Chapter 1: Computer Networks and the Internet

1.1 What is the Internet?

1.1.1 A nuts-and-bolts description

The Internet is a computer networks that interconnects hundreds of millions of computing devices through the world. Today not only computers and workstation are being connected to the network, therefore the term *computer network* may sound a bit dated.

All the devices connected to the Internet are called **hosts** or **end systems**. End systems are connected together by a network of **communication links** and **packets switches**.

Different links can transmit data at different rates, with the **transmission rate** of a link measured in bits/second.

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. The resulting packages of information, called **packets**, are then sent through the network to the destination and system where they a reassembled into the original data.

A packet switch takes a packet arriving on one of its incoming communication links and forwards that packet on one of its outgoing communication links. The two most prominent types of packets switches are **routers** and **link switches**. 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 **route** or **path**.

End systems access the Internet through **Internet Service Providers (ISPs)**, including residential ISPs (cable or phone company), corporate, university ISPs ... Each ISP in itself is a network of packet switches and communication links. *Lower tier* (which interconnect end-systems) ISPs are interconnected through national and international *upper tier* ISP. An upper-tier ISP consists of high speed routers interconnected with high-speed fiber-optic links. Each ISP network is managed independently.

End systems, packet switches and other pieces of the Internet run **protocols** that control the sending and receiving of information within the Internet.

1.1.2 A Services Description

The Internet can be described as *an infrastructure that provides services to applications.* These applications (Web, social networks, VoIP...) are said to be **distributed** since they involve multiple end systems that exchange data with each other. **Internet applications run on end systems, not in the packet switches or routers**, packet switches facilitate the exchange of data, but they are not concerned with the application that is the source or sink of data.

End systems attached to the Internet provide and **Application Programming Interface (API)** that specifies how a program running on one end system asks the Internet infrastructure to deliver data to a specific destination program running on another end system.

1.1.3 What Is a Protocol?

All the activity in the Internet that involves two or more communicating remote entities is governed by a protocol.

A protocol defines the format and the order of messages exchanged between two or more communicating entities, as weel as the actions taken on the trasmission and/or receipt of a message or other event

1.2 The Network Edge

Computers and other devices connected to the Internet are often referred to as *end systems* as they sit at the edge of the Internet. They are also called *hosts* as they host, run, applications programs such as a Web Browser or an email client.

Hosts are sometimes further divided into two categories: *clients* and *servers.* The former being desktop, mobile pcs, smartphones, the latter being powerful machines that store and distribute Web pages, streams... Nowadays most of the servers reside in large *data centers*

1.2.1 Access Networks

They are the networks that physically connect end systems to the first router on a path from the end system to any other distant end system. Examples: mobile network, national or global ISP, local or regional ISP, home networks enterprise networks.

Home Access: DSL, Cable, FITH, Dial-Up and Satellite Today, the two most prevalent types of broadband residential access are **digital subscriber line (DSL)** and **cable**.

A residence typically obtains DSL access from the telephone company (telco) that provides its wired local phone access. The customer's telco is therefore its ISP. DSL modem use the existing telephone lines to exchange data with DSLAMs (digital subscriber line access multiplexer) located in the telco local central office. The DSL modem takes digital data and translates it to high-frequency tones for transmission over telephone wires, these analog signals from many houses are translated back into digital format at the DSLAM. The use of different frequencies allows the phone line to carry a high-speed downstream channel, a medium-speed upstream channel and an ordinary two-way telephone channel. Hundreds or even thousands of households connect to a single DSLAM.

DSL: 24 Mbps downstream and 2.5 Mbps upstream (MAX VALUES). Because of the difference between these two values, the access is said to be **asymmetric**.

Cable Internet access makes use of the cable television company's existing cable television infrastructure. Cable modems connect to CMTS (Cablem Modem Termination System) which does the same job the DSLAM does for phone lines. The access is typically asymmetric. CABLE: 42.8 Mbps downstream and 30.7 Mbps upstream (MAX VALUES). Cable Internet access is a shared broadcast medium: each packet travels downstream on every link to every home and viceversa. For this, if several users are simultaneously using the downstream channel, the actual rate will be significantly lower.

Another up-and-coming technology that promises very high speeds is **fiber to the home (FTTH)**. The concept is simple: provide an optical fiber path from the Central Office (CO)

Access in the Enterprise and the Home: Ethernet and WiFi 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. Ethernet is by far the most prevalent access technology is corporate, university and home networks. Ethernet uses twisted-pair copper wire to connect to an Ethernet switch which is connected into the larger Internet. The Internet is increasingly accessed wirelessly: wireless users transmit/receive packets to/from an access point connected into the enterprise's network which in turn is connected to the wired Internet.

Wide-Area Wireless Access: 3G and LTE Smartphones and Tablets employ the same wireless infrastructure used for cellular telephony to send/receive packets through a base station operated by the cellular network provider. Third generation (3G) wireless and fourth generation (4G) of wide-area network are being deployed. LTE ("Long-Term Evolution") has its root in 3G and can potentially achieve rates in excess of 10 Mbps.

1.2.2 Physical Media

le livre en parle en détail mais nous n'en avons pas parlé en cours

A bit, when traveling from source to destination, passes through a series of transmitter-receiver pairs, for each pair, the bit is sent by propagating electromagnetic waves or optical pulses across a **physical medium**. This can take many shapes and forms and doesn't have to be of the same type for each transmitter-receiver pair along the path. Physical media fall into two categories:

- **guided media**: the waves are guided along a solid medium (fiber-optic cable, twisted-pair copper wire, coaxial cable)
- **unguided media**: the waves propagate in the atmosphere and in outer space (wireless LAN, digital satellite channel)

1.3 The Network Core

1.3.1 Packet Switching

In a network application, end systems exchange **messages** with each other. 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**. 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**). Packets are transmitted over each communication link at a rate equal to the *full* transmission rate of the link. So, if a source end system or a packet switch is send a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds.

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. The link must **buffer** ("store") the packet's bits and only after the router has received all of the packet's bits can it begin to transmit ("forward") the packet onto the outbound link.

Queuing Delays and Packet Loss Each packet switch has multiple links attached to it. For each attached link, the packet switch has an **output buffer** (or **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, packets suffer output buffer **queuing delays** which are variable and depend on the level of congestion in the network. Since the amount of buffer space is finite, an arriving packet may find the buffer completely full. In this case, **packet loss** will occur, either the arriving packet or one of the already queued packets will be dropped.

Forwarding tables and routing protocols In the Internet, every end system has an address called an IP address. When a source end system wants to send a packet to a destination end system, the source includes the destination's IP address in the packet's header. Each router has a **forwarding table** that maps destination addresses (or portions of the destination addresses) to that router's outbound links. When a packet arrives at the router, the router examines the address and searches its forwarding table, using this destination address, to find the appropriate outbound link. A number of special **routing protocols** are used to automatically set the forwarding tables.

1.3.2 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 sessions. When two hosts want to communicate, the network establishes a **dedicated end-to-end connection** between them.

Multiplexing in Circuit-Switched Networks 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. The width of the band is called the **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.

