Course Notes

Gidon Rosalki

Contents

Part I
Lecture 1 - Introduction to Communication Networks — 2025-10-19

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

1 Administration

Homework will make up 25% of the final grade, and are done in pairs. There are available 5 bonus points if they are typed. Additionally, 2 interviews will be carried out during the semester, on the homework. Failing to pass the interview results in a failure of the homework in question. There are 2 bonus points available for attendance.

We will mostly follow the book *Computer Networking: A Top Down Approach* (5th or 6th edition) by J. Kurose and K. Ross.

1.1 Key concepts

We will study the concepts in networking, such as governing paradigms, key ideas and principles, and fundamental networking tasks. Furthermore, we will also discuss **Domain Specific knowledge** such as how the internet works, specific architectures / protocols, the contents of an IP packet, and "How can a single typo bring down a third of the internet?"

Networking is particularly interested due to the number of unsolved challenges. These include questions such as security and privacy, performance, evolvability (ability to update the networking protocols), and so on.

In this course, we will start with fundamental concepts, and terminology, and finish by building the internet from the ground up.

2 Experience

Let us consider a situation of streaming video. It has been established heuristically that a user will leave if a video does not begin playing within 2 seconds. This is a very short period of time, and does not allow us to download an entire video to the user's computer before beginning. To try and resolve this we have **Dynamic Adaptive Streaming over HTTP** or DASH. This requests chunks of the video at specific bitrates, allowing us to dynamically change the requested bitrate according to the bandwidth of the connection. Overall, we want to ensure a high **Quality of Experience (QoE)**: The network provides user with a level of performance needed for the desired function. For example, a phone call with 100ms of delay results in a very bad user experience, as you are unsure if the other person is pausing, or has finished speaking.

Depending where you are in the world, bandwidth limitation can be significant. For example, Utah to Berlin would take 2 days, 22 hours, and 46 minutes to transmit 100GB of data, where flying would only take 14 hours and 5 minutes.

The internet has become a massively important resource, between conversations, video streaming, information searching, and social media. Despite this, the modern internet is surprisingly fragile, with suboptimal performance, insecure protocols, and unpredictable problems. New challenges keep growing. We saw massive increase in demands on bandwidth in the last 2 decades with the advent of video streaming, and requirements for greater ability to handle slow connections despite requiring high bandwidth with the proliferation of mobile phones.

We consider the internet to "Only Just" work. We have many applications, and developing them is quick and easy. Between Zoom, Facebook, Google, YouTube, this space has rapid technology changes. However, the internet protocols upon which these sit are stagnant, and their methods for routing, congestion control, naming and so on have not changed in decades. However, the technologies upon which these sit, the computers, routers, and so on, have constant innovation. We **want** better internet protocols, and slow change is (finally!) on the horizon.

Consider transmitting information. To transmit information from Jerusalem to New York, the speed of light ensures that it will take at the very least 30.55ms to send a single bit. How much longer depends on the route, the propagation speed of the links, the transmission rate of the links (bits / sec), the number of hops (processing delay), and the congestion. In practice, it generally takes at least 70ms.

Let us consider, how many cycles does the PC execute before it can possibly get a reply? Since the round trip takes $\geq 140ms$, and a PC runs at (approximately) 4GHz, we get approximately 4.2×10^8 cycles, which is a *very* long time. So, communication feedback is always dated, and is fundamentally asynchronous. Even between machines that are directly connected, we can expect a latency of around $200\mu s$, or around 600,000 cycles, which is still a very long time.

3 Computer networks

The ultimate goal of a computer network is to transmit data between end users.

1. End user: These are generally hosts, terminals, stations, phones

2. Links: These connect between nodes, and can be wireless, optical fibre, copper, and so on
3. Nodes: Routers, switches, are the points within the network through which information flows

There are many requirements from a network:

- Size of transfers
- Bidirectionality (or not)
- Latency sensitive (or not)
- Tolerance of jitter (or not)
- Tolerance of packet drop (or not)
- Need for reliability (or not)
- Multicast (or not)

3.1 Computer networks vs Distributed systems

A **Computer Network** is the infrastructure and technologies needed to transmit data between hosts.
A **Distributed System** is the set of hosts using the computer network.

