CHAPTER 5

Computer Networks and Distributed Systems

So far, we have discussed the computer systems in isolation. Computers need to talk to each other to enable communication with other systems to enable higher-value services. When we talk about a set of computers communicating over a network, we are describing a distributed system. In this chapter, we will discuss how this happens.

History and Evolution of Networks and the Internet

Since the beginning of electronic computers, humans have had the desire to connect them. The US Department of Defense ARPANET was one of the earliest networks. By 1971, ARPANET was a 15-node network.

Roberts and Merrill proposed a common "messaging protocol" for heterogeneous computers to have a common language for sharing messages. Heterogeneous networks are defined as being made up of different computers, from different vendors. Wesley Clark, another researcher at ARPANET, proposed a two-layer approach, of an interface

P. D. Crutcher et al., *Essential Computer Science*,
https://doi.org/10.1007/978-1-4842-7107-0_5

layer and a communications layer. Hosts would provide the user interface, initiate the connection, and maintain the connection. And a communications layer of Interface Message Processors (IMPs) would move the data through the subnets to other hosts.

IMPs would break messages from the host into 8096-bit packets. A packet can be thought of as an envelope; it is a discrete set of bits that contain the message, or part of a message. A packet header contains the routing information and can be thought of as the address on the envelope. The IMP protocol added a common header that included source, destination, and control information. Routing is determining where to send a packet, so that it arrives at its proper destination. In IMPs, routing to the destination was not done by a central router; rather, each IMP kept a table of the routes with the amount of time it takes to send one of these packets. To ensure the arrival of the packets, an acknowledgement message was sent from the receiving IMP, and a checksum was used to verify the data was uncorrupted. If, after a certain period of time, the packet was not acknowledged, it would be sent again.

By 1971, a third layer had been added to the network stack, now application, host, and communications. Also, by 1971, the first application, a remote login program called telnet, was generally available. The File Transfer Protocol (FTP) and email were soon added and generally available by 1972. In the spring of 1972, ARPANET was demonstrated for the first time at the first International Conference on Computer and Communications (ICCC). The ARPANET that was demonstrated in 1972 was not the Internet though. It was a single network of 15 computers with one killer app in the form of email. What was learned in the development of ARPANET, however, led to the creation of the Internet.

Robert Kahn extended the work from ARPANET to see if the techniques could be applied to radio for both terrestrial transmission and satellite transmission. The biggest impact of this research was applied to local area networks.

ARPANET typically used leased phone lines to connect from computer to computer; however, in Hawaii, the phone lines were too noisy for clean data transmission. So ALOHAnet used radio to transmit the packets. ALOHAnet used two radio channels: one for machine data and one for user data. As you can imagine, without knowing when a transmission would be received, it was likely that two systems would transmit at the same time and collide with each other. It was impossible to know when to transmit to avoid a collision on the channel. ALOHAnet provided an elegant solution by not trying to avoid collisions. Recognizing that collisions would occur, the ALOHAnet researchers created an algorithm that would select a random time to wait and retransmit.

Robert Metcalfe improved on this algorithm with subsequent transmission collisions that would increase the random wait time exponentially to back off of a congested channel. Metcalfe applied this radio transmission technique to transmission on wires. Where transmitting data over radio at the time could carry thousands of bits per second, the transmissions over wires could transmit millions. Transmitting data on wires with this technique was named Ethernet.

Ethernet became the standard for data transmission for a local area network (LAN). By 1982, Ethernet products were commercially available for personal computers.

Robert Kahn and Vincent Cerf, both computer science researchers on ARPANET, created the Internet architecture. The Internet architecture was more flexible and more distributed than the architecture of ARPANET. The Internet architecture was adopted by not only the Internet itself but many other networks.

In 1973, Vincent Cerf organized a seminar to design the Internet host protocol, the Transmission Control Protocol (TCP). Cerf and Kahn asked the questions what would be the best protocol for unreliable networks and what would be the best way to connect different networks. Both Cerf and Metcalfe, as well as Gerard Le Lann, collaborated on TCP; as a result, TCP reflected the Ethernet protocol. TCP would be able to handle collisions

and out-of-order packet reception. The second question of how to connect different networks had two possible answers. The first possible answer was to continue doing what had been done, which is let each network have its own protocol and then translate between protocols at the network edge. Cerf and Khan realized this would not scale as the number of networks grew, so they pushed for the second possible answer, to have a common protocol, TCP. The advantages of a common protocol, such as a common address space and transparency of the network boundaries, were worth the cost of needing to upgrade legacy networks.

To connect to other LANs and potentially translate between different network protocols, Cerf proposed a special kind of computer called a gateway. A gateway would be connected to two or more networks, and those gateways would maintain routing tables between the networks. This allows the local networks to connect to other networks and eventually be part of the Internet without having total knowledge of the Internet.

TCP required that all packets were reliably delivered, but this was not needed in every case. In some cases, for instance, if you are broadcasting a message out to lots of subscribers and don't care if they get it or not, an unreliable protocol makes more sense. As such, in 1978, Vincent Cerf, Jon Postel, and Dan Cohen proposed that TCP was split into two protocols, TCP and IP. TCP was still responsible for reliable delivery, and IP was for simply passing packets between machines. This reduced the complexity of gateways, because they now only needed to handle IP.

By the end of the 1970s, TCP/IP had been developed and by the early 1980s had become a de facto standard for computer-to-computer communication. At the time, it was not the only standard floating around. A group of public telephone companies and communication equipment manufacturers had developed a standard called X.25 that largely overlapped with TCP/IP. X.25 varied from TCP/IP in that it defined network switching nodes to make virtual circuits between computers during the communication sessions.

Many in the community saw that X.25 was in direct competition with TCP/IP and a threat to open networks. Both network protocols were used during this period, with commercial networks using X.25, while the ARPA Internet used TCP/IP and private networks used a mix of X.25 and TCP/IP. While the debate about how to connect these disparate networks continued, the International Organization for Standardization (ISO) was focusing on computer manufacturers. To help keep the emerging network standards open, ISO created the Open Systems Interconnection (OSI) project.

