned. The NAT router must:

- For **outgoing datagrams**: replace (source IP address, port #) of every outgoing datagram with (NAT IP address, new port #)
- Remember the NAT translation table: every mapping of (source IP address, port #) to (NAT IP address, new port #)
- For **incoming datagram**: replace (NAT IP address, new port #) in the destination field of every incoming datagram with the corresponding (source IP address, port #) stored in the NAT table

Advantages of NAT:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network

Devices inside the local network are not explicitly addressable or visible by outside world
This can be an advantage if you want privacy, but it can also be a disadvantage because if you are a server, it will be

difficult for clients to find your IP address

When generating a new source port number, the NAT router can select any source port number that is not currently in the NAT translation table. (Note that because a port number field is 16 bits long, the NAT protocol can support over 60,000 simultaneous connections with a single WAN-side IP address for the router!) NAT in the router also adds an entry to its NAT translation table.

NAT is considered controversial because:

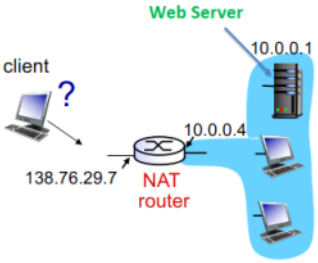
- Routers should only process the lower three layers (physical, link, network), yet NAT violates this by changing port numbers.
- The NAT protocol violates the so-called end-to-end argument; that is, hosts should be talking directly with each other, without interfering nodes modifying IP addresses and port numbers.
- We should use IPv6 to solve the shortage of IP addresses, rather than recklessly patching up the problem with a stopgap solution like NAT
- Any protocol that embeds IP or TCP-layer information in the application stream is likely to be broken by a basic NAT.

NAT: Practical Issues

- NAT modifies port # and IP address
  - *Requires recalculation of TCP and IP checksum*
- Some applications embed IP address or port numbers in their message payloads
  - DNS, FTP (PORT command), SIP, H.323
  - For legacy protocols, NAT must look into these packets and translate the embedded IP addresses/port numbers
  - Duh, What if these fields are encrypted ?? (SSL/TLS, IPSEC, etc)
  - **Q: In some cases why may NAT need to change TCP sequence number??**
- If applications change port numbers periodically, the NAT must be aware of this
- NAT Traversal Problems
  - E.g: How to setup a server behind a NAT router?
  - How to talk to a Skype user behind a NAT router?
  - Possible workarounds in next few slides

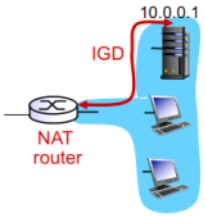
NAT traversal problem

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

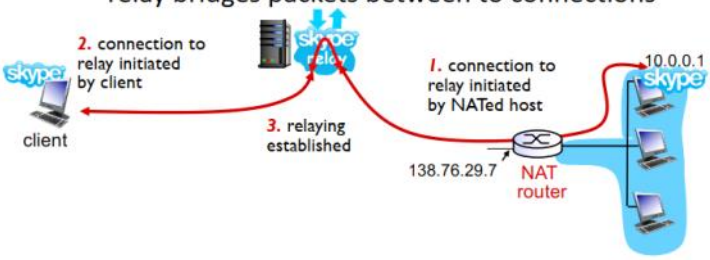


- *solution1:* statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 80) always forwarded to 10.0.0.1 port 80
- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

  i.e., automate static NAT port map configuration



- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between to connections



Network Layer: Control Plane

Wednesday, 10 April 2019 12:10 PM

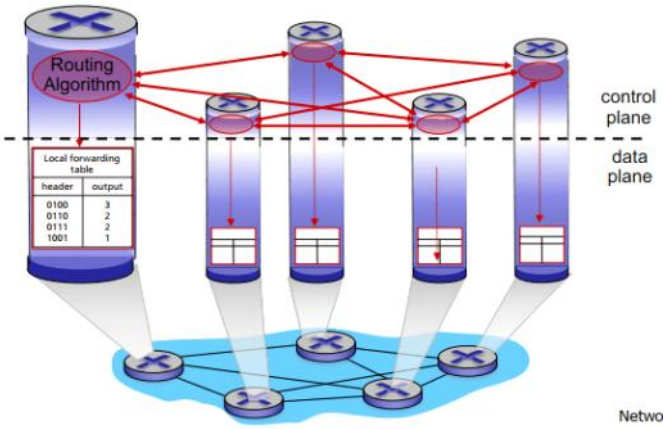
The purpose of the control plane is to compute, maintain and install forwarding and flow tables.

There are two network layer functions:

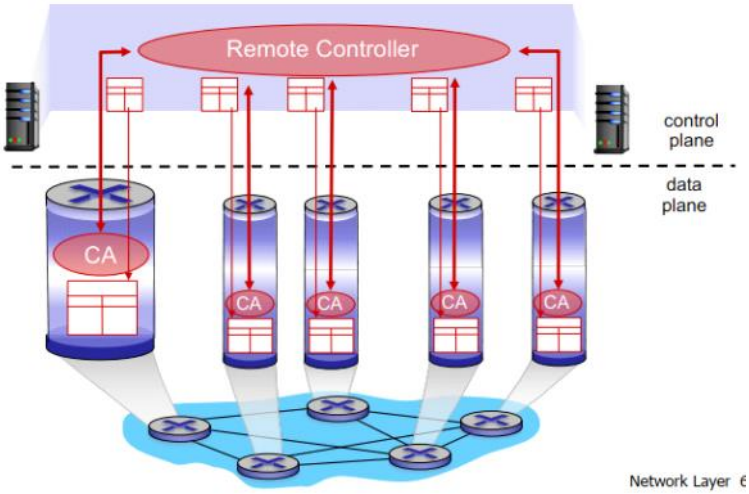
1. ***Forwarding*** - moving packets from the router's input to the appropriate router output (data plane)
2. ***Routing*** - determining the route taken by packets from source to destination (control plane)

There are two approaches to structuring the network control plane:

- ***Per-router control*** - there are individual routing components in each and every router. Each router interacts with each other in the control plane to compute forwarding tables