The network itself does **NOT** produce data (aside from control data to manage the network).

4 Telephone network and the internet

The telephone network uses circuit switching. Let us consider how this system works:

1. Establish: The source creates a circuit to the destination, and nodes along the path store connection information. Resources are reserved for the connection, and if the circuit is not available, then a "busy signal" is produced.
2. Transfer: The source sends data over the circuit. There is no destination address, since this is just a closed circuit. There is a continual stream of data.
3. Teardown: The source tears down the circuit when done, freeing resources, and breaking the circuit.

This kind of circuit switching involves physically connecting, and disconnecting wires to make circuits. This was originally done with a human operator, where you would ask them to connect you to a particular line, and they would physically move the wires.

In 1889, Almon Brown Strowger invented the *mechanical switching system*. This allows removing the human operators by having the line transmit information to the mechanical system, which would then automatically connect the lines.

4.1 Timing

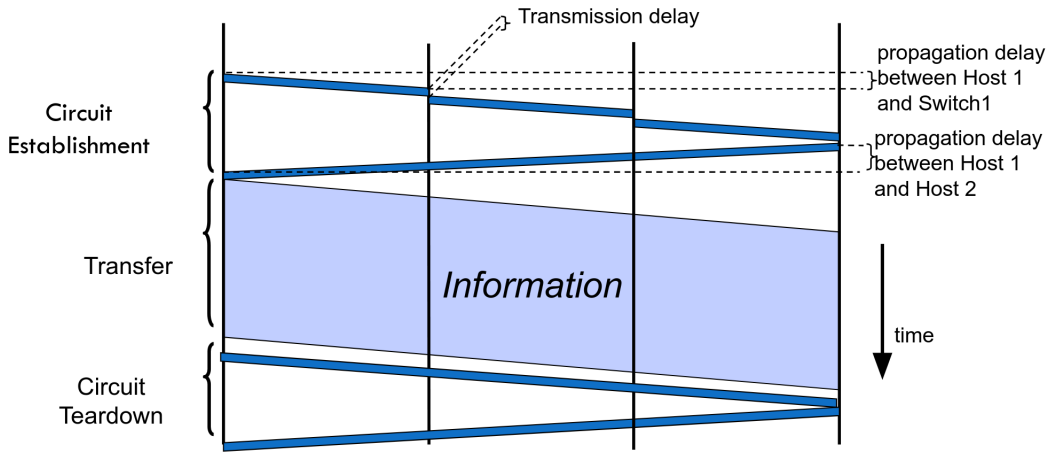


Figure 1: Timing

4.2 Multiplexing

All this talk of having a single circuit per link leaves us with the question of how do we share a link? We achieve this though **multiplexing**, and there are a couple of methods to achieve this. We may have each circuit allocated to

certain time slots, switching between them too quickly for the humans at either end to notice. However, this means that all users have to be well synchronised. We will also need a buffer to ensure that we do not run out of data before the turn of a particular user comes up again.
We may also use frequency division, where every circuit is given different frequencies for transmission of data. This way, they do not interrupt each other, but since we can only use a section of the available data pipeline, the bitrate has been significantly limited.

In our following examples, we will stick to time division multiplexing, for simplicity.

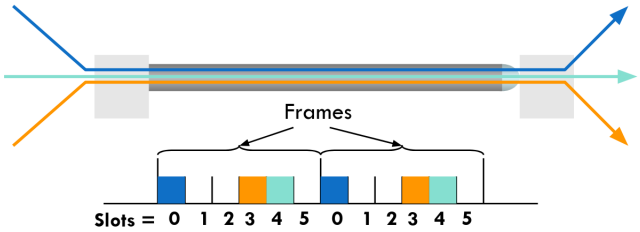


Figure 2: Timing

We divide the time into frames, and frames into slots. The relative slot position inside of a frame determines to which conversation the data belongs. For example, each slot 0 belongs to the blue conversation. However, this requires synchronisation between the sender, and the receiver. We also need to dynamically bind a slot to a conversation. If a conversation does not use its circuit, then capacity is lost.