Because networking computers were still new, ISO did not want to specify specific protocols or standards. Instead, they provided a standard model for creating network models. ISO based their model on the layering scheme that had been created by ARPANET. The OSI model consists of seven layers: physical, link, network, transport, session, presentation, and application (Figure 5-1). The layering scheme allowed ISO standards for network protocols to be slotted into the appropriate layer. A side effect of this layered approach to the network model was that it shaped the thinking of network protocols.

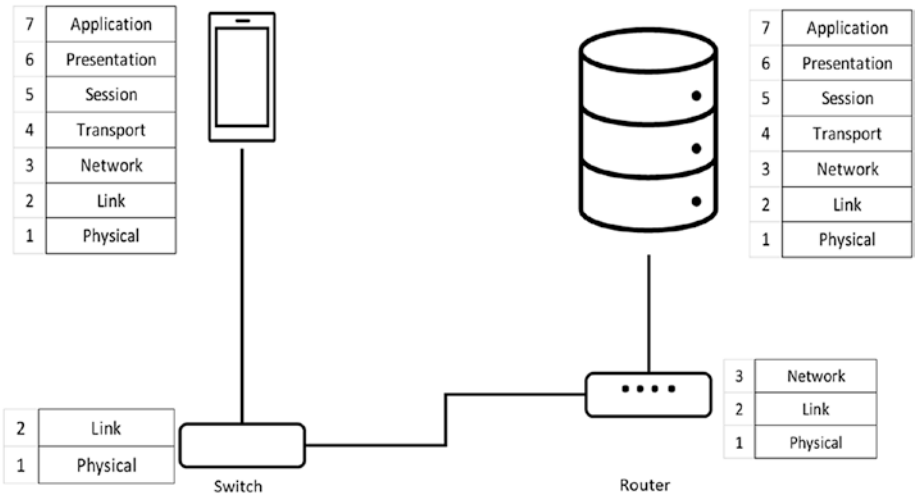

Figure 5-1. *OSI Layered Model Showing Layers Used for Connection*

Throughout the 1980s, the Internet grew from a small set of networks mostly related to defense research to an enormous network of computers and users. The Internet was transferred from military to civilian control. At the same time, the personal computer revolution was happening. One growing capability of personal computers was the ability to connect to other users through dial-up modems and bulletin board systems (BBSs). BBSs were sometimes connected to FidoNet, a network of bulletin board systems.

In 1989, Tim Berners-Lee invented the HyperText Markup Language (HTML) and the Hypertext Transfer Protocol (HTTP). This was the beginning of the World Wide Web as we know it. In 1993, Marc Andreessen and Eric Bina created a graphical user interface web browser for the National Center of Supercomputing Applications (NCSA) called Mosaic. The Mosaic client used HTTP to connect to servers on the Internet to download and display HTML content. Web browsers have continued to evolve as one of the primary clients of the Internet Protocols.

The World Wide Web Consortium (W3C) was founded in 1994 by Tim Berners-Lee, with a mission to lead the World Wide Web to its full potential by defining protocols and guidelines to ensure long-term growth of the Web.

Protocols: Stateful and Stateless

Protocols are the language used to communicate between computing systems on a network called nodes. The protocols carry the information about the connection as well as the content. Protocols define the rules of how to communicate between nodes. The protocol algorithm defines the rules such as who speaks next and what is expected. A protocol is implemented by a header containing the required data and an algorithm that utilizes that data.

Network protocols can be either stateful or stateless. Stateful protocols keep track of the connection between the two nodes as part of the protocol data itself. Stateless protocols do not track the state in the protocol, so, in general, there is no relation from one message to the next. There are advantages to both types of protocols, as we discuss in the following.

Internet Protocol (IP): TCP and UDP

The Internet Protocol suite handles the connections between the host systems on the Internet, covering the transport and network levels in the OSI model. The Transmission Control Protocol (TCP) is used for connection-oriented data traffic. The User Datagram Protocol (UDP) is used for connectionless data traffic. Connectionless data traffic is data that is sent but not guaranteed to be received by another node. We will describe why this is done in the UDP section. The underlying Internet Protocol (IP) provides the methods to instruct and route the traffic on the network. The current version of IP is IPv6; however, IPv4 is still in heavy use. One of the key differences between IPv4 and IPv6 is the available addressable space

in the IP address. IPv4 has 32-bit IP addresses with both the source and destination host addresses and has a 20-byte header (Figure 5-2). IPv6 has 128-bit IP addresses, again with both source and destination, and has a 40-byte header (Figure 5-3).

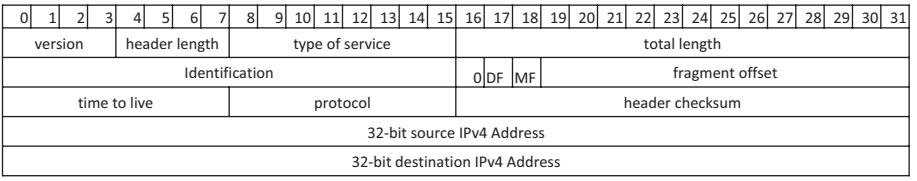

Figure 5-2. *IPv4 Header (32 Bits per Row)*

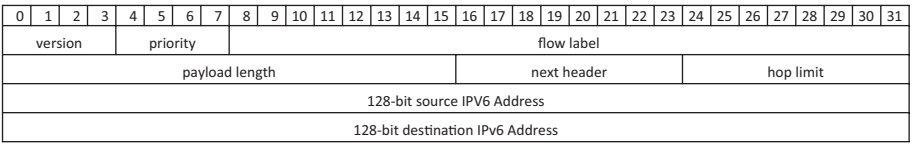

Figure 5-3. *IPv6 Header (32 Bits per Row)*

The Transmission Control Protocol, TCP, sits on top of IP (Figure 5-4). TCP is the same regardless of whether it uses IPv4 or IPv6. TCP is a connection-oriented protocol in that it provides a byte stream for user processes that is both full duplex and reliable. TCP is reliable because it guarantees the data is sequenced and can be reassembled in the same order it was sent. Many of the most common application protocols such as FTP, HTTP, and ISMP sit on top of TCP. TCP provides features like acknowledgement and timeouts to increase reliability. TCP is also full duplex, which means that data can simultaneously be sent and received by both endpoints on a single connection. TCP keeps track of state information such as the sequence numbers of the message segments.

