CS 6250 Computer Network Lecture Notes

Jie Wu, Jack

Summer 2023

1 Lesson 1: Introduction, History, and Internet Architecture

1.1 Why Study Computer Networks

The reasons include at least the following

- internet growth, the explosion of applications made available by internet and the ever increasing number of internet users
- networks play an instrumental role in our society, transforming every aspect of our lives
- networking is a playground for interdisciplinary research innovations
- networking offers multidisciplinary research opportunities with potential for impact

1.2 A Brief History of the Internet

1.2.1 (1962) J.C.R. Licklider proposed the "Galactic Network"

The first version of a Network, proposed as "Galactic Network", by J.C.R. Licklider in 1962 who led a group of researchers to experiment connecting two computers. Among them Lawrence G. Roberts connected one computer in Massachusetts to another computer located in California via low-speed dial-up telephone line.

1.2.2 (1969) the ARPANET

The results of the first experiments showed that time-shared infrastructure was working sufficiently well at that time. However next researchers indicated the need for packet switching technology. Roberts developed the first network which was connecting four nodes (from UCLA, Stanford Research Institute, UCSB, and Univ. of Utah, respectively) into the initial ARPANET by the end of 1969.

1.2.3 (1970) Network Control Protocol (NCP), an initial ARPANET protocol

As the number of computers that were added to the ARPANET increased quickly, research work proceeded to designing protocols. The initial ARPANET Host-to-Host protocol called Network Control Protocol (NCP) was introduced in 1970, and it allowed the network users to begin developing applications. One of the first applications that launched was email in 1972.

1.2.4 (1973) Internetworking and TCP/IP

A DARPA team of researchers led by Bob Kahn, introduced the idea of open-architecture networking. This enabled individual networks to be independently designed and developed. This led researchers to develop a new version of the NCP protocol which would eventually be called the Transmission Control Protocol/Internet Protocol (TCP/IP), the original TCP paper published in 1973. The first version of TCP later split its functionalities into two protocols:

simple IP: used only for addressing and forwarding individual packets

separate TCP: used for service features such as flow control and recovery from lost packets

1.2.5 (1983) the Domain Name System (DNS) and (1990) the World Wide Web (WWW)

The scale of the Internet was increasing so rapidly that it was no longer feasible to have a single table of hosts to store names and addresses. The Domain Name System (DNS), which was designed to translate domain names to IP addresses by a scalable distributed mechanism, was introduced by Paul Mockapetris at USC in 1983. More applications sprung up quickly. One of the first and most popular applications was the World Wide Web (WWW) introduced by a team of researchers led by Tim Berners-Lee.

1.3 Internet Architecture Introduction

The **internet architecture** is what enables us to connect hosts running the same applications but located in different types of networks. The designers of the network protocols provide structure to the network architecture by organizing the protocols into layers. The functionalities in the network architecture are implemented by dividing the architectural model into layers, each offering different services. Some of the advantages of having a layered network stack include scalability, modularity and the flexibility to add or delete components.

1.4 The OSI Model

The Internet architecture follows a layered model, where every layer provides some service to the layer above. The International Organization for Standardization (ISO) proposed the seven-layered **OSI model**, which consists of the following layers: (1) application layer (2) presentation layer (3) session layer (4) transport layer (5) network layer (6) data link layer and (7) physical layer.
Separating the functionalities into layers offers multiple advantages. But there are also disadvantages of the layered protocol stack model, some of which are:

- some layers functionality depends on the information from other layers, which can violate the goal of layer separation
- one layer may duplicate lower layer functionalities (e.g. the functionality of error recovery can occur in lower layers, but also on upper layers as well)
- some additional overhead that is caused by the abstraction between layers

In the following sections we will go through a brief overview of the layers, and more specifically we will focus on

service: what each layer does

interface: how the layer is accessed

protocol: how the layer is implemented

reference: how we refer to the packet of information it handles

1.5 The OSI Reference Model and the Internet Architecture Model

The traditional Internet architecture model however only has five layers, where the application, presentation, and session layers are combined into a single layer called the application layer. The interface between the application layer and the transport layer are the **sockets**.

1.6 Application, Presentation, and Session Layers

1.6.1 The application layer

The application layer includes multiple protocols. The most popular ones include

HTTP and SMTP: for web and email

FTP: for transferring files between two end hosts

DNS: for translating domain names to IP addresses

The services that this layer offers are multiple depending on the application that is implemented. The same is true for the interface through which it is accessed and the protocol that is implemented. At the application layer, we refer to the packet of information as a **message**.

1.6.2 The presentation layer

The presentation layer plays the intermediate role of formatting the information that it receives from the layer below and delivering it to the application layer, e.g. formatting a video stream or translating integers from big endian to little endian format.

1.6.3 The session layer

The session layer is responsible for the mechanism that manages the different transport streams that belong to the same session between end-user application processes, e.g. tying together the audio stream and the video stream for a teleconference.

1.7 Transport and Network Layer

1.7.1 The transport layer

The transport layer is responsible for the end-to-end communication between end hosts. In this layer there are two transport protocols

TCP: offers a connection-oriented service to the applications running in the layer above, with (1) guaranteed delivery of the application-layer messages (2) flow control matching the sender's and receiver's speed (3) congestion control (the sender slow its transmission rate when it perceives the network to be congested)

UDP: offers a connectionless best-effort service to the applications running in the layer above, without guaranteed delivery, flow control, or congestion control

At the transport layer, we refer to the packet of information as a **segment**.

1.7.2 The network layer

In this layer, we refer to the packet of information as a **datagram**. A source Internet host sends the segment along with the destination address from the transport layer to the network layer. The network layer is responsible to deliver the datagram to the transport layer in the destination host. The protocols in the network layer are

IP protocol: defines (1) the fields in the datagram and (2) how the source/destination hosts and the intermediate routers use these fields, all Internet hosts and devices that have a network layer must run the IP protocol

routing protocol: determine the routes that the datagrams can take between sources and destinations

1.8 Data Link Layer and Physical Layer

1.8.1 The data link layer

In this layer, we refer to the packets of information as **frames**. Some example protocols in this layer include Ethernet, PPP, and WiFi. The network layer will route the datagram through multiple routers across the path between the sender and the receiver. At each node across this path the network layer passes the datagram to the data link layer, which in turn delivers the datagram to the next node. Then at that node the link layer passes the datagram up to the network layer.
The data link layer offers services that depend on the data link layer protocol that is used over the link, some example being guaranteed delivery. However this reliable delivery service is different from the one offered by the TCP protocol, because the delivery is from one transmitting node across one link to the receiving node rather than from the source host to the destination host.

1.9 The physical layer

The physical layer facilitates the interaction with the actual hardware. It is responsible to transfer bits within a frame between two nodes that are connected through a physical link. The protocols in this layer depend on the actual transmission medium of the link. For example one of the main protocols in the data link layer, Ethernet, has different physical layer protocols for twisted-pair copper wire, coaxial cable, and single-mode fiber optics.

1.10 Layers Encapsulation

At each layer the message is a combination of two parts: (1) the payload which is the message from the layer above (2) the new appended header information. This process of appending header information layer by layer is called **encapsulation**. For example

transport layer: segment = message + transport layer header information (Ht) thus encapsulates the application layer message, the header information can help the receiving host to (1) inform the receiver-side transport layer about which application to deliver the message (2) perform error detection and determine whether bits in the message have been changed along the route

network layer: datagram = segment + network layer header information (Hn) thus encapsulates the network layer segment, the header information includes the source and destination addresses of the end hosts

data link layer: frame = datagram + data link layer header information (Hl)

At the receiving end the process is reversed, with headers being stripped off at each layer. This reverse process is known as **de-encapsulation**.
Intermediate devices such as transport layer switches and network layer routers implement protocol stacks similarly to end-hosts. However they do not implement all the layers in the protocol stack, for example switches implement layers 1 to 2 while routers implement layers 1 to 3. But when data leave the sending host and are received by the layer-2 switch, the switch implements the same process of de-encapsulation to process the data and encapsulation to send the data forward to the next device. End-hosts implement all five layers while the intermediate devices don't. This design choice ensures that the Internet architecture puts much of its complexity and intelligence at the edges of the network while keeping the core simple.

1.11 The End-to-End Principle

principle: the network core should be simple and minimal, while the end systems should carry the intelligence

rationale: not all applications need the same features and network functions to support them, thus building such functions in the network core is rarely necessary

purpose: the original goal that led to the e2e principle is moving functions and services closer to the applications that use them, increases the flexibility and the autonomy of the application designer

consequence: the higher-level protocol layers are more specific to an application, whereas the lower-level protocol layers are free to organize the lower-level network resources to achieve application design goals more efficiently and independently

1.12 Violations of the End-to-End Principle and NAT Boxes

Some examples of the violation are

firewall: usually operate at the periphery of a network, violate the e2e principle because they are intermediate devices operated between two end hosts and can drop the end hosts' communication

NAT box: the network address translation (NAT) box maintains a translation table that provides a mapping between the public-facing IP address/ports and the IP addresses/ports that belong to hosts inside the private network (e.g. 10.0.0.4:3345 $\leftrightarrow$ 120.70.39.40:5001, violate the e2e principle because a host behind a NAT and a host on the public Internet cannot communicate by default without the intervention of a NAT box (workaround: STUN, UDP hole punching)

1.13 The Hourglass Shape of Internet Architecture

Researchers have suggested a model called the *Evolutionary Architecture model*, or EvoArch, that can help to study layered architectures and their evolution in a quantitative manner. Through this model researchers were able to explain how the hierarchical structure of the layer architecture eventually lead to the hourglass shape.

1.14 Evolutionary Architecture Model

Researchers have suggested a model called the Evolutionary Architecture model or EvoArch which can help illustrate layered architectures and their evolution in a quantitative manner. The EvoArch model considers an abstract model of the Internet's protocol stack that has the following components:

layers: a protocol stack is modeled as a directed and acyclic network with L layers

nodes: each network protocol is represented as a node, the layer of a node u is denoted by $l(u)$

edges: dependencies between protocols are represented as directed edges

node incoming edges: if a protocol u at layer l uses the service provided by a protocol w at the lower layer $l-1$, then this is represented by an "upwards" edge from w to u

node substrates: we refer to substrates of a node u, $S(u)$, as the set of nodes that u is using their services, every node has at least one substrate except the nodes at the bottom layer

node outgoing edges: the outgoing edges from a node u terminate at the products of u, the products of a node u are represented by $P(u)$

layer generality: each layer is associated with a probability $s(l)$, which we refer to as layer generality, a node u at layer $l+1$ selects independently each node of layer l as the substrate with probability $s(l)$, the layer generality decreases as we move to higher layers, protocols at lower layers are more general in terms of their functions or provided services than protocols at higher layers, e.g. in the case of the Internet protocol stack the protocols at layer 1 offer a very general bit transfer service between two connected points which most higher layer protocols would use

node evolutionary value: the value of a protocol node, $v(u)$, is computed recursively based on the products of u, which captures the fact that the value of a protocol u is driven by the values of the protocols that depend on it, the evolutionary value determines if the protocol will survive the competition with other protocols, at the same layer, that offer similar services

node competitors and competition threshold: we refer to the competitors of a node u, $C(u)$, as the nodes at layer l that share at least a fraction c of node u's products, we refer to the fraction c as the competition threshold

node death rate: the model has a death and birth process in place, to account for the protocols that cease or get introduced respectively, it is more likely that a protocol u dies if at least one of its competitors has a higher value than itself, when a node u dies, then its products also die if their only substrate is u

node basic birth process: the model has a basic birth process in place, the number of new nodes at a given time is set to a small fraction (say 1% to 10%) of the total number of nodes in the network at that time, so the larger a protocol stack is the faster it grows

EvoArch is iterated through the following steps: (1) introduce new nodes and place them randomly at layers (2) from the top to the bottom perform the following:

1. connect the new nodes by choosing substrates based on the generality probabilities of the layer below $s(l-1)$, and by choosing products for them based on the generality probability of the current layer $s(l)$
2. update the value of each node at each layer l, given that we may have new nodes added to it
3. examine all nodes in order of decreasing value in that layer and remove the nodes that should die

(3) finally we stop the execution of the model when the network reaches a given number of nodes.
The main takeaway message from Figure 4 in the referenced paper is that the layer width decreases as we move from the bottom layer to a middle layer and then it increases again as we move towards the top layer.
The EvoArch model can be used to explain the following

- The EvoArch model suggests that the TCP/IP stack was not trying to compete with the telephone network services. The TCP/IP managed to grow and increase its value without competing or being threatened by the telephone network.

- IPv4, TCP, and UDP provide a stable framework and EvoArch provides an explanation for this. A large birth rate at the layer above the waist can cause death for the protocols at the waist if these are not chosen as substrates by the new nodes at the higher layers. The transport layer acts as an "evolutionary shield" for IPv4, because any new protocols that might appear at the transport layer are unlikely to survive the competition with TCP and UDP which already have multiple products. In other words the stability of the two transport protocols adds to the stability of IPv4, which eliminate any potential new transport protocols that could select a new network layer protocol instead of IPv4.

For new Internet architectures the EvoArch model predicts that even if these brand new architectures do not have the shape of an hourglass initially, they will probably do so as they evolve, which will lead to new ossified protocols. The model suggests that one way to proactively avoid these ossification effects is to design the functionality of each layer so that the waist is wider, consisting of several protocols that offer largely non-overlapping but general services so that they do not compete with each other.

1.15 Architecture Redesign (Optional)

Some of the major design principles of the current Internet architecture are layering, packet switching, a network of collaborating networks, intelligent end-systems as well as the end-to-end argument.
An important aspect about designing new Internet architectures through a clean-slate approach, is the ability to deploy and thoroughly test them, which can take place at an appropriate experimental facility.
Given the fact that IP network layer provides little to no protection against misconfiguration or malicious actions which occur frequently, Researchers proposed network "accountability" in order to establish the foundation for defenses against those behaviors. The proposed work addressed two accountabilities and illustrated that both could be improved by a network layer called Accountable Internet Protocol (AIP)

source accountability: the ability to trace actions to a particular end host and stop that host from misbehaving through AIP addresses which are of the form AD:EID, where AD is the identifier for the network that the host belongs to, and EID is a globally unique host identifier

control-plane accountability: the ability to pinpoint and prevent attacks on routing through origin authentication (ensuring that the network that appears to originate paths is indeed the correct network) and path authentication (checking the integrity of the network path) to detect misleading route advertisements

1.16 Interconnecting Hosts and Networks

We have different types of devices that help to provide connectivity between hosts that are in the same network or help interconnect networks

repeaters and hubs: operate on the physical layer (L1), provide connectivity between hosts that are directly connected (in the same network)

bridges and layer-2 switches: operate on the data link layer (L2) based on MAC addresses, provide connectivity between hosts that are not directly connected, due to finite bandwidth if the arrival rate of the traffic is higher than the capacity of the outputs then packets are temporarily stored in buffers; if the buffer space gets full then this can lead to packet drops

routers and layer-3 switches: opeerate on the network layer (L3)

1.17 Learning Bridges

A bridge is a device with multiple inputs/outputs. A bridge transfers frames from an input to one or multiple outputs though it doesn't need to forward all the frames it receives. A learning bridge learns, populates, and maintains a forwarding table. The bridge consults its forwarding table so that it only forwards frames on specific ports. Whenever the bridge receives any frame it learns which hosts are reachable through which ports.

1.18 Looping Problem in Bridges and the Spanning Tree Algorithm

Using bridges to connect LANs fails if the network topology results in loops (cycles). The solution is to exclude links that lead to loops by running the spanning tree algorithm—represent the topology of the network as a graph, the bridges as nodes, and the links between the bridges as edges. The goal of the spanning tree algorithm is to have the bridges select which links (ports) to use for forwarding so as to eliminate loops.
The bridges eventually select one bridge as the root of the topology. The algorithm runs in rounds. At the very first round of the algorithm every node thinks that it is the root. At every round each node sends to each neighbor node a configuration message with three fields—(1) the sending node's ID (2) the ID of the root as perceived by the sending node (3) the number of hops between that perceived root and the sending node. At every round each node keeps track of the best configuration message that it has received so far, and it compares that against the configuration messages it receives from neighboring nodes at that round. Between two configurations, a node selects one configuration as better if

- the root of the configuration has a smaller ID
- the roots have equal IDs, but one configuration indicates a smaller distance from the root
- if both roots IDs and the distances are the same, then the node breaks the tie by selecting the configuration of the sending node that has with the smallest ID

A node stops sending configuration messages over a link (port) when it learns that all its neighbors are closer to the root.

2 Lesson 2: Transport and Application Layers

2.1 Introduction to Transport Layer and the Relationship between Transport and Network Layer

The transport layer provides an end-to-end connection between two applications running on different hosts regardless of whether they are in the same network. The transport layer on the sending host receives a message from the application layer and appends its own header. We refer to this combined message as a **segment**. We need the transport layer because the network layer does not guarantee the delivery of packets, nor does it guarantee the integrity of the data.
The two most common transport layer protocols are

User Datagram Protocol (UDP): provides basic functionality and relies on the application layer to implement the remaining

Transmission Control Protocol (TCP): provides strong primitives to make end-to-end communication more reliable and cost-effective, because of these primitives TCP is used for most applications today

2.2 Multiplexing: Why Do We Need It?

One of the desired functionalities of the transport layer is the ability for a host to run multiple applications to use the network simultaneously known as **multiplexing**. The network layer only uses the IP address and an IP address alone does not say anything about which processes on the host should get the packets. The transport layer do multiplexing by using ports. Each application binds itself to a unique port number by opening sockets and listening for any data from a remote application.
There are two ways we can use multiplexing: (1) connectionless and (2) connection-oriented.

2.3 Connection Oriented and Connectionless Multiplexing and Demultiplexing

The job of delivering the data included in the transport-layer segment to the appropriate socket as defined in the segment fields is called **demultiplexing**. Similarly, the sending host will need to gather data from different sockets and encapsulate each data chunk with header information (that will later be used in demultiplexing) to create segments, and then forward the segments to the network layer. We refer to this job as **multiplexing**.
The sockets are identified based on special fields in the segment such as the source port number and the destination port number.
We have two flavors of multiplexing/demultiplexing

connectionless: the identifier of a UDP socket is a two-tuple that consists of a destination IP and a destination port, if the destination host receives UDP segments with a specific destination port number, it will forward the segments to the same destination process via the same destination socket even if the segments are coming from different source hosts or different source port numbers

connection-oriented: the identifier for a TCP socket is a four-tuple that consists of the source IP, source port, destination IP, and destination port, a TCP client creates a socket (client socket) and sends a connection request, which is a TCP segment that has a source port number chosen by the client, a destination port number 12000 (welcoming socket), and a special connection establishment bit set in the TCP header, the TCP server creates a socket (connection socket) that is identified by the four-tuple source IP, source port, destination IP, and destination port, the server uses this socket identifier to demultiplex incoming data and forward them to this socket (the three-way handshake)
The client and the server may be using

persistent HTTP sessions: they exchange HTTP messages via the same server socket

non-persistent HTTP sessions: where a new TCP connection and a new socket are created and closed for every response/request

2.4 A Word About the UDP Protocol

UDP offers fewer delays and better control over sending data because with UDP we have

No congestion control or similar mechanisms: TCP "intervenes" with a congestion-control mechanism or retransmission of unacknowledged packets, These TCP mechanisms cause further delays

no connection management overhead: TCP uses a three-way handshake before it begins transferring data

For some real-time applications that are sensitive to delays UDP is a better option. Applications that typically uses UDP include DNS, remote file server RFS, network management SNMP, and routing protocol RIP.
UDP has a 64-bit header consisting of the following fields: (1) source port number (2) destination port number (3) length of the UDP segment (header and data) (4) checksum (for error checking). The UDP sender adds the bits of the source port, the destination port, and the packet length, and application data. It performs a 1's complement on the sum (all 0s are turned to 1 and all 1s are turned to 0s) which is the value of the checksum. The receiver adds all the four 16-bit words (including the checksum). The result should be all 1's unless an error has occurred.

2.5 The TCP Three-Way Handshake

Connection establishment consists of the following steps

step 1: the TCP client sends a special segment (containing no data) with the SYN bit set to 1, the client also generates an initial sequence number (client_isn) and includes it in this special TCP SYN segment

step 2: upon receiving this packet the TCP server allocates the required resources for the connection and sends back the special "connection-granted" segment which we call SYNACK segment, this packet has the SYN bit set to 1, the acknowledgment field of the TCP segment header set to client_isn+1, and a randomly chosen initial sequence number (server_isn) for the server

step 3: when the client receives the SYNACK segment, it also allocates buffer and resources for the connection and sends an acknowledgment with SYN bit set to 0

Connection teardown consists of the following steps

step 1: when the client wants to end the connection, it sends a segment with FIN bit set to 1 to the server

step 2: the server acknowledges that it has received the connection closing request and is now working on closing the connection

step 3: the server then sends a segment with FIN bit set to 1 indicating that connection is closed

step 4: The client sends an ACK for it to the server, it also waits for sometime to resend this acknowledgment in case the first ACK segment is lost

2.6 Reliable Transmission

TCP guarantees in-order delivery of the application-layer data without any loss or corruption. One way to achieve this is by having the receiver send acknowledgments indicating that it has successfully received the specific segment. If the sender does not receive an acknowledgment within a given period of time (**timeout**), the sender can assume the packet is lost and resend it. This method of using acknowledgments and timeouts is also known as **Automatic Repeat Request** or ARQ. The simplest ARQ would be for the sender to send a packet and wait for its acknowledgment from the receiver. This is known as **Stop and Wait ARQ**, for which the algorithm typically needs to figure out the waiting time after which it resends the packet. Typically the timeout value is a function of the estimated round trip time (RTT) of the connection.