This sort of system has several strengths. It has predictable performance, with known delays, and no drops. However, it comes with a few weaknesses:

1. It is not resilient to failure: Any failure along the path will prevent transmission. The entire transmission would then have to be restarted. We call this sort of delivery model “all or nothing”

2. Wastes bandwidth: We have a fixed number of slots within each frame. Having fewer uses than slots results in wasted bandwidth, with unused slots. Consider a network application with the peak bandwidth P, and the average bandwidth A. In order for the application to work, it must reserve the peak bandwidth, but then the resulting level of utilisation (average throughput) will be the ratio $\frac{A}{P}$. Some applications have relatively small P/A ratios. Voice over IP might have a ratio of 3:1 or so. Data applications tend to be rather bursty, resulting in ratios of 100 or greater to be rather common. Thus, circuit switching is too inefficient for bursty applications. Generally 2× performance penalties are considered acceptable, but when this gets to orders of magnitude larger, this is too much.

3. Design is tied to the application: This design revolves around the requirements of voice communication, and is not a general feature of circuit switching.

4. Setup time: Every connection requires at least the time of a round trip to set up, which results in a very significant penalty in short transfers, where there is little information to be transmitted.

Part II
Lecture 2 — 2025-10-26

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

5 Taxonomy of communication networks

Communication networks can be classified based off how the nodes exchange information.

5.1 Broadcast communication network

Information transmitted by *any* node is received by *every* other node in the network. This is usually only applicable in LANs (Local Area Networks), when using things such as WiFi, or classical Ethernet. Another such example is a lecture - everyone can hear what every other person in the room says.
This has a few problems. Firstly, there is a very limited range. Every node that wants to listen to node a needs to be directly connected to it. Secondly, one needs to coordinate access to the shared communication medium, two nodes cannot transmit at once (not everyone can talk at the same time in a lecture). Finally, there is no privacy. Everyone can hear everything said, since you cannot say something directly to another person.

5.2 Switched communication network

Here the information is switched, and only sent to specific nodes. Last lecture we saw an example of circuit switched communication networks. There are also *packet switched* communication networks, also known as *datagram* networks. Datagrams are comprised of 2 parts:

1. Header: Instructions to the network for how to handle the packet
2. Payload (body): The actual data being transferred to the recipient

Each packet is switched independently, containing the complete header with the destination address. No resources are preallocated in advance. In this design, each packet needs to be a discrete, relatively small size. This way, if a packet is lost, relatively little data is lost. Additionally, if we were to send large packets, then it would take longer to handle them upon receipt. Smaller packets will reduce latency. However, we also do not want them to be too small, since then we would be sending large amounts of header data for insignificantly small amounts of payload data.

An important impact of using datagram communication is that we can have different packet sending patterns, they may arrive smoothly, there may be gaps as packets arrive in bursts, and so on. Due to bursts, sometimes the transient arrival rate is larger than the transmission rate, even if the long term average arrival rate is smaller than the transmission rate. How do we resolve this? One option is to drop packets. This is obviously not great, since the information is being sent for a reason, and this would just lose parts of it. A better approach is to add a buffer in which we save the excess packets, and read from it instead. Whenever new packets arrive, we add them to the end of the buffer, and read from the start of the buffer. This way, when the burst ends, we are still reading the data from the buffer, and have not lost packets. However, this has its downsides. Even with buffers (even if they could be of infinite size), packets can still be lost. Additionally, buffers can add latency to the operations, especially when the network is busy.

5.2.1 Slots per frame

Let us assume that time is divided into frames, and that frames are divided into slots. *Flows* generate packets during each frame, with a peak number of packets per frame being defined to be P. The average number of packets per frame is defined as A. A single flow must allocate P slots in order to avoid packet drops. However, this is very wasteful, since P might be much larger than A. To avoid this, many flows can exploit the "Law of Large Numbers".

Law of Large Numbers: Consider any probability distribution, and take N samples from said distribution (in this case, that is one set of packets from each flow). The theorem states that the sum of the samples is very close to $N \times A$, and gets percentage-wise closer as N increase. Therefore, sharing between many flows (high aggregation) means that you only need to allocate slightly more than the average A slots per frame.