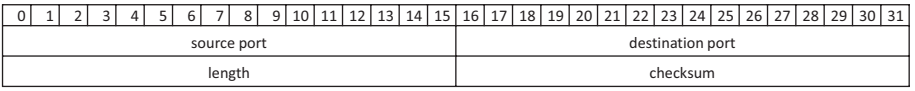

Figure 5-4. *TCP Header (32 Bits per Row)*

TCP defines a connection algorithm that is illustrated in Figure 5-5. First, a client will send a synchronization (SYN) message to the server with a sequence number (j). If and when that is received by the server, the server will send both its own synchronization (SYN) with a sequence number (k) and an acknowledgement (ACK) with the client's sequence number increased (j+1). Finally, if and when the client receives this message, it will respond back with an acknowledgement (ACK) with the server's sequence number increased (k+1). Once this handshake is done, then the client and server are connected and can communicate.

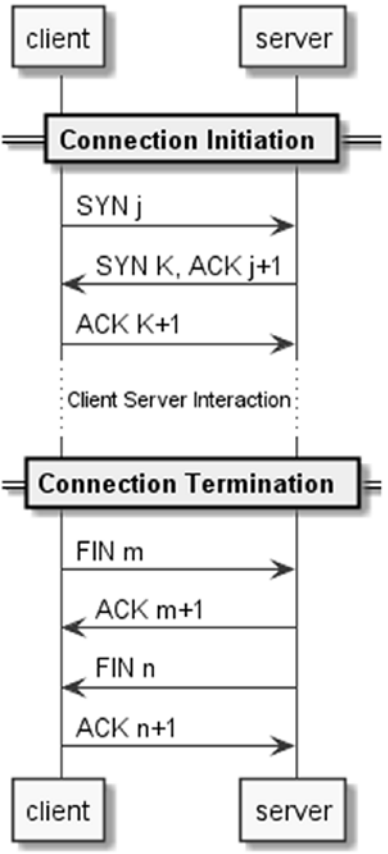

Figure 5-5. *TCP Connect and Disconnect*

When the client is done communicating with the server, it can terminate the connection by sending a finish (FIN) message and sequence number (m). The server responds with an acknowledgement (ACK) and sequence number (m+1) and then its own finish (FIN) message. Finally, the client responds with an acknowledgement (ACK) of the server's finish message, after which the client and server are not connected.

UDP or the User Datagram Protocol is the other part of the Internet Protocol suite. Unlike TCP, UDP is a connectionless protocol; this means that UDP can send a message to multiple receivers without disconnecting

from one receiver (Figure 5-6). This also means that there is no formal relationship between the senders and receivers, so receipt of the data is not guaranteed. UDP is typically used where performance and speed are more important than reliability. UDP messages are often referred to as datagrams. DHCP (Dynamic Host Configuration Protocol), RIP (Routing Information Protocol), and DNS (Domain Name System) are examples of protocols that are on top of UDP.

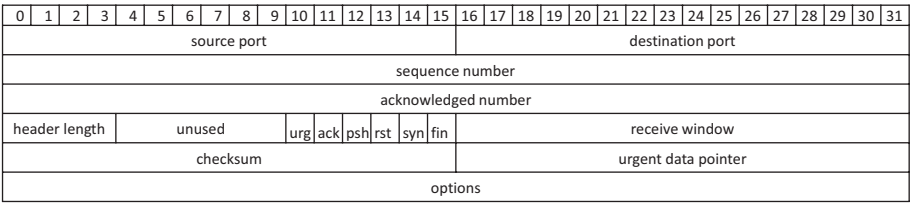

Figure 5-6. *UDP Header (32 Bits per Row)*

Host, IP Address, MAC Address, Port, Socket

The computers that are the various endpoints in the network are generically referred to as hosts. Hosts may have one or more physical layer connections to the network such as Ethernet adapters, Wi-Fi cards, or wireless WAN adapters. The link layer is this direct node-to-node connection from a physical connection on one system to a single other system. The MAC (media access control) address is the link layer address of these physical connections. The MAC address is a unique 48-bit number assigned to each device. With 48 bits, there is an addressable space for over 200 trillion devices on the network. The IEEE manages the assignment of MAC addresses to manufacturers of network equipment to prevent collisions of MAC addresses.

One or more IP addresses at the network layer can be assigned to the link layer MAC address. For IPv4, the IP address is a 32-bit number that is typically written as four dot-separated (between each byte) fields with values ranging from 0 to 255. With a 32-bit number, the addressable

space is about 4 billion possible IP addresses. With the explosion of the number of hosts on the Internet, especially Internet of Things (IoT) hosts, this 4 billion number is too small. An IPv6 address is a 128-bit number typically written as eight fields of four hexadecimal (16-bit/2-byte) digits separated by colons. 128 bits provides a sufficiently large address space for the future of the Internet. A loopback IP address represents the device to itself. The loopback addresses are 127.0.0.1 and 0:0:0:0:0:0:0:1 for IPv4 and IPv6, respectively. Multiple services or processes can run on the same host concurrently by using either TCP or UDP. Each service listens on a port number, which is a 16-bit number. Each service must have a unique port number to be accessible on a given host. When a client or a peer needs to connect to a particular service or peer, it needs to specify not only the IP address but the port that the service process is listening on. Sockets are an API (application programming interface) for connecting to network services. A socket is bound to a port and allows a program to send and receive data with another program. The Internet Assigned Numbers Authority (IANA) assigns ports and port ranges to various applications (Table 5-1) to avoid conflicts.