To improve the performance of Stop and Wait ARQ, the sender may send at most N unacknowledged packets typically referred to as the window size. As the sender receives an acknowledgment from the receiver, it can send more packets based on the window size. In implementing this (1) each packet is tagged with a unique byte sequence number (2) both sender and receiver would need to buffer more than one packet because the rate of consuming these packets (e.g. writing to a disk) is slower than the rate at which the packets arrive.
The receiver can notify the sender of a missing segment by sending an ACK for the most recently received in-order packet. The sender would then send all packets from the most recently received in-order packet even if some of them had been sent before. The receiver can simply discard any out-of-order received packets. This is called **Go-Back-N**. With this mechanism a single packet error can cause a lot of unnecessary retransmissions. To solve this problem TCP uses **selective ACKing**—the sender retransmits only those packets that it suspects were received in error. Then the receiver would acknowledge a correctly received packet even if it is not in order. In both cases TCP would need to use a timeout to judge the possibility of ACKs getting lost in the network.
In addition to timeout TCP also uses duplicate acknowledgments as a means to detect packet loss. A duplicate ACK is an additional acknowledgment of a segment for which the sender has already received acknowledgment earlier. When the sender receives say 3 duplicate ACKs for a packet, it considers the packet to be lost and will retransmit it instead of waiting for the timeout. This is known as **fast retransmit**.

2.7 Transmission Control

Transmission control is a fundamental function for most applications. Therefore implementing it in the transport layer is easier.

2.8 Flow Control

The first case where we need transmission control is to protect the receiver buffer from overflowing, as TCP uses a buffer at the receiver end to buffer packets that have not been transmitted to the application. TCP provides a rate control mechanism, also known as flow control, that helps match the sender's rate against the receiver's rate of reading the data. the sender maintains a variable receive window rwnd which is calculated as follows

$$\mathsf{rwnd} = \mathsf{RcvBuffer} - [\mathsf{LastByteRcvd} - \mathsf{LastByteRead}]$$

The receiver advertises the value rwnd in every ACK it sends back to the sender. To not overflow the receiver's buffer, the sender must ensure that the maximum number of unacknowledged bytes it sends is no more than the rwnd, i.e.

$$\mathsf{LastByteSent} - \mathsf{LastByteAcked} <= \mathsf{rwnd}$$

TCP makes the sender continue sending segments of size 1 byte even after $\mathsf{rwnd} = 0$. When the receiver acknowledges these segments, it will specify the rwnd value so that the sender will know as soon as the receiver has some room in the buffer.

2.9 What are the Goals of Congestion Control

efficiency: high throughput, high utilization

fairness: each user should have their fair share of the network bandwidth, for this context we assume that every flow under the same bottleneck link should get equal bandwidth

low delay: keep sending the packets to the network will lead to long queues in the network leading to delays

fast convergence: a flow should converge to its fair allocation fast, which is crucial since a typical network's workload is composed of many short flows and few long flows, if the convergence to fair share is not fast enough the network will still be unfair for these short flows

2.10 Congestion control flavors: E2E vs. Network-Assisted

There can be two approaches to implement congestion control

network-assisted congestion control: rely on the network layer to provide explicit feedback to the sender about congestion in the network, e.g. routers could use ICMP packets to notify the source that the network is congested, however even the ICMP packets could be lost under severe congestion rendering the network feedback ineffective

end-to-end congestion control: the hosts infer congestion from the network behavior and adapt the transmission rate

TCP ended up using the end-to-end approach largely aligns with the end-to-end principle adopted in the design of the networks.

2.11 How Does a Host Infer Congestion? Signs of Congestion

There are mainly two signals of congestion

packet delay: as the network becomes congested the queues in the router buffers buildup leading to increased packet delays, thus an increase in the round trip time, which can be estimated based on ACKs, can indicate congestion in the network, however packet delays in a network tend to be variable making delay-based congestion inference quite tricky

packet loss: as the network gets congested routers start dropping packets, although packets can also be lost due to other reasons it rarely happens

The earliest implementation of TCP used packet loss as a signal for congestion. This is mainly because TCP already detected and handled packet losses to provide reliability.

2.12 How Does a TCP Sender Limit the Sending Rate?

TCP congestion control allows each source to determine the network's available capacity and choose how many packets to send without adding to the network's congestion level. TCP uses a congestion window cwnd similar to the receive window rwnd used for flow control. The number of unacknowledged data that a sender can have is the minimum of the congestion window and the receive window

$$\mathsf{LastByteSent} - \mathsf{LastByteAcked} \leq \min\{\mathsf{cwnd}, \mathsf{rwnd}\}$$

TCP uses a **probe-and-adapt** approach in adapting the congestion window—under normal conditions TCP increases the congestion window trying to achieve the available throughput, once it detects congestion the congestion window is decreased.

2.13 Congestion Control at TCP: AIMD

TCP decreases the window when the level of congestion goes up and increases the window when the level of congestion goes down. We refer to this combined mechanism as **additive increase/multiplicative decrease (AIMD)**.

additive increase: the connection starts with a constant initial window, typically 2, every time the sending host successfully sends a cwnd number of packets it adds 1 to cwnd, in practice TCP does not wait for ACKs of all the packets from the previous RTT, instead it increases the congestion window size as soon as each ACK arrives, in bytes this increment is a portion of the MSS (Maximum Segment Size)

$$\text{increment} = \text{MSS} \times (\text{MSS}/\mathsf{cwnd})$$

$$\mathsf{cwnd} = \mathsf{cwnd} + \text{increment}$$

multiplicative decrease: when the TCP sender detects that a loss event has occurred, then it sets cwnd to half of its previous value

If we plot cwnd with respect to time we observe that it follows a sawtooth pattern.
TCP Reno uses two types of loss events as a signal of congestion

the triple duplicate ACKs: considered mild congestion, in which case cwnd is reduced to half the original

timeout: considered a more severe form of congestion, in which case cwnd is reset to the initial window size

"probing" refers to the fact that a TCP sender increases its transmission rate to probe for the rate at which congestion onset begins, backs off from that rate, and then begins probing again to see if the congestion onset rate has changed.

2.14 Slow Start in TCP

The AIMD approach is useful when the sending host is operating very close to the network capacity. AIMD approach reduces the congestion window at a much faster rate than it increases the congestion window. The main reason for this approach is that the consequences of having too large a window are much worse than those of it being too small. In contrast when we have a new connection that starts from a cold start, the sending host can take much longer to increase the congestion window by using AIMD. To handle this TCP Reno has a **slow start phase**, where the congestion window is increased exponentially instead of linearly as in the case of AIMD. The source host starts by setting cwnd to 1 packet. When it receives the ACK for this packet it adds 1 to the current cwnd and sends 2 packets. When it receives the ACK for these two packets it adds 1 to cwnd for each ACK it receives. So it now sends 4 packets. Once the congestion window becomes more than a threshold, often called the **slow start threshold**, it starts using AIMD.
Another scenario where slow start kicks in is when a connection dies while waiting for a timeout to occur. The source will eventually receive a cumulative ACK to reopen the connection. In this case the source will have a fair idea about the congestion window from the last time it had a packet loss, and thus will be able to use this value as slow start threshold.

2.15 TCP Fairness

The goal is to get the throughput achieved for each link to fall somewhere near the intersection of the equal bandwidth share line and the full bandwidth utilization line. Using AIMD leads to fairness in

bandwidth sharing, because the total utilized bandwidth line will shift back and forth so that the total transmission rate is less than or more than the capacity. During the process the bandwidth is always divided equally between the two hosts.

2.16 Caution About Fairness

There can be cases when TCP is not fair

- due to the difference in the RTT of different TCP connections—connections with smaller RTT values would increase their congestion window faster than those with longer RTT values
- due to a single application uses multiple parallel TCP connections

2.17 Congestion Control in Modern Network Environments: TCP CUBIC

TCP Reno has low network utilization especially when the network bandwidth is high or the delay is large. Such networks are also known as high bandwidth delay product networks. TCP CUBIC uses a CUBIC polynomial as the growth function

$$W(t) = C(t-K)^3 + W_{\max}$$

where $W_{\max}$ is the window when the packet loss was detected, C is a scaling constant, and K, the time it takes to increase W to $W_{\max}$ is calculated by

$$K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$$

Note that the time t here is the time elapsed since the last loss event instead of the usual ACK-based timer used in TCP Reno. This also makes TCP CUBIC RTT-fair.

2.18 The TCP Protocol: TCP Throughput

Assume that we have $p =$ the probability loss. So we assume that the network delivers $1/p$ consecutive packets followed by a single packet loss. With AIMD the height of the sawtooth is $W/2$ and the width of the base is $W/2$ thus the area underneath the sawtooth is $(1 + 1/2)(W/2)^2 = (3/8)W^2$, which is the number of packets sent in one cycle and is equal to $1/p$. Solving for W gives the maximum value

$$W = \sqrt{\frac{8}{3p}} = 2\sqrt{\frac{2}{3p}}$$

The bandwidth is given by

$$\text{BW} = \frac{\text{data per cycle}}{\text{time per cycle}} = \frac{\text{MSS} \cdot \frac{3}{8}W^2}{\text{RTT} \cdot \frac{W}{2}} = \frac{\text{MSS}}{\text{RTT}} \cdot \frac{3}{4}W = \frac{\text{MSS}}{\text{RTT}} \cdot \frac{C}{\sqrt{p}}$$

where $C = \sqrt{3/2}$.
In practice, because of additional parameters, such as small receiver windows, extra bandwidth availability, and TCP timeouts, our constant term C is usually less than 1. This means that bandwidth is bounded

$$\text{BW} \leq \frac{\text{MSS}}{\text{RTT}} \cdot \frac{1}{\sqrt{p}}$$

2.19 Datacenter TCP (Optional)

Data center (DC) networks are other networks where new TCP congestion control algorithms have been proposed and implemented. There are mainly two differences that have led to this:

- the flow characteristics of DC networks are different from the public Internet, e.g. there are many short flows that are sensitive to delay, thus the congestion control mechanisms are optimized for delay and throughput not just the latter alone
- a private entity often owns DC networks, this makes changing the transport layer easier since the new algorithms do not need to coexist with the older ones

3 Lesson 3: Intradomain Routing

3.1 Routing Algorithms

Each of the two hosts knows the default router (or first-hop router). We need the algorithms so that when a packet leaves the default router of the sending host, it will travel over a path towards the default router of the destination host. In this context **forwarding** refers to transferring a packet from an incoming link to an outgoing link within a single router. On the other hand **routing** refers to how routers work together using routing protocols to determine the good paths (or good routes as we call them) over which the packets travel from the source to the destination node. When we have routers that belong to the same administrative domain, we refer to the routing as **intradomain routing**. But when routers belong to different administrative domains, we refer to **interdomain routing**. This lecture focuses on intradomain routing algorithms or Interior Gateway Protocols (IGPs). The two major classes of intradomain routing algorithms are **link-state** and **distance-vector** algorithms. We use a graph to understand these algorithms.

3.2 Link-State Routing Algorithm

Also known as Dijkstra's algorithm, in the link-state routing protocol the link costs and the network topology are known to all nodes (for example by broadcasting these values). The basic terminology is

- u = source node
- v = every other node in the network
- $D(v)$ = the cost of current least-cost path from u to v
- $p(v)$ = the previous node along the current least-cost path from u to v
- $c(u, v)$ = the cost from u to directly attached neighbor v
- N' = the subset of nodes along the current least-cost path from u to v

The algorithm consists of two stages—initialization and loop. The pseudocode of initialization is

```
Initialization:
    N' = {u}
    for all nodes v:
        if v is a neighbor of u
            then D(v) = c(u,v)
        else
            D(v) = ∞
```

The pseudocode of loop is

Loop:
    find w not in N' such that $D(w)$ is a minimum
    add w to N'
    update $D(v)$ for each neighbor $v \notin N'$ of w as follows:
        $D(v) = \min(D(v), D(w) + c(w, v))$
until $N' = N$

At each iteration we select the node with the least cost from the previous iteration. The algorithm exits by returning the shortest paths and their costs from the source node u to every other node v in the network.

3.3 Link-State Routing Algorithm: Computational Complexity

In the first iteration we need to search through all n nodes to find the node with the minimum path cost. In the second iteration we search through $(n-1)$ nodes. This decrease continues at every step so by the end of the algorithm we will need to search through $n(n+1)/2$ nodes. Thus the complexity of the algorithm is $O(n^2)$.

3.4 Distance Vector Routing

The DV routing algorithm is

iterative: the algorithm iterates until the neighbors do not have new updates to send to each other

asynchronous: the algorithm does not require the nodes to be synchronized with each other

distributed: direct nodes send information to one another and then they resend their results back after performing their own calculations, so the calculations are not happening in a centralized manner

The DV algorithm is based on the Bellman-Ford Algorithm. Each node maintains its own distance vector with the costs to reach every other node in the network. Each node x updates its own distance vector using the **Bellman-Ford equation**

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\}$$

for each destination node y in the network. A node x computes the least cost to reach destination node y by considering the options that it has to reach y through each of its neighbor v.
The algorithm also consists of two stages—initialization and loop. The pseudocode of initialization is

Initialization:
    for all destinations $y \in N$:
        $D_x(y) = c(x, y)$ if y is a neighbor of x otherwise $D_x(y) = \infty$
    for each neighbor w:
        $D_w(y) = ?$ for all destinations $y \in N$
    for each neighbor w:
        send distance vector $\mathbf{D}_x = [D_x(y) : y \in N]$ to w

The pseudocode of loop is

Loop:

wait until observing a link cost change to neighbor w or receiving $\mathbf{D}_w$ from it

for each $y \in N$:

$D_x(y) = \min_v\{c(x,v) + D_v(y)\}$

if $D_x(y)$ changed for any destination y

send distance vector $\mathbf{D}_x = [D_x(y) : y \in N]$ to all neighbors

3.5 Link Cost Changes and Failures in DV: Count-to-Infinity Problem

The **count-to-infinity problem** refers to the scenario in which a link cost change took a long time to propagate among the nodes of the network, due to the formation of a routing loop resulting in a pair of neighbors keeping updating each other about their new cost to reach a common destination.

3.6 Poison Reverse

A solution to the previous problem is **poison reverse**: since y is on the least-cost path of z reaching x, z will advertise to y that its distance to x is infinity $D_z(x) = \infty$. This discourages y from sending packets to x via z, so z poisons the path from z to y. This technique will solve the problem with 2 nodes. However poisoned reverse will not solve a general count-to-infinity problem involving 3 or more nodes that are not directly connected.

3.7 Distance Vector Routing Protocol Example: RIP

The **Routing Information Protocol (RIP)** is based on the Distance Vector protocol. The first version, released as a part of the BSD version of Unix, uses hop count as a metric (i.e. assumes link cost as 1). The metric for choosing a path could be shortest distance, lowest cost, or a load-balanced path. In RIP routing updates are exchanged between neighbors periodically using RIP response messages. These messages, called RIP advertisements, contain information about sender's distances to destination subnets.

Each router maintains a **routing table**, which contains its own distance vector as well as the router's forwarding table. A routing table has three columns: (1) destination subnet (2) identification of the next router along the shortest path to the destination subnet (3) the number of hops to get to the destination subnet along the shortest path. A routing table will have one row for each subnet in the autonomous system. RIP version 2 allows subnet entries to be aggregated using route aggregation techniques.

If a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable (broken link). In this case the local routing table is modified and changes are propagated. Routers send request and response messages over UDP using port 520, which is layered on top of the network-layer IP protocol. RIP is actually implemented as an application-level process. Some of the challenges with RIP include updating routes, reducing convergence time, and avoiding loops/count-to-infinity problems.

3.8 Link-state Routing Protocol Example: OSPF

Open Shortest Path First (OSPF) is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm. It was introduced as an advancement of the RIP Protocol

operating in upper-tier ISPs. Advances include authentication of messages exchanged between routers, the option to use multiple same-cost paths, and support for hierarchy within a single routing domain.

hierarchy: An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link-state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area.
Exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between different areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well.
For packets routing between two different areas, it is required that the packets be sent through source → an area border router within the source area → the backbone → an area border router within the destination area → destination

operation: First a graph (topological map) of the entire AS is constructed. Then considering itself as the root node, each router computes the shortest-path tree to all subnets by running Djikstra's algorithm locally.
Whenever there is a change in a link's state the router broadcasts routing information to all other routers in the AS, not just to its neighboring routers. It also periodically broadcasts a link's state even if its state hasn't changed.

link state advertisements: Every router within a domain that operates on OSPF uses Link State Advertisements (LSAs). LSA communicates the router's local routing topology to all other local routers in the same OSPF area. In practice LSA is used for building a database (called the link state database) containing all the link states. LSAs are typically flooded to every router in the domain.

the refresh rate for LSAs: OSPF typically has a default refresh period of 30 minutes for LSAs. If a link comes alive before this refresh period is reached, the routers connected to that link ensure LSA flooding. Since the flooding process can happen multiple times, every router receives multiple copies of refreshes or changes. They stores the first received LSA change as new and the subsequent ones as duplicates.

3.9 Processing OSPF Messages in the Router

A simple model of a router consists of a route processor (the main processing unit) and interface cards that receive data packets and forward them via a switching fabric. Router processing consists of the following 7 steps

1. LS update packets which contain LSAs arrive at the router. For every LSA unpacked from the update packet the OSPF protocol checks whether it is a new or duplicate LSA. This is done by referring to the link-state database and checking the sequence number of the LSA for a matching LSA instance in the database.

2. For every new LSA the link-state database is updated, an shortest path first (SPF) calculation is scheduled, and which interface the LSA needs to be flooded out is determined. Since a SPF calculation is a CPU-intensive task, SPF calculations are scheduled and carried out over a period of time (usually when LSA's are changed) so as to offset the CPU costs.

3. Repeat the above step until all the LSAs from the arrived LS update packet have been processed.

4. Bundle the LSAs into an LS update packet and flood out the update packet to the next router.

5. Start the scheduled SPF calculation.

6. End the scheduled SPF calculation.

7. Use the SPF calculation result to update the Forwarding Information Base (FIB). The information in the FIB is used when a data packet arrives at an interface card of the router, where the next hop for the packet is decided and it is forwarded to the outgoing interface card.

3.10 Hot Potato Routing

In large networks, routers rely on both interdomain and intradomain routing protocols to route the traffic. In some cases there are multiple network exits (egress points) that the routers can choose from. Hot potato routing is a technique/practice of choosing a path within the network by choosing the closest egress point based on intradomain or IGP path cost. Hot potato routing simplifies computations for the routers as they are already aware of the IGP path costs. It ensures that the path remains consistent since the next router in the path will choose the same egress point. Hot potato routing also effectively reduces the network's resource consumption by getting the traffic out as soon as possible.

3.11 An Example Traffic Engineering Framework (Optional)

A traffic engineering framework involves three main components

measure: real time view of the network state which includes (1) the operational routers and links and (2) the link capacity and IGP parameters configuration, in addition to the current network state the network operator also requires an estimate of the traffic in the network that can be acquired either by prior history or by measurement

model: predicting the traffic flow through the network based on the IGP configuration, the best path between two routers is selected by calculating the shortest path between them when all the links belong to the same OSPF area, the path selection among routers in different areas is dependent on the summary information passed across the area boundaries

control: new link weights are applied on the affected routers by connecting to the router using telnet or ssh, since the convergence after a weight change involves a transient period in the network changing the link weights is not done frequently

4 Lesson 4: AS Relationships and Interdomain Routing

4.1 Autonomous Systems and Internet Interconnection

Internet is a complex ecosystem built of a network of networks. The basis of this ecosystem includes

Internet Service Providers (ISPs): ISPs can be categorized into three tiers or types: access ISPs (or Tier-3), regional ISPs (or Tier-2), and large global scale ISPs (or Tier-1), some example Tier-1 ISPs include AT&T, NTT, Level-3, and Sprint, in turn regional ISPs connect to Tier-1 ISPs, and smaller access ISPs connect to regional ISPs, smaller ISPs become the customer of a larger ISP, this leads to competition at every level of the hierarchy, but at the same time competing ISPs need to cooperate in providing global connectivity
there are many interconnection options for ISPs: Points of Presence (PoPs, routers which customer network can use to connect), multi-homing (connecting to one or more provider networks), and peering (settlement-free)

Internet Exchange Points (IXPs): interconnection infrastructures which provide the physical infrastructure where multiple networks (e.g. ISPs and CDNs) can interconnect and exchange traffic locally

Content Delivery Networks (CDNs): networks that are created by content providers with the goal of having greater control of how the content is delivered to the end-users while reducing connectivity costs, some examples of CDNs are Google and Netflix

The ecosystem of ISPs forms a hierarchical structure, but the dominant presence of IXPs and CDNs has caused the structure to begin morphing from hierarchical to flat.
An **Autonomous System (AS)** is a group of routers that operate under the same administrative authority. Each AS implements its own policies, makes its own traffic engineering decisions and interconnection strategies. An ISP may operate as a single AS or through multiple ASes. The border routers of the ASes use the **Border Gateway Protocol (BGP)** to exchange routing information with one another, contrary to the Interior Gateway Protocols (IGPs) operate that within an AS.

4.2 AS Business Relationships

provider-customer relationship (or transit): based on a financial settlement that determines how much the customer will pay the provider, the provider forwards the customer's traffic to destinations found in the provider's routing table

peering relationship: two ASes share access to a subset of each other's routing tables, the routes shared between two peers are often restricted to the respective customers of each one, the agreement holds as long as the traffic exchanged between the two peers is not highly asymmetric, peering relationships are formed between Tier-1 ISPs but also between smaller ISP, in the case of Tier-1 ISPs the two peers need to be of similar size, when two small ISPs peer they both save the money they would otherwise pay to their providers by directly forwarding traffic between themselves instead of through their providers

A provider usually charges in one of two ways

- based on a fixed price, given that the bandwidth used is within a predefined range
- based on the bandwidth used, the bandwidth usage is calculated based on periodic measurements (e.g. five-minute intervals), the provider then charges by taking the 95th percentile of the distribution of the measurements

We might observe complex routing policies. In some cases the driving force behind these policies is to increase traffic from a customer to its provider so that the provider gains more revenue.