Packet Switching Versus Circuit Switching Packet switching is more flexible, uses resources efficiently and is simpler to implement (even if it requires congestion control). Circuit switching offers performance guarantees but uses resources inefficiently

1.3.3 A Network of Networks

To create the Internet, ISPs must be interconnected, thus creating a *network of networks*. Much of the evolution of the structure of the Internet is driven by economics and national policy, rather than by performance consideration.

Today's Internet is complex, consisting of a dozen or so tier-1 ISPs and hundreds of thousands of lower-tier ISPs. The ISPs are diverse in their coverage, with some spanning multiple continents and oceans, and others limited to narrow geographic regions. The lower-tier ISPs connect to the higher-tier ISPs and the higher-tier ISPs interconnect with one another. Users and content providers are customers of lower-tier ISPs and lower-tier ISPs are customers of higher-tier ISPs. Recently, major content providers (Google) have also created their own networks and connect directly into lower-tier ISPs where possible.

1.4 Delay, Loss and Throughput in Packet-Switched Networks

Computer networks necessarily constrain **throughput** (the amount of data per second that can be transferred) between end system, introduce delays between end systems and can actually lose packets.

1.4.1 Overview of Delay in Packet-Switched networks

As a packet travels from one node (host or router) to the subsequent host along his path, it suffers from several types of delays at *each* node along the path.

Types of Delay

Processing Delay The **processing delay** consists of the time required to examine the packet's header and determine where to direct the packet. It may also include other factors, such as the time needed to check for bit-level errors occurred during transmission. They typically are of the order of microseconds or less. After processing the packet is sent to the queue preceding the link to the next router

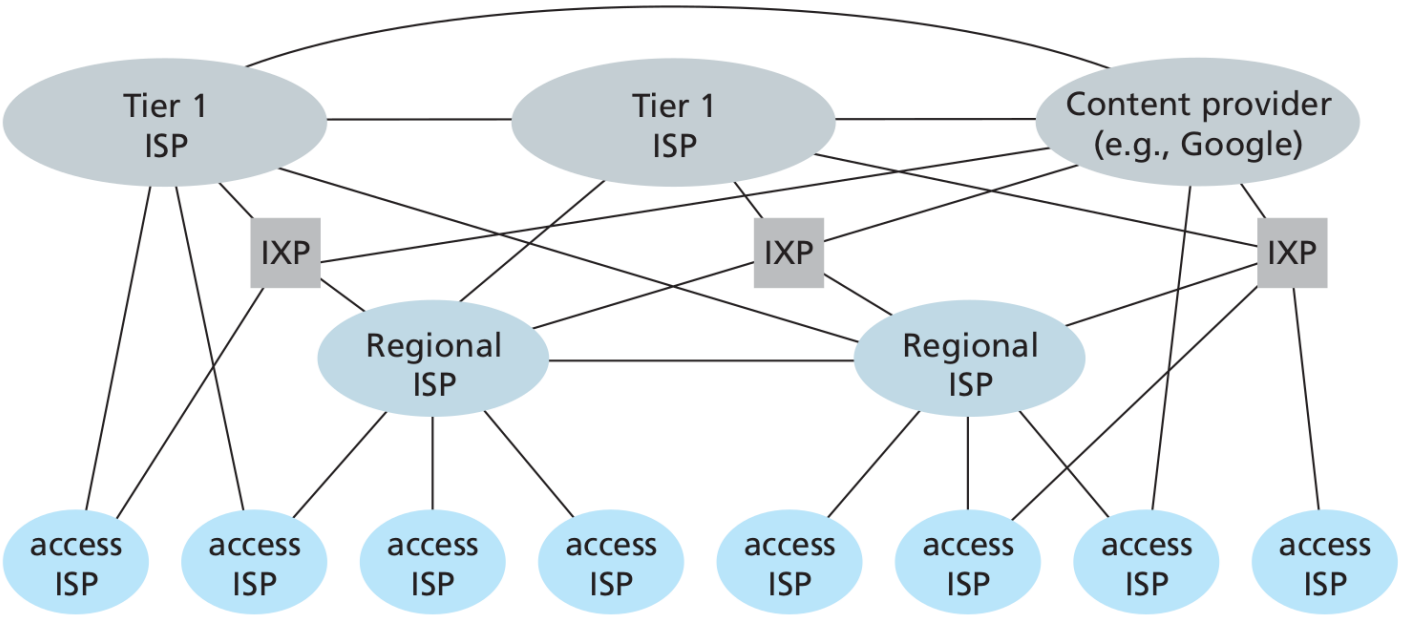


Figure 1: Alt text

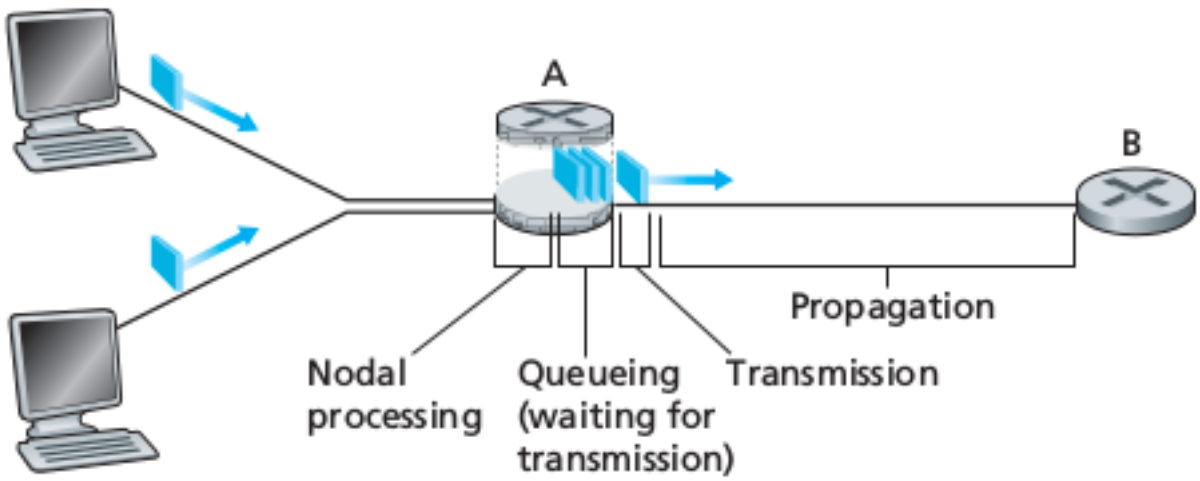


Figure 2: Alt text

Queuing Delay At the queue, the packet experiences a **queuing delay** as it waits to be transmitted onto the link. It depends on the number of earlier-arriving packets, therefore if the queue is empty, then the packet's queuing delay will be 0. Typically of the order of microseconds or milliseconds.

Transmission delays If the length of the packet is L bits, and the **transmission rate** of the link is R bits/sec, then the **transmission delay** is L/R. This is the amount of time required to push (transmit) all of the packet's bits into the link. Typically on the order of microseconds to milliseconds.

