Computer Networks

2018A7PS0193P

January 30, 2021

1 Networks

Definition 1.1. A **network** is a shared infrastructure that allows users to communicate with each other.

The basic building blocks of a network are **nodes** and **links**. Nodes may be hosts or forwarding nodes. The end hosts communicate with one another through the **core network**, consisting of forwarding nodes. These nodes are connected via links called **network edges**.

End hosts are physically connected to the core network via the **access network**. This consists of many parts, such as the ethernet switch, router, etc.

1.1 Communication Models

Networks could have multiple communication models:

- **Client-Server model**: The client host requests, and receives the service from an always-on server. This server is also an end host, but has some special privileges.
- **Peer-peer model**: Here, there is minimal or no use of dedicates servers, as in Bit-Torrent. The clients directly communicate with one another.

1.2 Core Network Models

1.2.1 Circuit Switching

How is our core network made? One way to do this is via **circuit switching**. There are end-to-end resources reserved for a "call", like on a telephone network. There is no sharing of resources. Call setup needs to be done as a preparatory step. Circuit switching generally can be implemented by two different methods:

- **FDM**, which stands for Frequency Domain Multiplexing. The total frequency bandwidth is divided among the users, allowing them to send data simultaneously.
- **TDM**, which stands for Time Domain Multiplexing. Here the time is divided among the users (perhaps in a round robin fashion). As such, one user gets access to the entire bandwidth of the circuit, but only for a period in time.

1.2.2 Packet Switching

Another way is through **packet switching**, where data is sent over the net in discrete "chunks", called packets. This is how it is done on the Internet. The host takes the application message, and breaks into packets of length L bits. It then transmits packets into the access networks at transmission rate R, also called the bandwidth. Of course, this means that each packet faces a transmission delay of L/R.

Packet switching uses **store and forward**. The packets are stored at intermediate nodes before sending to the next node. The intermediate node checks for errors in the packets before transmitting, assuring the integrity of the packets.

When using packet switching, there may be four sources of packet delay:

- Nodal processing : The node performs error checking and checks the header for the destination of the packet.
- Queueing: When the arrival rate of the packets is faster than the sending rate, the node will keep the packets in a buffer queue. As such, there is a delay when the packet wait in the node queue.
- Propagation: This is the delay from propagation of the packets from a node to the next node, i.e., the outgoing delay. This depends on the medium of the wire, and is given by d/s, where d is the length of the connection and s is the speed.
- Transmission : This is the delay from transmission of packets into the node, i.e. the incoming delay.

1.3 Performance of a Network

The performance of a network can be measured by the following parameters:

- Delay

- Packet loss : This is the number of packets lost when transmitting. Some applications, like streaming, might not care too much about this.
- Throughput : This is the amount of bits transferred in unit time. This is important in some applications, such as for file transfer.

2 The Internet

The Internet is, in fact, a network of networks. The networks must be able to communicate despite using different applications running on different devices - i.e. it is heterogeneous.

As such, the Internet is full of different access ISP networks. How do end hosts on different access ISPs communicate with one another. Of course, if we directly connect them all, it would not be scalable as it would need $O(N^2)$ connections. We also cannot use a single global hub, since it would be difficult to find a single place to put it and connect the entire world.

Since a single global ISP cannot scale to connect the entire world, we use multiple global ISPs. These must be interconnected themselves. One way to do this is using **peering links**, which directly link two global ISPs. Another is to use **Internet Exchange Points**, called IXPs, to which multiple global ISPs can connect.

The Internet uses this system in a tiered manner - end hosts might connect to a regional ISP, which may then connect to a higher level country ISP, and so on.

Some corporations, like Google, have their own Content Distribution Networks (CDNs), and have their own network to bring services and content closer to users.

2.1 Layered Network Model

The Internet is based on a Layered Network Model known as **OSI**. Any device under OSI can have the following layers:

1. Physical
2. Data Link
3. Network
4. Transport
5. Session

6. Presentation
7. Application

Each of these layers depend on the one below (lower number) and export their services to the ones above (higher number).

The end hosts implement all 7 layers of the model. Those which implement the first 3 layers are called **routers** or Layer 3 devices. Routers are used to connect two different networks. Those which implement the first 2 layers are called **switches** or Layer 2 devices. They connect devices within a network, i.e., in Local Area Networks.

The Internet stack does not actually use all 7 layers - in fact it uses only 5. It removes the Presentation layer, which allows applications to interpret the meaning of data. It also removes the Session layer, which is used for synchronization, check pointing and recovery of data exchange. These functions are generally performed by the Application layer and/or the Transport layer. This Internet stack is known as **TCP/IP model**.

In the context of the Internet, these layers perform the following functions:

1. **Physical:** This layer delivers bits between the two endpoints of a link, e.g. copper, fiber, wireless, etc.
2. **Data Link:** This layer delivers packets between two hosts in a local area network. These are bridges and switches.
3. **Network:** This layer connects multiple networks, e.g. routers. This uses the Internet Protocol (IP).
4. **Transport:** This layer does process-process data transfer. It may use a multitude of protocols, including TCP, UDP, etc.
5. **Application:** This layer supports network applications. It may use FTP, SMTP, HTTP, etc.

2.2 IP Hourglass Architecture

One way to imagine the Internet architecture is as a hourglass. The IP interconnects multiple existing networks, and hides the underlying technology from applications. This provides minimal functionality (has a ”narrow waist”). The tradeoff of this approach is that there are no assumptions being made, and as such no guarantee that something works.