4.3 BGP Routing Policies: Importing and Exporting Routes

4.3.1 Exporting routes

Deciding which routes to advertise for a destination is a policy decision implemented through route filters. **Route filters** are rules that determine which routes an AS's router should advertise to the routers of neighboring ASes. There are different types of routes that an AS can advertise

routes learned from customers: routes received as advertisements from its customers, since a provider is getting paid to provide reachability to a customer AS, it makes sense that it wants to advertise these customer routes to as many other neighboring ASes as possible

routes learned from providers: routes received as advertisements from its providers, advertising these routes does not make sense since an AS does not have the financial incentive to carry traffic for its provider's routes

routes learned from peers: routes received as advertisements from its peers, lack of financial incentive it does not make sense for an AS to advertise either

4.3.2 Importing routes

Similar to exporting routes an AS receives route advertisements from its customers, providers, and peers. When an AS receives multiple route advertisements towards the same destination from multiple ASes, it needs to rank the routes before selecting which one to import. In descending order of preference the imported routes are the customer routes, then the peer routes, and finally the provider routes. The reasoning behind this ranking is that an AS wants to

- ensure that routes towards its customers not traverse other ASes unnecessarily generating costs
- use routes learned from peers since these are usually free under the peering agreement

4.4 BGP and Design Goals

The design goals of the BGP protocol include

scalability: as the size of the Internet grows, the same is true for the number of ASes and the number of prefixes in the routing tables, one of the design goals of BGP is to manage the complications of this growth while achieving convergence in reasonable timescales and providing loop-free paths

express routing policies: BGP has defined route attributes that allow ASes to implement policies (which routes to import and export) through route filtering and route ranking

allow cooperation among ASes: each individual AS can still make local decisions (which routes to import and export) while keeping these decisions confidential from other ASes

security: originally the design goals for BGP did not include security, there have been several efforts to enhance BGP security ranging from protocols, additional infrastructure, public keys for ASes etc. but these solutions have not been widely deployed or adopted for multiple reasons that include difficulties in transitioning to new protocols and lack of incentives

4.5 BGP Protocol Basics

A pair of routers, known as **BGP peers**, exchange routing information over a semi-permanent TCP port connection called a **BGP session**. In order to begin a BGP session a router will send an OPEN message to another router. Then the sending and receiving routers will send each other announcements (BGP messages) from their routing tables. A BGP session between a pair of routers in two different ASes is called an **external BGP (eBGP)** session, and a BGP session between routers that belong to the same AS is called an **internal BGP (iBGP)** session.
There are two types of BGP messages

UPDATE: contain information about the routes that have changed since the previous update, there are two kinds of updates:

announcements: advertise new routes and updates to existing routes, they include several standardized attributes

withdrawals: inform that a previously announced route is no longer available, the removal could be due to some failure or a change in the routing policy

KEEPALIVE: exchanged between peers to keep a current session going

In the BGP protocol destinations are represented by IP prefixes. Each prefix represents a subnet or a collection of subnets that an AS can reach. Gateway routers running eBGP advertise the IP prefixes they can reach according to the AS's specific export policy to routers in neighboring ASes. Then using separate iBGP sessions the gateway routers disseminate these routes for external destinations to other internal routers according to the AS's import policy.
In addition to the reachable IP prefix field, advertised BGP routes consist of several BGP attributes. Two notable ones are

ASPATH: each AS is identified by its **autonomous system number (ASN)**, as an announcement passes through various ASes their identifiers are included in the ASPATH attribute, this attribute prevents loops and is used to choose between multiple routes to the same destination—the route with the shortest path

NEXT HOP: refers to the next-hop router's IP address (interface) along the path towards the destination, internal routers use this field to store the IP address of the border router, if there is more than one such router and each advertises a path to the same external destination, NEXT HOP allows internal routers to store in the forwarding table the best path according to the policy

4.6 iBGP and eBGP

Both iBGP and eBGP are used to disseminate routes for external destinations. The eBGP speaking routers learn routes to external prefixes and disseminate them to all routers within the AS. The dissemination of routes within the AS is done by establishing a full mesh of iBGP sessions—each eBGP speaking router has an iBGP session with every other BGP router in the AS.
iBGP is not another IGP-like protocol (e.g. RIP or OSPF). IGP-like protocols are used to establish paths between the internal routers of an AS based on specific costs within the AS. In contrast iBGP is only used to disseminate external routes within the AS.

4.7 BGP Decision Process: Selecting Routes at a Router

When a router receives incoming BGP messages, it

1. applies the import policies to exclude routes from further consideration
2. implements the decision process to select the best routes that reflect the policy in place
3. the newly selected routes are installed in the forwarding table
4. the router decides which neighbors to export the route to by applying the export policy

When a router receives multiple route advertisements to the same destination, it goes through the list of attributes in the route advertisements to compare routes. In the simplest scenario where there is no policy in place (meaning it does not matter which route will be imported), the router uses the attribute of the path length to select the route with the fewest number of hops. However this simple scenario rarely occurs in practice.
A router compares a pair of routes by going through the list of attributes. For each attribute it selects the route with the attribute value that will help apply the policy. If for a specific attribute the values are the same, then it goes to the next attribute. Two exemplary attributes are

LocalPref: prefer routes learned through a specific AS over other ASes, it will influence which routers will be selected as exit points for the traffic that leaves the AS (outbound traffic)
as discussed earlier an AS ranks the routes it learned by preferring the routes learned from its customers, then the routes learned from its peers, and finally the routes learned from its providers, an operator can assign a non-overlapping range of values to the LocalPref attribute according to the type of relationship, if B prefers to route its traffic through A due to peering or business, it can assign a higher LocalPref value to routes it learns from A

MED: the MED (Multi-Exit Discriminator) value is used by ASes to designate which links are preferred for inbound traffic, e.g. AS A will be influenced to choose border router R1 instead of R2 to forward traffic to AS B if R1 has a lower MED value (all other attributes being equal)
an AS does not have an economic incentive to export routes that it learns from providers or peers to other providers or peers, an AS can reflect this by tagging routes with a MED value to "staple" the type of business relationship

The attributes are set either locally by the AS (e.g. LocalPref), by the neighboring AS (e.g. MED), or by the protocol (e.g. if a route is learned through eBGP or iBGP).
Note that influencing the route exports will also affect how the traffic enters an AS.

4.8 Challenges with BGP: Scalability and Misconfigurations

The BGP protocol in practice can suffer from two significant limitations: (1) **misconfigurations** and (2) **faults**. A possible misconfiguration or an error can result in an excessively large number of updates, resulting in route instability, router processor and memory overloading, outages, and router failures. One way that ASes can help reduce the risk that these events will happen is by

limiting routing table size: an AS can limit its routing table size using filtering, e.g. long specific prefixes can be filtered to encourage route aggregation, in addition routers can limit the number of prefixes advertised from a single source on a per-session basis, some small ASes also have the option to configure default routes into their forwarding tables, ASes can likewise protect other ASes by using route aggregation and exporting less specific prefixes where possible

limiting the number of route changes: an AS can explicitly limit the propagation of unstable routes by using a mechanism known as **flap damping**—track the number of updates to a specific prefix over a certain amount of time, if the tracked value reaches a configurable value then the AS can suppress that route until a later time
because flap damping can affect reachability, an AS can be strategic about how it uses this technique for certain prefixes, e.g. more specific prefixes could be more aggressively suppressed (lower thresholds), while routes to known destinations that require high availability could be allowed higher thresholds

4.9 Peering at IXPs

ASes can either peer with one another directly or peer at **Internet Exchange Points (IXPs)**, which are infrastructures that facilitate peering and provide more services. IXPs are physical infrastructures that provide the means for ASes to interconnect and directly exchange traffic with one another. The ASes that interconnect at an IXP are called participant ASes. The physical infrastructure of an IXP is usually a network of switches, locally or distributed. Typically the infrastructure has a fully redundant switching fabric that provides fault tolerance.
IXPs become increasingly popular and important because

IXPs are interconnection hubs handling large traffic volumes: for some large IXPs (mostly located in Europe), the daily traffic volume is comparable to that of global Tier 1 ISPs

an important role in mitigating DDoS attacks: IXPs can play the role of a shield to mitigate DDoS attacks based on BGP blackholing and stop DDoS traffic before it hits a participant AS

real-world infrastructures with a plethora of research opportunities: studying IXPs and the traffic that traverses these facilities can help us understand how the Internet landscape is changing, IXPs also provide an excellent research playground for multiple applications such as security applications

IXPs are active marketplaces and technology innovation hubs: IXPs provide an expanding list of services that go beyond interconnection most notably DDoS mitigation and SDN-based services, as a result IXPs is evolving from interconnection hubs to technology innovation hubs

For an AS to peer at an IXP it must have a public Autonomous System Number (ASN). Each participant brings a router to the IXP facility and connects one of its ports to the IXP switch. The router of each participant must be able to run BGP since the exchange of routes across the IXP is via BGP only. In addition each participant must agree to the IXP's General Terms and Conditions (GTC). Each network incurs a one-time cost to establish a circuit from the premises to the IXP. Then there is a monthly charge for using a chosen IXP port. The entity that owns and operates the IXP might also charge an annual membership fee. Nevertheless exchanging traffic over an established public peering link at an IXP is in principle settlement-free as IXPs typically do not charge for exchanged traffic volume.
Networks choose to peer at IXPs because they

keep local traffic local: traffic exchanged between two networks do not need to travel unnecessarily through other networks if both networks are participants in the same IXP facility

lower costs: typically peering at an IXP is offered at a lower cost than relying on third parties to transfer the traffic

network performance: improved due to reduced delay

incentive: critical players in today's Internet ecosystem often incentivize other networks to connect at IXPs, e.g. a prominent content provider may require another network to be present at a specific IXP or IXPs in order to peer with them

Services offered at IXPs include

public peering: the most well-known use of IXPs is public peering service, the IXPs do not usually charge based on the amount of exchanged volume, even with the setup costs, IXPs are generally cheaper than other conventional methods of exchanging traffic, IXP participants also often experience better network performance and Quality-of-Service (QoS) because of reduced delays and routing efficiencies

private peering: doesn't use the IXP's public peering infrastructure, commonly used when the participants want a well-provisioned dedicated link capable of handling high-volume

route servers and service level agreements: many IXPs also include service level agreements (SLAs) and free use of the IXP's route servers, this allows participants to arrange instant peering using essentially a single agreement/BGP session

remote peering through resellers: third parties resell IXP ports wherever they have infrastructure connected to the IXP, these third parties can offer the IXP's service remotely, which will enable networks that have little traffic also to use the IXP, this also enables remote peering

mobile peering: for the interconnection of mobile GPRS/3G networks

DDoS blackholing: a few IXPs support customer-triggered blackholing, which allows users to alleviate the effects of DDoS attacks against their network

free value-added services: e.g. Internet Routing Registry (IRR), consumer broadband speed tests, DNS root name servers, country-code top-level domain (ccTLD) nameservers, as well as distribution of the official local time through NTP

4.10 Peering at IXPs: How Does a Route Server Work?

Generally two ASes exchange traffic through the switching fabric utilize a two-way BGP session called a **bilateral BGP session**. However this option does not scale with many participants. To mitigate this some IXPs operate a **route server** which facilitates and manages how multiple ASes can talk on the control plane simultaneously thus establishes a muli-lateral BGP peering session. It does the following

- collects and shares routing information from its peers or participants of the IXP that connect to the RS
- executes its own BGP decision process and re-advertises the resulting information (e.g. best route selection) to all RS's peer routers

In a modern route server architecture, a typical routing daemon maintains a **Routing Information Base (RIB)** which contains all BGP paths that it receives from its peers—the Master RIB. In addition the route server also maintains AS-specific RIBs to keep track of the individual BGP sessions they maintain with each participant AS.
Route servers maintain two types of route filters

import filters: applied to ensure that each member AS only advertises routes that it should advertise

export filters: typically triggered by IXP members themselves to restrict the set of other IXP member ASes that receive their routes

The typical steps AS X and AS Z exchange routes through a multi-lateral peering session are

1. AS X advertises a prefix $p1$ to the RS, which is added to the route server's RIB specific to AS X
2. the route server uses the peer-specific import filter to check whether AS X is allowed to advertise p1, and if true it adds the prefix $p1$ is added to the Master RIB
3. the route server applies the peer-specific export filter to check if AS X allows AS Z to receive $p1$, and if true it adds the prefix $p1$ to the AS Z-specific RIB
4. the route server advertises $p1$ to AS Z with AS X as the next hop

4.11 Remote Peering (Optional)

Remote peering (RP) is peering at the peering point without the necessary physical presence. The remote peering provider is an entity that sells access to IXPs through their own infrastructure. The primary method of identifying remote peering is to measure the round-trip time (RTT) between a vantage point (VP) inside the IXP and the IXP peering interface of a member. However this method fails to account for the changing landscape of IXPs today and even misinfers latencies of remote members as local and local members as being remote. Instead a combination of methods can achieve detection of remote peering in a more tractable way, some of which include

information about the port capacity: IXPs offer ASes connectivity to ports with capacity typically between 1 and 100 Gbit/s, but resellers usually offer connectivity through their virtual ports with smaller capacities and lower prices

gathering colocation information: an AS needs to be physically present (i.e. actually deploy routing equipment) in at least one colocation facility where the IXP has deployed switching equipment, it should be easy to locate the colocation facilities where both AS and IXPs are colocated although this information is imperfect in practice

multi-IXP router inference: an AS can operate a multi-IXP router which is a router connected to multiple IXPs to reduce operational costs, suppose we infer the AS as local or remote to one of these IXPs from a previous step, then we can extend the inference to the rest of the involved IXPs based on whether they share colocation facilities or not

private connectivity with multiple existing AS participants: if an AS has private peers over the same router that connects it to an IXP, and the private peers are physically colocated to the same IXP facilities, it can be inferred that the AS is also local to the IXP

4.12 BGP Configuration Verification (Optional)

Control of BGP configuration is complex and easily misconfigured where route propagation happens in a full mesh or via "route reflectors". In addition configuration languages vary among routing manufacturers and may not be well-designed. Adding to the complexity is the distributed nature of BGP's implementation. Two main aspects of persistent routing defining BGP correctness are

path visibility: route destinations are correctly propagated through the available links in the network

route validity: the traffic meant for a given destination reaches it

The router configuration checker or rcc is a tool researchers propose that detects BGP configuration faults. rcc uses static analysis to check for correctness before running the configuration on an operational network before deployment. It can be used to analyze the configuration of live systems and potentially detect live faults. While analyzing real-world configurations it was found that most Path Visibility Faults were the result of

- problems with full mesh and route reflector configurations in iBGP settings leading to signaling partitions
- route reflector cluster problems
- incomplete iBGP sessions where an iBGP session is active on one router but not the other

while Route Validity Faults were determined to stem from filtering and dissemination problems. These issues suggest that routing might be less prone to faults if there were improvements to iBGP protocols when it comes to making updates and scaling.

5 Lesson 5: Router Design and Algorithms (Part 1)

5.1 Introduction

When a packet arrives at the input link, the router's job is to look at the destination IP address of the packet and determine the output link by consulting the forwarding table.

5.2 What's Inside a Router?

The main job of a router is to implement the forwarding plane functions and the control plane functions.

5.2.1 Forwarding/switching function

Forwarding is the router's action to transfer a packet from an input link interface to the appropriate output link interface. Forwarding occurs at very short timescales (typically a few nanoseconds) and is typically implemented in hardware.
The main components of a router are

input ports: perform the following functionalities

1. physically terminate the incoming links to the router
2. the data link processing unit decapsulates the packets
3. perform the lookup function—consult the forwarding table to ensure that each packet is forwarded to the appropriate output port through the switch fabric

switching fabric: makes the connections between the input and the output ports and moves packets from input to output ports, there are three types of switching fabrics: memory, bus, and crossbar

output ports: perform the following functionalities

1. receive and queue the packets from the switching fabric
2. the data link processing unit encapsulates the packets
3. physically terminate the outgoing links from the router

5.2.2 Control function

By **control** function we refer to implementing routing protocols, maintaining the routing tables, and computing the forwarding table. All these functions are implemented in software in the routing processor, or as we will see in the SDN chapter, these functions could be implemented by a remote controller.

5.3 Router Architecture

Often input links and output links are put together. When a packet arrives at an input link, the most time-sensitive tasks are

lookup: the router looks at the destination IP address and determines the output link by looking up the forwarding table or Forwarding Information Base (FIB), the FIB provides a mapping between destination prefixes and output links, routers use the **longest prefix matching** algorithms to resolve any disambiguities, some routers offer a more specific and complex type of lookup called **packet classification**, where the lookup is based on destination or source IP address, port, and other criteria

switching: after lookup the switching system takes over to transfer the packet from the input link to the output link, modern fast routers use crossbar switches for this task, although scheduling the switch (matching available inputs with outputs) is difficult because multiple inputs may want to send packets to the same output

scheduling: after the packet has been switched to a specific output, it will be queued if the link is congested, the queue may be as simple as First-In-First-Out (FIFO) or more complex weighted fair queuing to provide delay guarantees or fair bandwidth allocation

Some less time-sensitive tasks that take place in the router are

header validation and checksum: the router checks the packet's version number, decrements the time-to-live (TTL) field, and recalculates the header checksum

route processing: routers build their forwarding tables using routing protocols such as RIP, OSPF, and BGP, these protocols are implemented in the routing processors

protocol processing: routers need to implement the following protocols to implement their functions

Simple Network Management Protocol (SNMP): for remote inspection

TCP and UDP: for remote communication with the router

Internet Control Message Protocol (ICMP): for sending error messages e.g. when time-to-live (TTL) time is exceeded

5.4 Different Types of Switching

switching via memory: Input/Output ports operate as I/O devices in an operating system controlled by the routing processor. When an input port receives a packet it sends an interrupt to the routing processor, and the packet is copied to the processor's memory. Then the processor extracts the destination address and looks into the forward table to find the output port. Finally the packet is copied into that output's port buffer.

switching via bus: The routing processor does not intervene as we saw the switching via memory. When an input port receives a new packet it puts an internal header that designates the output port and sends the packet to the shared bus. All output ports will receive the packet but only the designated one will keep it. When the packet arrives at the designated output port the internal header is removed from the packet. Only one packet can cross the bus at a given time thus the speed of the bus limits the speed of the router.

switching via interconnection network: A crossbar switch is an interconnection network that connects N input ports to N output ports using $2N$ buses. Horizontal buses meet the vertical buses at crosspoints controlled by the switching fabric. Crossbar network can carry multiple packets at the same time as long as they are using different input and output ports.

5.5 The Challenges Routers Face

The fundamental problems that a router faces revolve around

bandwidth and Internet population scaling: caused by

- increasing number of devices that connect to the Internet
- increasing volumes of network traffic due to new applications
- new technologies such as optical links that can accommodate higher volumes of traffic

services at high speeds: new applications require services such as protection against delays in the presence of congestion and protection during attacks or failures, but offering these services at high speed is a challenge for routers

Some exemplary bottlenecks are

longest prefix matching: The increasing number of Internet hosts and networks has made it impossible for routers to have explicit entries for all possible destinations. So routers group destinations into prefixes instead. But then routers run into the problem of more complex algorithms for efficient longest prefix matching.

service differentiation: Routers can also offer different quality-of-service (or security guarantees) to different packets. In turn this requires routers to classify packets based on more complex criteria beyond destination.

switching limitations: A way to deal with high-speed traffic is to use parallelism by crossbar switching. But at high speeds this comes with its problems and limitations such as head-of-line blocking.

bottlenecks about services: Providing performance guarantees (quality-of-service) at high speeds is nontrivial, as is providing support for new services such as measurements or security guarantees.

5.6 Prefix-Match Lookups

One way to help with the scalability problem is to group multiple IP addresses with the same prefix. There are different ways to denote prefix

dot decimal: for the 16-bit prefix 132.234: binary form of the first octet is 10000100, binary form of the second octet is 11101010, thus the binary prefix of 132.234 is 1000010011101010∗

slash notation: of the format A/L where A = Address and L = Length, e.g. 132.234.0.0/16 where 16 denotes that only the first 16 bits are relevant for prefixing

masking: we can use a mask instead of the prefix length, e.g. the prefix 132.234.0.0/16 is written as 132.234.0.0 with a mask 255.255.0.0, the mask 255.255.0.0 denotes that only the first 16 bits are important

In the earlier days of the Internet, we used an IP addressing model based on **classes** (fixed-length prefixes). With the rapid exhaustion of IP addresses, in 1993, the Classless Internet Domain Routing (CIDR) came into effect. CIDR essentially assigns IP addresses using arbitrary-length prefixes. CIDR has helped to decrease the router table size, but at the same time it introduced us to a new problem—longest-matching prefix lookup.
There are various challenges that the router needs to overcome when performing a lookup to determine the output port. These challenges revolve around lookup speed, memory, and update time. Measurement studies on network traffic had shown

- a large number (∼250,000) of concurrent flows of short duration, this already large number has only been increasing, and as a consequence caching solutions will not work efficiently
- a large part of the cost of computation for lookup is accessing memory
- an unstable routing protocol may adversely impact the update time, inefficient routing protocols increase this value up to additional milliseconds
- a vital trade-off is memory usage—we can use expensive fast memory or cheaper but slower memory

5.7 Unibit Tries

One of the simplest techniques for prefix lookup is the unibit trie. Every node has a 0 or 1 pointer. Starting with the root, 0-pointer points to a subtrie for all prefixes that begin with 0, and 1-pointer points to a subtrie for all prefixes that start with 1. Moving forward this way we construct more subtries by allocating the remaining bits of the prefix. The steps we follow to perform a prefix match are

1. begin the search for a longest prefix match by tracing the trie path
2. continue the search until we fail (no match or an empty pointer)
3. when our search fails, the last known successful prefix traced in the path is our match and our returned value

Two final notes on the unibit trie

- If a prefix is a substring of another prefix, the smaller string is stored in the path to the longer (more specific prefix), e.g. P4 = $1*$ is a substring of P2 = $111*$ thus P4 is stored inside a node towards the path to P2.
- There may be nodes that only contain one pointer e.g. P3 = 11001. After matching 110 we will be expecting to match 01. But in our prefix database we don't have any prefix that share more than the first 3 bits with P3. The nodes with only one pointer each are called **one-way branches**. For efficiency we compress these one-way branches to a single text string with 2 bits (P9).

5.8 Multibit Tries

While a unibit trie is very efficient and offers advantages such as fast lookup and easier updates, its most significant problem is the number of memory accesses required to perform a lookup (32 memory accesses for 32-bit addresses). Instead, we can implement lookups using a **stride**. The stride is the number of bits that we check at each step. So an alternative to unibit tries is multibit tries. A multibit trie is a trie where each node has 2^k children, where k is the stride. We have two flavors of multibit tries: fixed-length stride tries and variable-length stride tries.