Propagation Delay The time required to propagate a bit from the beginning of the link to the next router is the **propagation delay**. The bit propagates at the propagation speed of the link, which depends on the physical medium of the link. The propagation delay is the distance between two routers divided by the propagation speed of the link.

Total nodal delay it is the summation of the previous delays

1.4.2 Queuing Delay and Packet Loss

The queuing delay depends can vary from packet to packet, therefore when characterizing queuing delay, one typically uses statistical measures, such as *average queuing delay, variance of queuing delay, and the probability that the queuing delay exceeds some specified value.*

Packet Loss A queue preceding a link has finite capacity. If a packet finds a full queue, then the router will **drop** it, the packet will be lost. The fraction of lost packets increases as the traffic intensity increases.

1.4.3 End-to-End Delay

Let's now consider the **total delay, from source to destination** (not only the nodal delay). Let's suppose there are *N-1* routers between the source host and the destination host, then the nodal delays accumulate and give an **end-to-end delay**:

$$d_{end_end} = N(d_{proc} + d_{trans} + d_{prop})$$

1.4.4 Throughput in Computer Networks

Another critical performance measure in computer networks is *end-to-end throughput.* The **instantaneous throughput** at any instant of time is the rate (in bits/sec) at which host B is receiving a file. If the file consists of F bits and the transfers takes T seconds to transfer the whole file, then the **average throughput** of the file is F/T *bits/sec.* For a simple two-link network, the throughput is the min of all the throughputs, that is the transmission rate of the **bottleneck link**. Therefore, the constraining factor for throughput in today's Internet is typically the *access network.*

1.5 Protocol Layers and Their Service Models

1.5.1 Layered Architecture

A layered architecture allows us to discuss a well-defined, specific part of a large and complex system. This simplification itself is of considerable value by providing *modularity*, making it much easier to change the implementation of the service provided by the layer: as long as the layer provides the same service to the layer above it, and uses the same services from the layer below it, the remainder of the system remains unchanged when a layer's implementation is changed.

Protocol Layering To provide structure to the design of network protocols, the network designers organize protocols in **layers**. **Each protocol belongs to one of the layers**. We are interested in the **services** that a layer offers to the layer above, **service model** of a layer. When taken together, the protocols of the various layers are called the **protocol stack**. The Internet protocol stack consists of five layers:

- Application
- Transport
- Network
- Link
- Physical

Application Layer Where network applications and their applications-layer protocols reside. The Internet's application layer includes many protocols: HTTP, SMTP, FTP, DNS. 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. This packet of information at the application layer is called **message**.

Transport Layer It transports application-layer messages between application endpoints. In the Internet there are two transport protocols: TCP and UDP. TCP provides a connection-oriented service to its application: the service includes guaranteed delivery of application-layer messages to the destination and flow control unit. 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. HTTP and SMTP use TCP

UDP provides a connectionless service to its applications: it's a no-frills service that provides no guarantees, no reliability, no flow control and no congestion control. A transport-layer packet is called **segment** Skype uses UDP (speed required)

Network Layer It is responsible for moving network-layer packets known as **datagrams** from one host to another. The Internet's network layer includes the IP Protocol. There is only one IP Protocol and all the Internet components that have a network layer must run it. The Internet's network layer also contains routing protocols that determine the routes that datagrams take between sources and destinations. The Internet has many routing protocols. Often it is simply referred to as the IP protocols, forgetting that it includes routing too.

Link Layer To move a packet from one node to the next, the network layer relies on the services of the link layer. The services provided by the link layer depend on the specific link-layer protocol that is employed over the link. Examples are Ethernet, WiFi. We will refer to the link-layer packets as **frames**

Physical Layer The job of the physical layer is to move the individual bits within the frame from one node to the next. The protocols are link dependent and further depend of the actual transmission medium of the link.

1.5.2 Encapsulation

Routers and link-layer switches are both packet switches but routers and link-layer switches do not implement all of the layers in the protocol stack: link-layer switches implement Physical and Link while router add the Network Layer too.

From the Application Layer, the message passes to the transport layer, which appends additional information to it (the **Header**) that will be used by the receiver-side transport layer. The transport layer then adds its own header and passes the datagram to the link layer which adds it own link-layer header information. Thus, we see that at each layer, a packet has two types of fields: **header fields** and a **payload field**, the payload typically being the packet from the layer above.

The process of encapsulation can be more complex: for example a large message may be divided into multiple transport-layer segments, which will be divided into multiple datagrams....

1.6 Networks Under Attack

Malware

Along with all the good files we exchange on the Internet, come malicious software, collectively known as **malware** that can also enter and infect our devices. Once a device infected, the malware can do all kinds of evil things: deleting files, install spyware... A compromised host may also be enrolled in a network of thousands of similarly compromised devices, known as **botnet** which can be used for spam or distributed denial-of-service. Much of the malware is **self-replicating**: it seeks entry into other hosts from the infected machines. Malware can spread in the from of a virus or a worm.

- **Viruses** are malware that requires some form of user interaction to infect the user's device.
- **Worms** are malware that can enter a device without any explicit user interaction.

DoS

Denial-of-Service attacks render a network, host, or other piece of infrastructure unusable by legittimate users. Most of them fall into one of the three categories:

- *Vulnerability Attack*: a few well-crafted messages are sent to a vulnerable application or operating system running on the targeted host. The service might stop or the host might crash.
- *Bandwidth flooding*: a deluge of packets is sent to the targeted host, so many packets that the target's access link becomes clogged preventing legitimate packets from reaching the server

- *Connection flooding*: a large number of half-open or fully open TCP connections are established at the targeted host, which can become so bogged down that it stops accepting legitimate connections.

In a **distributed DoS** (**DDoS**) attack the attacker controls multiple sources and has each source blast traffic at the target.

Sniffing

A passive receiver can record a copy of every packet that passes through the network. It is then called a **packet sniffer**. Because packet sniffers are *passive* (they do not inject packets into the channel), they are difficult to detect. Some of the best defenses against packet sniffing involve cryptography.

Spoofing

The ability to inject packets into the Internet with a false source address is known as **IP Spoofing** and is but one of many ways in which one user can masquerade as another user. To solve this problem we will need *end-point authentication.*

The history of the Internet shaped is structure

The Internet was originally designed to be based on the model of a *group of mutually trusting users attached to a transparent network*, a model in which there is no need for security. Many aspects of the original Internet architecture deeply reflect this notion of mutual trust, such as the ability for one to send a packet to any other user is the default rather than a requested/granted capability. However today's Internet certainly does not involve "mutually trusted users": communication among mutually trusted users is the exception rather the rule.

History of Computer Networking and the Internet

Chapter 2: Application Layer

Network applications are the *raison d'être* of a computer network. They include text email, remote access to computers, file transfers, the WorldWideWeb (mid 90s), web searching, e-commerce, Twitter/Facebook, Amazon, Netflix, Youtube, WoW...

2.1 Principles of Network Applications

At the core of network application development is writing programs that run on different **end systems** and communicate with each over the network. The programs running on end systems might be different (server-client architecture) or identical (Peer-to-Peer architecture). Importantly we write programs that run on end systems/hosts, not on network-core devices (routers/link-layer switches).