Table 5-1. *IANA Common Port Numbers and Ranges*

Port	Description
20	FTP Data
21	FTP Control
22	SSH
23	Telnet
25	Simple Mail Transfer
80	HTTP

(*continued*)

Table 5-1. (*continued*)

Port	Description
92	Network Printing Protocol
443	HTTPS
546	DHCP Client
547	DHCP Server
631	Internet Printing Protocol
8080	HTTP Alternate
1-1023	IANA well-known ports
1024-49151	IANA registered ports
49152-65535	IANA dynamic or private ports

DNS and DHCP

IP addresses are a great way of uniquely identifying hosts on the network, but it can be very difficult for humans to understand and remember the addresses of various hosts. The Domain Name Service (DNS) is a protocol to map human-understandable names to IP addresses. DNS sits on top of UDP. DNS servers maintain a mapping of domain names or human-understandable addresses to hosts on the Internet and the corresponding IP addresses. A DNS server will respond to a DNS resolution request with the IP address (Figure 5-7). If the DNS server does not have a matching name to IP address, it forwards the request up to a more authoritative DNS server, which may forward the request to other DNS servers. Once there is a name match, the IP address is returned to the original requestor. The remaining interactions between those hosts will be done with IP addresses.

DNS names follow a specific set of rules. The names must end in a top-level domain (TLD) such as .com or .org. Various countries each have

top-level domains. Preceding the top-level domain is a subdomain. This is usually the name of the organization that manages the host. Proceeding the subdomain and top-level domain is an arbitrary name for the specific host. Domain names are registered by a domain name registrar under the supervision of ICANN, the Internet Corporation for Assigned Names and Numbers.

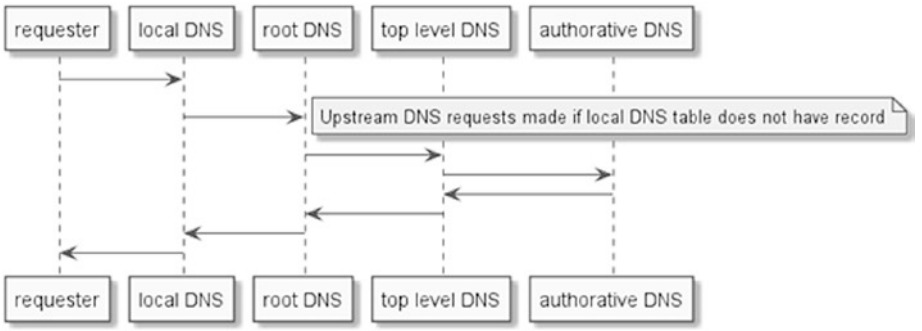

Figure 5-7. *Domain Name Lookup Sequence*

The Dynamic Host Configuration Protocol or DHCP is a protocol to dynamically assign IP addresses to hosts on a LAN or IP segment. It is very common to use DHCP on local area networks so a person does not have to explicitly assign IP addresses to every host on the network.

With DHCP, the host sends out a DHCP service discovery message on the network. When a DHCP host receives a service discovery message, it responds with an IP address for the requesting system, the network mask, and the IP address lease time.

A network mask is a bit pattern that indicates which bits in the IP address cannot change. This indicates the range of possible IP addresses the host can reach. The network mask is sometimes called the subnet mask because it defines the subnet that the host is part of. A subnet is one or more hosts connected to a router. As an example (Figure 5-8), we have two

subnets 143.11.38.0/24 and 143.11.40.0/24, where the first 24 bits or three fields of the IP addresses in the subnets will be the same.

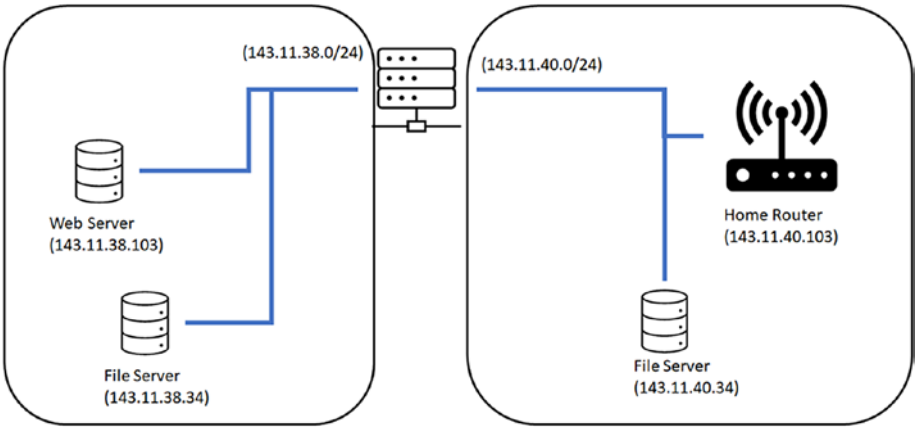

Figure 5-8. *Two Subnets Connected to a Router*

The DHCP lease time is how long the requesting client will have that IP address before needing to request a new one. Once the client selects the IP address, it will respond back to the DHCP server with a request for that IP address. Finally, the DHCP server will respond, acknowledging that the client is associated with that IP address. In addition to the IP address information, a DHCP server can also provide the address to a DNS server.

Proxy, Firewall, Routing

Routers are computers that have the responsibility of moving network packets through the network. A router does this by forwarding packets from an inbound link to an outbound link. A router uses a forwarding table to determine which outbound link to send the packet, by inspecting the destination IP address in the packet.

The forwarding table is kept up to date with the current network topology with the Routing Information Protocol (RIP). RIP is a UDP

with datagrams from other routers and systems. Because RIP is a connectionless UDP, the packets sometimes get lost. This is ok because the routing table will just get updated with the next RIP datagram. RIP provides a distance measurement to a router, by counting how many hops (number of routers it passes through) between the source and destination.

Another routing protocol is OSPF (Open Shortest Path First), which provides information to routers to build a complete map of the network topology, allowing packet forwarding to be based on a shortest path to the destination. OSPF is used by upper-tier Internet Service Providers, ISPs, where RIP is used inside enterprise networks.