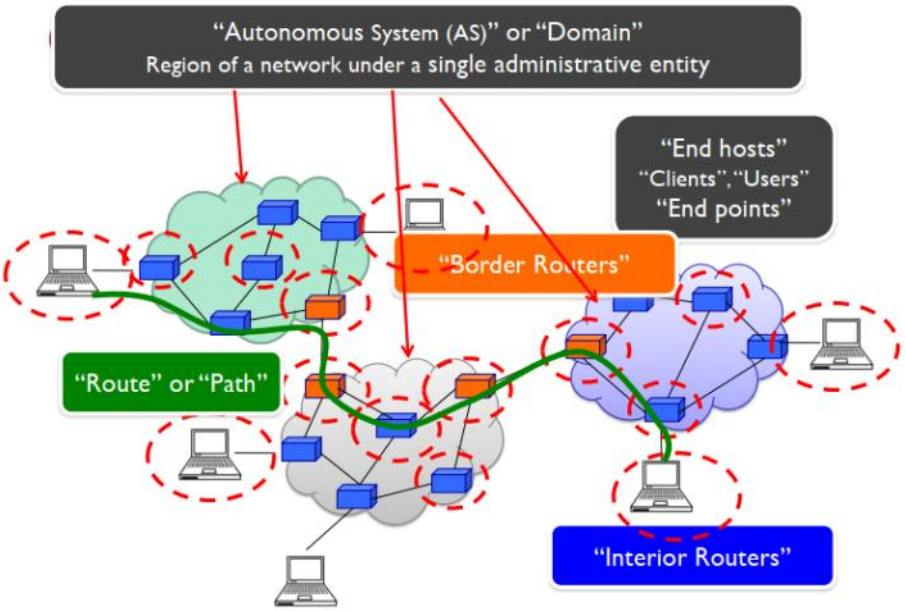


- ***Logically centralised control*** - a distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute and distribute forwarding tables to be used by each and every router



Routing Algorithms and Protocols

Recall that a routing algorithm determine the end-end path through a network, while a forwarding table determines the local forwarding at this router.



Internet routing works at two levels. Each Autonomous System runs an ***intra-domain*** routing protocol that establishes routes within its domain. An Autonomous System is a region of a network under a single administrative authority. It uses ***Link State*** algorithms such as Open Shortest Path First (OSPF) and ***Distance Vector*** algorithms such as Routing Information Protocol (RIP)

Autonomous Systems also participate in ***inter-domain*** routing protocols that establish routes between domains. It uses a ***Path Vector*** algorithm such as Border Gateway Protocol (BGP).

We will only focus on intra-domain routing protocols.

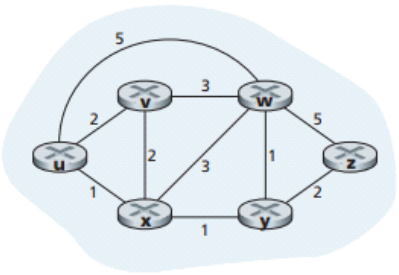


Figure 4.27 ♦ Abstract graph model of a computer network

We will use graph abstraction to formulate routing problems. In a graph $G = (N, E)$. N is our set of routers and E is the set of edges (physical links) that connect our routers. An edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link, the link speed, or the monetary cost associated with the link.
Assume that each host has a unique ID (address) and there is no particular structure to those IDs.

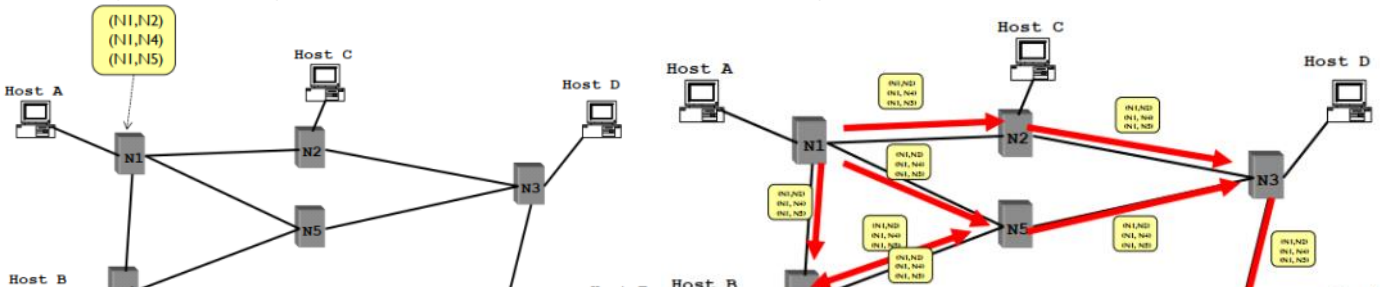
Our routing algorithms want to find a path that costs the least. For simplicity, we will assume that all links are equal, hence the *least cost path will be shortest path* and it will have the least number of hops.

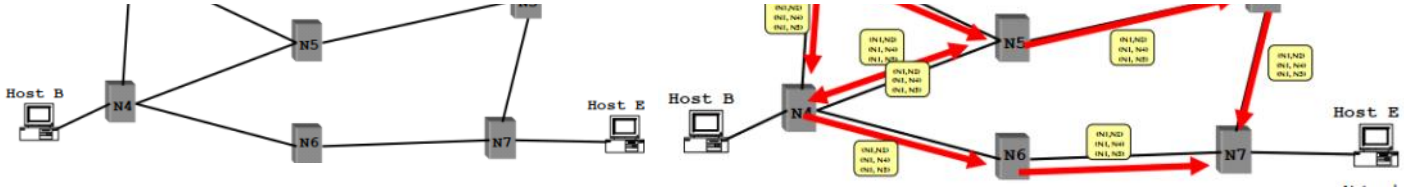
There are two classes of routing algorithms we will cover:

Link State (Global)	***Distance Vector (Decentralised)***
• Routers maintain the cost of each link in the network • Connectivity/cost changes flooded to all routers • Converges quickly (less inconsistency, looping) • Limited network sizes	• Routers maintain next hop and the cost of each destination • Connectivity/cost changes iteratively propagate from neighbour to neighbour • Require multiple rounds to converge • Scales to large networks

Link-State (LS) Routing Algorithm

In link state routing, each node maintains its local *link state* (LS); that is a list of its directly attached links and their costs. Each node floods its local link state to all other nodes in the network. On receiving a new LS message, a router forwards the message to all its neighbours other than the one it received the message from.





Flooding LSAs

Routers transmit ***Link State Advertisement (LSA)*** on their links. A neighbouring router forwards out on all links except the incoming link. It also keeps a copy locally, so that it does not forward the LSA if it has already been received before.

This flooding raises challenges of packet loss and out of order arrival. However, it can also be resolved through acknowledgements and retransmissions, sequence numbers, TTL for each packet

The result of the flood is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node. We use Dijkstra's algorithm to compute the shortest path between nodes.

Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as *u*) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the *k*th iteration of the algorithm, the least-cost paths are known to *k* destination nodes, and among the least-cost paths to all destination nodes, these *k* paths will have the *k* smallest costs. Let us define the following notation:

- ***D*(*v*)**: cost of the least-cost path from the source node to destination *v* as of this iteration of the algorithm.
- ***p*(*v*)**: previous/predecessor node (neighbor of *v*) along the current least-cost path from the source to *v.*
- ***c*(*u*,*v*)**: is the link cost from node *u* to *v*. it is ∞ if *u* and *v* are not direct neigbours.
- ***N*'** : subset of nodes; *v* is in *N*' if the least-cost path from the source to *v* is definitively known.