2.1.1 Network Application Architectures

From the application developer's perspective, the network architecture is fixed and provides a specific set of services to applications. The **application architecture**, on the other hand, is chosen by him. In choosing the application architecture, a developer will likely draw one of the two predominant architectural paradigms used in modern network applications:

- **Client-server architecture**: there is an always on host, called the *server* which serves requests from many other hosts, called *clients*: [Web Browser and Web Server]. Clients do not communicate directly with each other. The server has a fixed, well-known address, called an IP address that clients use to connect to him. Often, a single server host is incapable of keeping up with all the requests from clients, for this reason, a **data center**, housing a large number of hosts, is often used to create a powerful virtual server (via *proxyin*).
- **P2P architecture**: there is minimal or no reliance on dedicated servers in data centers, the application exploits direct communication between pairs of intermittently connected bots, called *peers*. They are end systems owned and controlled by users. [Bittorrent, Skype]. P2P applications provide **self-scalability** (the network load is distributed) They are also **cost-effective** since they don't require significant infrastructure and server bandwidth. P2P face challenges:
  1. ISP Friendly (asymmetric nature of residential ISPs)
  2. Security
  3. Incentives (convincing users to participate)

Some applications have hybrid architectures, such as for many instant messaging applications: a server keeps track of the IP addresses of users, but user-to-user messages are sent directly between users.

2.1.2 Processes Communicating

In the jargon of operating systems, it's not programs but **processes** that communicate. A process can be thought of as a program that is running within an end system. Processes on two different end systems communicate with each other by exchanging **messages** across the computer network: a sending process creates and sends messages into the network, a receiving process receives these messages and possibly responds by sending messages back.

Client and Server Processes A network application consists of pairs of processes that send messages to each other over a network. For each pair of communicating processes we label:

- the process that initiates the communication as the **client** [web browser]
- the process that waits to be contacted to begin the session as the **server** [web server]

This labels stand even for P2P applications in the *context of a communication session.*

The Interface Between the Process and the Computer Network A process sends messages into, and receives messages from, 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 of the socket but has little control of the transport-layer side of the socket. The only control that he has over the transport-layer is:

1. The choice of the transport protocol
2. Perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes

Addressing Processes In order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address. To identify the receiving processes, two pieces of information need to be specified:

1. The address of the host. In the Internet, the host is identified by its **IP Address**, a 32-bit (or 64) quantity that identifies the host uniquely.
2. An identifier that specifies the receiving process in the destination host: the destination **port number**. Popular applications have been assigned specific port numbers (web server -> 80)

2.1.3 Transport Services Available to Applications

What are the services that a transport-layer protocol can offer to applications invoking it?

Reliable Data Transfer For many applications, such as email, file transfer, web document transfers and financial applications, packet's drops and data loss can have devastating consequences. If a protocol provides guarantees that the data sent is delivered completely and correctly, it is said to provide **reliable data transfer**. The sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.

Throughput A transport-layer protocol could provide guaranteed available throughput at some specific rate. Applications that have throughput requirements are said to be **bandwidth-sensitive applications**.

Timing A transport-layer protocol can also provide timing guarantees. Example: guarantees that every bit the sender pumps into the socket arrives at the receiver's socket no more than 100 msec later, interesting for real-time applications such as telephony, virtual environments...

Security A transport-layer protocol can provide an application with one or more security services. It could encrypt all data transmitted by sending process and in the receiving host decrypt it.

2.1.4 Transport Services Provided by the Internet

The Internet makes two transport protocols available to applications: TCP and UDP.

TCP Services TCP includes a connection-oriented service and a reliable data transfer service:

- **Connection-oriented service**: client and server exchange transport-layer control information *before* the application-level messages begin to flow. This so-called *handshaking* procedure alerts the client and server, allowing them to prepare for an onslaught of packets. Then a **TCP connection** is said to exist between the sockets of the two processes. When the application finishes sending messages, it must tear down the connection

SECURING TCP Nether TCP nor UDP provide encryption. Therefore the Internet community has developed an enhancement for TCP called **Secure Sockets Layer (SSL)**, which not only does everything that traditional TCP does but also provides critical process-to-process security services including *encryption*, *data integrity* and *end-point authentication*. It is not a third protocol, but an enhancement of TCP, **the enhancement being implemented in the application layer** in both the client and the server side of the application (highly optimized libraries exist). SSL has its own socket API, similar to the traditional one. Sending processes passes cleartext data to the SSL socket which encrypts it.

- **Reliable data transfer service** The communicating processes can rely on TCP to deliver all data sent without error and in the proper order.

TCP also includes a **congestion-control mechanism**, a service for the general welfare of the Internet rather than for the direct benefit of the communicating processes. It throttles a sending process when the network is congested between sender and receiver.

UDP Services UDP is a no-frills, lightweight transport protocol, providing minimal services. It is connectionless, there's no handshaking. The data transfer is unreliable: there are no guarantees that the message sent will ever reach the receiving process. Furthermore messages may arrive out of order. UDP does not provide a congestion-control mechanism neither.

Services Not Provided by Internet Transport Protocols These two protocols do not provide timing or throughput guarantees, services not provided by today's Internet transport protocols. We therefore design applications to cope, to the greatest extent possible, with this lack of guarantees.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	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)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figure 2.5 ♦ Popular Internet applications, their application-layer protocols, and their underlying transport protocols

Figure 3: Alt text

2.1.5 Application-Layer Protocols

An **application-layer protocol** defines how an application's processes, running on different end systems, pass messages to each other. It defines:

- The type of the messages exchanged (request/response)

- The syntax of the various message types
- The semantics of the fields (meaning of the information in fields)
- The rules for determining whem and how a process sends messages and responds to messages

2.2 The Web and HTTP

In the early 1990s, a major new application arrived on the scene: the World Wide Web (Berners-Lee 1994), the first application that caught the general public's eye. The Web operates *on demand*: users receives what they want, when they want it. It is enormously easy for an individual to make information available over the web, hyperlinks and search engines help us navigate through the ocean of web sites...

2.2.1 Overview of HTTP

The **HyperText Transfer Protocol (HTTP)**, the Web's application-layer protocol is a the heart of the Web. It is implemented in two programs: a client program and a server program. The two programs talk to each other by exchanging HTTP messages. A **Web page** (or document) consists of objects. An **object** is simply a file (HTML file, jpeg image...) that is *addressable by a single URL*. Most Web pages consist of a **base HTML file** and several referenced objects. The HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the hostname of the server that houses the object and the object's path name. **Web Browsers** implement the client side of HTTP. **HTTP** uses TCP as its underlying transport protocol. The server sends requested files to clients without storing any state information about the client: it is a **stateless protocol**

2.2.2 Non-Persistent and Persistent Connections

In many Internet applications, 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, periodically at regular intervals or intermittently. When this is happening over TCP, the developer must take an important decision: should each request/response pair be sent over a *separate* TCP connection or should all of the requests and their corresponding responses be sent over the *same* TCP connection? In the former approach, the application is said to use **non-persistent connections** and in the latter it is said to use **persistent connections** By default HTTP uses non-persistent connections but can be configured to be use persistent connections. To estimate the amount of time that elapses when a client requests the base HTML file until the entire file is received by the client we define the **round-trip time** (**RTT**) which is the time it takes for a small packet to travel from client to server and then back to the client.