2.3 Application Layer

2.3.1 Network Applications

A **Network application** is a program that runs on different end systems and communicates over a network. These are run only on the end hosts - core network devices do not run user application code.

The application architecture can run different application architectures:

- **Client-Server**: The server is an "always on" host, which has a permanent IP address. To be able to scale, there are generally large data centers acting as a virtual server. The clients communicate with the server. Unlike the server, they may be intermittently connected, and may have a changing dynamic IP address. These clients never directly communicate with one another.
- **Peer to peer** : Here, there is not always on server. Instead the end hosts directly communicate with one another. The peers are connected and may change IP addresses dynamically.
- **Hybrid of client server and peer to peer** : This is the case in Instant Messaging and Skype. In instant messaging, the chatting between two users is P2P, but to get the IP addresses of a user's friends, a central server is needed.

Processes communicate within the same host using interprocess communication, but they must communicate with different hosts by exchanging messages.

Definition 2.1. A **socket** is the interface between the application later and the transport layer within the host.

A process (a network application) send and receives messages using it's socket. This is a software entity, not a physical one.

To receive messages, each process must have some identifier. The IP address is not enough since it will only uniquely identify the host, but not the individual processes running on the host. So, we also use the port number to identify a process. For instance, an HTTP server would use port number 80, and a mail server would use port number 25.

2.3.2 Application Transport Services

What transport services does an application need? This changes from application to application. Here are some examples:

- **Data Loss** : Depending on the application, it might be necessary that data transfer is 100% reliable. For instance, in the case of file transfer, there must be no data loss, but some loss can be tolerated in the case of streaming.
- **Bandwidth** : Applications may also have bandwidth requirements. Streaming needs some minimum amount of bandwidth to work, but elastic applications like email and file transfer will make do with whatever bandwidth is available.
- **Timing** : Another parameter to consider is timing. Some applications, like games, require low delay, but for others this is unnecessary.

2.3.3 HTTP

The application layer for web pages is HTTP. A web page consists of objects, like HTML files, JPEG images, etc. Each of these objects is addressable by a URL. HTTP applications use TCP as it's transfer protocol.

1. Client initiates the TCP connection on port 80. The time taken to establish this is called the **Round Trip Time** or RTT.
2. Client sends a request to the server
3. Server receives message through it's socket and sends the response.
4. Server attempts to close the TCP connection. The client may refuse to do so if the client has not received the message.
5. The client receives the message and closes the connection.

It is important to remember that the time taken to send a file would be $2RTT +$ File Transmission Time - one RTT to establish connection, one RTT to send the first message and the time taken to send the file. Let us say the received file is an HTML file, which as 10 images embedded. Then, the client will request these images as well, by reopening the TCP connection and sending requests.

If at most one object is sent over a TCP connection, it is called **Non-persistent HTTP**. This was the case in HTTP version 1.0. In HTTP version 1.1, **Persistent HTTP** was introduced, which allows multiple objects to be sent over a single TCP connection between client and server. This could be done with or without a pipeline. If there is no pipeline, the client requests for each object, waits for the response, requests the next, and so on. To speed this up, we can use a pipeline and send requests for multiple objects one after another, while waiting for the response. The responses are always received in the same order in which they were sent.

2.3.4 HTTP Requests

A HTTP request message consists of:

- A request line. It contains the method (POST,GET,HEAD), the URL being requested, and the HTTP version being used.
- Header lines. Each header line has the header field name and a value. An example of a header line can be `User-Agent: Firefox/3.6.10` or `Keep-Alive:115`.
- The body, which contains all the data of an entity

Each line is delimited by a carriage return character \r and a line feed character `\n`. The request methods could have the following meanings:

- `GET` : This method is used to transfer from the server to the client
- `POST` : This is used to upload something to the server
- `HEAD` : This asks the server to leave requested object out of response
- `PUT` : This is present in HTTP version 1.1. It uploads a file in entity body to the path specified in the URL field.
- `DELETE` : This is present in HTTP version 1.1. It deletes the file specified in the URL field.

2.3.5 HTTP Responses

A HTTP response message consists of:

- A status line, which consists of the protocol, a status code and a status phrase. An example is `HTTP/1.1 200 OK`.
- Header lines, which are in the same format as in the request message. It could contain the time, the OS, etc.
- The body, which contains the requested data.

The HTTP response status codes are:

- `200 OK` : Request succeeded, requested object later in this message.
- `301 Moved Permanently` : Requested object moved, new location specified in the Location header line.
- `400 Bad Request` : Request message not understood by the server.
- `404 Not Found` : Requested document not found on this server.
- `505 HTTP Version not supported` : The HTTP version used in the request is not supported by the server.

2.3.6 States and HTTP

HTTP is a stateless protocol - it does not save anything from it's previous actions. To save the state in HTTP, we use **cookies**. When a client sends a http request to the server, it will create an ID for the user and create an entry in the backend database. Now, when the server sends it's response, it will include a header line called `set-cookie`. From then onwards, the usual HTTP request message will send the cookie ID through a header line called `cookie`.

Hence, using cookies, we can save state with HTTP. This can be used to save user information and track them on other sites.

2.3.7 Proxy Servers

Proxy servers are also called web caches. Clients send their requests to a proxy server, which would forward it to the destination. The incoming responses are also forwarded to their respective clients. The proxy server can cache information, and hence answer requests itself to save time. This is especially useful in institutions when the content that users access has a large overlap.

One issue with this is that the cached content might turn stale. To fix this we use conditional GETs. When the client sends a request, it adds a header line called `if-modified-since` which gives the last date at which an access was made.