The next routing protocol is the Border Gateway Protocol (BGP). BGP is used by subnets to advertise that subnet is part of the Internet.

Network Address Translation (NAT) does a similar job to routers of taking incoming packages and sending them out to a specific destination. Private IP addresses are IP addresses that can be used in multiple local area networks without conflicting as they cannot be routed out to the broader Internet. This is typically the type of IP address a DHCP server will serve up. To send and receive packets to and from these private networks, a NAT table is used to associate a private network IP address and port to a public IP address and port. For instance (Figure 5-9), you may be running a web server on your private network at 10.0.0.11 on port 80 and an FTP server at 10.0.0.9 on port 22 with NAT to your ISP-assigned address 143.11.38.34. The Internet only sees one device 143.11.38.34 and can send packets to that device. The NAT will inspect the packet it receives at 143.11.38.34 to check the port destination and then forward that packet to one of the two machines on the private network.

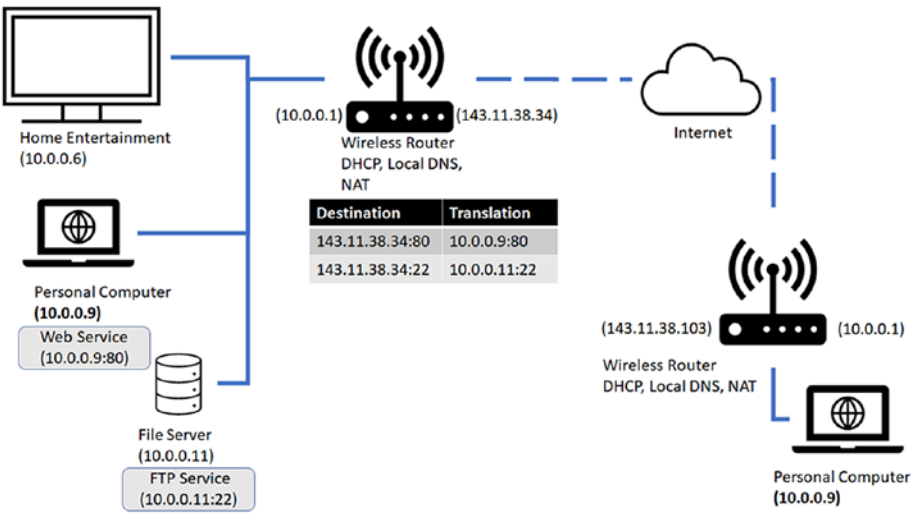

Figure 5-9. *Typical Network with DHCP and Network Address Translation*

A firewall works similarly to a NAT in that it inspects the incoming packets. Depending on certain criteria, it will either forward that packet or drop it. The destination application and port number are common rules that are set up in firewalls. Other rules include destination IP addresses and hostnames.

A proxy server is another service similar to a firewall in that it usually is part of the edge of a network before packets are sent out to the broader Internet to help secure your traffic. Even with encryption of the data, with TCP/IP, the headers are not encrypted, so your source and destination IP addresses are exposed. If you want to hide your source address, a proxy service will replace your source address with a proxy address and send it onto the destination. The destination will then respond back to the proxy server, which will reassemble the received packet with the original source address as the destination of the response.

Distributed Systems: Prominent Architectures

Now that we have looked at some of the fundamentals of what makes up a distributed system, let's look at some of the application architectures that are built on these network configurations.

Client Server

A client-server architecture is one of the oldest and most common architectures you will see on a network. In this architecture, you will see a centralized server that multiple clients connect to in order to accomplish a task. Many of the common Internet applications use a client-server architecture today, such as HTTP and email.

A client-server architecture has the advantage of centralizing access to data, so there won't be multiple potentially out-of-sync copies of the data. Data synchronization is a common problem with distributed systems in general. Data can be processed across multiple nodes, and that processing takes time. If data is changed during the time of processing in one node, but remains the same on another node, then data can be out of sync. The client-server architecture with its central access to data maintains what data to use and manages any synchronization issues.

With well-known protocols, a client-server architecture (Figure 5-10) can have a diverse set of clients that do not need to be implemented in the same programming language or even in the same operating system.

A microservice architecture is a modern variation of the client-server architecture with a client connecting to one or more (micro-, or smaller) services that provide a single capability, or a small set of related capabilities. A microservice has a smaller API and usually less code. Both of these features make individual microservices easier to maintain and secure. However, as the number of microservices grows, coordinating the microservices can become overly complex.

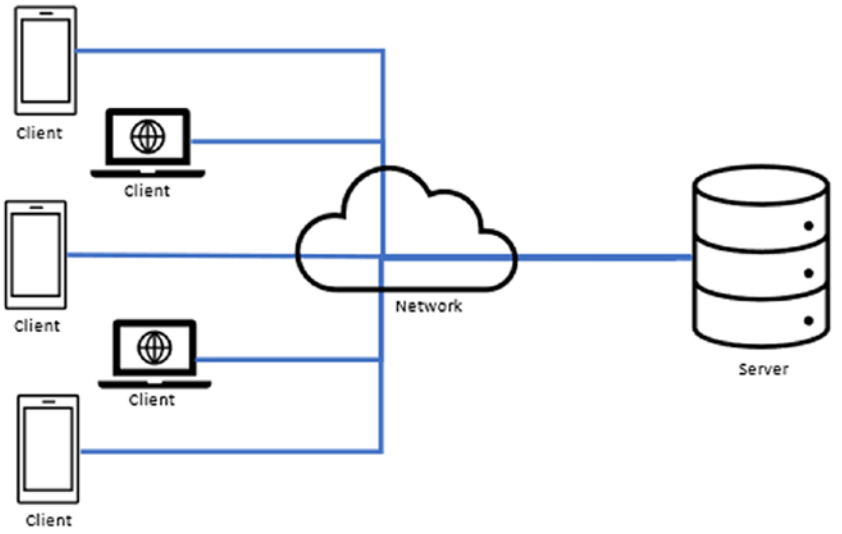

Figure 5-10. *Client-Server Architecture*

Peer to Peer

A peer-to-peer architecture (Figure 5-11) has two or more homogenous nodes in the network that can act as both client and server. This architecture is commonly used for distributed computing where each node does a portion of the computation on a portion of the data. It is also used for file sharing where each node shares distributing part of the files, which is then reassembled at the requesting node.