HTTP with Non-Persistent Connections For the page and each object it contains, a TCP connection must be opened (handshake request, handshake answer), we therefore observe an addition RTT, and for each object we will have a request followed by the reply This model can be expensive on the server side: a new connection needs to be established for each requested object, for each connection a TCP buffer must be allocated along some memory to store TCP variables.

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 will be sent over the same connection. In particular an entire web page (text + objects) ca be sent over a single persistent TCP connection, multiple web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection. **These requests can be make back-to-back** without waiting for replies to pending requests (**pipelining**). When the server receives back-to-back requests, it sends the objects back-to-back. If connection isn't used for a pre-decided amount of time, it will be closed.

2.2.3 HTTP Message Format

Two types of HTTP messages:

HTTP Request Message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

- Ordinary ASCII text
- First line: **request line**

- Other lines: **header lines**
- the first lines has 3 fields: method field, URL field, HTTP version field:
  - method field possible values: `GET, POST, HEAD, PUT, DELETE`

The majority of HTTP requests use the GET method, used to request an object.

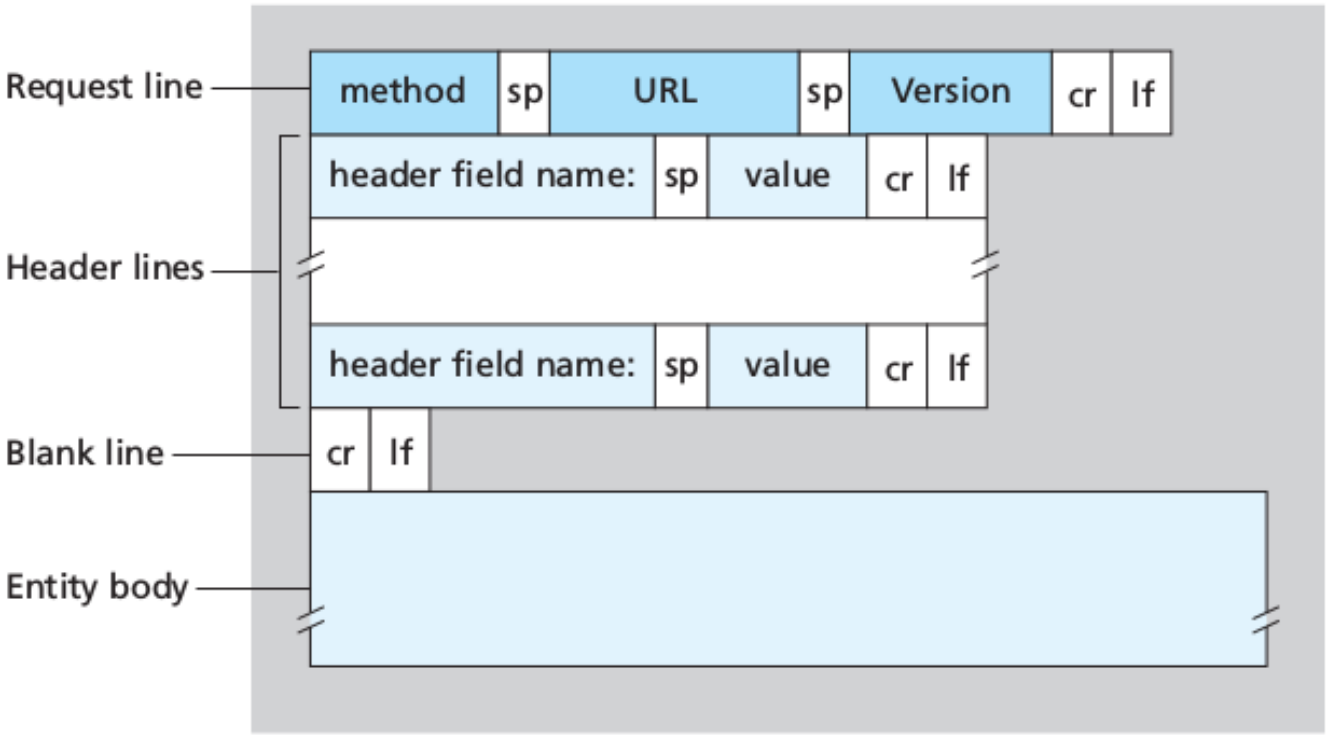


Figure 4: Alt text

The entity body (empty with `GET`) is used by the `POST` method, for example for filling out forms. The user is still requesting a Web page but the specific contents of the page depend on what the user entered into the form fields. When `POST` is used, the entity body contains what the user entered into the form fields. Requests can also be made with `GET` including the inputted data in the requested URL. The `HEAD` method is similar to `GET`, when a server receives it, it responds with an HTTP message but it leaves out the requested object. It is often used for debugging. `PUT` is often used in conjunction with web publishing tools, to allow users to upload an object to a specific path on the web servers. Finally, `DELETE` allows a user or application to delete an object on a web server.

HTTP Response Message A typical HTTP response message:

```
HTTP/1.1 200 OK
Connection: close
Date: ...
Server: ...
Last-Modified: ...
Content-Length: ...
Content-Type: text/html

(data data data data data ...)
```

- Status line: protocol version, status code, corresponding status message
- six header lines:
  - the connection will be closed after sending the message
  - date and time when the response was created (when the server retrieves the object from the file system, insert object in the message, sends the response message)
  - Type of the server / software
  - Last modified: useful for object caching
  - Content-Length: number of bytes in the object
  - Content-Type

- entity body: contains the requested object itself (data)

Some common status codes:

`200 OK`: request succeeded, information returned `301 Moved Permanently`: the object has moved, the new location is specified in the header of the response `400 Bad Request`: generic error code, request not understood `404 Not Found`: The requested document doesn't exist on the server `505 HTTP Version Not Supported`: The requested HTTP protocol version is not supported by the server

2.2.4 User-Server Interaction: Cookies

An HTTP server is *stateless* in order to simplify server design and improves performances. A website can identify users using **cookies**. Cookie technology has 4 components:

1. Cookie header in HTTP response message
2. Cookie header in HTTP request message
3. Cookie file on the user's end-system managed by the browser
4. Back-end database at the Website

User connects to website using cookies:

- Server creates a unique identification number and creates an entry in its back-end database indexed by the identification number -server responds to user's browser including in the header: `Set-cookie: identification number`
- The browser will append to the cookie file the hostname of the server and the identification number header
- Each time the browser will request a page, it will consult the cookie file, extract the identification number for the site and put a cookie header line including the identification number

The server can track the user's activity: it knows exactly what pages, in which order and at what times that identification number has visited. This is also why cookies are controversial: a website can learn a lot about a user and sell this information to a third party.

Therefore **cookies can be used to create a user session layer on top of stateless HTTP**.