5.9 Prefix Expansion

Consider a prefix such as $101*$ (length 3) and a stride length of 2 bits. If we search in 2-bit lengths we will miss out on prefixes like $101*$. To combat this we use a strategy called **controlled prefix expansion**, where we expand a given prefix to more prefixes. We ensure that the expanded prefix is a multiple of the chosen stride length. At the same time we remove all lengths that are not multiples of the chosen stride length. We end up with a new database of prefixes, which may be larger (in terms of the actual number of prefixes) but with fewer lengths. So the expansion gives us more speed with an increased cost of the database size.
When we expand our prefixes there may be a collision, i.e. when an expanded prefix collides with an existing prefix. In that case that expanded prefix gets dropped.

5.10 Multibit tries: Fixed-Stride

Every element in a trie represents two pieces of information: a pointer and a prefix value. The prefix search moves ahead with the preset length in n-bits (3 in this case). Because $n > 1$ the root node of the trie may contain more than one prefix. When the path is traced by a pointer we remember the last matched prefix (if any). Our search ends when an empty pointer is met. At that time we return the last matched prefix as our final prefix match.

5.11 Multibit Tries: Variable Stride

With this scheme we can examine a different number of bits every time, e.g. P3 has 5 bits in total thus the nodes left to the 3-bit root only have 2 bits. Every node can have a different number of bits to be explored. By varying the strides we could make our prefix database smaller and optimize for memory (least memory accesses). An optimum variable stride is selected by using dynamic programming.

6 Lesson 6: Router Design and Algorithms Part 2

6.1 Why Do We Need Packet Classification?

As the Internet becomes increasingly complex, packet forwarding based on the longest prefix matching of destination IP addresses is insufficient. We need to handle packets based on multiple criteria such as TCP flags, source addresses, and so on. We refer to this finer packet handling as **packet classification**. Some examples include:

firewalls: routers implement firewalls at the entry and exit points of the network to filter out unwanted traffic or to enforce other security policies

resource reservation protocols: e.g. DiffServ has been used to reserve bandwidth between a source and a destination

traffic type: routing based on the specific type of traffic helps avoid delays for time-sensitive applications

6.2 Packet Classification: Simple Solutions

linear search: firewall implementations perform a linear search of the rules database and keep track of the best-match rule, reasonable for a few rules but the time to search through thousands of rules can be prohibitive

caching: cache the results so that future searches can run faster, the problem is that we will need to perform searches for missed hits, and even with a high 90% hit rate cache a slow linear search of the rule space will perform poorly

passing labels: Multiprotocol Label Switching (MPLS) and DiffServ use this technology, in MPLS

1. a label-switched path is set up between sites A and B
2. before traffic leaves site A, a router does packet classification and maps the web traffic into an MPLS header
3. the intermediate routers between A and B apply the label without having to redo packet classification

DiffServ follows a similar approach, applying packet classification at the edges to mark packets for special quality-of-service.

6.3 Fast Searching Using Set-Pruning Tries

Suppose we want to classify packets using both the source and the destination IP address. The simplest way to approach the problem would be to build a trie on the destination prefixes in the database, and then for every leaf-node at the destination trie to "hang" source tries. In other words for every

destination prefix D in the destination trie, we "prune" the set of source rules to those compatible with D. We first match the destination IP address in a packet in the destination trie. Then we traverse the corresponding source trie to find the longest prefix match for the source IP. The algorithm concludes with the least-cost rule. The problem that we need to solve now is which source prefixes to store at the sources tries. It turns out that this problem is memory explosion, because a source prefix can occur in multiple destination tries.

6.4 Reducing Memory Using Backtracking

The set pruning approach has a high cost in memory to reduce time. The opposite approach is to pay in time to reduce memory. The backtracking approach has each destination prefix D point to a source trie that stores the rules whose destination field is exactly D. The search algorithm then performs a "backtracking" search on the source tries associated with all ancestors of D. Since each rule is stored exactly once, the memory requirements are lower than the previous scheme.

6.5 Grid of Tries

The set-pruning approach has a high cost in memory. Because we construct a trie on the destination prefixes and then for every destination prefix we have a trie on the source prefixes. On the other hand, the backtracking approach has a high cost in terms of time. Because we first traverse the destination trie to find the longest prefix match, and then we need to work our way backward to the destination trie to search for the source trie associated with every prefix of our previous match. With the grid of tries approach we can reduce the wasted time in the backtracking search by using precomputation. When there is a failure point in a source trie, we precompute a switch pointer. Switch pointers take us directly to the next possible source trie containing a matching rule. The precomputed switch pointers allow us to avoid backtracking to find an ancestor node and then traversing the source trie, skipping source tries with source fields that are shorter than our current source match.

6.6 Scheduling and Head-of-Line Blocking

Assume that we have an N-by-N crossbar switch with N input lines, N output lines, and N^2 crosspoints. We want to maximize the number of input/output links pairs that communicate in parallel for better performance. A simple scheduling algorithm is the **take-the-ticket** algorithm. Each output line maintains a distributed queue for all input lines that want to send packets to it. When an input line intends to send a packet to a specific output line it requests a ticket. Then the input line waits for the ticket to be served. At that point the input line connects to the output line, the crosspoint is turned on, and the input line sends the packet.
In the example while A sends its packet in the first iteration, the entire queue for B and C is waiting. We refer to this problem as **head-of-line (HOL) blocking**, because the entire queue is blocked by the progress of the head of the queue.

6.7 Avoiding Head-of-Line Blocking

6.7.1 Avoiding head-of-line blocking via output queuing

A practical implementation of this approach is the **knockout scheme**. It relies on breaking up packets into fixed sizes (cell). Suppose that the same output rarely receives N cells and the expected number is k (smaller than N), then we can have the fabric running k times as fast as an input link instead of N. To accommodate the scenarios where the expected case is violated, we have one or more of a primitive switching element that randomly picks the chosen output e.g.

$k = 1$ and $N = 2$: randomly pick the output that is chosen, the switching element in this case is called a *concentrator*

$k = 1$ and $N > 2$: one output is chosen out of N possible outputs, we can use the same strategy of multiple 2-by-2 concentrators in this case

arbitrary k and N: create k knockout trees to calculate the first k winners

The drawback with this approach is that is it is complex to implement.

6.7.2 Avoiding head-of-line blocking by using parallel iterative matching

The main idea is that we can still allow queuing for input lines but in a way that avoids the head-of-line blocking. The algorithm runs in three rounds

request phase: all inputs send requests in parallel to all outputs they want to connect with

grant phase: the outputs that receive multiple requests pick a random input

accept phase: inputs that receive multiple grants randomly pick an output to send to

This is more efficient than the take-a-ticket.

6.8 Scheduling Introduction

Busy routers rely on scheduling to handle routing updates, management queries, and data packets. For example scheduling enables routers to allow certain types of data packets to get different services from other types. Scheduling is done in real-time.

FIFO with tail drop: simplest method of router scheduling, packets placed on an output port is a FIFO (first-in, first-out) queue, if the output link buffer is completely full then incoming packets to the tail of the queue are dropped, this results in fast scheduling decisions but a potential loss in important data packets

need for quality-of-service (QoS): there are other methods of packet scheduling such as priority, round-robin, etc. to provide quality-of-service (QoS) guarantees to a flow of packets on measures such as delay and bandwidth, a **flow of packets** refers to a stream of packets that travels the same route from source to destination and requires the same level of service at each intermediate router and gateway, in addition flows must be identifiable using fields in the packet headers

The reasons to make scheduling decisions more complex than FIFO with tail drop are

router support for congestion: while most traffic is based on TCP (which has its own ways to handle congestion), additional router support can improve the throughput of sources by helping handle congestion

fair sharing of links among competing flows: during periods of backup these packets tend to flood the buffers at an output link, which would freeze clients' connection if we use FIFO with tail drop

providing QoS guarantees to flows: one way to enable fair sharing is to guarantee certain bandwidths to a flow, another way is to guarantee certain delay through a router for a flow which is noticeably important for (e.g. video flows)

6.9 Deficit Round Robin

We consider round-robin to avoid tail drop and introduce fairness in servicing different flows. However if we alternate between packets from different flows, the difference in packet sizes could result in some flows getting serviced more frequently. To avoid this researchers came up with **bit-by-bit round robin**. Since it's impossible to split up the packets in the real world, we consider an imaginary bit-by-bit system to calculate the packet-finishing time and send a packet as a whole.
Let $R(t)$ be the current round number at time t. If the router can send μ bits per second and the number of active flows is N, the rate of increase in round number is given by

$$\frac{dR(t)}{dt} = \frac{\mu}{N}$$

The rate of increase in round number is inversely proportional to the number of active flows. An important takeaway is that the number of rounds required to transmit a packet does not depend on the number of backlogged queues.
Let a packet of size $p(i)$ bits arrive as the ith packet in the flow. If it arrives at an empty queue it reaches the head of the queue at the current round $R(t)$. If not it reaches the head after the packet in front of it finishes it. Combining both the scenarios the round number at which the packet reaches the head is given by

$$S(i) = \max\{R(t), F(i-1)\}$$

where $F(i-1)$ is the round at which the packet ahead of it finishes. The round number at which a packet finishes, which depends only on the size of the packet, is given by

$$F(i) = S(i) + p(i)$$

Using the above two equations, the finish round of every packet in a queue can be calculated.

6.9.1 Packet-level Fair Queuing

This strategy emulates the bit-by-bit fair queueing by sending the packet with the smallest finishing round number. At any round the packet chosen to be sent out is garnered from the previous round of the algorithm. The packet which had been starved the most while sending out the previous packet from any queue is chosen. Although this method provides fairness, it also introduces new complexities—we will need to keep track of the finishing time at which the head packet of each queue would depart and choose the earliest one. This requires a priority queue implementation which has a time complexity that is logarithmic in the number of flows. Additionally if a new queue becomes active all timestamps may have to change—an operation with time complexity linear in the number of flows. Thus the time complexity of this method makes it hard to implement at gigabit speeds.

6.9.2 Deficit Round Robin (DRR)

Although the bit-by-bit round-robin gave us bandwidth and delay guarantees, the time complexity was too high. We can use a simple constant-time round-robin algorithm with a modification to ensure fairness. We assign a quantum size Q_i and a deficit counter D_i to each flow. The quantum size determines the share of bandwidth allocated to that flow. For each turn of round-robin the algorithm will serve as many packets in the flow i with size less than $(Q_i + D_i)$. Initially the deficit counters for all flows are set to 0. If packets remain in the queue it will store the remaining bandwidth in D_i for the next run; if all packets in the queue are serviced in that turn it will clear D_i to 0 for the next turn.

6.10 Traffic Scheduling: Token Bucket

There are scenarios where we want to set bandwidth guarantees for flows in the same queue without separating them. The technique of **token bucket shaping** can limit the burstiness of a flow by (1) limiting the average rate and (2) limiting the maximum burst size. The bucket shaping technique assumes a bucket per flow that fills with tokens with a rate of R per second, and it also can have up to B tokens at any given time. If the bucket is full with B tokens additional tokens are dropped. When a packet arrives it can go through if there are enough tokens (equal to the size of the packet in bits). If not the packet needs to wait until enough tokens are in the bucket. Given the max size of B a burst is limited to B bits per second.
In practice the bucket shaping idea is implemented using a counter (can't go more than max value B and gets decremented when a bit arrives) and a timer (to increment the counter at a rate R). The problem with this technique is that we have one queue per flow, because a flow may have a full token bucket, whereas other flows may have an empty token bucket and therefore will need to wait.
We use a modified version of the token bucket shaper to maintain one queue called **token bucket policing**. Here if a packet arrives and there are no tokens in the bucket, it is dropped.

6.11 Traffic Scheduling: Leaky Bucket

Traffic policing and traffic shaping are both mechanisms to limit the output rate of a link. The output rate is both controlled by identifying traffic descriptor violations. However they respond to them in two different ways

policer: when the traffic rate reaches the maximum configured rate, excess traffic is dropped, the output rate appears as a saw-toothed wave

shaper: a shaper typically retains excess packets in a queue or a buffer which is scheduled for later transmission, the result is that excess traffic is delayed instead of dropped, thus the flow is shaped or smoothed when the data rate is higher than the configured rate

Traffic shaping and policing can work in tandem.
Leaky Bucket is an algorithm that can be used in both traffic policing and traffic shaping. It is analogous to water flowing into a leaky bucket with the water leaking at a constant rate. The bucket, say with capacity B, represents a buffer that holds packets, and the water corresponds to incoming packets. The leak rate r is the rate at which packets are allowed to enter the network, which is constant irrespective of the rate at which packets arrive. If an arriving packet does not cause an overflow when added to the bucket it is said to be *conforming*. Otherwise it is said to be non-conforming. Packets classified as conforming are added to the bucket while non-conforming packets are discarded. Hence if the bucket is full any new packet that arrives to the bucket is dropped. Irrespective of the input rate of packets the output rate is constant, which leads to uniform distribution of packets sent to the network. This algorithm can be implemented as a single server queue.

7 Lesson 7: SDN Part 1

7.1 What Led Us to SDN

Computer networks are difficult to manage for two main reasons

diversity of equipment on the network: the network has to handle different software adhering to different protocols for each of these equipment, which is true even with a network management tool offering a central point of access

proprietary technologies for the equipment: equipment like routers and switches tend to run software which is closed and proprietary thus configuration interfaces vary between vendors

Software Defined Networking (SDN) offers new ways to redesign networks to make them more manageable. It employs a simple idea—separation of tasks. Similarly SDN divides the network into two planes—control plane and data plane.

7.2 A Brief History of SDN: The Milestones

The history of SDN can be divided into three phases

7.2.1 Active networks

This phase took place from the mid-1990s to the early 2000s, when the internet took off resulting in an increase in the applications and appeal of the internet. The required standardization of new protocols by the IETF (Internet Engineering Task Force) was a slow and frustrating process, which led to the growth of active networks aiming at opening up network control. Active networking envisioned a programming interface (a network API) that exposed resources/network nodes and supported customization of functionalities for subsets of packets passing through the network nodes. This was the opposite of the popular belief in the internet community that the simplicity of the network core was important to the internet success.
There were two types of programming models in active networking

capsule model: code is carried in-band in data packets, caching is also used to make code distribution more efficient

programmable router/switch model: established by out-of-band mechanisms, made decision making a job for the network operator

Although the capsule model was most closely related to active networking, both have some effect on the current state of SDNs.
The technology pushes that encouraged active networking were

reduction in computation cost: enabled putting more processing into the network

advancement in programming languages: Java, VM

advances in rapid code compilation and formal methods:

funding from agencies with long-term vision: promoted interoperability among projects

The use pulls for active networking were

- network service provider's frustration concerning the long timeline to develop and deploy new network services
- third party's interest to add value by implementing control in a more individualistic manner, dynamically meeting the needs of specific applications or network conditions
- researchers' interest in having a network that would support large-scale experimentation
- unified control over individually managed middleboxes, which foreshadows the current trends in network functions virtualization

The use pulls for active networks in the mid-1990s are similar to those for SDN now.
Active networks made three major contributions related to SDN

programmable functions in the network to lower the barrier to innovation: active networks were one of the first to introduce the idea of using programmable networks to overcome the slow speed of innovation in computer networking, while many early visions for SDN concentrated on increasing programmability of the control-plane, active networks focused on the programmability of the data-plane, in addition the concept of isolating experimental traffic from normal traffic had emerged from active networking and is heavily used in OpenFlow and other SDN technologies

network virtualization: active networks produced a framework that described a platform capable of demultiplexing different programming models based on packet headers, this was the need that led to network virtualization

the vision of a unified architecture for middlebox orchestration: the last use pull, unified control over middleboxes, was never fully realized in the era of active networking, while it did not directly influence network function virtualization (NFV), some lessons from its research is useful while trying to implement unified architecture now

The downfalls for active networking were

- it was too ambitious, requiring end users to write Java code
- it was more involved in redesigning the architecture of networks, not as much emphasis was given to performance and security which users were more concerned about

7.2.2 Control and data plane separation

This phase lasted from around 2001 to 2007, when there was a steady increase in traffic volumes and thus network reliability, predictability, and performance became more important. They identified the challenge in network management lay in the way existing routers and switches tightly integrated the control and data planes. Once this was identified efforts to separate the two began.
The technology pushes that encouraged control and data plane separation were

- higher link speeds in backbone networks led vendors to implement packet forwarding directly in the hardware, thus separating it from the control-plane software
- Internet Service Providers (ISPs) found it hard to meet the increasing demands for greater reliability and new services (such as VPNs), and struggled to manage the increased size and scope of their networks
- servers had substantially more memory and processing resources such that a single server could store all routing states and compute all routing decisions for a large ISP network, this also enabled simple backup replication strategies thus ensuring controller reliability
- open-source routing software lowered the barrier to creating prototype implementations of centralized routing controllers

These pushes inspired two main innovations: (1) open interface between control and data planes (2) logically centralized control of the network
This phase was different from active networking in the following ways

- it focused on spurring innovation by and for network administrators rather than end users and researchers

- it emphasized programmability in the control domain rather than the data domain
- it worked towards network-wide visibility and control rather than device-level configurations

Some use pulls for the separation of control and data planes were

- selecting between network paths based on the current traffic load
- minimizing disruptions during planned routing changes
- redirecting/dropping suspected attack traffic
- allowing customer networks more control over traffic flow
- offering value-added services for VPN customers

Most work during this phase tried to manage routing within a single ISP, but there were some proposals about ways to enable flexible route control across many administrative domains.
The attempt to separate the control and data planes resulted in a couple of concepts which were used in further SDN design: (1) logically centralized control using an open interface to the data plane (2) distributed state management.
Initially many people thought separating the control and data planes was a bad idea. However this concept of separation of planes helped researchers think clearly about distributed state management. Several projects exploring clean-slate architecture commenced and laid the foundation for OpenFlow API.

7.2.3 OpenFlow API and network operating systems

This phase took place from around 2007 to 2010. OpenFlow was born out of the interest in the idea of network experimentation at a scale. OpenFlow built on the existing hardware and enabled more functions than earlier route controllers. Although this dependency on hardware limited its flexibility it enabled immediate deployment.
The basic working of an OpenFlow switch is as follows. Each switch contains a table of packet-handling rules. Each rule has a pattern, list of actions, set of counters and a priority. When an OpenFlow switch receives a packet, it determines the highest priority matching rule, performs the action associated with it, and increments the counter.
The main technology push that encouraged OpenFlow was that unlike its predecessors OpenFlow was adopted in the industry. This could be due to the fact that before OpenFlow switch chipset vendors had already started to allow programmers to control some forwarding behaviors. This allowed more companies to build switches without having to design and fabricate their own data plane. On the other hand early OpenFlow versions built on technology that the switches already supported. This meant that enabling OpenFlow initially was as simple as performing a firmware upgrade.
The use pulls for OpenFlow were

- OpenFlow met the need of conducting large scale experimentation on network architectures—in the late 2000s OpenFlow were deployed across many college campuses to show its capability
- OpenFlow was useful in data-center networks because there was a need to manage network traffic at large scales
- companies started investing more in programmers to write control programs and less in proprietary switches that could not support new features easily, which allowed many smaller players to become competitive in the market by supporting capabilities like OpenFlow

Some key effects that OpenFlow had were

- generalizing network devices and functions
- the vision of a network operating system
- distributed state management techniques

7.3 Why Separate the Data Plane from the Control Plane

The reason is two-fold

independent evolution and development: in the traditional approach routers are responsible for both routing and forwarding functionalities, this meant that a change to either of the functions would require an upgrade of hardware, in this new approach routers only focus on forwarding, thus innovation in this design can proceed independently of other routing considerations, similarly improvement in routing algorithms can take place without affecting any of the existing routers

control from high-level software program: in SDN we use software to compute the forwarding tables, thus we can easily use higher-order programs to control routers' behavior, the decoupling of functions makes debugging and checking the behavior of the network easier

In addition this separation leads to opportunities in

data centers: SDN helps to make management of such large network easier

routing: the interdomain routing protocol used today, BGP, constrains routes, and it is hard to make routing decisions using multiple criteria, SDN makes it easier to update the router's state and can provide more control over path selection

enterprise networks: SDN can improve the security applications for enterprise networks, e.g. protecting DDoS by dropping the attack traffic at strategic locations of the network

research networks: SDN allows research networks to coexist with production networks

7.4 Control Plane and Data Plane Separation

Two important functions of the network layer are

forwarding: a router looks at the header of an incoming packet and consults the forwarding table to determine the outgoing link to send the packet to, it could also entail blocking a packet from exiting the router if it is suspected to have been sent by a malicious router and duplicating the packet to send it along multiple output links

routing: determine the path from the sender to the receiver across the network based on some routing algorithms

SDN differs from traditional networks in that

traditional approach: routing algorithms (control plane) and forwarding function (data plane) are closely coupled, the router runs routing algorithms to construct its forwarding table which is consulted by its forwarding function

SDN approach: there is a remote controller that computes and distributes the forwarding tables to be used by every router, this controller is physically separated from the router, so that the routers are solely responsible for forwarding and the remote controllers are solely responsible for computing and distributing the forwarding tables, since the controller is implemented in software we say the network is software-defined

7.5 The SDN Architecture

The main components of an SDN network are

SDN-controlled network elements: sometimes called the infrastructure layer, responsible for the forwarding of traffic in a network based on the rules computed by the SDN control plane

SDN controller: logically centralized entity that acts as an interface between the network elements and the network-control applications

network-control applications: programs that manage the underlying network by collecting information about the network elements with the help of SDN controller

The four defining features in an SDN architecture

flow-based forwarding: the rules for forwarding packets in SDN-controlled switches can be computed based on any number of header field values in various layers such as the transport layer, network layer and link layer (e.g. OpenFlow allows up to 11 header field values to be considered), this differs from the traditional approach where only the destination IP address determines the forwarding of a packet

separation of data plane and control plane: SDN-controlled switches operate on the data plane and they only execute the rules in the flow tables

network control functions: the SDN control plane consists of two components

- **controller:** maintains up-to-date network state information about the network devices and elements (e.g. hosts, switches, links) and provides it to network-control applications
- **network applications:** used the network state information provided by controller to monitor and control the network devices

a programmable network: example applications can include network management, traffic engineering, security, automation, analytics etc.