5.3 Comparison

There are some advantages to **circuit** switching. There is a guaranteed bandwidth, with predictable communication performance, rather than "best effort" delivery, with no real guarantees. There is also a simple abstraction (for the application), with a reliable communication channel between hosts, and no worries about lost, or out of order packets. We also have simple forwarding (for the network devices) where the forwarding is based on a time slot, or frequency, with no need to inspect a packet header. Finally, there is a low overhead per packet, where we forward based off time slot or frequency, and have no headers on each packet.

There are also some disadvantages to circuit switching. There's wasted bandwidth, since bursty traffic leads to idle connections during the silent period. It is also unable to achieve gains from "statistical multiplexing". Connections also

get refused when there are insufficient resources, resulting in blocking, and at times it cannot offer a "good enough" service to everybody. The network must store state, with network nodes storing per connection information. This means that failures are more disruptive. Additionally, we have a communication setup delay, where nothing can be sent before the connection is set up. It is also unable to avoid extra latency for small data transfers.

There is a reliability advantage to packet switching: Routers do not know about individual conversations, and when a router or link fails, it is easy to fail over to a different path. It is also more efficient, thanks to the ability to exploit statistical multiplexing (law of large numbers). Next, there is an advantage to how easy it is to deploy, it is easier for different parties to link their networks together because they are not promising to reserve resources for each other. However, packet switching must handle congestion, which is automatically handled in circuit switching. This is done through adding complexity to routers (buffering, sophisticated dropping), and so it is harder to provide good network services, since we cannot provide such good guarantees with regards to the delay, and bandwidth.

6 Modularity

6.1 Computer modularity

In computers, we partition the system into modules, and abstractions. Well defined interfaces enable flexibility, where we hide the implementation, allowing it to change freely, but limiting their scope to enable the abstraction. We may also extend the functionality of a system by adding new modules. This allows for continuing evolution, and innovation. All of this work isolates assumptions, presenting high level abstractions, making implementation easier, but this comes at the cost of performance (consider the size of an asm hello world, vs the most basic C implementation).

6.2 Network modularity

This is like computer modularity, but the implementation is distributed across many machines (routers and hosts). One must decide how to break the system into modules (layering), where the modules are implemented (on the hosts, or on the network), and where the state is stores (fate sharing).

6.2.1 Layering

From the bottom up, networking has the following tasks:

- Electrons in the wire (more accurately - the electric field in the wire / different voltage levels).
- Bits in the wire
- Packets in the wire
- Delivering packets across a local network (LAN) with local addresses
- Delivering packages across multiple LANs, with global addresses
- Ensuring that packets arrive (handle losses)
- Do something with the data

So how do we handle these tasks? By putting them into "layers". Layering is a particularly strict form of modularity, where the interactions are limited to interfaces above and below.
So, if we add in the layers that we will use:

- Electrons in the wire
- **Bits in the wire (physical layer)**
- Packets in the wire
- **Delivering packets across a local network (LAN) with local addresses (link layer)**
- **Delivering packages across multiple LANs, with global addresses (network layer)**
- **Ensuring that packets arrive (handle losses) (transport layer)**
- **Do something with the data (application layer)**

In short, we have 5 layers, and the top two layers are implemented only by the hosts.

- Application (host)
- Transport (host)
- Network
- Datalink
- Physical

Peers within each layer interact, and communication goes down to the physical network, then from the network peer to peer, before finally rising up to the relevant layer. This means that when my app sends a message, it goes down to the physical layer, and to the router. The router then raises it up to the network layer, understands what there is to do, before it goes back to the physical layer, and is transmitted to the recipient, where the message is raised up to the application layer once more.

6.2.2 Protocol

The *protocol* handles all the communication, but what is a protocol? It is an agreement on how to communicate, handling the exchange of data, and coordinating sharing resources. Protocols specify *syntax* and *semantics*. The *syntax* is how the protocol is structured, handling the format, and message order, where the *semantics* are how to respond to various messages and events.
We have various examples of protocols in real life, such as how we ask questions in class, or having a conversation, or perhaps answering the phone.

Since we want many machines to work together, it is very important to **standardise** the protocol, and for everyone to follow the same protocol. Very small modifications can make very significant differences, and potentially even prevent it from working. Standards allow us to have multiple implementations of the same protocol. To create this standardisation, we have the Internet Engineering Task Force (IETF).