2.2.5 Web Caching

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

1. The browser establishes a TCP connection to the web cache, sending an HTTP request for the object to the Web cache.
2. The web cache checks to see if it has a copy of the object stored locally. If yes, it will return it within an HTTP response message to the browser.
3. If not, the Web cache opens a TCP connection to the origin server, which responds with the requested object.
4. The Web caches receives the object, stores a copy in its storage and sends a copy, within an HTTP response message, to the browser over the existing TCP connection.

Therefore a **cache is both a server and a client at the same time**. Usually caches are purchased and installed by ISPs. They can substantially reduce the response time for a client request and substantially reduce traffic on an institution's access link to the Internet.

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

2.2.6 The Conditional `GET`

Caches introduce a new problem: what if the copy of an object residing in the cache is stale? The **conditional GET** is used to verify that an object is up to date. An HTTP request message is a conditional get if

1. the request message uses the `GET` method
2. the request message includes an `If-modified-since:` header line.

A conditional get message is sent from the cache to server which responds only if the object has been modified.

2.5 DNS - The Internet's Directory Service

One identifier for a host is its **hostname** [`cnn.com`, `www.yahoo.com`]. Hostnames are mnemonic and therefore used by humans. Hosts are also identified by **IP addresses**.

2.5.1 Services provided by DNS

Routers and use IP addresses. The Internet's **domain name system (DNS)** translates hostnames to IP addresses. The 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.

DNS servers are often UNIX machines running the **B**erkeley **I**nternet **N**ame **D**omaine (BIND) software.

DNS runs over UDP and uses port 53 It is often employed by other application-layer protocols (HTTP, FTP...) to translate user-supplied hostnames to IP addresses.

How it works:

- The user machine runs the client side of the DNS application
- The browser extracts `www. xxxxx . xxx` from the URL and passes the hostname to the client side of the DNS application
- The DNS sends a query containing the hostname to a DNS server
- The DNS client eventually receives a reply including the IP address for the hostname
- The browser can initiate a TCP connection.

DNS adds an additional delay

DNS provides other services in addition to translating hostnames to IP addresses:

- **host aliasing**: a host with a complicated hostname can have more alias names. The original one is said to be a **canonical hostname**.
- **mail server aliasing**: to make email servers' hostnames more mnemonic. This also allows for an e-mail server and an Web server to have the same hostname.
- **load distribution**: replicated servers can have the same hostname. In this case, a set of IP addresses is associated with one canonical hostname. When a client make a DNS query for a name mapped to a set of addresses, the server responds with the entire set, but rotates the ordering within each reply.

2.5.2 Overview of How DNS Works

From the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service. Having one single global DNS server would be simple, but it's not realistic because it would a **single point of failure**, it would have an impossible **traffic volume**, it would be **geographically too distant** from some querying clients, its **maintenance** would be impossible.

A Distributed, Hierarchical Database The DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world.

[Alt text][./dns-servers.png]

The three classes of DNS servers:

- **Root DNS servers**: In the Internet there are 13 root DNS servers, most hosted in North America, each of these is in reality a network of replicated servers, for both security and reliability purposes (total: 247)
- **Top-level domain (TLD) servers**: responsible for top-level domains such as com org net edu and govand all of the country top-level domains uk fr jp
- **Authoritative DNS servers**: every organization with publicly accessible hosts must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization can choose to implement its own authoritative DNS server or to pay to have the records stored in an authoritative DNS of some service provider.

Finally there are **local DNS servers** which is central to the DNS architecture. They are hosted by ISPs. When a hosts connects to one of these, the local DNS server provides the host with the IP addresses of one or more of its local DNS servers. Requests can ho up to the root DNS servers and back down.

[Alt text][./distributedDNS.png]

We can have both **recursive** and **iterative queries**. In **recursive queries** the user sends the request its nearest DNS which will ask to a higher-tier server, which will ask to lower order… the chain goes on until it reaches a DNS that can reply, the reply will follow the inverse path that the request had. In **iterative queries** the same machine sends requests and receives replies. Any DNS can be iterative or recursive or both.

DNS Caching DNS extensively exploits DNS caching in order to improve the delay performance and to reduce the number of DNS messages ricocheting around the Internet. In a query chain, when a DNS receives a DNS reply it can cache the mapping in its local memory.

2.5.3 DNS Records and Messages

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

A resource record is a four-tuple that contains the fields: `(Name, Value, Type, TTL)` `TTL` is the time to live of the resource record (when a resource should be removed from a cache). The meaning of `Name` and `Value` depend on `Type`:

| Type | Name | Value | |——- |—————————— |————————————————————————————————— | | A | a hostname | IP address | | NS | a domain (foo.com) | hostname of an authoritative DNS server which knows how to obtain the IP addresses for hosts in the domain. Used to route queries further along in the query chain | | CNAME | a alias name | canonical hostname for the name in Name | | MX | alias hostname | canonical hostname of a mail server that has an alias hostname Name | #### DNS Messages The only types of DNS messages are DNS queries and reply messages. They have the same format:

- first 12 bytes in the *header section*: 16-bit number identifying the query, which will be copied into the reply query so that the client can match received replies with sent queries. 1 bit query/reply flag (0 query, 1 reply). 1 bit flag authoritative flag set in reply messages when DNS server is an authoritative for a queried name. 1 bit recursion flag if the client desires that the server performs recursion when it doesn't have a record, 1 bit recursion-available field is set in the reply if the DNS server supports recursion
- *question section*: information about the query: name field containing the name being queried, type field
- *answer section*: resource records for the name originally queried: Type, Value, TTL. Multiple RRs can be returned if the server has multiple IP addresses
- *authority section*: records for other authoritative servers.
- *additional section*: other helpful records: canonical hostnames…

Inserting Records into the DNS Database We created a new company. Next we register th domain name `newcompany.com` at a registrar. A **registrar is a commercial entity that verifies the uniqueness of the domain name, enters it into the DNS database and collects a small fee for these services**. When we register the address, **we need the provide the registrar with the IP address of our primary and secondary authoritative DNS servers**, that will make sure that a Type NS and a Type A records are entered into the TLD com servers for our two DNS servers.

Focus on security: DNS vulnerabilities

- DDoS bandwidth-flooding attack
- MITM: the mitm answers queries with false replies tricking the user into connecting to another server.
- The DNS infrastructure can be used to launch a DDoS attack against a targeted host

To date, there hasn't been an attack that that has successfully impeded the DNS service, DNS has demonstrated itself to be surprisingly robust against attacks. However there have been successful reflector attacks, these can be addressed by appropriate configuration of DNS servers.

2.6 Peer-to-Peer Applications

2.6.1 File Distribution

In P2P file distribution, each peer can redistribute any portion of the file it has received to any peers, thereby assisting the server in the distribution process. As of 2012 the most popular P2P file distribution protocol is BitTorrent, developed by Bram Cohen.

Scalability of P2P architectures Denote the upload rate of the server's access link by u_s, the upload rate of the ith peer's access link by u_i and the download rate of the ith access link by d_i, tthe size of the to be distributed in bits () Comparison client-server and P2P.