7.6 The SDN Controller Architecture

An SDN controller can be broadly split into three layers

communication layer: communicating between the controller and the network elements, it consists of a protocol through which the SDN controller and the network controlled elements communicate, using this protocol devices send locally observed events to the SDN controller providing it with a current view of the network state, e.g. a new device joining, heartbeat indicating the device is up etc. the communication between the SDN controller and the controlled devices is known as the "southbound" interface, OpenFlow is an example of southbound API

network-wide state-management layer: stores information of network state, network state includes any information about the state of the hosts, links, switches and other controlled elements in the network, it also includes copies of the flow tables of the switches, network state information is needed by the SDN control plane to configure the flow tables

interface to the network-control application layer: communicating between controller and applications, also known as the controller's "northbound" interface, through which the SDN controller interacts with network-control applications, the SDN controller can notify applications of changes in the network state based on the event notifications sent by the SDN-controlled devices, and then network-control applications can read/write network state and flow tables in controller's state-management layer, a REST interface is an example of northbound API

The SDN controller, although viewed as a monolithic service by external devices and applications, is implemented by distributed servers to achieve fault tolerance, high availability, and efficiency.

7.7 OpenDayLight Architecture Overview (Optional)

The OpenDaylight controller architecture consists of

southbound interface: used to communicate with network devices and support third-party vendor specific protocols

northbound interface: through which SDN applications can talk to the controller platform

model driven service abstraction layer (MD-SAL): the abstraction layer provided by OpenStack for developers to add new features to the controller Using karaf, it consists of 2 components

shared datastore: there are two tree-based structures

config datastore: manages the representation of the network. sanity check of any new services, manages how these changes are pushed

operation datastore: has the true representation of the network state based on data from the managed network elements

message bus: provides a way for various services and protocol drivers to notify and communicate with each other

8 Lesson 8: SDN Part 2

8.1 Revisiting the Motivation for SDN

As IP networks grew in adoption worldwide, there were a few challenges that became more and more pronounced

handling the ever growing complexity and dynamic nature of networks: the implementation of network policies required changes right down to each individual network device, which were often carried out by vendor-specific commands and required manual configurations, traditional IP networks are quite far away from achieving automatic response mechanisms to dynamic network environment changes

tightly coupled architecture: traditional IP networks consist of a control plane (handles network traffic) and a data plane (forwards traffic based on the control plane's decisions) that are contained inside networking devices thus are not flexible to work on, as a result any new protocol update takes as long as 10 years because changes need to percolate down to every networking device

As an attempt to overcome limitations of the legacy IP networking paradigm, software defined networking (SDN) starts by separating out the control logic (in the control plane) from the data plane, which is achieved by using a programming interface between the SDN controller and the switches—the SDN controller controls the data plane elements via the API. An example of such an API is OpenFlow. An OpenFlow switch can be instructed by the controller to behave like a firewall, switch, router, or even perform other roles like load balancer, traffic shaper etc.
Traditionally viewed computer networks have three planes of functionality, which are all abstract logical concepts

data plane: functions and processes that forward data in the form of packets or frames

control plane: functions and processes that determine which path to use by using protocols to populate forwarding tables of data plane elements

management plane: services that monitor and configure the control functionality e.g. SNMP-based tools

A network policy is defined in the management plane, enforced in the control plane, and executed in the data plane. As opposed to traditional IP networks, SDN principles allow for a separation of concerns introduced between the definition of networking policies, their implementation in hardware, and the forwarding of traffic.

8.2 SDN Advantages

Traditional networks come with a tightly coupled data and control plane thereby making the networking components physically embedded. As a result to add a new networking feature one has to go through the process of modifying all control plane devices. To avoid this traditionally a new specialized equipment was introduced (known as middlebox) through which concepts and features such as load balancers, intrusion detection systems, firewalls etc. were introduced. Since these middleboxes are required to be carefully placed in the network topology, it is much harder to later change or reconfigure them.
In contrast since SDN decouples the control plane from physical networking devices, it isolates itself as an external entity (SDN controller). With this middlebox services can be viewed as SDN controller applications. This approach has several advantages:

shared abstractions: middlebox services (or network functionalities) can be programmed easily since the abstractions provided by the control platform and network programming languages can be shared

consistency of same network information: all network applications have the same global network information view, leading to consistent policy decisions while reusing control plane modules

locality of functionality placement: previously the location of middleboxes was a strategic decision and big constraint, however in this model middlebox applications can take actions from anywhere in the network

simpler integration: integrations of networking applications are smoother, e.g. load balancing and routing applications can be combined sequentially

8.3 The SDN Landscape

8.3.1 Data plane

infrastructure: similar to traditional networks an SDN infrastructure consists of networking equipments (routers, switches, and other middlebox hardware), the difference is that these physical networking equipments are merely forwarding elements, any logic to operate them is directed from the centralized control system, popular examples of such infrastructure equipment include OpenFlow (software) switches

southbound interface: connecting bridges between control and forwarding elements, and since they sit in between control and data plane they play a crucial role in separating control and data plane functionality, the most popular implementation of Southbound APIs for SDNs is OpenFlow

8.3.2 Control plane

network virtualization: for a complete virtualization of the network the network infrastructure needs to provide support for arbitrary network topologies and addressing schemes, existing virtualization constructs are able to provide full network virtualization, however these technologies are connected in a box-by-box basis configuration and there is no unifying abstraction that can be leveraged to configure these in a global manner, new advancements in SDN network virtualization are promising

network operating system: the promise of SDN is to ease network management and solve networking problems by using a logically centralized controller by way of a network operating system (NOS), the value of a NOS is in providing abstractions, essential services, and common APIs to developers, such systems propel more innovation by reducing inherent complexity of creating new network protocols and network applications, some popular NOSs are OpenDayLight

northbound interface: a standard for northbound interface is still an open problem as are its use cases, what is relatively clear is that northbound interfaces are supposed to be a mostly software ecosystem as opposed to the southbound interfaces, another key requirement is the abstraction that guarantees programming language and controller independence

8.3.3 Management plane

language-based virtualization: an important characteristic of virtualization is the ability to express modularity and allow different levels of abstraction, this takes the complexity away from application developers without compromising on security which is inherently guaranteed

network programming languages: network programmability can be achieved using either low-level or high-level programming languages

- **low-level languages:** difficult to write modular code reuse it, which generally leads to more error-prone development
- **high-level languages:** provide abstractions, make development more modular and code more reusable in control plane, do away with device specific and low-level configurations, which generally allows faster development

network applications: functionalities that implement the control plane logic and translate to commands in the data plane, SDNs can be deployed on traditional networks and home area networks, data centers, IXPs etc. due to this there is a wide variety of network applications such as routing, load balancing, security enforcement, end-to-end QoS enforcement, power consumption reduction, network virtualization, mobility management etc.

8.4 SDN Infrastructure Layer

The SDN infrastructure composes of networking equipment (routers, switches, and appliance hardware) performing simple forwarding tasks. An important difference in these networks is that they are built on top of open and standard interfaces which ensure configuration and communication compatibility and interoperability among different control plane and data plane devices as opposed to traditional networks that use proprietary and closed interfaces. In the SDN architecture a data plane device is a hardware or software entity that forwards packets, while a controller is a software stack running on commodity hardware. A model derived from OpenFlow is currently the most widely accepted design of SDN data plane devices. It is based on a pipeline of flow tables where each entry of a flow table has three parts: (1) a matching rule (2) actions to be executed on matching packets (3) counters that keep

statistics of matching packets.
In an OpenFlow device, when a packet arrives, the lookup process starts in the first table and ends either with a match in one of the tables of the pipeline or with a miss (when no rule is found for that packet). Some possible actions for the packet include

- forward the packet to outgoing port
- encapsulate the packet and forward it to controller
- drop the packet
- send the packet to normal processing pipeline
- send the packet to next flow table

8.5 SDN Southbound Interfaces

The southbound interfaces or APIs are the separating medium between the control plane and data plane functionality, and traditionally represent one of the major barriers for the introduction and acceptance of any new networking technology. API proposals like OpenFlow have received good reception because they promote interoperability and deployment of vendor-agnostic devices.
Currently OpenFlow is the most widely accepted southbound standard for SDNs. There are three information sources sent by OpenFlow-enabled forwarding devices to controller

event-based message: where there is a link or port change

flow statistics: generated by forwarding devices

packet message: when forwarding devices do not know what to do with a new incoming flow

These three channels are key to providing flow-level info to the Network Operating System (NOS).
Besides SDN there are others API proposals, e.g.

ForCES: provides a more flexible approach to traditional network management without changing the current architecture of the network

OVSDB: allows the control elements to create multiple vSwitch instances, set QoS policies on interfaces, and attach interfaces to the switches

8.6 SDN Controllers: Centralized vs Distributed

The biggest drawback of traditional networks is that they are configured using low-level, device-specific instruction sets and run mostly proprietary network operating systems. This challenges the notion of device-agnostic developments and abstraction. SDN offers these by means of a logically centralized control. Functions such as topology, statistics, notifications, device management, along with shortest path forwarding and security mechanisms are what we consider the essential functionality all controllers should provide. For example high priority services' rules should always take precedence over rules created by applications with low priority.
SDN controllers can be categorized into

centralized controller: a single entity that manages all forwarding devices in the network, which is a single point of failure and may have scaling issues, some enterprise class networks and data centers use such architectures, they use multi-threaded designs to explore parallelism of multi-core computer architectures

distributed controller: can be scaled to meet the requirements of potentially any environment, distribution can occur in two ways—a centralized cluster of nodes or physically distributed set of elements, typically a cloud provider uses a hybrid approach to distribute—clusters of controllers inside each data center and distributed controller nodes in different sites, properties of distributed controllers include weak consistency semantics and fault tolerance

8.7 An Example Controller: ONOS

ONOS (Open Networking Operating System) is a distributed SDN control platform. It aims to provide a global view of the network to applications, scale-out performance, and fault tolerance. The prototype was built based on Floodlight, an open-source single-instance SDN controller. Owing to the distributed architecture of ONOS there are several ONOS instances running in a cluster. The management and sharing of network state across these instances is achieved by maintaining a global network view. Titan, a graph database and, Cassandra, a distributed key-value store are used to implement the view. The applications interact with the network view using the Blueprints graph API.
The distributed architecture of ONOS offers

scale-out performance: each ONOS instance serves as the master OpenFlow controller for a subset of switches, the propagation of state changes between a switch and the network view is handled solely by the master instance of that switch, workload can be distributed by adding more instances to the ONOS cluster

fault tolerance: ONOS redistributes the work of a failed instance to other remaining instances, each switch in the network connects to multiple ONOS instances with only one instance acting as its master, Upon failure of an ONOS instance, an election is held among the remaining instances on a consensus basis to choose a master for each of the switches that were controlled by the failed instance

Zoopkeeper is used to maintain the mastership between the switch and the controller.

8.8 Programming the Data Plane: The Motivation

P4 (Programming Protocol-independent Packet Processors) is a high-level programming language to configure switches which works in conjunction with SDN control protocols. The popular vendor-agnostic OpenFlow interface started with a simple rule table to match packets based on a dozen header fields. This specification has grown over the years to include multiple stages of the rule tables with increasing number of header fields to allow better exposure of a switch's functionalities to the controller. Thus to manage the demand for increasing number of header fields, a need arises for an extensible flexible approach to parse packets and match header fields while also exposing an open interface to the controllers to leverage these capabilities. P4 is used to configure the switch programmatically and acts as a general interface between the switches and the controller with its main aim of allowing the controller to define how the switches operate.
The following are the primary goals of P4

reconfigurability: the way parsing and processing of packets takes place in the switches should be modifiable by the controller

protocol independence: to make the switches independent of any particular protocol the controller defines a packet parser and a set of tables mapping matches and their actions, the packet parser extracts the header fields which are then passed on to the match+action tables to be processed

target independence: the packet processing programs should be programmed independent of the underlying target devices, these generalized programs written in P4 should be converted into target-dependent programs by a compiler which are then used to configure the switch

8.9 Programming the Data Plane: P4's Forwarding Model

The switches using P4 use a programmable parser and a set of match+action tables to forward packets. The tables can be accessed in multiple stages in a series or parallel manner. This contrasts with OpenFlow which supports only fixed parsers based on predetermined header fields and only a series combination of match+actions tables. The P4 model allows generalization of packet processing across various forwarding devices such as routers, load balancers etc. using multiple technologies such as fixed function switches, NPUs etc. This generalization allows the design of a common language to write packet processing programs that are independent of the underlying devices.
The two main operations of the P4 forwarding model are

configure: used to program the parser, specify the header fields to be processed in each match+action stage and also define the order of these stages

populate: the entries in the match+action tables specified during configuration may be altered using the populate operations, it allows addition and deletion of the entries in the tables

In short configuration determines the packet processing and the supported protocols in a switch (configuration of a switch), whereas population decides the policies to be applied to the packets (processing of a packet).

8.10 An introduction To The P4 Programming Language (Optional)

P4 is a packet processing language with the following characteristics

legal header types: declared to let the parser be aware of the possible packet formats, e.g. the format of an IPv4 header may be specified with the list of headers allowed

control flow program: which uses the declared header types and a set of actions to specify how the headers are processed, e.g. checksums may be computed

table dependency graph (TDG): used to identify the dependencies between header fields and help determine the order in which the tables can be executed, tables with no dependencies may be executed in parallel, each node in the TDG maps to a match+action table with the control flow between tables acting as the edges between the nodes, the analysis of dependencies between the nodes determine where a table may be placed in the pipeline, however these graphs are not readily available to the programmer, the compiler then maps the TDG to a specific target switch

8.11 SDN Applications: Overview

8.11.1 Traffic engineering

This is one of the major areas of interest for SDN applications with main focus on optimizing the traffic flow so as to minimize power consumption, judiciously use network resources, perform load balancing etc.

minimize power consumption: ElasticTree is one such application which identifies and shut downs specific links and devices depending on the traffic load

load balancing: Plug-n-Serve and Aster*x achieve scalability by creating rules based on wildcard patterns which enables handling of large numbers of requests from a particular group

Another use case of SDN applications is to automate the management of router configuration to reduce the growth in routing tables due to duplication of data. Large scale service providers also use SDN for traffic optimization to scale dynamically.

8.11.2 Mobility and wireless

SDN-based wireless networks offer a variety of features including on-demand virtual access points (VAPs), usage of spectrum dynamically, sharing of wireless infrastructure etc.

- OpenRadio, the OpenFlow for wireless, enables decoupling of wireless protocols from the underlying hardware by providing an abstraction layer
- Light Virtual Access Points (LVAPs) offer an improved way of managing wireless networks by using a one-to-one mapping between LVAPs and clients, so that users can move between APs without any visible lag in contrast to traditional wireless networks

8.11.3 Measurement and monitoring

The first class of applications in this domain aims to add features to other networking services such as new functions for measurement systems. A second class of these applications aim to improve the existing features of SDNs using OpenFlow such as reducing the load on the control plane arising from collection of data plane statistics using various sampling and estimation techniques.

8.11.4 Security and dependability

The applications in this area focus on improving the security of networks. One approach of using SDN to enhance security is to impose security policies on the entry point to the network e.g. DDoS detection. Another approach is to use programmable devices to enforce security policies on a wider network e.g. IP address random mutation. With regards to improving the security of SDN itself, there have been simple approaches like rule prioritization for applications.

8.11.5 Data center networking

Data center networking can be revolutionized by the use of SDN through

- detecting anomalous behavior by defining different models and building application signatures from observed information, e.g. FlowDiff
- performing dynamic reconfigurations of virtual networks involved in a live virtual network migration, e.g. LIME

8.12 SDN Application Example: A Software Defined Internet Exchange

The two main limitations of BGP are

routing only on destination IP prefix: there is no flexibility to customize rules for example based on the traffic application or the source/destination network

networks have little control over end-to-end paths: networks can only select paths advertised by direct neighbors, networks cannot directly control preferred paths but instead have to rely on indirect mechanisms such as AS Path prepending

SDN can perform multiple actions on the traffic by matching over various header fields, not only destination prefix. In the context of the IXPs researchers have proposed an SDN based architecture called SDX, which was proposed to implement multiple applications including

application specific peering: custom peering rules can be installed for certain applications such as high-bandwidth video applications

traffic engineering: controlling inbound traffic based on source IP or port numbers by setting forwarding rules

traffic load balancing: destination IP address can be rewritten based on any field in the packet header to balance the load

traffic redirection through middleboxes: targeted subsets of traffic can be redirected to middleboxes

In a traditional IXP the participant ASes connect their BGP-speaking border router to a shared layer-2 network and a BGP route server. The layer-2 network is used for forwarding packets (data plane) while the BGP route server is used for exchanging routing information (control plane). In the SDX architecture each AS has the illusion of its own virtual SDN switch that connects its border router to every other participant AS. Each AS can define forwarding policies as if it is the only participant at the SDX without influencing how other participants forwarding packets on their own virtual switches. Each AS can have its own SDN applications for dropping, modifying, or forwarding their traffic. The policies can also be different based on the direction of the traffic (inbound or outbound). The SDX is responsible for combining the policies from multiple participants into a single policy for the physical switch. To write policies the SDX uses the Pyretic language to match header fields of the packets and express actions on the packets (the sequential operator "≫" for forwarding to the next function and the parallel operator "+" for combining outputs).

8.13 SDN Applications: Wide Area Traffic Delivery

Applications of SDX in the domain of wide area traffic delivery include

8.13.1 Application specific peering

ISPs prefer dedicated ASes to handle the high volume of traffic flowing from high bandwidth applications such as Netflix. This can be achieved by identifying a particular application's traffic using packet classifiers and directing the traffic to a different path. However this involves configuring additional and appropriate rules in the edge routers of the ISP. This overhead can be eliminated by configuring custom rules for flows matching a certain criteria at the SDX.

8.13.2 Inbound traffic engineering

An SDN enabled switch can be installed with forwarding rules based on the source IP address and source port of the packets, thereby enabling an AS to control how the traffic enters its network, in contrast with BGP which performs routing based solely on the destination address of a packet.

8.13.3 Wide-area server load balancing

The existing approach of load balancing across multiple servers of a service involves a client's local DNS server issuing a request to the service's DNS server. As a response the service DNS returns the IP address of a server such that it balances the load in its system. This involves DNS caching which

can lead to slower responses in case of a failure. In SDX the destination IP addresses of packets can be modified at the exchange point to the desired backend server based on the request load thus is more efficient.

8.13.4 Redirection through middleboxes

The placement of middleboxes (firewalls, load balancers etc.) are usually targeted at important junctions such as the boundary of the enterprise networks with their upstream ISPs. To avoid the high expenses involved in placing middleboxes at every location in case of geographically large ISPs, the traffic is directed through a fixed set of middleboxes by the ISPs. This is done by manipulating routing protocols such as internal BGP to essentially hijack a subset of traffic and sending it to a middlebox. This approach could result in unnecessary additional traffic being redirected, and is also limited by the fixed set of middleboxes. To overcome these issues an SDX can identify and redirect the desired traffic through a sequence of middleboxes.

9 Lesson 9: Internet Security

9.1 Internet Security

The Internet was not designed or built with security in mind.

9.2 Properties of Secure Communication

There are certain properties that are important to ensure the communication is secure

confidentiality: an intruder can eavesdrop on the communication by sniffing or recording the exchanged messages, a countermeasure is to encrypt the message

integrity: an intruder can attack by modification, insertion, or deletion of part of the messages send, a countermeasure is check for the integrity

authentication: an intruder can steal information by impersonating another entity on the network, a countermeasure is authentication mechanism

availability: an intruder can render the system unavailable by denial of service attacks

9.3 DNS Abuse

Attackers have developed techniques of abusing the DNS protocol so as to extend the uptime of domains that are used for malicious purposes. The ultimate goal is to remain undetectable for longer.

9.3.1 Round Robin DNS (RRDNS)

This method is used by large websites to distribute the load of incoming requests to several servers at a single physical location. It responds to a DNS request with a list of DNS A records which it then cycles through in a round robin manner. The DNS client can then choose a record using different strategies. Each A record also has a *Time to Live* (TTL) for this mapping which specifies the number of seconds the response is valid. If the lookup is repeated while the mapping is still active, the DNS client will receive the same set of records albeit in a different order.

9.3.2 DNS-based content delivery

Content Distribution Networks (CDNs) also use DNS-based techniques to distribute content but using more complex strategies. For example CDNs not only distribute the load amongst multiple servers at a single location but also distribute these servers across the world. When accessing the name of the service using DNS, the CDN computes the "nearest edge server" and returns its IP address to the DNS client. This results in the content being moved "closer" to the DNS client which increases responsiveness and availability. CDNs can react quickly to changes in link characteristics as their TTL is lower than that in RRDNS.

9.3.3 Fast-Flux Service Networks (FFSN)

Using RRDNS a DNS request receives multiple A records (each containing a different IP address), this makes it harder to shut down online scams. Similarly spreading across several servers makes the shut-down of these scams more complex. Fast-Flux Service Networks (FFSN) is an extension of the ideas behind RRDNS and CDN. As its name suggests it is based on a "rapid" change in DNS answers with a TTL lower than that of RRDNS and CDN. One key difference between FFSN and the other methods is that after the TTL expires, it returns a different set of A records from a larger set of compromised machines.
The mothership of the domain found in a spam email is the control node where the actual content of the scam is being hosted. In the content retrieval process for content hosted in a FFSN, the DNS lookup from the client returns several different IP addresses, all belonging to compromised machines in the network (flux agents). Each time the TTL expires, the lookup returns completely different IP addresses. The flux agent then relays the request it receives (HTTP GET) to the control node which sends content to the flux agent which is then delivered to the DNS client. These flux agents are usually located in different countries and belong to different Autonomous Systems (AS).
An important aspect of Internet abuse is the infrastructure that attackers use to support the abuse. For example, the attackers need Internet infrastructure to support illegal content hosting, C&C infrastructure hosting etc.

9.4 How to Infer Network Reputation: Evidence of Abuse

FIRE or FInding Rogue nEtworks is a system that monitors the Internet for **rogue networks**, which are networks whose main purpose is malicious activity such as phishing, hosting spam pages, hosting pirated software etc. It uses three data sources to identify hosts that likely belong to rogue networks:

botnet command and control providers: several botnets still rely on centralized command and control (C&C), thus a bot-master would prefer to host their C&C on networks where it is unlikely to be taken down, the two main types of botnets this system considers are (1) IRC-based botnets and (2) HTTP-based botnets

drive-by-download hosting providers: drive-by-download is a method of malware installation without interaction with the user, it commonly occurs when the victim visits a web page that contains an exploit for their vulnerable browser

phish housing providers: phishing pages usually mimic authentic sites to steal login credentials, credit card numbers, and other personal information, these pages are hosted on compromised servers and usually are up only for a short period of time

Each of these data sources produces a list of malicious IP addresses daily. FIRE combines the information from these three lists to identify rogue AS (organizations are considered equivalent to autonomous systems). The approach is to identify the most malicious networks as those that have the highest ratio

of malicious IP addresses as compared to the total owned IP addresses of that AS.
The key difference between rogue and legitimate networks is the longevity of malicious behavior. Legitimate networks are usually able to remove malicious content within a few days, whereas rogue networks may let the content be up for weeks to more than a year. By disregarding IP addresses that have been active for a short time, we ignore phishing attacks hosted on legitimate networks and web servers that were temporarily abused for botnet communication.