A peer-to-peer architecture is advantageous when centralized access is not needed, and portions of the work can be done independently. One of the challenges of a peer-to-peer architecture is discovering the peers. Multicast DNS or mDNS is one solution to this challenge. Using mDNS, a peer will send DNS information as a multicast UDP datagram on a network to advertise its presence. Other peers will receive this message to discover a peer. This only works on a single subnet. An alternative approach to discovery is that each peer will register with a central node and ask the central node about the other peers.

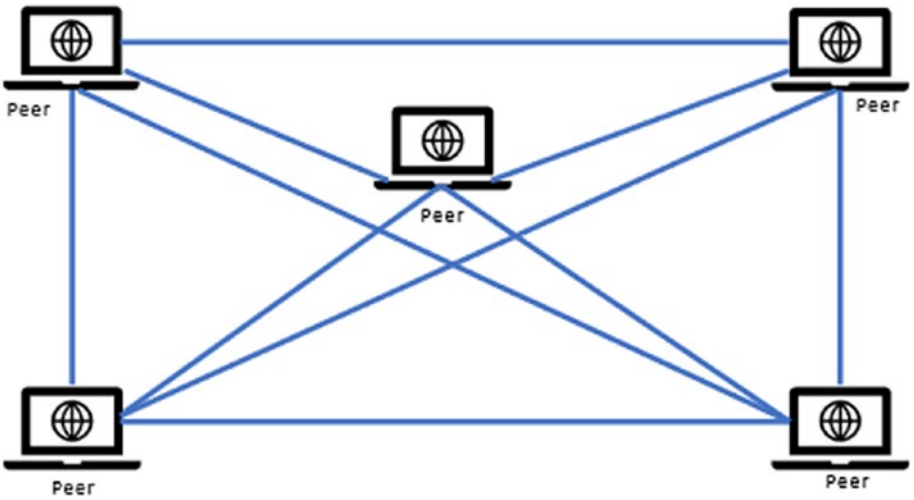

Figure 5-11. *A Peer-to-Peer Architecture*

N-Tiered

An N-tiered architecture (Figure 5-12) is when multiple nodes in the network have specific roles as part of the total solution. One of the most common N-tiered architectures is the three-tiered Model-View-Controller (MVC). The Model service provides the data for a particular model that the View service presents to the user. The Controller service operates on the model and transforms the data as defined by the business logic. This separation of concerns in the architecture provides the advantage of a flexible architecture that holds even when the underlying implementation changes. Model-View-View-Model (MVVM) and Model-View-Presentation (MVP) are other N-tiered architectures you may encounter.

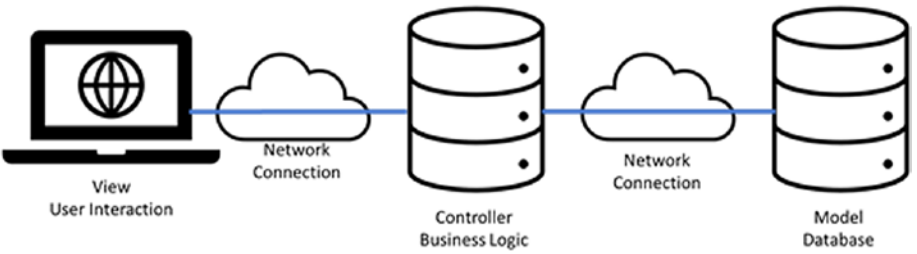

Figure 5-12. *An N-Tiered Architecture*

Distributed System Examples

File transfer (FTP) and the World Wide Web (HTTP) are two examples of distributed systems that we can look into at a detailed level.

FTP

FTP is one of the oldest protocols on the Internet. FTP is implemented with a client-server architecture and is considered a stateful protocol in that the server maintains the state of the connection with the client.

Let's examine what happens when a user wants to download a file from an FTP server (Figure 5-13). First, the user will start the FTP client on their host machine specifying the FTP server by hostname, for instance, `ftp.example.com`. The FTP client will first resolve the hostname to an IP address via DNS. Once the client has the IP address for `ftp.example.com`, for instance, 143.11.38.34, the FTP client can create a TCP/IP packet with 143.11.38.34 as the destination and port 21 to designate FTP. This packet gets sent and is received by the first router, which then forwards to the next router and so on until it gets to 143.11.38.34. The FTP server will set up a session for that client and then send a response packet, which will be routed back to the client host. Once the packet is received by the client, it is decoded, and the user is presented with connection information. The user can then log into the FTP server by entering a username and password.

The username is sent to the FTP server as one packet, which the FTP server associates with the session, and the password is sent in clear text as a separate packet to the FTP server. Once the FTP server has both the username and password, the user is authenticated. Now the user can send one or more commands to the FTP server. For the List command, the FTP server will respond with a listing of the files available for download. From here the user can send a Get command to get a specific file. This will open a separate connection to the FTP server on port 22 to receive the requested file. Finally, the user will send the Logout command to terminate the connection. When this packet is received by the FTP server, it "forgets" all of the information for this connection session and sends a connection terminated response back to the FTP client.