Client-Server The server must transmit one copy of the file to N peers, thus it transmits NF bits. The time to distribute the file is at least **NF/u_s**. Denote $d_min = min\{d_i\}$ the link with the slowest download rate cannot obtain all F bits in less than F/d_min seconds Therefore:

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_min}\right\}$$

P2P When a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers.

- At the beginning of the distribution only the server has the file. It must send all the bits at least once. $D \geq F/u_s$
- The peer with the lowest download rate cannot obtain all F bits of the file in less than $F/d_min $ seconds.
- The total upload capacity of the system is equal to the summation of the upload rates of the server and of all the peers. The system must upload F bits to N peers, thus delivering a total of NF bits which can't be done faster that u_total.

We obtain:

$$D_{P2P} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_j}\right\}$$

BitTorrent In BitTorrent 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. At the beginning a peer has no chunks, it accumulates more and more chunks over time. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file it may leave the torrent or remain in it and continue to upload chunks to other peers (becoming a *seeder*). Any peer can leave the torrent at any and later rejoin it.

Each torrent has infrastructure node called a *tracker*: when a peer joins a torrent, it registers itself with the tracker and periodically inform it that it is still in the torrent. The tracker keeps track of the peers participating in the torrent. A torrent can have up to thousands of peers participating at any instant of time.

User joins the torrent, the tracker randomly selects a subset of peers from the set of participating peers. User establishes concurrent TCP connections with all of these peers, called *neighboring peers*. The neighboring peers can change over time. The user will ask each of his neighboring peers for the list of chunks they have (one list per neighbor). The user starts downloading the chunks that have the fewest repeated copies among the neighbors (**rares first** technique). In this manner the rarest chunks get more quickly redistributed, roughly equalizing the numbers of copies of each chunk in the torrent.

Every 10 seconds the user measures the rate at which she receives bits and determines the four peers that are sending to her at the highest rate. It then reciprocates by sending chunks to these same four peers. The four peers are called **unchocked**. Every 30 seconds it also choses one additional neighbor at sends it chunks. These peers are called **optmistically unchocked**.

2.6.2 Distributed Hash Tables (DHTs)

How to implement a simple database in a P2P network? In the P2P system each peer will only hold a small subset of the totality of the (key, value) pairs. Any peer can query the distributed database with a particular key, the database will locate the peers that have the corresponding pair and return the pair to querying peer. Any peer can also insert a new pair in the databse. Such a distributed database is referred to as a **distributed hash table (DHT)**. In a P2P file sharing application a DHT can be used to store the chunks associated to the IP of the peer in possess of them.

An approach: let's assign an identifier to each peer, where the identifier is an integer in the range $[0, 2^n - 1]$ for some fixed n. Such identifier can be expressed by an n-bit representation. A hash function is used to transform non-integer values into integer values. We suppose that this function is available to all peers. How to assign keys to peers? We assign each (key,value) pair to the peer *whose identifier is the closest to key*, which is the identifier defined as *the closest successor of the key.* To avoid having each peer keeping track of all other peers (scalability issue) we use

Circular DHT If we organize peers into a circle, each peer only keeps track of its immediate successor and predecessor (modulo 2^n). This circular arrangement of peers is a special case of an **overlay network**: the peers form an abstract logical network which resides above the "underlay" computer network, the overlay links are not physical but virtual liaisons between pairs of peers. A single overlay link typically uses many physical links and physical routers in the underlying network.

In the circle a peer asks "who is responsible for key k?" and it sends the message clockwise around the circle. Whenever a peer receives such message, it knows the identifier of its predecessor and predecessor, it can determine whether it is responsible (closest to) for the key in question. If not, it passes the message to its successor. When the message reaches the peer responsible for the key, this can send a message back to the querying peer indicating that it is responsible for that key. Using

this system *N/2* messages are sent on average (N = number of peers). Designing a DHT there is a tradeoff between the number of neighbors for each peer and the number of DHT messages needed to resolve a single query. (1 message if each peer keeps track of all other peers, *N/2* messages if each knows only 2 neighbors). To improve our circular DHT we could add shortcuts so that each peer not only keeps track of its immediate successor and predecessor but also of relatively small number of shortcut peers scattered about the circle. How many shortcut neighbors? Studies show that DHT can be designed so that bot the number of neighbors per peer as well as the number of messages per query is O(log N) (N the number of peers).

Peer Churn In a P2P system, a peer can come or go without warning. To keep the DHT overlay in place in presence of a such peer churn we require each peer to keep track (know to IP address) of its predecessor and successor, and to periodically verify that its two successors are alive. If a peer abruptly leaves, its successor and predecessor need to update their information. The predecessor replaces its first successor with its second successor and ask this for the identifier and IP address of its immediate successor.

What if a peer joins? If it only knows one peer, it will ask him what will be his predecessor and successor. The message will reach the predecessor which will send the new arrived its predecessor and successor information. The new arrived can join the DHT making its predecessor successor its own successor and by notifying its predecessor to change its successor information.

2.7 Socket Programming: Creating Network Applications

Only code explication —-> skipping

Chapter 3: Transport Layer

3.1 Introduction and Transport-Layer Services

A transport-layer protocol provides for **logical communication** (as if the hosts running the processes were directly connected) between application processes running on different hosts. 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. **Transport-layer protocols are implemented in the end systems but not in network routers**. On the sending side, the transport layer converts the application messages into transport-layer packets, known as transport-layer **segments**. This is done by breaking them into smaller chunks and adding a transport-layer header to each chunk. The transport-layer then passes the segment to the network-layer packet at the sending end-system. On the receiving side, the network layer extracts the transport-layer segment from the datagram and passes the segment up to the transport-layer which then processes the received segment, making the data in the segment available to the received application.

3.1.1 Relationship Between Transport and Network Layers

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

3.1.2 Overview of the Transport Layer in the Internet

A TCP/IP network (such as the Internet) makes two distinct transport-layer protocols available to the application layer:

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

We need to spend a few words on the network-layer protocol: the Internet network-layer protocol is the IP (Internet Protocol). It provides a logical communication between hosts. The IP service model is a **best-effort delivery service**: it makes the best effort to deliver segments between hosts, *but it makes guarantees*:

- it doesn't guarantee segment **delivery**
- it doesn't guarantee **orderly** delivery of segments
- it doesn't guarantee the **integrity** of the data in the segments

Thus IP is said to be an **unreliable service**. Every host has **at least one network-layer address** a so-called IP address.

UDP and TCP extend IP's delivery service between to end systems to a delivery service between two processes running on the end systems. Extend host-to-host delivery to process-to-process delivery is called **transport-layer multiplexing and demultiplexing**. UDP provides process-to-process delivery and error checking are the only services provided by UDP (therefore it is an unreliable service). TCP provides **reliable data transfer** using flow control, sequence numbers,

acknowledgements and timers. **TCP thus converts IP's unreliable service between end systems into a reliable data transport service between processes** TCP also provides **congestion control** a service not really provided to the invoking application as it is to the Internet as a whole: it prevents any TCP connection from swamping the links and routers between communication hosts with an excessive amount of traffic giving each connection traversing a congested link an equal share of the bandwidth.