9.5 How to Infer Network Reputation: Interconnection Patterns

With data plane monitoring only when a network has a large enough concentration of blacklisted IPs it will be flagged as malicious. But in practice this approach has several disadvantages:

- it is not feasible to monitor the traffic of all networks to detect malicious behaviors from the data plane
- it may take a long time until a very large fraction of IPs makes it into a blacklist
- it does not differentiate well between networks that are legitimate but abused, and those which are likely operated by cyberactors

As a complementary approach ASwatch uses information exclusively from the control plane (i.e. routing behavior) to identify malicious networks. Also, this approach aims to detect malicious networks that are likely run by cyberactors, or bulletproof as they are called, rather than networks that may be badly abused. The approach is based on the observation that bulletproof ASes have distinct interconnection patterns and overall different control plane behavior from most legitimate networks. These networks are found to

- change upstream providers more aggressively than most legitimate networks
- engage in customer-provider or peering relationships with likely shady networks instead of connecting directly with legitimate networks

These behaviors help the bulletproof network to remain unnoticeable for longer, and when complaints may start the bulletproof network can simply change to another upstream provider.
The design of ASwatch is based on monitoring global BGP routing activity to learn the control plane behavior of a network. The system has two phases:

training phase: the system is given a list of known malicious and legitimate ASes, it then tracks the their business relationships with other ASes and their BGP updates/withdrawals patterns over time, ASwatch then computes statistical features of each AS
there are three main families of features

rewiring activity: based on changes in the AS connecting activities, frequent changes in customers/providers, connecting with less popular providers etc. is usually suspicious behavior

IP space fragmentation and churn: malicious ASes are likely to use small BGP prefixes to partition their IP address spaces, and advertise only a small section of these to avoid all of them being taken down at once if detected

BGP routing dynamics: the BGP announcements and withdrawals for malicious ASes follow different patterns from legitimate ones e.g. periodically announcing prefixes for short periods of time

The system then uses supervised learning to capture the known behaviors and patterns with a trained model.

operational phase: given an unknown AS it calculates the features for this AS and then uses the trained model to assign a reputation score to the AS, if the system assigns the AS a low reputation score for several days in a row indicating consistent suspicious behavior, it then identifies the AS as malicious

9.6 How to Infer Network Reputation: Likelihood of Breach

We look at a system that uses features to train a Random Forest to predict the likelihood of a security breach within an organization. There are three classes of features used for this model:

mismanagement symptoms: If there are misconfigurations in an organization's network, it indicates that there may not be policies in place to prevent such attacks, or may not have the technological capability to detect these failures. This increases the likelihood of breach. The features used are:

open recursive resolvers: misconfigured open DNS resolvers

DNS source port randomization: many servers still do not implement this

BGP misconfiguration: short-lived routes can cause unnecessary updates to the global routing table

untrusted HTTPS certificates: can detect the validity of a certificate by TLS handshake

open SMTP mail relays: servers should filter messages so that only those in the same domain can send mails/messages

malicious activities: Another factor to consider is the level of malicious activities that are seen to originate from the organization's network and infrastructure. We can determine this using spam traps, darknet monitors, DNS monitors etc. We create a reputation blacklist of the IP addresses that are involved in some malicious activities. There are three such types of malicious activities:

spam activity: e.g. CBL, SBL, SpamCop

phishing and malware activities: e.g. PhishTank, SURBL

scanning activity: e.g. Dshield, OpenBL

security incident reports: Data based on actual security incidents gives us the ground truth on which to train our machine learning model on. The system uses three collections of such reports to ensure a wider coverage area:

VERIS community database: this is a public effort to collect cyber security incidents in a common format, it is maintained by the Verizon RISK team and contains more than 5000 incident reports

Hackmageddon: this is an independently maintained blog that aggregates security incidents on a monthly basis

the web hacking incidents database: this is an actively maintained repository for cyber security incidents

This system uses a Random Forest (RF) classifier and compares it to a baseline provided by a Support Vector Machine (SVM). It uses 258 features—the features described above are divided into features based on the time span for which they are valid, secondary features based on statistics from the other features, and the size of the organization. These inputs are processed, then fed to a RF which produces a risk probability (a float). By thresholding this value we obtain the binary class label. Since this data is sequential, the training-testing splits of the data are strictly based on the time of each datapoint. The best combination of parameters gives this model an accuracy of 90%.

9.7 Traffic Attraction Attacks: BGP Hijacking

BGP hijacking attacks can be classified into the following groups:

classification by affected prefix: primarily concerned with the IP prefixes that are advertised by BGP, there are different ways the prefix can be targeted such as:

- **exact prefix hijacking:** when two different ASes (one is genuine and the other one is counterfeit) announce a path for the same prefix, this disrupts routing in such a way that traffic is routed towards the hijacker wherever its AS-path route is shorter
- **sub-prefix hijacking:** an extension of exact prefix hijacking except that the hijacking AS works with a sub-prefix of the genuine prefix of the real AS, this exploits the characteristic of BGP to favor more specific prefixes thus route large/entire amount of traffic to the hijacking AS
- **squatting:** the hijacking AS announces a prefix that has not yet been announced by the owner AS

classification by AS-path announcement: an illegitimate AS announces the AS-path for a prefix for which it doesn't have ownership rights, there are different ways this can be achieved:

- **type-0 hijacking:** simply an AS announcing a prefix not owned by itself
- **type-N hijacking:** the counterfeit AS announces an illegitimate path for a prefix that it does not own to create a fake link (path) between different ASes, where N denotes the position of the rightmost fake link in the illegitimate announcement, e.g. {AS2, ASy, AS1 – 10.0.0.0/23} is a type-2 hijacking
- **type-U hijacking:** the hijacking AS does not modify the AS-path but may change the prefix

classification by data-plane traffic manipulation: the intention of the attacker is to hijack the network traffic and manipulate the redirected network traffic on its way to the receiving AS, there are three ways the attack can be realized:

- **dropped:** so that it never reaches the intended destination, this attack falls under the category of **blackholing (BH) attack**
- **eavesdropped or manipulated:** before it reaches the receiving AS, which is also called **man-in-the-middle (MM) attack**
- **impersonated:** the network traffic of the victim AS is impersonated and the response to this network traffic is sent back to the sender, this attack is called **imposture (IM) attack**

9.8 Traffic Attraction Attacks: Motivations

The attacks can be classified as caused by:

human error: accidental routing misconfiguration due to manual errors, this can lead to large-scale type-0 hijacking

targeted attack: the hijacking AS usually intercepts network traffic (MM attack) while operating in stealth mode to remain under the radar on the control plane (Type-N and Type-U attacks)

high impact attack: the attacker is obvious in their intent to cause widespread disruption of services (e.g. Pakistan Telecom's type-0 sub-prefix hijacking blackholing YouTube's services)

9.9 Defending Against BGP Hijacking: An Example Detection System

ARTEMIS is a self-operated system that is run locally by network operators to safeguard their own prefixes against malicious BGP hijacking attempts. The key ideas behind ARTEMIS are:

configuration file: all the prefixes owned by the network are listed here for reference, this configuration file is populated by the network operator

mechanism for receiving BGP updates: allows receiving updates from local routers and monitoring services, this is built into the system

Using local configuration file as a reference, for the received BGP updates ARTEMIS can check for prefixes and AS-PATH fields and trigger alerts when there are anomalies. The ARTEMIS system also allows the network operator to choose between (a) accuracy and speed and (b) false negative (FN) that are inconsequential (less impact on control plane) for less false positive (FP).

9.10 Defending Against BGP Hijacking: Example Mitigation Techniques

The ARTEMIS system uses two automated techniques in mitigating these attacks:

prefix disaggregation: in a BGP attack scenario the affected network can either contact other networks, or it can simply disaggregate the prefixes that were targeted by announcing more specific prefixes of a certain prefix

mitigation with multiple origin AS (MOAS): the idea is to have third party organizations and service providers do BGP announcements for a given network, this is akin to the current model that exists for legitimizing network traffic by third parties to mitigate DDoS attacks, when a BGP hijacking event occurs the following steps take place:

1. the third party receives a notification and immediately announces from their locations the hijacked prefix(es)
2. network traffic from across the world is attracted to the third party organization, which then scrubbs it and tunnels it to the legitimate AS

The authors of the ARTEMIS paper put forth two main findings from their research work:

outsource the task of BGP announcement to third parties: having even just one single external organization to mitigate BGP attacks is highly effective against BGP attacks

comparison of outsourcing BGP announcements vs prefix filtering: prefix filtering, which is the current standard defense mechanism, is less optimal compared with BGP announcements

9.11 A Hijacking Case Study: Background (Optional)

Linktel, a Russian ISP under attack, had gone through financial struggles and thus had not renewed its DNS domain, which allowed administrative take-over of its Internet resources. The attacker starts with announcing one prefix and takes over other prefixes over time. To exploit an unallocated address space of Linktel, the attacker brings in another hijacked AS which is also being served by the same upstream provider as that of Linktel. The attacker uses this as a decoy channel to test announcing the unallocated address space without affecting the whole attack.

9.12 A Hijacking Case Study: Attack Progression (Optional)

The Linktel incident ran its course for about 6 months, starting from the attack phase to the productive phase to the recovery phase.

hijacking phase: the vulnerability exploited is that Linktel had let its DNS domain (link-telecom) expire, which consequently could be taken over by anyone and be used maliciously, the attacker re-registered the domain within six hours of its expiration, announced prefix one by one, and finally issued a forged letter of authorization to Internap two months after Internap had been announcing the attacker's routes without authorization

productive phase: the attacker stopped announcing the unallocated prefix, since the attacker's motivation was to send spam it was eventually caught by Spamhaus and the prefixes were blacklisted

recovery phase: Linktel itself was recovering from a financial crisis, and in its attempt to bring its prefixes back online it recognized that its prefixes had been blacklisted by Spamhaus, Eventually Linktel complains to its upstream ISPs and redelegates reverse DNS entries and announces more specific prefixes, the upstream ISPs comply and withdraw routes from hijacked prefixes

9.13 DDoS: Background and Spoofing

A **Distributed Denial of Service (DDoS)** attack is an attempt to compromise a server or network resources with a flood of traffic. To achieve this the attacker first compromises and deploys flooding servers (slaves), then the attacker instructs these flooding servers to send a high volume of traffic to the victim. This results in the victim host either becoming unreachable or in exhaustion of its bandwidth. This master-slave configuration amplifies the intensity of the attack while also making it difficult to protect against it. Because the attack traffic sent by the slaves contain spoofed source addresses making it difficult for the victim to track the slaves, and since the traffic is sent from multiple sources it is harder for the victim to isolate and block the attack traffic.
IP spoofing is the act of setting a false IP address in the source field of a packet with the purpose of impersonating a legitimate server. In DDoS attacks this can happen in two forms:

- the source IP address is spoofed, resulting in the response of the server sent to some other client instead of the attacker's machine, this results in wastage of network resources and the client's resources while also causing denial of service to legitimate users
- the attacker sets the same IP address in both the source and destination IP fields resulting in the server sending the replies to itself, causing it to crash

9.14 DDoS: Reflection and Amplification

There are two more techniques that the attackers are using to amplify the impact of DDoS attack—**reflection** and **amplification**.

9.14.1 Reflection

In a reflection attack the attackers use a set of reflectors to initiate attack on the victim. A **reflector** is any server that sends a response to a request, e.g. SYN ACK in response to SYN, and Host Unreachable response to an IP query. Here the master directs the slaves to send spoofed requests to a very large number of reflectors, usually in the order of 1 million. The slaves set the source address of the packets to the victim's IP address thereby redirecting the response of the reflectors to the victim. This is in contrast with the conventional DDoS attack where the slaves directly send traffic to the victim.

Note that the victim can easily identify the reflectors from the response packets. However the reflector cannot identify the slave sending the spoofed requests.

9.14.2 Amplification

If the requests are chosen in such a way that the reflectors send large responses to the victim, it is a reflection and amplification attack. Not only would the victim receive traffic from millions of servers, the response sent would be large in size, making it further difficult for the victim to handle it.

9.15 Defenses Against DDoS Attacks

9.15.1 Traffic scrubbing services

A scrubbing service diverts the incoming traffic to a specialized server, where the traffic is "scrubbed" into either clean or unwanted traffic. The clean traffic is then sent to its original destination. Although this method offers fine-grained filtering of the packets, it has three limitations

- there are monetary costs required for an in-time subscription, setup, and other recurring costs
- reduced effectiveness due to per packet processing and challenges in handling Tbps level attacks
- decreased performance as the traffic may be rerouted and becoming susceptible to evasion attacks

9.15.2 ACL filters

Access Control List (ACL) filters are deployed by ISPs and IXPs at their AS border routers to filter out unwanted traffic. These filters, whose implementation depends on the vendor-specific hardware, are effective when the hardware is homogeneous and the deployment of the filters can be automated. The drawbacks of these filters include limited scalability and since the filtering does not occur at the ingress points, it can exhaust the bandwidth to a neighboring AS.

9.15.3 BGP Flowspec

The flow specification feature of BGP, called Flowspec, is an extension to the BGP protocol, which allows rules to be created on the traffic flows and take corresponding actions. This feature of BGP can help mitigate DDoS attacks by specifying appropriate rules. The AS domain borders supporting BGP Flowspec are capable of matching packets in a specific flow based on a variety of parameters such as source IP, destination IP, packet length, protocol used etc. and can also be based on the drop rate limit. In a BGP Flowspec a "traffic-rate" action with value 0 discards the traffic. The other possible actions include rate limiting, redirecting, or filtering. If no rule is specified the default action for a rule is to accept the incoming traffic.
In contrast to ACL filters Flowspec leverages the BGP control plane making it easier to add rules to all the routers simultaneously. Although Flowspec is seen to be effective in intra-domain environment, it is not so popular in inter-domain environments as it depends on trust and cooperation among competitive networks. In addition it may not scale for large attacks where the attack traffic originates from multiple sources as it would multiply rules or combine the sources into one prefix.

9.16 DDoS Mitigation Techniques: BGP Blackholing

With **BGP blackholing** all the attack traffic to a targeted DDoS destination is dropped to a null location. The premise of this approach is that the traffic is stopped closer to the source of the attack and before it reaches the targeted victim. For a high volume attack it proves to be an effective strategy

when compared to other mitigation options. This technique is implemented either with the help of the upstream provider or with the help of the IXP if the network is peering at an IXP. With this technique the victim AS uses BGP to communicate the attacked destination prefix to its upstream provider or IXP, which then advertises a more specific prefix and modifies the next-hop address that will divert the attack traffic to a null interface. The blackhole message is tagged with a specific BGP blackhole community attribute, usually publicly announced by the upstream provider or IXP, to differentiate it from regular routing updates.
A network that offers blackholing service is known as a blackholing provider, which is also responsible for providing the blackholing community that should be used. Network or customer providers act as blackholing providers at the network edge, while Internet Service Providers (ISPs) or Internet Exchange Points (IXPs) act as blackholing providers at the Internet core.

9.17 DDoS Mitigation Techniques: BGP Blackholing Limitations and Problems

One of the major drawbacks of BGP blackholing is that the destination under attack becomes unreachable since all the traffic including the legitimate one is dropped. For the ASes that accept the blackholing announcement, collateral damage occurs since all the traffic including the legitimate one via the ASes is dropped. For the ASes that doesn't accept the announcement, the mitigation is ineffective as legitimate traffic along with attack traffic will come through them.
The possible reasons for an AS rejecting the blackholing announcement are voluntarily choosing not to participate in blackholing, rejecting updates that require additional config changes, or it could simply be that the AS made a misconfiguration mistake.

10 Lesson 10: Internet Surveillance and Censorship

10.1 Introduction

Internet censorship is a special case of Internet security.

10.2 DNS Censorship: What Is It?

DNS censorship is a large-scale network traffic filtering strategy opted by a network to enforce control and censorship over Internet infrastructure to suppress material which they deem objectionable, e.g. the Great Firewall of China (GFW). The GFW works by injecting fake DNS record responses so that access to a domain name is blocked. Researchers have tried to reverse engineer the GFW to understand how it works and have started to identify some of the properties:

locality of GFW nodes: there are two differing notions on whether the GFW nodes are present only at the edge ISPs or whether they are also present in non-bordering Chinese ASes, the majority view is that censorship nodes are present at the edge

centralized management: since the blocklists obtained from two distinct GFW locations are the same, there is a high possibility of a central management (GFW Manager) entity that orchestrates blocklists

load balancing: GFW load balances between processes based on source and destination IP address, the processes are clustered together to collectively send injected DNS responses

10.3 Example DNS Censorship Techniques (1)

DNS injection is one of the most common censorship technique employed by the GFW, which uses a rule set to determine when to inject DNS replies to censor network traffic. To start with it is important to identify and isolate the networks that use DNS injection for censorship. The steps involved in DNS injection are:

1. DNS probe is sent to the open DNS resolvers
2. the probe is checked against the blocklist of domains and keywords
3. for domain-level blocking a fake DNS A record response is sent back, there are two levels of blocking domains: (1) directly blocking the domain (2) blocking it based on keywords present in the domain

10.4 Example DNS Censorship Techniques (2)

We provide an overview of different DNS censorship techniques and look at their strengths and weaknesses

10.4.1 Packet dropping

In packet dropping all network traffic going to a set of specific IP addresses is discarded.

strength: easy to implement, low cost

weakness: maintenance of blocklist—it is challenging to stay up to date with the list of IP addresses to block, overblocking—if two websites share the same IP address and the intention is to only block one of them then there is a risk of blocking both

10.4.2 DNS poisoning

When a DNS receives a query for resolving hostname to IP address, there is no answer returned or an incorrect answer is sent to redirect or mislead the user request. This scenario is called **DNS Poisoning**.

strength: no overblocking as there is an extra layer of hostname translation, access to specific hostnames can be blocked versus blanket IP address blocking

weakness: blocks the entire domain, it is impossible to allow email contact while blocking the website

10.4.3 Content inspection

proxy-based content inspection: more sophisticated in that it allows for all network traffic to pass through a proxy where the traffic is examined for content, and the proxy rejects requests that serve objectionable content

- **strength:** precise censorship—a very precise level of censorship can be achieved down to the level of single web pages or even objects within the web page, flexible—works well with a combination of other censorship techniques like packet dropping and DNS poisoning
- **weakness:** not scalable—they are expensive to implement on a large-scale network as the processing overhead is large (through a proxy)

intrusion detection system (IDS) based content inspection: use parts of an IDS to inspect network traffic, an IDS is easier and more cost effective to implement than a proxy-based system as it is more responsive than reactive in nature in that it informs the firewall rules for future censorship

10.4.4 Blocking with resets

The GFW employs this technique where it sends a TCP reset (RST) to block individual connections that contain requests with objectionable content. After a client sends the request containing flaggable keywords, it receives 3 TCP RSTs corresponding to one request, possibly to ensure that the sender receives a reset.

10.4.5 Immediate reset of connections

Censorship systems like GFW have blocking rules in addition to inspecting content to suspend traffic coming from a source immediately for a short period of time. The reset packet received by the client is from the firewall. It does not matter whether the client sends out legitimate GET requests following one questionable request. The client will continue to receive resets from the firewall for a particular duration. Running different experiments suggests that this blocking period is variable for questionable requests.

10.5 Why is DNS Manipulation Difficult to Measure

Anecdotal evidence suggests that more than 60 countries are currently impacted by control of access to information through the Internet's DNS manipulation. However our understanding of censorship around the world is relatively limited, because

diverse measurements: such understanding would need a diverse set of measurements spanning different geographic regions, ISPs, countries, and regions within a single country, because political dynamics can vary different ISPs can use various filtering techniques, and different organizations may implement censorship at multiple layers of the Internet protocol stack and using different techniques, therefore we need widespread longitudinal measurements to understand the heterogeneity of DNS manipulation

need for scale: at first the methods to measure Internet censorship were relying on volunteers who were running measurement software on their own devices, this method is unlikely to reach the scale required, there is a need for methods and tools that are independent of human intervention and participation

identifying the intent to restrict content access: while identifying inconsistent or anomalous DNS responses can help to detect a variety of underlying causes such as misconfigurations, identifying DNS manipulation is different and it requires that we detect the intent to block access to content, so we need to rely on identifying multiple indications to infer DNS manipulation

ethics and minimizing risks: there are risks associated with involving citizens in censorship measurement studies based on how different countries may be penalizing access to censored material, therefore it is safer to stay away from using DNS resolvers or DNS forwarders in the home networks of individual users, and it is safer to rely on open DNS resolvers that are hosted in Internet infrastructure

10.6 Example Censorship Detection Systems and Their Limitations

One the most common systems/approaches is the OpenNet Initiative where volunteers perform measurements on their home networks at different times since the past decade. Relying on volunteer efforts make continuous and diverse measurements very difficult. In addition Augur is a new system created to perform longitudinal global measurements using TCP/IP side channels. However this system focuses on identifying IP-based disruptions as opposed to DNS-based manipulations.