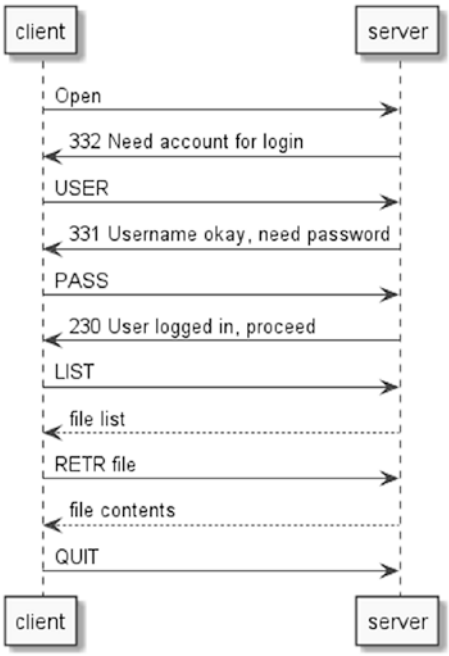

Figure 5-13. *FTP Login and File Transfer Sequence*

The World Wide Web

The modern World Wide Web is a collection of technologies that deliver a variety of services from video and music to banking and retail. One key technology that makes the modern Web so successful is SSL, the Secure Sockets Layer. SSL provides a method using asynchronous keys to encrypt the HTTP payload of a TCP/IP packet. This includes the HTTP headers and body.

For the purpose of simplifying this discussion, we will focus on nonencrypted HTTP (Hypertext Transfer Protocol) in comparison to FTP.

Like FTP, HTTP is a client-server architecture primarily for transferring files. Unlike FTP, HTTP is a stateless protocol, meaning the server does not keep any state about the client. This means HTTP needs to provide all connection information in each packet.

The World Wide Web uses the Uniform Resource Locator (URL) scheme to describe resources on the Web. This scheme defines the protocol, domain hostname or IP address, port, and path to the file. This scheme looks like this with each item in brackets indicating a parameter to specify: `<protocol>://<hostname>:<port>/path/to/file`. The protocol we will use for our example will be HTTP. FTP and HTTPS are two other protocols that can be addressed with an URL. HTTPS for HTTP secure is used to address HTTP through the Secure Sockets Layer and FTP for File Transfer Protocol. For our example, we will use the URL `http://example.com:80/index.html`. This example has HTTP as the protocol, example.com as the hostname, and port 80, which is the default port for HTTP, as the port number. Because we are using the default number for HTTP, we can exclude the port number from the URL.

The user opens a browser and enters the URL into the location field. The browser will decode the URL into its component parts. Just like the FTP client, the first thing the browser will do is resolve the hostname to an IP address. It will then create a TCP/IP packet with the IP address associated with example.com and port number equal to 80, the default

port for HTTP. Included in this packet is the HTTP command Get and the requested path. Like all TCP/IP packets, this will be forwarded from router to router until it reaches the server. Here is where HTTP is significantly different than FTP. When the HTTP server receives this packet, it will build a response packet including the contents of the file at the path, in our case index.html. The server will send this response packet back to the client and forget everything about that transaction (it won't keep state). When the response packet is received by the browser, the data content is parsed and rendered in your browser window. Table 5-2 lists the HTTP response code sent back to the client.

Table 5-2. *HTTP Response Codes*

Class	Code text	Code	Meaning
Success	OK	200	Request successfully fulfilled.
Success	Created	201	Used by the Post method to indicate newly created document.
Success	Accepted	202	Request has been accepted for processing, but is not yet processed.
Success	Partial Information	203	Not the definitive document requested but metainformation.
Success	No Response	204	Server received the request but does not send any information back.
Redirection	Moved	301	Requested document has permanently moved to a new URL. Header will contain a new URL.
Redirection	Found	302	Requested document has a different URL, but this is a valid URL.

(*continued*)

Table 5-2. (*continued*)

Class	Code text	Code	Meaning
Redirection	Method	303	Requested document not available with this method.
Redirection	Not Modified	304	Requested document has not changed; the client should use the cache.
Client Errors	Bad Request	400	Request had bad syntax or is impossible to fulfill.
Client Errors	Unauthorized	401	Request does not have a suitable Authorization header.
Client Errors	Payment Required	402	The request requires a ChargeTo header information.
Client Errors	Forbidden	403	Request is forbidden; there is no suitable Authorization header.
Client Errors	Not Found	404	The server has not found anything matching the URL.
Server Error	Internal Error	500	The server encountered an error.
Server Error	Not Implemented	501	The server has not implemented the facility to fulfill the request.
Server Error	Service Temporarily Overloaded	502	The server cannot fulfill this request do to load.
Server Error	Gateway Timeout	503	Similar to 500 error, but indicates the server cannot access another server.

While HTTP was originally developed for transferring HTML documents, "web pages," the versatility of a stateless protocol has allowed for a wider variety of applications to be implemented on top of the World Wide Web. ReST (Resource Stateless Transfer) is a method of creating general-purpose APIs using HTTP. ReST APIs will typically transfer documents that contain data. JavaScript Object Notation or JSON and the Extensible Markup Language (XML) are two common formats that are used for these data-rich documents. The data is sent to a client that may or may not be a browser.

Case Study: Web Application

As a case study, we will build a simple web application. This application will provide a browser-based form to request a user-specified number of files to recommend to the user and then allow the user to select one of those files to download from the back end.

System Architecture

This system will have three main components (Figure 5-14). On the front end will be a browser form that provides the user input. In the middle will be the HTTP ReST server that receives the requests from the front end. Finally, we have the data source that the server will use to choose files from.

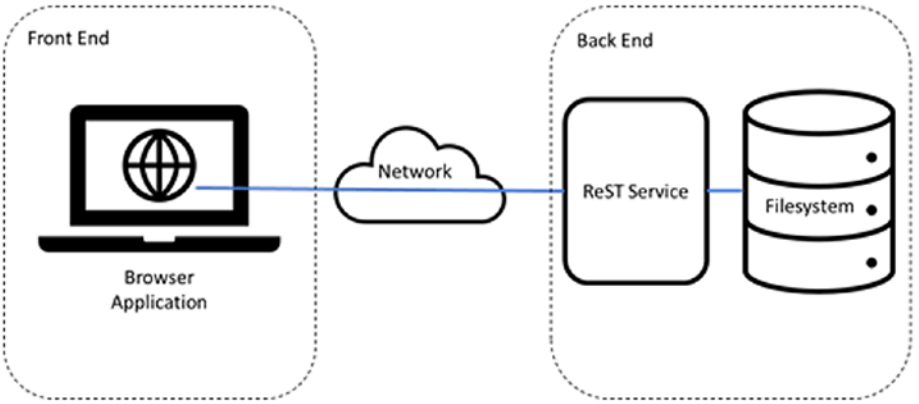

Figure 5-14. *Web Application Three-Tiered Architecture*

As part of the architecture for a ReST service, it will be important to define the resources that will be available before implementation. We will define the first resource as filelist with a URL of http://example.com/filelist/<count>. Each file that is available will also have a URL that will be http://example.com/files/<filename>, and finally the HTML content for the front end will be served from http://example.com/app.html.