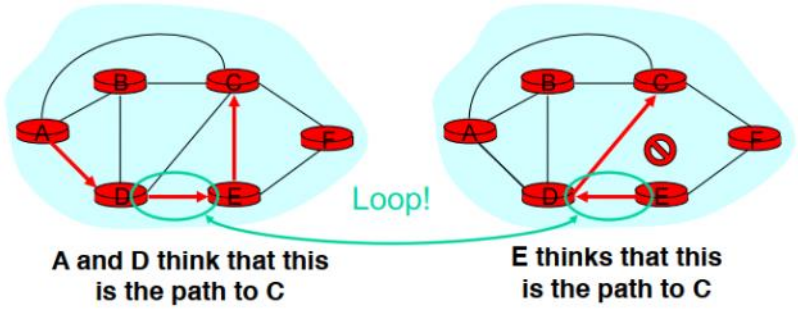
The global routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node *u* to every other node in the network.

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v is a neighbor of u
5        then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for each neighbor v of w and not in N':
12         D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    least path cost to w plus cost from w to v */
15 until N'= N
```

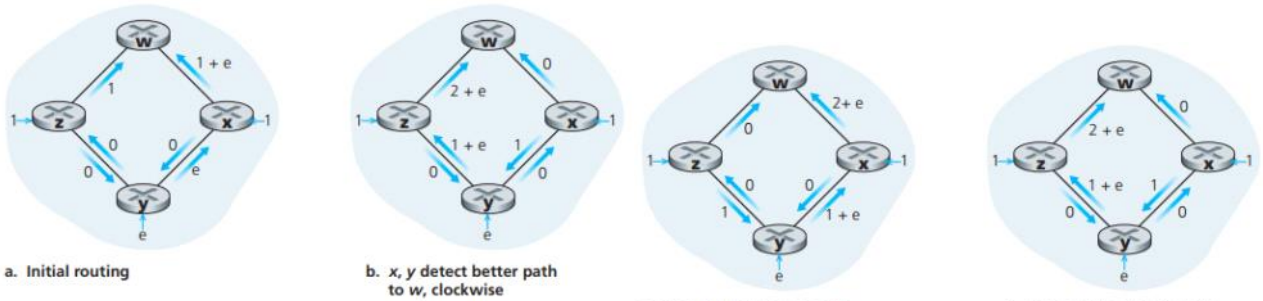
When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node. For each predecessor, we also have *its* predecessor, and so in this manner we can construct the entire path from the source to all destinations. The forwarding table in a node, say node *u*, can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from *u* to the destination.

Issues with Link State Routing

1. Scalability
   - How many messages needed to flood link state messages?
     - O(N x E), where N is #nodes; E is #edges in graph
   - Processing complexity for Dijkstra's algorithm?
     - $O(N^2)$, because we check all nodes w not in S at each iteration and we have O(N) iterations
     - more efficient implementations: O(N log(N) +E) using min-heap
   - How many entries in the LS topology database? O(E)
   - How many entries in the forwarding table? O(N)
2. Transient disruptions - an inconsistent link-state database means that some routers will know about link failures before others. This means that the shortest paths are no longer consistent. This causes transient forward loops.



3. Oscillations - if we make link costs equal to the load carries on the link, oscillations can occur due to congestion sensitive routing. One solution would be to mandate that link costs do no depend on the amount of traffic carried, but this does not help with avoiding congested links. Another solution is to ensure that not all routers run the LS algorithm at the same time.



Distance Vector (DV) Routing Algorithm

The ***distance vector (DV)*** algorithm is iterative, asynchronous, and distributed.
It is ***distributed*** in that each node receives some information from one or more of its *directly attached* neighbours, performs a calculation, and then distributes the results of its calculation back to its neighbours. It does not have a global view of the network.
It is ***iterative*** in that this process continues on until no more information is exchanged between neighbours. (Interestingly, the algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.)
The algorithm is ***asynchronous*** in that it does not require all of the nodes to operate in lockstep with each other.

Messages exchanged between routers have two components; the destination network (vector) and the cost to the destination (distance). It is also known as the ***Bellman-Ford algorithm***.

Each router maintains a routing table. Periodically, each router sends a copy of its table (destination, cost columns only) to directly connected routers. When *K* receives a table from its neighbouring router *J*, *K* updates its table if:

1. *J* knows a shorter route for a given destination
2. *J* knows a destination, *K* did not know about
3. *K* currently routes to a destination through *J*, and *J*'s cost to that destination has changed

If *K* updates or adds an entry in response to *J*'s message, it assigns the *next hop* as router *J* and it updates the cost; if *X* is the destination, the cost to $X = p(K,J) + p(J,X)$, where p(u,v) is the total cost of the path from node *u* to node *v*

Example: we are given these two routing tables. The cost of the path from router *K* to router *J* is 1. If router *J* is given router *K*'s routing table, router *J*'s table will be updated to the table on the left

Existing routing table for router K

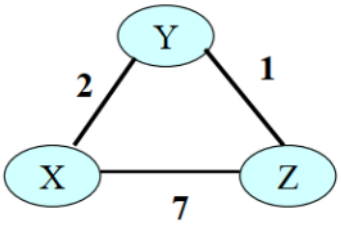
Destination	Cost	Next Hop
Net 1	0	Direct
Net 2	0	Direct
Net 4	8	Router L
Net 17	5	Router M
Net 24	6	Router J
Net 30	2	Router Q
Net 42	2	Router J

Routing table from neighbouring router J:

Destination	Cost
Net 1	2
Net 4	3
Net 17	6
Net 21	4
Net 24	5
Net 30	10
Net 42	3

Destination	Cost	Next Hop
Net 1	0	Direct
Net 2	0	Direct
Net 4 (*)	4	Router J
Net 17	5	Router M
Net 21 (*)	5	Router J
Net 24	6	Router J
Net 30	2	Router Q
Net 42 (*)	4	Router J

At bootup, each router initialises its routing table with directly connected network information. For example, the following shows a network with three nodes. The cost of an edge is shown next to the edge.



The initial routing table for X is:

Destination	Cost	Next Hop
Y	2	Direct
Z	7	Direct

After initialising their routing tables, the nodes start exchanging their routing tables with their neighbouring nodes. When a node receives a routing table, it updates its routing table according to the three distance vector update rules.

If we compute the initial routing tables for all the nodes in our example above and exchange their routing tables, we get:

Routing table for X		
Destination	Cost	Next Hop
Y	2	Direct
Z	7	Direct

Routing table for Y		
Destination	Cost	Next Hop
X	2	Direct
Z	1	Direct

Routing table for Z		
Destination	Cost	Next Hop
X	7	Direct
Y	1	Direct

Note: * indicates an entry has been updated.

Routing table for X		
Destination	Cost	Next Hop
Y	2	Direct
Z	3 *	Y *

Routing table for Y		
Destination	Cost	Next Hop
X	2	Direct
Z	1	Direct

Routing table for Z		
Destination	Cost	Next Hop
X	3 *	Y *
Y	1	Direct

After this update, X sends its routing table to neighbouring nodes. The routing tables before and after this update is sent is shown below.

In fact, the tables remain the same.

Routing table for X		
Destination	Cost	Next Hop
Y	2	Direct
Z	3	Y

Routing table for Y		
Destination	Cost	Next Hop
X	2	Direct
Z	1	Direct

Routing table for Z		
Destination	Cost	Next Hop
X	3	Y
Y	1	Direct

Routing table for X		
Destination	Cost	Next Hop
Y	2	Direct
Z	3	Y

Routing table for Y		
Destination	Cost	Next Hop
X	2	Direct
Z	1	Direct

Routing table for Z		
Destination	Cost	Next Hop
X	3	Y
Y	1	Direct

The aim of distance vector routing algorithm is to find the least path route, the update is not causing any changes because the least cost routes have been found.

Changes in the table may occur if:

- There is a change of cost of an attached link
- There is receipt of an update message from a neighbour

At each node, x:

```
1  Initialization:
2     for all destinations y in N:
3        D_x(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        D_w(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector D_x = [D_x(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       D_x(y) = min_v{c(x,v) + D_v(y)}
15
16    if D_x(y) changed for any destination y
17       send distance vector D_x = [D_x(y): y in N] to all neighbors
18
19 forever
```

Link Layer

Saturday, 13 April 2019 12:10 AM

We will refer to:

- Any device that runs a link-layer protocol as a ***node***. This can include hosts, routers, switches, and WiFi access points.
- Communication channels that connect adjacent nodes along the communication path as ***links***. This includes wired links, wireless links, LANs

Over a given link, a transmitting node encapsulates the datagram in a ***link-layer frame*** and transmits the frame into the link.

The link layer has the responsibility of transferring datagrams from one node to a *physically adjacent* node over a link. A datagram is transferred by different link protocols over different links. e.g Ethernet on an initial link, frame relay on intermediate links and 802.11 on the last link. Each link protocol provides different services, so they may or may not provide reliable data transfer over a link.

Link Layer Services

Possible services that can be offered by a link-layer protocol includes:

- ***Framing*** - encapsulating a datagram into a frame before transmission over a link. Headers and trailers? are added.
- ***Link access*** - a medium access control (MAC) protocol specifies the rules by which a frame is transmitted on a link. MAC addresses are used in frame headers to identify the source and destination (and is different from IP addresses).
- ***Reliable delivery*** - guarantees that each network-layer datagram is moved across a link without error. Often used for links that are prone to high error rates such as a wireless link. It has the goal of correcting an error locally - on the link where the errors occur - rather than forcing an end-to-end retransmission of the data by a transport or application layer protocol
- ***Error detection*** - detect errors caused by signal attenuation and electromagnetic noise. The receiver detects the presence of errors, signals the sender for retransmission or drops the frame.
- ***Error correction*** - the receiver identifies *AND corrects* the bit error(s) without resorting to retransmission.
- ***Flow control*** - pacing between adjacent sending and receiving nodes
- ***Half-duplex*** and ***full-duplex*** - half duplex -> both nodes can transmit but one at a time, full-duplex -> both nodes can transmit at the same time

Where is the Link Layer Implemented?

The link layer is implemented in each and every host. It is implemented in a ***network adapter*** (AKA a network interface card - NIC) or a chip. The network adapter is the link-layer controller, usually a single, special purpose chip that implements many of the link-layer services. Thus, much of a link layer controller's functionality is implemented in hardware. Nowadays, network adaptors are integrated onto the host's motherboard (host's system bus)

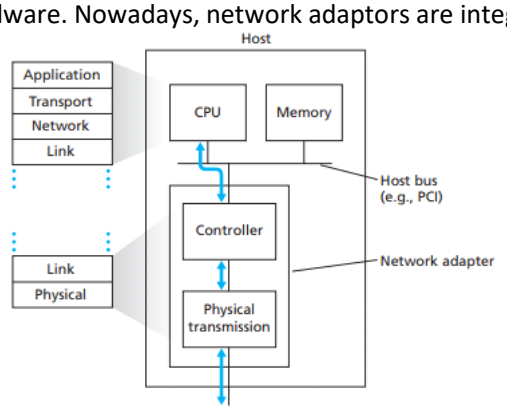
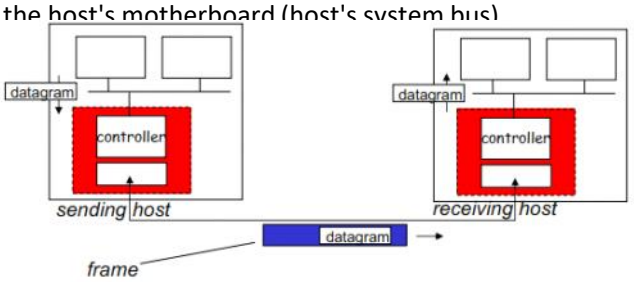


Figure 5.2 ♦ Network adapter: its relationship to other host components and to protocol stack functionality



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

Switched Local Area Networks

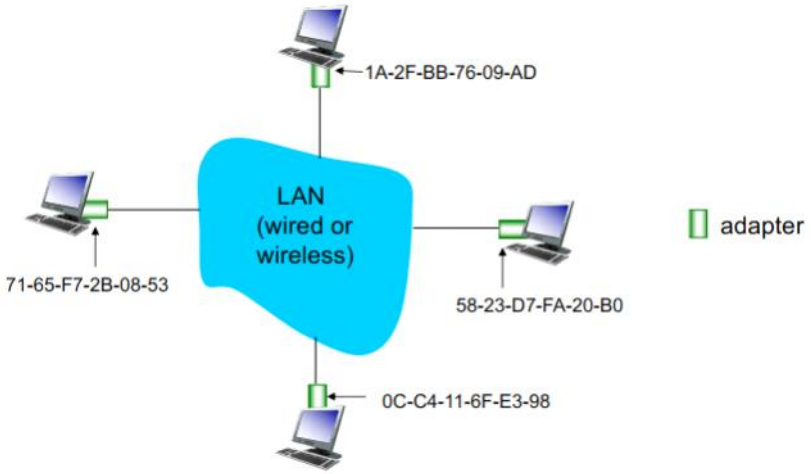
MAC Addresses

A 32-bit IP address is a network-layer address interface. It is used for layer 3 (network layer) forwarding.
The ***MAC*** (or ***LAN*** or ***physical*** or ***Ethernet***) ***address*** is a 48-bit address used *locally* to get frames from one interface to another physically connected interface (on the same network in an IP addressing sense). It is (for most LANs) burned in the NIC ROM, and is also sometimes software settable. MAX addresses are also represented using hexadecimal (base 16)

notation. An example of a MAC address is 1A-2F-BB-76-09-AD.

MAC addresses were designed to be permanent, but it is now possible to change an adapter's MAC address via software.

Each adapter on a LAN has a unique ***LAN address***, so no two adapters have the same address.



MAC address allocation is administered by IEEE. A manufacture buys a portion of a MAC address space (to ensure uniqueness). IEEE allocates the chunk of 2^{48} addresses by fixing the first 24 bits of a MAC address and letting the company create unique combinations of the last bits for each adapter.
MAC addresses are not hierarchical. They are a *flat* address, meaning they do not change no matter where the adapter goes. This makes them portable; you can move a LAN card from one LAN to another. This differs from IP's hierarchical address, where the address depends on the IP subnet, which to node is attached to.

MAC Addresses vs. IP Addresses

MAC	IP
• Hard-coded in read-only memory when adapter is built • Like a social security number • Flat name space of 48 bits (in hexadecimal) • Portable, and can stay the same as the host moves • Used to get packets between interfaces on the same network	• Configured, or learned dynamically • Like a postal mailing address • Hierarchical name space of 32 bits • Not portable, depends on where the host is attached • Used to get a packet to destination IP subnet

Layer	Examples	Structure	Configuration	Resolution Service
App. Layer	www.cse.unsw.edu.au	organizational hierarchy	~ manual	DNS
Network Layer	129.94.242.51	topological hierarchy	DHCP	ARP