10.7 DNS Censorship: A Global Measurement Methodology

We explore a method to identify DNS manipulation via machine learning with a system called Iris. The lack of diversity is an issue while studying DNS manipulation. In order to counter that Iris uses open DNS resolvers located all over the globe. In order to avoid using home routers (which are usually open due to configuration issues) this dataset is then restricted to a few thousand that are part of the Internet infrastructure. There are two main steps associated with this process:

1. scanning the Internet's IPv4 space for open DNS resolvers
2. identifying Infrastructure DNS Resolvers

Now that we've obtained a global set of open DNS resolvers we need to perform the measurements. The steps involved in this measurement process are:

1. performing global DNS queries—Iris queries thousands of domains across thousands of open DNS resolvers, to establish a baseline for comparison the creators included 3 DNS domains that were under their control to help calculate metrics used for evaluation DNS manipulation
2. annotating DNS responses with auxiliary information—to enable the classification Iris annotates the IP addresses with additional information such as their geo-location, AS, port 80 HTTP responses etc. this information is available from the Censys dataset
3. additional PTR and TLS scanning—one IP address could host several websites via virtual hosting, so when Censys retrieves certificates from port 443 it could differ from the one retrieved via TLS's Server Name Indication (SNI) extension, this results in discrepancies that could cause Iris to label virtual hosting as DNS inconsistencies, to avoid this Iris adds PTR and SNI certificates

After annotation the dataset techniques are performed to clean the dataset and identify whether DNS manipulation is taking place or not. Iris uses two types of metrics to identify this manipulation:

consistency metrics: domain access should have some consistency in terms of network properties, infrastructure, or content even when accessed from different global vantage points, using one of the domains Iris controls gives a set of high-confidence consistency baselines, some consistency metrics used are IP address, AS, HTTP Content, HTTPS certificate, PTRs for CDN

independent verifiable metrics: Iris also use metrics that could be externally verified using external data sources, some of the independent verifiable metrics used are HTTPS certificate without and with SNI

If any consistency metric or independent verifiable metric is satisfied the response is correct. Otherwise the response is classified as manipulated.

10.8 Censorship Through Connectivity Disruptions

The highest level of Internet censorship is to completely block access to the Internet. Although this seems simple it may not be feasible as the infrastructure could be distributed over a wide area. A more subtle approach is to use software to interrupt the routing or packet forwarding mechanisms. There are two such methods:

routing disruption: a routing mechanism decides which part of the network can be reachable, routers use BGP to communicate updates to other routers in the network, if this communication is disrupted or disabled on critical routers it could result in unreachability of the large parts of a network, this approach can be easily detectable as it involves withdrawing previously advertised prefixes or re-advertising them with different properties and therefore modifying the global routing state of the network which is the control plane

packet filtering: typically packet filtering is used as a security mechanism in firewalls and switches, but to disrupt a network's connectivity packet filtering can be used to block packets matching a certain criteria disrupting the normal forwarding action, this approach can be harder to detect and might require active probing of the forwarding path or monitoring traffic of the impacted network

Connectivity disruption can include multiple layers apart from the two methods described above. It can include DNS-based blocking, deep packet inspection by an ISP, or the client software blocking the traffic.

10.9 Connectivity Disruptions: A Case Study (Optional)

10.9.1 Egypt

During the outage in Egypt it was observed that all the routes to Egyptian networks were withdrawn from the Internet's global routing table. The primarily state-owned Internet infrastructure of Egypt, with a small number of parties providing international connectivity and a state telecommunications provider controlling the physical connectivity, enables such manipulation of the system. The analysis of the darknet traffic also identified DoS attacks to the Egyptian government sites.

10.9.2 Libya

A single AS which is owned by the state dominates Libya's Internet infrastructure, with only two submarine cables providing international connectivity, making it easier to manipulate. During the first two outages it was observed that 12 out of the 13 delegated prefixes to Libya were withdrawn by its local telecom operator with a reasonable exception of a prefix that was controlled by an outside company. However it was also observed that the darknet received some small traffic during the second outage suggesting that other censorship techniques might have been used. During the third outage it was observed that none of the BGP routes were withdrawn. Unlike the first two outages where the primary technique was BGP disruption, the third outage was caused by packet filtering. This can be concluded by the small amount of traffic observed by the darknet that is consistent with the behavior of packet filtering.

10.10 Connectivity Disruptions: Detection

Augur is a system that aims to detect if filtering exists between two hosts, a reflector and a site. A reflector is a host which maintains a global IP ID. A site is a host that may be potentially blocked. To identify if filtering exists it makes use of a third machine called the measurement machine.

10.10.1 IP ID

The strategy used by Augur takes advantage of the fact that any packet that is sent by a host is assigned a unique 16-bit IP identifier called IP ID, which the destination host can use to reassemble a fragmented packet. This IP ID should be different for different packets that are generated by the same host. Although there are multiple methods available to determine the IP ID of a packet (randomly, per-connection counter etc.), maintaining a single global counter is the most commonly used approach. The global counter is incremented for each packet that is generated and helps in keeping track of the total number of packets generated by that host. Using this counter we can determine if and how many packets are generated by a host.
In addition to the IP ID counter Augur also leverages the fact that when an unexpected TCP packet is sent to a host it sends back a RST (TCP Reset) packet. It also assumes there is no complex factors involved such as cross-traffic or packet loss. Two important mechanisms using IP ID are:

probing: a mechanism to monitor the IP ID of a host over time, we use the measurement machine sends a TCP SYN-ACK to the reflector and receives a TCP RST packet as the response, the RST packet received would contain the latest IP ID that was generated by the reflector, thus the measurement machine can track the IP ID counter of the reflector at any given point of time

perturbation: a mechanism which forces a host to increment its IP ID counter by sending traffic from different sources such that the host generates a response packet, the flow is as follows:

1. the measurement machine sends a spoofed TCP SYN packet to the site with source address set to the reflector's IP address
2. the site responds to the reflector with a TCP SYN-ACK packet
3. the reflector returns a TCP RST packet to the site while also incrementing its global IP ID counter by 1

With the reflector, the site, and the measurement machine, let the initial IP ID counter of the reflector be 5.

no filtering: When there is no filtering the sequence of events is as follows:

1. the measurement machine probes the IP ID of the reflector by sending a TCP SYN-ACK packet, it receives a RST response packet with IP ID set to 6 which is IPID(t1)
2. the measurement machine performs perturbation by sending a spoofed TCP SYN to the site
3. the site sends a TCP SYN-ACK packet to the reflector and receives a RST packet as a response, the IP ID of the reflector is now incremented to 7
4. the measurement machine again probes the IP ID of the reflector and receives a response with the IP ID value set to 8 which is IPID(t4)

The measurement machine thus observes that the difference in IP IDs between steps 1 and 4 is two and infers that communication has occurred between the two hosts.

inbound blocking: The scenario where filtering occurs on the path from the site to the reflector is termed as inbound blocking. In this case the SYN-ACK packet sent from the site in step 3 does not reach the reflector. Hence there is no response generated and the IP ID of the reflector does not increase. The returned IP ID in step 4, IPID(t4), will be 7. Since the measurement machine observes the increment in IP ID value is 1, it detects filtering on the path from the site to the reflector.

outbound blocking: Outbound blocking is the filtering imposed on the outgoing path from the reflector. In step 3 the reflector receives the SYN-ACK packet and generates a RST packet and the IP ID increments to 7. However the RST packet does not reach the site. When the site doesn't receive a RST packet it continues to resend the SYN-ACK packets at regular intervals depending on the site's OS and its configuration. This results in further increment of the IP ID value of the reflector, and the probe by the measurement machine reveals that the IP ID has again increased by 2 which shows that retransmission of packets has occurred. In this way outbound blocking can be detected.

11 Lesson 11: Applications: Video

11.1 Multimedia Applications: A Perspective From the Network

In this lesson we will be discussing multimedia applications, which are defined in Kurose and Ross as "any network application that employs audio or video".

11.2 Background: Video and Audio Characteristics

The first defining property for video is high bit rate, generally somewhere between 100 kbps to over 3 Mbps. However there are techniques for compressing video.
Audio has a lower bit rate than video, but glitches in audio are generally more noticeable than glitches in video. Like video audio can be compressed at varying quality levels.

11.3 Types of Multimedia Applications and Characteristics

The different kinds of multimedia applications can be organized into three major categories:

streaming stored audio and video: such course video clips on Udacity

conversational voice and video over IP: such as Skype

streaming live audio and video: such as the graduation ceremony on graduation day

Streaming stored video is

streamed: the video starts playing within a few seconds of receiving data instead of waiting for the entire file to download first

interactive: the user can pause, fast forward, skip ahead or move back in the video

continuous playout: it should play in the same way it was recorded without freezing in the middle

Generally streaming stored videos are stored on a CDN rather than just one data center. This type of multimedia application can also be implemented with the peer-to-peer model instead of the client-server model.
Since streaming live audio and video are live and broadcast-like, there are generally many simultaneous users. They are also delay-sensitive, but not as much as conversational voice and video applications are, generally a ten second delay is OK.
VoIP stands for "Voice over IP" which is like phone service that goes over the Internet instead of through traditional circuit-switched telephony network. Since these calls and conferences are real-time and involve human users interacting, these applications are highly delay-sensitive—a longer delay such as over 400 milliseconds can be frustrating as people end up accidentally talking over each other. On the other hand these applications are loss-tolerant—even if a word in the conversation gets garbled, human listeners are generally able to ask the other side to just repeat themselves.

11.4 How Does VoIP Work

VoIP falls in the category of conversational voice over IP. We will focus on conversational audio although video works in similar ways. VoIP has the challenge that it is transmitted over the Internet. Since the Internet is best effort and makes no promises that a datagram will eventually make it to its final destination or even on time. The three major topics in VoIP are

encoding: audio is encoded by taking thousands of samples per second and then rounding each sample's value to a discrete number within a particular range, this rounding to a discrete number is called **quantization**, with more samples per second or a larger range of quantization values, the digital approximation gets closer to the actual analog signal which means a higher quality, the three major categories of encoding schemes are narrowband, broadband, and multimode (which can operate on either)

signaling: in traditional telephony a signaling protocol takes care of how calls are set up and torn down, signaling protocols are responsible for four major functions

user location: the caller locating where the callee is

session establishment: handling the callee accepting, rejecting, or redirecting a call

session negotiation: the endpoints synchronizing with each other on a set of properties for the session

call participation management: handling endpoints joining or leaving an existing session

VoIP uses signaling protocols just like telephony to perform the same functions. The SIP (Session Initiation Protocol) is just one example of a signaling protocol used in many VoIP applications.

QoS metrics:

11.5 QoS for VoIP: Metrics

There are three major quality-of-service metrics for VoIP (1) end-to-end delay (2) jitter (3) packet loss.

11.6 QoS for VoIP: End-to-End Delay

The end-to-end delay, which is basically the total delay from mouth to ear, includes delay from:

- the time it takes to encode the audio (discussed earlier)
- the time it takes to put it in packets
- all the normal sources of network delay that network traffic encounters such as queuing delays, playback delay which comes from the receiver's playback buffer—a mitigation technique for delay jitter (discussed next)
- decoding delay, the time it takes to reconstruct the signal

In general an end-to-end delay of below 150ms is not noticeable by human listeners. A delay between 150ms and 400ms is noticeable but perhaps acceptable e.g. we might be more accepting of delays if we are calling a more remote region. However an end-to-end delay greater than 400ms starts becoming unacceptable as people start accidentally talking over each other. Since delays are so impactful for VoIP, VoIP applications frequently have delay thresholds such as 400ms, and discard any received packets with a delay greater than that threshold.

11.7 QoS for VoIP: Delay Jitter

Due to all the different buffer sizes, queueing delays, and network congestion levels that a packet might experience, different voice packets can end up with different amounts of delay. One voice packet may be delayed by 100ms while another by 300ms, we call this phenomenon **jitter**, packet jitter, or delay jitter. Jitter is problematic for VoIP because it interferes with reconstructing the analog voice stream. With large jitter we end up with more delayed packets that end up getting discarded, which can lead to a gap in the audio. Too many dropped sequential packets can make the audio unintelligible. Although human ear is pretty intolerant of audio gaps, audio gaps should ideally be kept below 30ms. Nonetheless depending on the type of voice codec used and other factors, audio gaps between 30 to 75ms can be acceptable.
The main VoIP application mechanism for mitigating jitter is maintaining a buffer called the **jitter**

buffer or the play-out buffer, which smooths out and hides the variation in delay between different received packets by buffering them and playing them out for decoding at a steady rate. There is a tradeoff here though. A longer jitter buffer reduces the number of packets that are discarded but that adds to the end-to-end delay, whereas a shorter jitter buffer will not add to the end-to-end delay as much, but can lead to more dropped packets which reduces the speech quality.

11.8 QoS for VoIP: Packet Loss

Packet loss is pretty much inevitable because the Internet is a best-effort service. VoIP protocols could use TCP. However

- TCP eliminates packet loss by retransmitting lost packets, and those retransmitted packets are no good if they are received too late.
- TCP congestion control algorithms drop the sender's transmission rate every time there is a dropped packet. Thus we can end up with the sender dropping the transmission rate lower than the receiver's drain rate from playing out the audio.

Therefore most of the time VoIP protocols use UDP.
When we consider packet loss for VoIP we adopt a different definition. For VoIP a packet is lost if it either never arrives or if it arrives after its scheduled playout. This is a harsher definition than for other applications such as file transfer. But thankfully VoIP can tolerate loss rates of between 1% and 20%.
VoIP protocols have three major methods of dealing with packet loss

forward error correction (FEC): FEC comes in a few different flavors, but in general FEC works by transmitting redundant data alongside the main transmission which allows the receiver to replace lost data with the redundant data, this redundant data could be

- a copy of the original data by breaking the audio into chunks and cleverly using exclusive OR (XOR) with n previous chunks
- a lower-quality audio stream transmitted alongside the original stream similar to the spare tire in a car

Regardless of which method is used there is a tradeoff:

bandwidth: the more redundant data transmitted the more bandwidth is consumed

delay: some of these FEC techniques require the receiving end to receive more chunks before playing out the audio and that increases playout delay

interleaving: interleaving does not transmit any redundant data, interleaving works by mixing chunks of audio together so that if one set of chunks is lost the lost chunks aren't consecutive, the idea is that many smaller audio gaps are preferable to one large audio gap
the tradeoff for interleaving is that the receiving side has to wait longer to receive consecutive chunks of audio and that increases latency, unfortunately that means this technique is limited in usefulness for VoIP, although it can have good performance for streaming stored audio

error concealment: basically guessing what the lost audio packet might be, this is possible because generally with really small audio snippets (like between 4ms and 40 ms) there is some similarity between one audio snippet and the next audio snippet, this is the same principle that makes audio compression possible
two simple versions of error concealment are

repetition: replace the lost packet with a copy of the previous packet, it is computationally cheap and works pretty well in a lot of cases

interpolation: use the audio before and after the lost packet and interpolate an appropriate packet to conceal the lost packet, this is a better solution than packet repetition but it is more computationally expensive

11.9 Live/On-Demand Streaming Introduction

Streaming media content over the Internet accounts for nearly 60-70% of the Internet traffic. Various enabling technologies and trends have led to this development of consuming media content over the Internet

- the bandwidth for both the core network and last-mile access links have increased tremendously over the years
- the video compression technologies have become more efficient, enabling streaming high-quality video without using a lot of bandwidth
- the development of Digital Rights Management culture has encouraged content providers to put their content on the Internet

The types of content that is streamed over the Internet can be divided into two categories

live: the video content is created and delivered to the clients simultaneously, e.g. streaming of sports events, music concerts etc.

on-demand: streaming stored video based on users' convenience, e.g. Netflix, YouTube

The constraints for streaming live and on-demand content differ slightly. One of the main constraints being that there is not a lot of room for pre-fetching content in the case of live streaming. Nonetheless it turns out that most of the basic principles behind streaming live at large-scale and on-demand content are similar apart from the few details such as video encoding.

11.10 Video Streaming Bigger Picture

The video content is first created. The raw recorded content is typically at a high quality. It is then compressed using an encoding algorithm. This encoded content is then secured using DRM and hosted over a server. The end-users download the video content over the Internet. The downloaded video is decoded and rendered on the user's screen.

11.11 Sources of Redundancy in Video Compression (Optional)

The compression can be of two types

lossy: we can not recover the high quality video

loss-less: the original video can be recovered

It is intuitive that lossy compression gives higher savings in terms of bandwidth and hence that is what is used by video compression algorithms these days.
The goal of a compression algorithm is to cut down data by removing similar information from the video while not compromising a lot with the video quality. As a video is essentially a sequence of digital pictures where each picture is a 2D sequence of pixels, an efficient compression can be achieved in two ways:

spatial redundancy: within an image—pixels that are nearby in a picture tend to be similar

temporal redundancy: across images—in a continuous scene consecutive pictures are similar

11.12 Image Compression (Optional)

The Joint Photographics Experts Group (JPEG) standard was developed. The main idea behind JPEG is how to represent an image using components that can be further compressed. It essentially consists of four steps:

1. Assume that we are given an RGB image, the first step is to transform it into color component (chrominance or Cb, Cr) and brightness component (luminance or Y). This is done because human eyes are more sensitive to brightness and at the same time we could further compress Cb and Cr. Now our image is represented with 3 matrices: Y, Cb, Cr. We obtain the 3 matrices Y, Cb and Cr from RGB by using the conversion below:

$$\begin{pmatrix} \mathrm{Y} \\ \mathrm{C_b} \\ \mathrm{C_r} \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.168 & 0.330 & 0.500 \\ 0.500 & 0.410 & 0.081 \end{pmatrix} \begin{pmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{pmatrix}$$

2. We compress and encode each of the 3 matrices (Y Cb Cr). We perform the same steps below: divide the image into 8×8 blocks (sub-image), apply the Discrete Cosine Transformation to each of the sub-images to transform it into the frequency domain, the output (for each sub-image) is an 8×8 table of DCT coefficients with each element representing the spectral power (weight) present at that spatial frequency. This is done because the human eye is less sensitive to high-frequency coefficients as compared to low-frequency coefficients. Thus the latter can be compressed significantly using quantization.

3. This is the lossy step of the process. We compress the matrix of the coefficients using a predefined Quantization table Q, $\mathrm{F}^{\mathrm{quantized}}(\mathrm{u}, \mathrm{v}) = \mathrm{F}(\mathrm{u}, \mathrm{v})/\mathrm{Q}(\mathrm{u}, \mathrm{v})$. Each entry of the coefficient table is divided by the corresponding entry in the quantization table and then it is rounded to the nearest integer. Note that the weights are higher for higher spectral frequencies. In addition the Quantization table is stored in the image header so the image can be later decoded.

4. After obtaining the quantized matrices for each of the 8×8 block in the image, we perform a lossless encoding to store the coefficients. This finishes the encoding process. This encoded image is then transferred over the network and is decoded at the user-end.

11.13 Video Compression and Temporal Redundancy (Optional)

Consecutive frames in a video are similar to each other. Instead of encoding every image as a JPEG we encode the first known as **I-frame** as a JPEG and then encode the difference between two frames. The in-between encoded frame is known as predicted or **P-frame**. Note that we do not want to use the difference all the way, because

- when a scene changes the frame becomes drastically different from the last frame, and even if we are using the difference it is almost like encoding a fresh image, this can lead to loss in image quality
- it also increases the decoding time in case a user skips ahead in the video, since it will need to download all frames in between the last I-frame and the current frame to decode the video

Thus a typical solution is to insert an I-frame periodically say after every 15 frames. All the frames between two consecutive I-frames is known as a Group of Pictures (GoP).
An additional way to improve encoding efficiency is to encode a frame as a function of the past and the future I (or P)-frames. Such a frame is known as a bi-directional or **B-frame**. Note that to decode these both frames are required. Therefore an example sequence of frames could be written as: IBBBPIBPPPIBBBBI...

11.14 VBR vs CBR (Optional)

compression algorithms provide knobs to control the trade-offs between the output image quality and its size. There are two ways to use these knobs for encoding videos:

constant bitrate (CBR) encoding: the output size of the video is fixed over time

variable bitrate (VBR) encoding: the output size remains the same on an average, but varies here and there based on the underlying scene complexity

The image quality is better in VBR as compared to CBR for the same bitrate. However VBR is more computationally expensive as compared to CBR because video compression in general requires heavy-computation. In fact for live streaming wherein the video compression has to be done real-time, specialized (but expensive) hardware encoders are used by content providers.

11.15 UDP vs TCP

Video needs to be decoded at the client. This decoding might fail if some data is lost. Thus between UDP and TCP content providers ended up choosing TCP for video delivery as it provides reliability. An additional benefit of using TCP was that it already provides congestion control which is required for effectively sharing bandwidth over the Internet.

11.16 Why Do We Use HTTP

The original vision was to have specialized video servers that remembered the state of the clients. These servers would control the sending rate to the clients. In the case if the clients paused the video, it would send a signal to the server and the server would stop sending video. However this required content providers to buy specialized hardware.
Another option was to use the already existing HTTP protocol. In this case the server is essentially stateless and the intelligence to download the video will be stored at the client. A major advantage of this is that content providers could use the already existing CDN infrastructure. Moreover it also made bypassing middleboxes and firewalls easier as they already understood HTTP. Because of the above advantages the original vision was abandoned and content providers ended up using HTTP for video delivery.

11.17 Progressive Download vs Streaming

One way to stream the video would be to send an HTTP GET request for video. It is essentially like downloading a file over HTTP. The server will send the data as fast as possible with the download rate limited only by the TCP rate control mechanisms. While quite simple this has some disadvantages:

- users often leave the video midway, thus downloading the entire file is a waste of network resources

- the video content that has been downloaded but not played would have to be stored i.e. we will need a video buffer at the client to store this content in memory, which can be an issue particularly with long videos as large buffer would be required to store the content

Thus instead of downloading the content all at once the client tries to pace it. This can be done by sending **byte-range** requests for parts of the video instead of requesting the entire video. Once the video content has been watched it sends request for more content. However throughput over the Internet is variable, which can lead to unnecessary stalling if the client is doing pure streaming i.e. downloading the video content just before it is to be played. To account for these variations client pre-fetches some video ahead and stores it in a playout buffer. The playout buffer is usually defined in terms of number of seconds of video that can be downloaded in advance or in terms of size in bytes. Once the video buffer becomes full, the client will wait for it to get depleted before asking for more content. Streaming in this manner typically has two states:

filling state: happens when the video buffer is empty and the client tries to fill it as soon as possible (e.g. the beginning of the playback)

steady state: after the buffer has become full the client waits for it to become lower than a threshold, after which the client sends a request for more content, the steady state is characterized by these ON-OFF patterns

11.18 How to Handle Network and User Device Diversity

A client's streaming context can be quite diverse, which varies in terms of device (smartphone vs. HDTV), network environment (WIFI vs. 5G), and the Internet throughput (family member downloading over the same connection could cut available bandwidth to half). Hence a single-bitrate encoded video is not the best solution. Instead content providers encode their video at multiple bitrates chosen from a set of pre-defined bitrates. Specifically the video is chunked into segments which are usually of equal duration. Each of these segments is then encoded at multiple bitrates and stored at the server. The client request while requesting for a segment also specifies its quality. This is known as **bitrate adaptation**. At the beginning of every video session the client first downloads a *manifest* file, again over HTTP. The manifest file contains all the metadata information about the video content and the associated URLs.