3.2 Multiplexing and Demultiplexing

Here we'll cover m&d in the context of the Internet but **a multiplexing/demultiplexing service is needed for all computer networks**.

- The 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 (which will be used in demultiplexing) to create segments and passing the segments to the networks layer is called **multiplexing**.

Therefore sockets need to have unique identifiers and each segment needs to have special fields that indicate the socket to which the segment is delivered. These fields 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. Port numbers ranging from 0 to 1023 are called **well-known port numbers** and are restricted, reserved for us by well-known application protocols such as HTTP (80) and FTP (21). Designing an application, we should assign it a port number.

Connectionless Multiplexing and Demultiplexing A UDP socket is fully identified by the **two-tuple**: `(destination IP address , destination port number)` therefore if two UDP segments have different source IP address and/or source port numbers but have the same destination IP address and destination port number, than the two segments will be directed to the same destination process via the same destination socket. The source port number serves as part of the "return address"

Connection-oriented Multiplexing and Demultiplexing A TCP socket is identified by the **four-tuple**: `(source IP address, source port number, destination IP address, destination port number)` When a TCP segment arrives from the network to a host, the **host uses all four values to 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 carrying the original connection establishment request) be directed to two different sockets.

Routine:

- The TCP server application always has a **welcoming socket** that waits for connection establishment requests from TCP clients on port number X
- The TCP client creates a socket and sends a connection **establishment request** (a TCP segment including destination port, source port number and *a special connection-establishment bit set in the TCP header*)
- The server OS receives the incoming connection-request segment on port X, it locates the server process that is waiting to accept a connection on port number X, then creates **a new socket** which will be identified by `(source port number in the segment (cleint), IP address of source host (client), the destination port number in the segment (its own), its own IP address)`
- With the TCP connection in place, client and server can now send data to each other

The server may support many simultaneous TCP connection sockets, with each socket attached to a process and each socket identified by its own four-tuple. When a TCP segment arrives at the host, all the fours fields are used to demultiplex the segment to the appropriate socket.

Port Scanning Can be used both by attackers and system administrator to find vulnerabilities in the target or to know network applications are running in the network. The most used port scanner is **nmap** free and open source. For TCP it scans port looking for port accepting connections, for UDP looking for UDP ports that respond to transmitted UDP segments. It then returns a list of open, closed or unreachable ports. A host running nmap can attempt to scan any target *anywhere* in the Internet

Web Servers and TCP In a web server, all segments have destination port 80 and both the initial connection-establishment segments and the segments carrying HTTP request messages will have destination port 80, the server will distinguish clients using the source IP addresses and port numbers. Moreover in today's high-performing Web, servers often use only one process and *create a new thread with a new connection soket for each new client connection.*

If using persistent HTTP, client and server will exchange messages via the same server socket. If using non-persistent HTTP, a new TCP connection is created and closed for every request/response and hence a new socket is created and closed for every request/response.

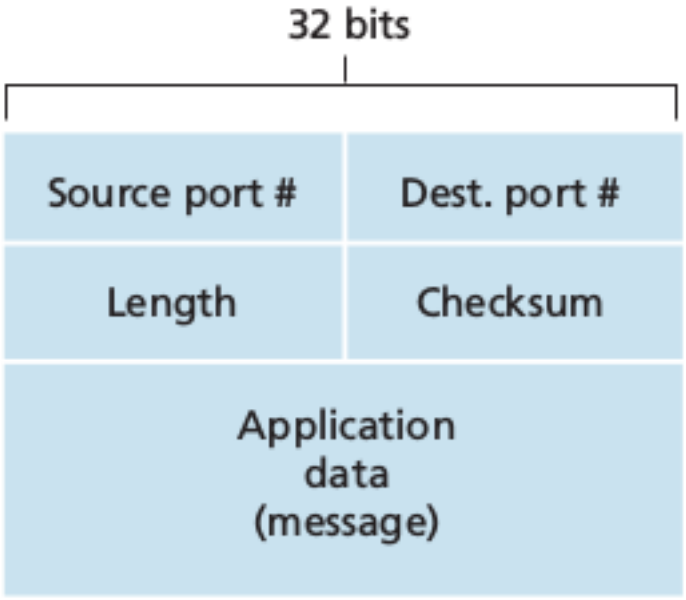
3.3 Connectionless Transport: UDP

UDP does multiplexing/demultiplexing, light error checking, nothing more. If the developer chooses UDP, the application is almost directly talking with IP. 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**. DNS is an example of an application layer protocol that typically uses UDP: there is no handshaking and when a client doesn't receive a reply either it tries sending the query to another name server or it informs the invoking application that it can't get a reply. Why should a developer choose UDP?

- *Finer application-level controll over what data is sent and when*: as soon as the application passes data to UDP, UDP will package the data inside a segment and immediately pass it to the network layer. TCP's congestion control can delay the sending of the segment and will try sending the packet until this is received. In real time applications the sending rate is important, so we can trade off some data loss for some sending rate.
- *No connection establishement* UDP justs send data without any formal preliminaries without introducing any delay, probably the reason why DNS runs over UDP.
- *No connection state*: because a UDP application doesn't need to keep track of the users or to keep connections alive, it can typically support many more active clients than a TCP application
- *Small packet header overhead* TCP has 20 bytes of header overhead in every segment versus the 8 of UDP

It is possible for an application developer to have reliable data transfer when using UDP. This can be done if reliability is built into the application itself (eg adding acknowledgement and retransmission mechanisms) but it is a nontrivial task and may keep the developer busy for a long time.

3.3.1 UDP Segment Structure



UDP segment structure

The UDP header has only four fields, each consisting of two bytes: source and destination port number, checksum and length (which specifies the number of bytes in the UDP segment, header + data). This last field is needed since the size of the data field may differ from one UDP segment to the next. The checksum is used for error detection.

3.3.2 UDP Checksum

Provides for error detection, to determine whether the bits in the segment have been altered as it moves from source to destination.

At the send side, UDP performs the 1s complement of the sum of all the 16-bit (max 64) 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 header.

UDP implements error detection according to the **end-end principle**: 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".

3.4 Principles of Reliable Data Transfer

It is the responsibility of a **realiable data transfer protocol** to implement reliable data service: no transferred data bits are corrupted or lost and all are delivered in the order in which they were sent. We will consider the following actions:

- The sending side of the data transfer protocol will be invoked from above by a call to `rdt_send()`
- On the receiving side `rdt_rcv()` will be called when a packet arrives while `deliver_data()` will be called when the `rdt` protocol wants to deliver data to the upper layer.

We use the term packet rather than segment because the concepts explained here applies to computer networks in general. We will only consider the case of **unidirectional data transfer** that is data transfer from the sending to the receiving side. The case of reliable **bidirectional** (full-duplex) **data transfer** is not more difficult but more tedious to explain. Nonetheless sending and receiving side will need to transmit packets in *both directions*.

3.4.1 Building a Reliable Data Transfer Protocol