Link layer	45-CC-4E-12-F0-97	vendor (flat)	hard-coded	

Address Resolution Protocol (ARP)

ARP translates between network-layer addresses (IP addresses) and link-layer addresses (MAC addresses)
ARP resolves IP addresses only for hosts and router interfaces on the same network. Each host and router has an ***ARP table*** in its memory, which contains mappings of IP addresses to MAC addresses. It also contains a time-to-live (TTL) value, which indicates when each mapping will be deleted from the table. Note that the table does not necessarily contain an entry for every host and router on the subnet; some may have never been entered into the table, and other may have expired. A typical expiration time for an entry is 20 minutes from when an entry is placed in an ARP table.

IP Address	MAC Address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Sending a Datagram in the Subnet
If host A wants to send a datagram to host B, which is on the same subnet. The sending host needs to obtain the MAC address of the destination given the IP address. If the sender's ARP table already has the entry this is easy. If it doesn't, the sender constructs a special packet called and ***ARP packet***, which has several fields; sending and receiving IP address, the sender's MAC address and the destination MAC address set as FF-FF-FF-FF-FF. The destination address is a ***broadcast address***, which transmits the frame to every adapter in the subnet. Each adapter passes the ARP packet within the frame up to its ARP module and checks if its IP address matches the destination IP address. If it matches it sends an ARP response packet to the querying host with the desired mapping. The querying host then updates its ARP table and sends its datagram, encapsulated in a link-layer frame.

Note that the query ARP message is sent within a broadcast frame, while the response ARP message is sent within a standard frame. ARP is ***plug-and-play***. This means that nodes create their ARP table automatically, it does not need to be configured by a system administrator.

Sending a Datagram outside the Subnet
Host A wants to send a datagram to host B. We will assume:

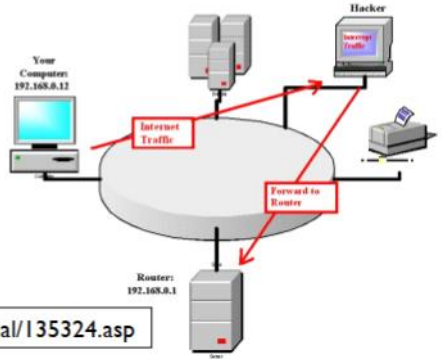
- A knows B's IP address
- A knows the IP address of its first hop router, R
- A know the MAC address of R

A creates an IP datagram with IP source address A, and destination address B. It then creates a link-layer frame with R's MAC address as the destination, but with the same IP addresses. The frame is sent from A to router R and received at R. Then the datagram is removed, passed up to the network layer. The router then determines the correct interface on which the datagram is to be forwarded. This is done by consulting the forwarding table in the router. R then creates the link-layer frame with B's MAC address as the destination, and the same IP addresses.

Security Issues: ARP cache poisoning

- Denial of Service - Hacker replies back to an ARP query for a router NIC with a fake MAC address
- Man-in-the-middle attack - Hacker can insert his/her machine along the path between victim machine and gateway router
- Such attacks are generally hard to launch as hacker needs physical access to the network

Solutions -
• Use Switched Ethernet with port security enabled (i.e. one host MAC address per switch port)
• Adopt static ARP configuration for small size networks
• Use ARP monitoring tools such as ARPWatch



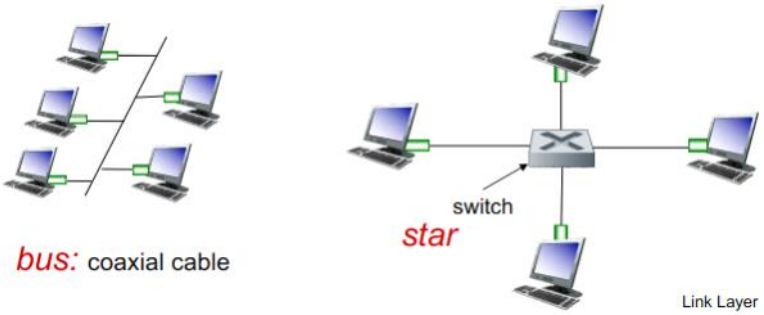
http://www.watchguard.com/infocenter/editorial/135324.asp

Ethernet

Ethernet was the first widely developed high-speed LAN. Its competitors, token ring, FDDI and ATM, were more complex and expensive discouraging network administrators to switch. Ethernet was also able to keep up with the higher data rate of new technology reducing the desire to switch.

The original Ethernet LAN used a ***bus topology***; a coaxial bus to interconnect the nodes. It was popular through the mid 90s. Its structure made all nodes in the same collision domain and requires CSMA/CD multiple access protocol for media access control.

Nowadays, Ethernet LANs use a ***switch-based star topology***. The hosts (and routers) are directly connected to a switch with twisted-pair copper wire. Each node runs a (separate) Ethernet protocol, so nodes do not collide with each other. Since there is no link sharing, there is no need for CSMA/CD.



Ethernet Frame Structure

The sending adapter encapsulates the IP data gram within an Ethernet frame and passes the frame to the physical later. The receiving adapter receives the frame from the physical layer, extracts the IP datagram and passes the IP datagram to the network layer.

Figure 5.20 ♦ Ethernet frame structure

The six fields of the Ethernet frame:

- ***Preamble*** (8 bytes) - each of the first 7 bytes of the preamble has the value of 10101010; the last byte is 10101011. this *wakes up* the receiving adapter and synchronises the clock to that of the sender.
- ***Destination address and source addresses*** (6 bytes each) - the MAC addresses of the source and destination. If an adapter receives a frame with the matching destination address, or with the broadcast address, it passes the data in the frame to the network layer
- ***Type field*** (2 bytes) - indicates the higher layer protocol (mostly IP but other are possible; Novell IPX, AppleTalk)
- ***Data field*** (46-1500 bytes)- carries the IP datagram. The MTU size of Ethernet is 1500 bytes.
- ***Cyclic redundancy check (CRC)*** (4 bytes) - allows the receiving adapter to detect bit errors in the frame

Ethernet technologies:

- provide ***connectionless*** service to the network layer; so there is no handshaking between the sending and receiving adapters.
- Provide ***unreliable*** service to the network layer. It does not acknowledge when a frame passes the CRC check nor sends negative acknowledgement when a frame fails the CRC check. This lack of reliable transport makes Ethernet cheap. Data in dropped frames are recovered only if the initial sender uses higher layer reliable data transport (e.g. TCP), otherwise the data is lost.

Ethernet comes in *many* different flavours; 10BASE-T, 10BASE-2, 100BASE-T, 1000BASE-LX, 10GBASE-T and 40GBASE-T. Many have been standardised over the years by the IEEE 802.3 CSMA/CD (Ethernet) working group. The first part of the acronym refers to the speed of the standard: 10 -> 10 Megabit, 100 -> 100 Megabit, 1000 -> 1000 Megabit, 10G -> 10 Gigabit. "BASE" refers to baseband Ethernet, meaning the physical media only carries Ethernet traffic. The final part refers to the physical media; coaxial, fiber, twisted-pair copper wires.

Ethernet Switches

A switch is a link-layer device, whose role is to receive incoming link-layer frames and forward them onto outgoing links. It examines an incoming frame's MAC address, and selectively forwards the from to one-or-more outgoing links when the frame is to be forwarded on segment. The switch is ***transparent*** to hosts and routers in the network, meaning that they are unaware of its presence. Switches are also full-duplex, meaning any switch interface can send and receive at the same time.

The rate at which frames arrive to an one of the switch's output interfaces may temporarily exceed the link capacity of that interface. To accommodate this problem, switch output interfaces have buffers.

Filtering is the switch function that determines whether a frame should be forwarded to some interface or should just be dropped. **Forwarding** is the switch function that determines the interfaces to which a frame should be directed, and then moves the frame to those interfaces. Switch filtering and forwarding are done with a **switch table**. The switch table contains entries for some, but not necessarily all, of the hosts and routers on a LAN. An entry in the switch table contains (1) a MAC address, (2) the switch interface that leads toward that MAC address, and (3) the time at which the entry was placed in the table.

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Switches are ***plug-and-play/self-learning*** and do not need to be configured. Its table is built automatically, dynamically, and autonomously. This capability is accomplished as follows:

1. The switch table is initially empty
2. For each incoming frame received on an interface, the switch stores in its table the source MAC address, the interface it came from, the current time. If every host in the LAN eventually sends a fame, then every host will eventually get recorded
3. Addresses are deleted from the table if no frames are received with that address as the source address after some period of time.

So, when a frame is received at a switch:

1. Record the frames information in the table
2. Index the destination MAC address to find the outgoing interface
    a. If the outgoing interface is the same as the incoming interface, drop the frame
    b. If the outgoing interface is different, send it to that specific interface
    c. If the destination's interface is unknown, forward the frame to all interfaces except the arriving interface.

Switches vs. Routers

Switches	Routers
• Link-layer devices, hence they examine link-layer headers • Compute forwarding tables using flooding, learning MAC addresses	• Network-layer devices, hence they examine network-layer header • Compute forwarding tables using routing algorithms, IP addresses

Security Issues

- In a switched LAN once the switch table entries are established frames are not broadcast
    - Sniffing frames is harder than pure broadcast LANs
    - Note: attacker can still sniff broadcast frames and frames for which there are no entries (as they are broadcast)
- Switch Poisoning: Attacker fills up switch table with bogus entries by sending large # of frames with bogus source MAC addresses
- Since switch table is full, genuine packets frequently need to be broadcast as previous entries have been wiped out

Quiz: Ethernet Review

1. C will not process the IP datagrams sent by A
2. C will not pass the IP datagram to the network layer, because when it processes the header of the frame it will know that the frame is not destined for it
3. If the frames were addressed to C, then the IP datagram will be passed to C's network layer.

WLAN

Wednesday, 24 April 2019 1:24 PM

There are two important (but different) challenges provided by wireless networks:

1. ***Wireless***: communication over wireless links
2. ***Mobility***: handling the mobile user who changes point of attachment to network

Elements in a wireless network:

- ***Wireless hosts*** - the end-system devices that run applications. They may be stationary or mobile; wireless does not always mean mobility.
- ***Base station*** - responsible for sending and receiving data (packets) to and from a wireless host that is associated with that base station. It is typically connected to a wired network and is often responsible for coordinating the transmission of multiple wireless host with which it is associated (i.e. a host within the communication distance of the base station, or a host that uses the base station to relay data between it and the larger network). Examples include cell towers in cellular networks and access points in 802.11 wireless LANs