HTML, CSS, and JavaScript

Before we dive into this solution, let's do a brief intro on HTML and some related topics. HTML stands for HyperText Markup Language and was the original intended format to be sent by HTTP. HTML provides a way of marking up a document into different sections using tags such as <head>, <body>, <script>, for division, the paragraph tag <p>, and many others, as shown in Listing 5-1. The sections are separated by a beginning tag <body> and an ending tag </body>. Tags can and do contain other tags.

Listing 5-1. Simple HTML Example

```
<html>
    <header><title>Network Example Client</title></header>
    <body>
        <div>
            <p>Hello Today</p>
        </div>
    </body>
</html>
```

Cascading Style Sheets or CSS is a method of providing styling information to the sections or tags of the documents. The style can be applied directly in the HTML document using a <style> tag or defined in a separate document and linked to the HTML document.

JavaScript is a programming language that is embedded in most web browsers and provides a programmatic access to the contents of the HTML document and the ability to alter the contents of the HTML document in the browser's memory. Similar to CSS, JavaScript code can be embedded in the HTML document using a <script> tag or defined in a separate document and linked to the HTML document. JSON is the native object definition syntax for JavaScript, allowing JavaScript code to easily read and manipulate JSON documents.

Front End

For the front end (Listing 5-2), we could use an HTML form, but we would like to get a little more dynamic and be able to update the view of the form without making additional requests to the app.html. The app.html will include a form to ask the user how many files they would like to see as options. JavaScript will connect, get the user input, and then form an HTTP request packet with the URL to the filelist route that includes the number of files to be provided as options. An HTTP request will be sent via

the browser to the back-end server. The server will send back a response containing JSON-formatted data that includes the files that the back end selected for options and the URLs for each file in the back end. The front end will then interpret this JSON data and update the form in the browser to show the file options to the user. Then the user will select one of the options, after which the front-end client will build an HTTP request to get the selected file. The back end will then respond with the contents for the file to the front end.

Listing 5-2. HTML and JavaScript for a Client

```
<html>
    <header><title>Network Example Client</title>
        <script >
            function load(){
                var xhttp = new XMLHttpRequest();
               var count = document.getElementById("count").value
                xhttp.onreadystatechange = function() {
                   if (this.readyState == 4 && this.status == 200) {
                         suggestions(this)
                    }
                };
                xhttp.open("GET", "filelist/"+count, true);
                xhttp.send();
            }
            function suggestions(resp) {
                var item_list = JSON.parse(resp.responseText);
                var suggest_html = "<ul>"
                item_list.forEach(element => {
                    suggest_html = suggest_html+'<li><a href="'
+element[1]+'">'+element[0]+"</a></li>"

                });
```

```
                suggest_html+="</ul>"
                document.getElementById("list").innerHTML =
                suggest_html;
            }
        </script>
    </header>
        <body>
        <H1>Network Demo</H1>
        <div>
            <input id="count" value=5>
            <button onclick=load()>Select</button>
        </div>
        <div id="list">
            <p>Select a Random list books</p>
        </div>
        </body>
</html>
```

Back End

The back end (Listing 5-3) will be a ReST service running on a system with a set of files on a disk. The first request the back end expects to get is a request for the front-end application at the route app.html. This is not required to be the first request because the back-end server is stateless. When it gets the request for app.html, the back end will return the HTML file to the browser, which the browser will render. Then the back end is ready to receive the next request. The next request could be a request for a number of files, for instance, to the route "filelist/3." With this request, the back end will parse the value 3 from the path and use that in a pseudo-random selection of three of the files from the disk. The back end will then encode a JSON object containing the name and URL for each of the

files and respond back to the front end. At this point, the back end will be ready to receive another request. The next request we might expect from the front end is a request for one of the files presented in the last response. Here the back end will read the file from the disk and create a response containing the contents of the file to send to the front end.

Listing 5-3. Python Flask Code for Serving the Back End

```
@routes.get('/filelist/{count}')
async def filelist(request):
    count = int(request.match_info.get("count",0))
    filelist = list(get_example(EXAMPLES, count))
    headers = {"Cache-Control": "no-cache"}
    return web.json_response(filelist, headers=headers)

@routes.get("/")
async def index(request):
    index = pathlib.Path(pathlib.Path(__file__).parent,
    "index.html")
    resp_text = index.read_text()
    return web.Response(text=resp_text, content_type="text/html")
```

Summary

In this chapter, we have covered a wide range of topics related to distributed systems and networks. We started with the history and evolution of the networking protocols that have brought us to the modern Internet. Next, we looked into the IPs that enable networks to work, such as TCP and UDP. Building on this, we examined specific protocols on top of UDP such as DNS and DHCP that help define the networks. And then we looked at the capabilities provided by TCP, such as FTP. After that we saw some common architectures for distributed systems including

client-server and peer-to-peer. Finally, we pulled all this knowledge together to create a simple client-server application using HTTP and related technologies, HTML and JavaScript.

References and Further Reading

- James Kurose and Keith Ross. *Computer Networking: A Top-Down Approach, Seventh Edition.* Pearson, 2016
- W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *Unix Network Programming, Volume 1: The Sockets Networking API, Third Edition.* Addison-Wesley Professional, 2004
- Janet Abbate. *Inventing the Internet*. MIT Press, 1999