11.19 Bitrate Adaptation in DASH

A content provider encodes the video and stores it on a web server. The video player at the client-side downloads the video content using HTTP/TCP over the network. The client dynamically adjusts the video bitrate based on the network conditions and device type. This is known as Dynamic Streaming over HTTP or DASH where dynamic streaming signifies the dynamic bitrate adaptation. There have been multiple implementations of this with HLS and MPEG-DASH being the most popular.
A video in DASH is divided into chunks and each chunk is encoded into multiple bitrates. Each time the video player needs to download a video chunk it calls the **bitrate adaptation function** f, which takes in some input and outputs the bitrate of the chunk to be downloaded

$$R(n) = f(\text{input}) \qquad \text{where } R(n) \in \{R_1, R_2, \ldots, R_m\}$$

where $R(n)$ denotes the set of available bitrates. In the following topics we will see different variations of the bitrate adaptation function f.

11.20 What are the Goals of Bitrate Adaptation

A bitrate adaptation algorithm essentially tries to optimize the user's viewing quality of experience. A good quality of experience (QoE) is usually characterized by the following:

low or zero re-buffering: users typically tend to close the video session if the video stalls a lot

high video quality: the better the video quality the better the user QoE, a higher video quality is usually characterized by high bitrate video chunk

low video quality variations: a lot of video quality variations are also known to reduce user QoE

low startup latency: startup latency is the time it takes to start playing the video since the user first requested to play the video, players typically fill up the video buffer a little before playing the video

For this lesson we will skip considering startup latency and focus on the other three metrics.
It is interesting to note that the different metrics characterizing QoE are conflicting. For instance to avoid re-buffering a user can change either download the lowest bitrate, which leads to a low video quality, or change the video bitrate as soon as it notices a change in the network conditions, which leads to high video quality variations. The goal of a good bitrate adaptation algorithm then is to consider these trade-offs and maximize the overall user QoE.

11.21 Bitrate Adaptation Algorithms

The signals that can serve as an input to a bitrate adaptation algorithm include

network throughput: ideally we want to select a bitrate that is equal or lesser than the available throughput, bitrate adaptation using this signal are known as **rate-based adaptation**

video buffer: if the video buffer is high then the player can possibly afford to download high-quality chunks; if the video buffer is low the player can download low-quality chunks so as to quickly fill up the buffer and avoid any re-buffering, bitrate adaptation based on the video buffer is known as **buffer-based adaptation**

In practice players do end up using both network throughput and video buffer for bitrate adaption.

11.22 Throughput-Based Adaptation and its Limitations

Recall that video players keep a buffer of video chunks to absorb any variations in the network. We can model this video buffer as a queue, which gets filled as a new chunk is downloaded and depletes as the video content is played. The buffer-filling rate $C(t)$ is essentially the network bandwidth divided by the chunk bitrate e.g. $C(t) = 10$ if the bandwidth is 10 Mbps and the bitrate of the chunk is 1 Mbps, whereas the buffer-depletion rate $R(t)$ is simply 1, because 1s of video content gets played in 1s. In order to have a stall-free streaming the buffer-filling rate should be greater than the buffer-depletion rate i.e. $C(t) > R(t)$. However $C(t)$ is the future bandwidth and there is no way of knowing it. A good estimate of the future bandwidth is the bandwidth observed in the past. Therefore it uses the previous chunk throughput to decide the bitrate of the next chunk.

11.23 Rate-Based Adaptation Mechanisms

A simple rate-based adaptation algorithm has the following steps:

estimation: The first step involves estimating the future bandwidth. This is done by considering the throughput of the last few downloaded chunks. Typically a smoothing filter such as moving average or the harmonic mean is used over these throughputs to estimate the future bandwidth.

quantization: The continuous throughput is mapped to a discrete bitrate. Basically we select the maximum bitrate that is less than the estimate of the throughput, including a factor in this selection.

We add a factor in the quantization because

- we want to be a little conservative in our estimate of the future bandwidth to avoid any re-buffering
- if the chunks are VBR-encoded their bitrate can exceed the nominal bitrate
- there are additional application and transport-layer overheads associated with downloading the chunk and we want to take them into account

Once the chunk-bitrate is decided player sends the HTTP GET request for the next chunk if the video buffer is not full. If the video buffer is full the player waits for the buffer to deplete before sending the next request. Once the new chunk is downloaded, its download throughput is also taken into account in estimating the next chunk's bitrate and the same process is repeated for downloading the next chunk.

11.24 Issues with Bitrate Adaptation

Rate-based adaptation may end up either overestimating or underestimating the future bandwidth which can lead to selection of a non-optimal chunk bitrate.

11.25 Problem of Bandwidth OVER-Estimation with Rate-Based Adaptation

Suppose the bandwidth is 5 Mbps for the first 20 seconds and is then reduced to 375 kbps. The video player has no way of knowing that the bandwidth has reduced. Meanwhile the video player buffer will deplete and the video will eventually re-buffer. Hence it takes some time for the player to converge to the right estimate of the bandwidth. If the player uses weighted average then it may take even more time to reflect the drop in the network bandwidth.

11.26 Problem of Bandwidth UNDER-Estimation with Rate-Based Adaptation

Suppose another client joins in and starts downloading a large file. Ideally we would expect both clients to end up getting equal network bandwidth i.e. 2.5 Mbps as they are both using TCP. However the client ends up picking a lower bitrate. This is because DASH clients have an ON-OFF pattern in the steady state. This happens when the video client has the buffer filled up and it is waiting to be depleted before requesting the next chunk. During the OFF period the TCP connection reset the congestion window, which impacts the throughput observed for the chunk download as the TCP flow has another competing flow. As a result it ends up picking a lower bitrate. What is worse it would not stabilize at that lower bitrate, because as the bitrate becomes lower the chunk size reduces. In the presence

of a competing flow a smaller chunk size would lower the probability for the video flow to get its fair share. Therefore the player ends up further underestimating the network bandwidth and picks up an even lower bitrate.
While we have seen this problem for DASH and a competing TCP flow, it can also happen in competing DASH players leading to an unfair allocation of network bandwidth.

11.27 Rate-Based Adaptation Conclusion

Rate-based adaptation pick the chunk bitrate based on estimation of available network bandwidth. While the actual available bandwidth is unknown and variable, it uses the past throughput as a proxy for the available bandwidth. This reactive estimation can lead the player to sometimes underestimate or overestimate the bandwidth under different scenarios.

11.28 Bitrate Adaptation Algorithm: Buffer-Based (Optional)

A second alternative is to use the buffer occupancy to inform bitrate selection. In other words the bitrate of the chunk is a function of the buffer occupancy

$$R_{\text{next}} = f(\text{buffer\_now})$$

If the buffer occupancy is low the player should download a low bitrate chunk; if the buffer occupancy increases it should increase the chunk quality accordingly. Thus the bitrate adaptation function should be an increasing function with respect to the buffer occupancy.
Using buffer-based adaptation can overcome the errors in bandwidth estimation that we saw in rate-based adaptation:

- It avoids unnecessary re-buffering. As long as the download throughput is more than the minimum available bitrate the video will not re-buffer. This is because as the buffer occupancy becomes very low, it selects the minimum available bitrate.
- It fully utilizes link capacity and does not suffer from bandwidth underestimation. This is because it avoids the on-off behavior as long as the video bitrate is less than the maximum available bitrate. Once the buffer occupancy is close to full it starts requesting the maximum possible video bitrate.

11.29 Buffer-Based Adaptation Example (Optional)

Here is an example buffer-based function which has three regions of buffer occupancy

reservoir: it corresponds to low buffer occupancy and the player always selects the minimum available bitrate in this region

cushion: the bitrate is a linear function of the buffer occupancy

upper reservoir: when the buffer occupancy is in the upper reservoir region the highest available bitrate is selected

11.30 Issues with Buffer-Based Adaptation (Optional)

Even buffer-based adaptation has some issues:

- In the startup phase the buffer occupancy is zero. Therefore the player will download a lot of low quality chunks which may be unnecessary.

- It can lead to unnecessary bitrate oscillations. For instance if the available bitrate is 1.5 Mbps and the available video bitrates are 1 Mbps and 2Mbps. The player will keep oscillating between these two. This is because if the player downloads 1 Mbps chunks the buffer occupancy will increase and then it will start downloading 2 Mbps chunks. Since the available bandwidth is 1.5 Mbps the buffer occupancy will decrease and then it will switch back again to 1 Mbps.
- It requires a large buffer to implement the algorithm efficiently which may not always be feasible. For instance in the case of live streaming the buffer size is not typically more than 8-16 seconds.

11.31 Bitrate Adaptation Conclusion

In practice most video players use both signals—throughput and buffer occupancy—to decide the bitrate of the next chunk.

12 Lesson 12: Applications: CDNs and Overlay Networks

12.1 Introduction to Content Distribution Networks

The classic way of providing content on the Internet was to put the content on a single publicly accessible web server. This classic method is simple even at scale. However there are three major drawbacks to this traditional approach:

- Users are located all over the globe. No matter where a single massive data center is placed there is potentially vast geographic distance between the users and the data center. That is a problem because the server-to-client packets will have to traverse lots of communication links between many ISPs. There will be annoying interruptions and delays for the user if just one of those links has a throughput lower than the video playout rate.
- Popular videos result not only in a spike in demand but many requests for the exact same video clip. It is wasteful for a single massive data center to be repeatedly sending the exact same data over the same communication link over and over again.
- A single massive data center is that it is also a single point of failure.

Therefore almost all major video-streaming companies use Content Distribution Networks or CDNs. Basically CDNs are networks of multiple geographically distributed servers and/or data centers with copies of content that direct users to a server or server cluster that can best serve the user's request.

12.2 Content Delivery Challenges

There are six major challenges that Internet applications face:

peering point congestion: There's the business and financial motivation to upgrade the "first mile" (like web hosts) and the "last mile" (end users), but not for the "middle mile" where expensive peering points between networks with no revenue happen. These points end up being bottlenecks causing packet loss and increased latency.

inefficient routing protocols: Using only AS hop count without taking into account other factors in addition to its well-documented vulnerabilities to malicious actions, BGP is not an efficient inter-domain routing protocol for the modern Internet.

unreliable networks: Outages happen all the time. Some are accidents. Others are malicious. And others yet are from natural disasters.

inefficient communication protocols: Since TCP requires an ACK for each window of data packets sent, the distance between the server and the end user becomes the overriding bottleneck. Despite all the research into ways to improve TCP, enhancements are slow to get implemented across the whole Internet.

scalability: Scaling up infrastructure is expensive and takes time, and it is difficult to forecast what capacity needs will be.

application limitations and slow rate of change adoption: Even if there are better protocols to meet the needs of the modern Internet, especially with video streaming, historically it took a long time for better protocols to get adopted, e.g. clients need to upgrade their browser to one that supports the new protocol.

12.3 CDNs and the Internet Ecosystem

There are two major shifts that have impacted the evolution of the Internet ecosystem.

- Content and video accounts for the largest fraction of today's Internet traffic. This demand has spurred the development and growth of CDNs in place of traditional single massive data centers.
- The traditional topology (hierarchical) has become flatter due to the popularity of IXPs.

Both of these shifts mean that in general we are seeing more traffic being generated and exchanged locally instead of traversing the complete hierarchy. This process has also been driven by major players (e.g. Google, Facebook) which have shifted the focus from traditional tier-1 ISPs to the edge and the end users. Of course operating a CDN comes with its own challenges e.g. there is the factor of cost. Private CDNs are CDNs that are owned by the content provider such as Google's CDN, which distributes YouTube videos and other content. Other CDNs are third party such as Akamai and Limelight, which distribute content on behalf of multiple content providers.

12.4 CDNs Server Placement Approaches

There is a tradeoff between the two different philosophies

enter deep: the CDNs place many smaller server clusters "deep" into the access networks around the world with the goal of making the distance between a user and the closest server cluster as small as possible, which reduces the delay and increases the available throughput for each user, the downside is that it is much more difficult to manage and maintain so many clusters, Akamai is a good example of a third party CDN with the "enter deep" philosophy

bring home: the CDNs place fewer larger server clusters at key points, typically in IXPs not in access networks—"bringing the ISPs home", there is not as many server clusters to manage or maintain so those tasks are easier, but the downside is that the users will experience higher delay and lower throughput

CDNs also employ a hybrid of these approaches. For instance Google has 16 mega data centers, around 50 clusters of hundreds of servers at IXPs, and many hundreds of clusters of tens of servers at access ISPs. These different server clusters deliver different types of content (e.g. those at access networks store the static portions of search result web pages).

12.5 How a CDN Operates

When a user requests some content the CDN will intercept the request in order to decide which server cluster should service the request. By intercepting the requests with DNS, CDNs have the opportunity to choose where to direct users based on location and/or current conditions.

12.6 CDN Server Selection

There are two main steps in this process

1. map the client to a cluster—a CDN constitutes of geographically distributed clusters with each cluster containing a set of servers
2. a server is selected from the cluster

12.7 Cluster Selection Strategies

The simplest strategy is to pick the geographically closest cluster. This simple strategy can work well in a lot of cases. However it has some limitations.

1. Since the name server system (where the server selection happens) didn't directly interact with the user but rather with the user's local domain name server (LDNS), picking the geographically closest cluster is really picking the cluster closest to the LDNS and not closest to the user. Most of the time this is OK but some customers use a remote LDNS. To overcome this there have been suggestions made to include the client IP in the interaction.
2. The cluster which is geographically closest may not be the best choice in terms of actual end-to-end network performance, because
   - due to routing inefficiencies in routing protocols such as BGP, the network path to the geographically closest server may have a higher RTT than the path to an alternative cluster
   - the path to the geographically closest cluster or the cluster itself could be congested

The second limitation suggests that relying on a static cluster selection policy (e.g. based on geographical distance) can be sub-optimal as the underlying network conditions are dynamic. Instead cluster selection should be based on more real-time measurements of end-to-end performance metrics such as delay.
There are two key aspects to note here

which end-to-end performance metrics to consider: one could use network-layer metrics such as delay, available bandwidth or both, a better strategy could be to use application-layer metrics e.g. re-buffering ratio and average bitrate for video streaming and page load time for web browsing

how to obtain real-time measurements: there are two approaches

active measurements: send a ping request to monitor RTT, however most of the LDNS are not equipped to perform such active measurements, furthermore this would also create a lot of traffic

passive measurements: keep track of the performance metrics based on current traffic conditions, this information can be kept at an aggregate level because IPs under the same subnet are most likely to have similar paths to different clusters thus experiencing similar end-to-end performance, however this requires a centralized controller which has a real-time view of the network conditions between all client-cluster pairs, given the scale of today's networks it can be challenging to design a purely centralized system

Researchers have proposed the design of a distributed system that uses a two-layered system

modeling layer: a coarse-grained global layer operates at larger time scales (a few tens of seconds or minutes) which has a global view of client quality measurements, it builds a data-driven prediction model of video quality

decision layer: a fine-grained per-client decision layer that operates at the millisecond timescale, which makes decisions upon a client request based on the latest but possibly stale pre-computed global model and up-to-date per-client state

12.8 Policy for Server Selection

Routing a client to the least-loaded server is still not optimal. Because a CDN server is responsible for distributing content for a variety of content providers, similarly it could also be serving the same type of content for a variety of content providers. Thus not all servers have all the content at a time. Instead the data is fetched to these servers from an origin server in a lazy manner, which takes additional time although it can cache this content for future request. Therefore if we use the simple load balancing technique for server selection, it could happen that another server already had the content requested, but it was not selected because the client was loaded to a less loaded server. This leads to higher delay and unnecessary requests.
A simple solution is to map the requests based on the content. Essentially requests for the same piece of content can be mapped to the same machine by using some sort of content-based hashing. However when there is a change in the cluster environment such as when a machine fails, the hash table used to map the content to servers needs to be recomputed. An ideal solution would be to only move the objects that were assigned to the server.

12.9 Consistent Hashing

Consistent hashing, which is an example of distributed hash table, has been developed to balance load by assigning roughly the same number of content IDs. It requires relatively little movement of these content IDs when nodes join and leave the system. The main idea behind consistent hashing is that servers and the content objects are mapped to the same ID space. First we map the servers uniformly to the edge of a circle. Next we map the objects to the circle. The successor server for the object ID can be responsible for serving the object. When a server leaves the objects that it was responsible for can now be served by the next successor server. It can be proved that the solution is optimal, which means that least number of keys need to be remapped to maintain load-balance on an average.

12.10 Network Protocols Used for Cluster/Server Selections

We will look at three different network protocols that can be used for server selection—DNS, HTTP redirection, and IP anycast.

12.11 Server Selection Strategies: The DNS Protocol

The advantage of using hostnames is that they can be easily remembered by humans. Its main disadvantage is that hostnames consist of variable characters thus it is difficult for routers to process them. This is where the Domain Name System (DNS) comes in. It helps translate hostnames to IP addresses. The DNS is an application layer protocol that provides a distributed database implemented over a hierarchy of servers. There are other service offered by DNS:

mail server/host aliasing: canonical hostname can be hard to remember and DNS is used to get the canonical hostname (and IP address) for an alias hostname, a host can have one or more names, and if there are two hostnames then they are usually a combination of canonical and mnemonic hostnames, in this case DNS can be used to find the canonical hostname for a given host and also obtain an IP for that host

load distribution: busy websites may be replicated over multiple servers, when a client makes a DNS query the DNS server responds with the entire set of addresses but rotates the address ordering with each reply, this helps in distributing the traffic across servers

A single DNS server containing all the hostnames-to-IP mappings would not be the best option because

- it introduces a single point of failure
- it would be very difficult for a single server to handle all the volume of the querying traffic
- it is based on a centralized database which cannot be close to all querying clients
- maintaining this centralized database would be a big problem as we would have to update a huge database with updates for every single host in the Internet

Due to the above reasons the DNS uses a hierarchical database to solve the scalability problem, which consists of the following types of servers:

root DNS servers: there are 13 servers each of which is a network of replicated servers mostly located in North America, as of May 2019 the total number of server instances is 984

top level domain (TLD) servers: responsible for the top level domains such as .com, .org, .edu, etc and also all of the country top level domains such as .uk, .fr, .jp.

authoritative servers: an organization's authoritative DNS server keeps the DNS records that need to be publicly accessible such as the domain-to-IP mappings for the web servers and mail servers of that organization

local DNS servers: even though this type of servers is not considered as strictly belonging to the DNS hierarchy it is considered central to the overall DNS architecture, each ISP has one or more local DNS servers, hosts that connect to an ISP are provided with the IP addresses of one or more local DNS servers which acts as a proxy and forwards client query to the DNS hierarchy

A client will first send a query to its local DNS which forwards this query to the root server. The root server returns the IP addresses of a list of top level domain servers. Then the local DNS queries one of the TLD servers to receive a referral to the authoritative server. Finally the local DNS queries that authoritative server to receive the domain-to-IP mapping. This process of obtaining the mapping of a hostname to an IP address is known as **name-address resolution**. During the process the host interacts with the DNS hierarchy using two types of queries:

iterative query: the querying host is referred to a different DNS server in the chain until it can fully resolve the request

recursive query: the querying host and each DNS server in the chain query the next server and delegate the query to it

The usual pattern is for the first query from the requesting host to the local DNS server to be recursive and the remaining queries to be iterative.

12.12 More on DNS: Making Responses Faster with Caching

One way to make the DNS resolution faster is to cache the responses. The idea of DNS caching is that, in both iterative and recursive queries, after a server receives the DNS reply of mapping from any host to IP address it stores this information in the cache memory before sending it to the client.

12.13 More on DNS: Resource Records and Messages

The DNS servers store the mappings between hostnames and IP addresses as resource records (RRs). A DNS resource record has four fields (name, value, type, TTL). The Time-to-Live or TTL specifies the time in second a record should remain in the cache. The name and the value depend on the type of the resource record. The most common types are four:

type=A: name is the domain name, value is the IP address of the hostname

type=NS: name is the domain name, value is the authoritative DNS server that can provide the IP addresses for hosts in that domain

type=CNAME: name is the alias hostname, value is the canonical name

type=MX: name is the alias hostname of a mail server, value is the canonical name of that mail server

A DNS message consists of

identification: an ID that is an identifier for the query which allows the client to match queries with responses

flag: contains multiple fields, e.g. a field specifying whether the DNS message is a query or response, a field specifying whether a query is recursive or not

question: contains information about the query being made (e.g. hostname), the type of the query (A, MX, etc.)

answer: if the message is a reply then it will have the resource records for the hostname queried

authority: resource records for more authoritative servers

additional: contains other helpful records, e.g. for an MX record the answer section will contain the resource record for the canonical hostname of the mail server, while the additional section will contain the IP address for the canonical hostname

12.14 Server Selection Strategies: IP Anycast

The main goal of IP anycast is to route a client to the closest server as determined by BGP. This is achieved by assigning the same IP address to multiple servers belonging to different clusters so that multiple BGP routes for the same IP address corresponding to different cluster locations will propagate in the public Internet. The shortest path will then be stored and used for routing packets. Using this strategy can enable CDNs to deliver content using the closest server to a client. However the path closest to the server in terms of BGP route does not always provide the best end-to-end performance due to congestion etc. thus it is not commonly used in practice for CDNs. One particular instance, though not of content delivery, where IP anycasting is used is routing to the DNS server. For example a client that is using public Google DNS servers for domain name resolution will be routed to the closest server in terms of the BGP path thus making the DNS query faster.

12.15 Server Selection Strategies: HTTP Redirection

As the name suggests the protocol works at the HTTP-layer in the network stack. Essentially when a client sends a GET request to a server A, it can redirect the client to server B by sending an HTTP response with a code 3xx and the name of the new server. Note that this would incur at least one additional HTTP request, which can correspond to one or more RTTs for the client to fetch the content. While this makes the protocol inefficient it can be still useful in certain cases. One particular case is for load balancing. For instance a popular video is requested by a large number of clients from the same region. In that case the server can send HTTP redirects to some of the clients. YouTube uses this method to share load within a cluster first and then across clusters if not enough.