- ***Wireless links*** - typically used to connect mobile hosts to a base station and is also used as a backbone link. A multiple access protocol coordinates link access. Different wireless link technologies have different transmission rates and can transmit over different distances.

Hosts associated with a base station are often referred to as operating in ***infrastructure mode***, since all traditional network services are provided by the network to which a host is connected via the base station. In ***ad hoc networks***, wireless hosts have no such infrastructure with which to connect. In the absence of such infrastructure, the hosts themselves must provide for services such as routing, address assignment, DNS-like name translation and more.

When a mobile host move beyond the range of one base station and into the range of another, it will change its point of attachment into the larger network. This processes is known as a ***handoff***.

WiFi: 802.11 Wireless LANs

802.11 Architecture

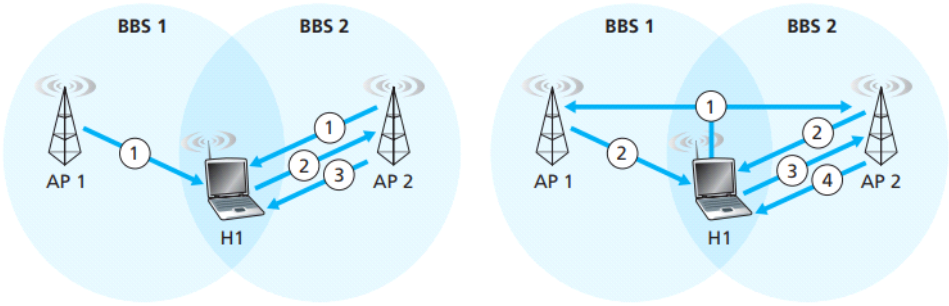
The fundamental building block of the 802.11 architecture is the ***basic service set (BSS)***. A BSS contains one or more wireless stations and a central ***base station*** known as an ***access point (AP)***. In a typical home network, there is one AP and one router (typically integrated together as one unit) that connects the BSS to the Internet.

Channels and Association

When a network administrator installs an AP, the administrator assigns a one or two-word **Service Set Identifier (SSID)** to the access point. The administrator must also assign a channel number to the AP. 802.11 operates in the frequency range of 2.4 GHz to 2.485 GHz. Within this 85 MHz band, 802.11 defines 11 partially overlapping channels. Any two channels are non-overlapping if and only if they are separated by four or more channels.

To gain Internet access, your wireless device needs to join exactly one of the subnets of an AP; it needs to ***associate*** with exactly one of the APs. Associating means the wireless device creates a virtual wire between itself and the AP.
APs are required to periodically send ***beacon frames*** (containing the AP's SSID and MAC address). Your devices finds APs by scanning channels for beacon frames. Having learned about available APs, you select one to associate with.

The process of scanning channels and listening for beacon frames is known as ***passive scanning***. A wireless device can also perform ***active scanning***, by broadcasting a probe frame that will be received by all APs within the wireless device's range. APs respond to the probe request frame with a response frame, and the wireless device chooses which AP to associate with among the responding APs.



a. Passive scanning
1. Beacon frames sent from APs
2. Association Request frame sent: H1 to selected AP
3. Association Response frame sent: Selected AP to H1

a. Active scanning
1. Probe Request frame broadcast from H1
2. Probes Response frame sent from APs
3. Association Request frame sent: H1 to selected AP
4. Association Response frame sent: Selected AP to H1

Figure 6.9 ♦ Active and passive scanning for access points

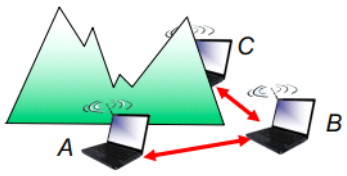
802.11 MAC Protocol

802.11 uses collision avoidance techniques instead of collision detection techniques. Hence, it uses CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance), meaning that each station senses the channel before transmitting, and refrains from transmitting when the channel sensed is busy.

The 802.11 MAC protocol does not implement collision detection for two reasons:

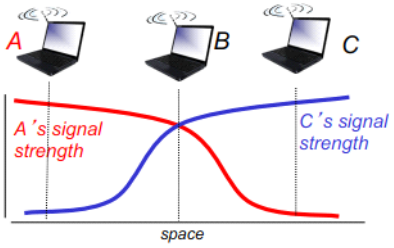
1. Collision detection requires a host to send and receive at the same time. The strength of the signal received it typically very weak and hard to detect.
2. The adapter would not be able to detect all collisions due to the hidden terminal problem and fading.

Multiple wireless senders and receivers create additional problems (beyond multiple access):



Hidden terminal problem

- *B, A hear each other*
- *B, C hear each other*
- *A, C can not hear each other means A, C unaware of their interference at B*
- *Carrier sense will be ineffective*



Signal attenuation:

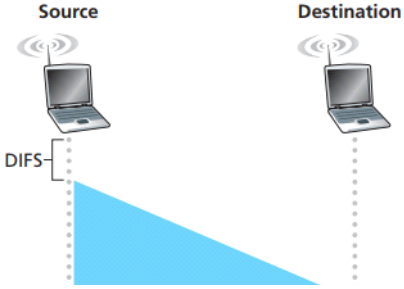
- *B, A hear each other*
- *B, C hear each other*
- *A, C can not hear each other interfering at B*

Some key points about multiple access is that

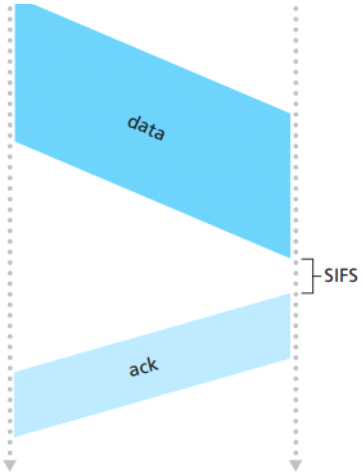
- There is no concept of a global collision - different receivers hear different signals and different senders reach different receivers.
- Collisions are at the receiver not the sender - we only care if the receiver can hear the sender clearly, it does not matter if the sender can hear someone else. As long as the signal does not interfere with the receiver
- The goal of the protocol is to detect if the receiver can hear the receiver and tell the sender who might interfere with the receiver to not send messages

Suppose that a station (wireless station or an AP) has a frame to transmit.

1. If initially the station senses the channel idle, it transmits its frame after a short period of time known as the **Distributed Inter-frame Space (DIFS)**
2. Otherwise, the station chooses a random back-off value using binary exponential back-off and counts down this value when the channel is sensed idle. While the channel is sensed busy, the counter value remains frozen



2. Otherwise, the station chooses a random back-off value using binary exponential back-off and counts down this value when the channel is sensed idle. While the channel is sensed busy, the counter value remains frozen.
3. When the counter reaches zero (note that this can only occur while the channel is sensed idle), the station transmits the entire frame and then waits for an acknowledgment.
4. If an acknowledgment is received, the transmitting station knows that its frame has been correctly received at the destination station. If the station has another frame to send, it begins the CSMA/CA protocol at step 2. If the acknowledgment isn't received, the transmitting station re-enters the back-off phase in step 2, with the random value chosen from a larger interval.

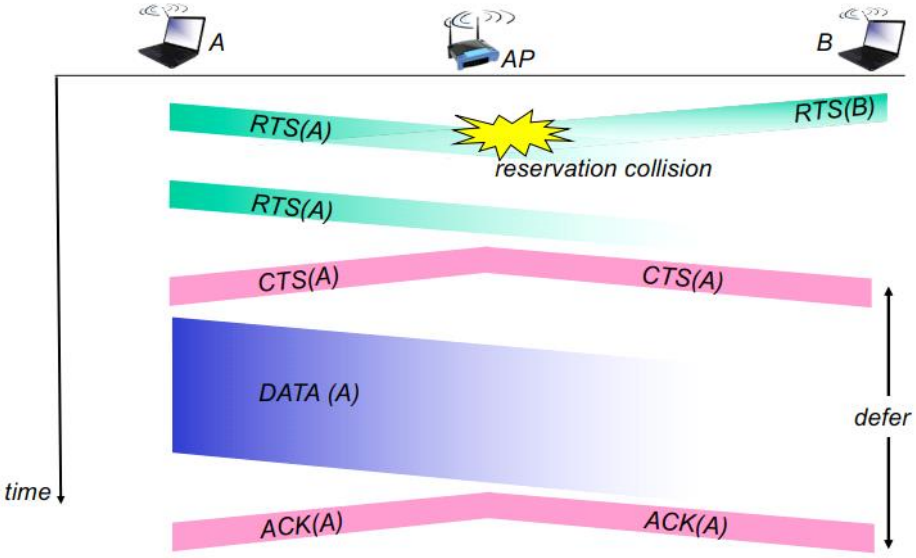


At the receiver, if the frame was successfully receive it returns an ACK after SIFS (Short Inter-frame Spacing)

Figure 6.10 ♦ 802.11 uses link-layer acknowledgments

Dealing with Hidden Terminals: RTS and CTS

A sender transmits a short ***Request to Send (RTS)*** control frame to the AP, indicating the total time to transmit the DATA frame and the ACK frame. When the AP receives the RTS frame; it responds by broadcasting a ***Clear to Send (CTS)*** frame. This CTS frame gives the sender explicit permission to send and also instructs the other stations not to send for the reserved duration. Note that RTSs may still collide with each other, but they are short hence collision is insignificant.



802.11: Mobility in the Same IP Subnet

We have two interconnected BSSs with a host, H1, moving from BSS1 to BSS2. The interconnection device that connects the two BSSs is *not* a router, all of the stations in the two BSSs, including the APs, belong to the same IP subnet. Thus, when H1 moves from BSS1 to BSS2, it may keep its IP address and all of its ongoing TCP connections. If the interconnection device were a router, then H1 would have to obtain a new IP address in the subnet in which it was moving. This address change would disrupt (and eventually terminate) any on-going TCP connections at H1.

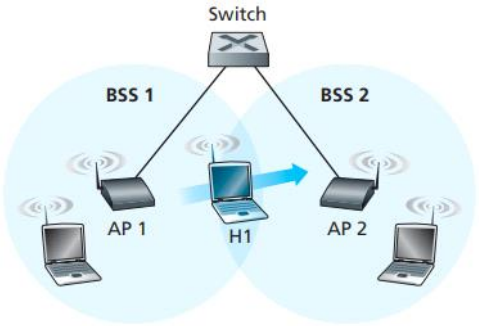


Figure 6.15 ♦ Mobility in the same subnet

As H1 wanders away from AP1, H1 detects a weakening signal from AP1 and starts to scan for a stronger signal. H1 receives beacon frames from AP2. H1 then disassociates with AP1 and associates with AP2, while keeping its IP address and maintaining its ongoing TCP sessions.

Since switches are self-learning, if H1 associates with AP2, AP2 can send a broadcast Ethernet frame with H1's source address to the switch. When the switch receives the frame, it updates its forwarding table, allowing H1 to be reached via AP2.

Advanced Features in 802.11

Some 802.11 implementations have rate adaptability capability that adaptively selects the underlying physical-layer modulation technique to use based on current or recent channel characteristics. The rate adaption mechanism shares the same *probing philosophy* as TCP's congestion-control mechanism - when conditions are good, the transmission rate is increased until something *bad* happens; when something *bad* happens, the transmission rate is reduced.

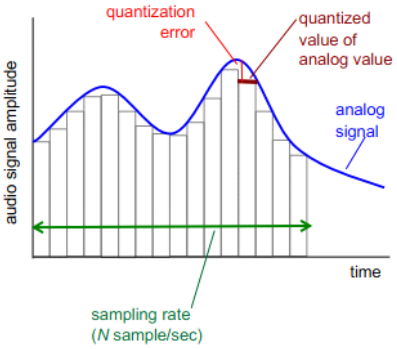
Multimedia

Saturday, 13 April 2019 4:27 PM

Multimedia Network Applications

Properties Audio

To convert analog audio to a digital signal, the analog audio signal is sampled at some fixed rate (8000 samples/sec for telephone, 44100 samples/sec for CD music). Each of the samples is then rounded to one of a finite number of values. This operation is called ***quantization***. The number of such values (called ***quantization values***) is typically a power of two. e.g. $2^8 = 256$ possible quantized values. Each quantized value is represented by a fixed number of bits. e.g. 8 bits for 256 values.

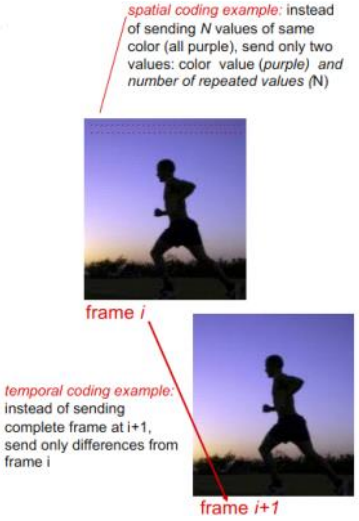


For playback, the digital signal is converted back (decoded) to an analog signal. By increasing the sampling rate and the number of quantization values, the decoded signal can better approximate the original analog signal.

Properties of Video

A video is a sequence of images displayed at a constant rate (say 24fps). An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and colour. There are two types of redundancy in video, which can be exploited by ***video compression***.

1. ***Spatial redundancy*** within the image
2. ***Temporal redundancy*** for the repetition from one image to a subsequent image.



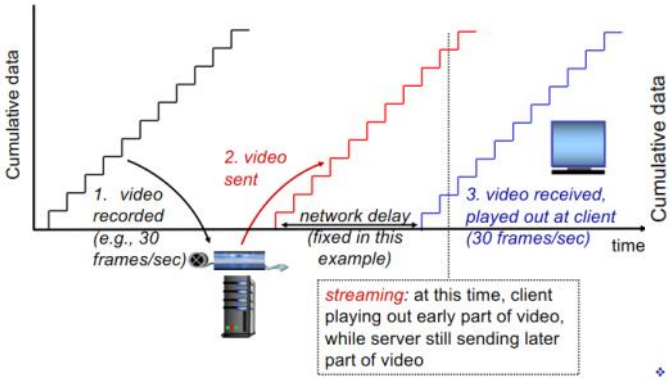
Constant bit rate (CBR) is the fixed rate at which the video is encoded
Variable bit rate (VBR) is when the video encoding rate changes as the amount of spatial and temporal coding changes

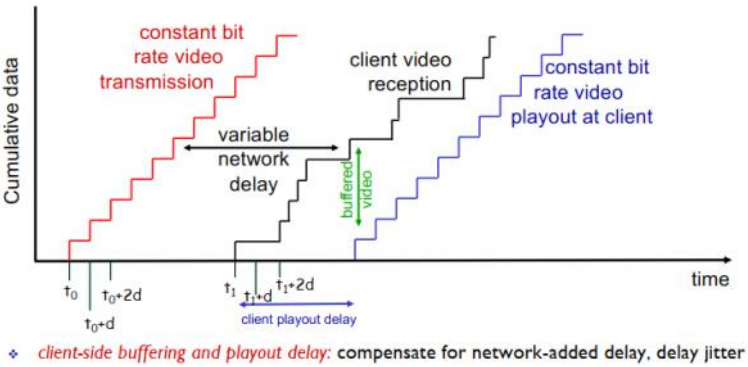
Types of Multimedia Applications

We classify multimedia applications into three broad categories:

1. Streaming stored audio/video - avoid having to download the entire video/audio file before playout begins
2. Conversational voice/voice-over-IP - the interactive nature of human-to-human conversation limits delay tolerance
3. Streaming live audio/video - user receives *live* radio or television transmission such as live sporting events

Streaming Stored Video





- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

There is extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client. When the video starts to arrive at the client, the client does not immediately play out, instead it builds up a reserve of the video in the application buffer. Once the client has built up a reserve of several seconds of buffered-but-not-yet-played video, the client can then begin video playout.

Other challenges include:

- Client interactivity: pausing, fast-forwarding, rewinding, jumping through the video
- Video packets being lost and retransmitted