Part III
Tutorial 1 — 2025-10-30

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

7 Introduction

There will be 4 exercises, with submission in pairs. There will be a 5 point bonus if the exercise is typed, and there will be 2 interviews per pair through the semester, which will have a binary pass / fail marking system. A pass is required in order for the homework to be accepted.

The final grade is comprised of 25% from the homework, 75% from the final exam, with a two point bonus for attendance in lectures. 2 lectures can be missed to still get the bonus. In order to pass the course one needs at least 55% in the final exam, a homework average of at least 55%, and a final grade of at least 60%.

We are going to be studying the basic building blocks of modern computer networks. Computers communicate with clearly defined languages, called protocols. We will mostly be discussing established protocols, internet architecture. Towards the end we will begin discussing things likes the performance analysis of network protocols, mobile communication, and security, which has become a bigger issue since the internet was not originally founded on a basis of security.

8 Probability

Today we will be discussing probability. The concept of probability is the measure of the chance that some event will occur, for example, "a coin toss will land on heads". Most things in life (and in CS) are not deterministic, so we need to model when events could happen.
In networking, it is mostly a game of chance, a transmission over WiFi may get lost in background noise, a link can fail, so transmitted messages may not arrive. We will mostly use probability for modelling link / packet failure rates, and analysing efficiency of protocols / networks.

8.1 Definitions

Ω is the **sample space**, with elements ω_i. It describes the states in our system. It may be either *finite*, or *infinite*. As an example, in a coin toss it may be $\Omega = \{H, T\}$. An **event** is a subset $A \subseteq \Omega$, so symbolises a set of possible outcomes. Events are disjoint if they have no intersection, and the complement of the event is every other event in Ω: $\overline{A} = \Omega \setminus A$.
A **probability function** is a function $\mathbb{P} : X \to [0, 1]$ that satisfies the following:

- $\forall A \subseteq \Omega \; \mathbb{P}[A] \in [0, 1]$
- $\mathbb{P}[\Omega] = 1$

For a set of disjoint events $\{A_1, \ldots, A_n\}$ it holds that:

$$\mathbb{P}\left[\bigcup_{i=1}^{n} A_i\right] = \sum_{i=1}^{n} \mathbb{P}[A_i]$$

We have **conditional probability** to denote the chance that B will occur, if we *already know* that A occurred:

$$A, B \subseteq \Omega, \; \mathbb{P}(B) > 0 \; : \; \mathbb{P}(B|A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$
$$\mathbb{P}(A \cap B) = \mathbb{P}(A|B)\,\mathbb{P}(B)$$

This has some useful properties:

- A is independent of B if
$$\mathbb{P}(A|B) = \mathbb{P}(A) \implies \mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$$
- $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$
- Bayes theorem:
$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\,\mathbb{P}(A)}{\mathbb{P}(B)}$$

  We also have complete probability: For a disjoint set
$$\Omega = \{B_i\}_{i=1}^{n}$$
$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A|B_i) \cdot \mathbb{P}(B_i)$$

8.1.1 Example

There are only two types of packets in some network: "Good" packets which survive in the network at least T seconds with probability e^{-T}, and "Bad" packets, which survive in the network at least T seconds with probability e^{-1000T}. The probability of creating a good packet is p.

Example 1. *Assuming a single (either good or bad) packet was created at time $T = 0$, what is the probability that it still exists at time $T = t$?*

Solution. Let us define the following events:

$$\begin{aligned} A &= \text{ a packet survived for } t \text{ seconds} \\ B &= \text{ a good packet was created} \end{aligned}$$

Thus:

$$\begin{aligned} \mathbb{P}(A) &= \mathbb{P}(A|B) \cdot \mathbb{P}(B) + \mathbb{P}\left(A|\overline{B}\right) \cdot \mathbb{P}\left(\overline{B}\right) \\ &= e^{-t} \cdot p + e^{-1000t} \cdot (1-p) \end{aligned}$$

□

9 Random variables

A random variable X is a function $X : \Omega \to \mathbb{R}$. It may be said that X describes some numerical property that our sample space could have. It may either be discrete (like a coin toss), or continuous (such as time). For example, in a coin toss $\Omega = \{H, T\}$, we may define a random variable (called an indicator in this case) $\mathbb{1}_H$:

$$\mathbb{1}_H = \begin{cases} 1, & \text{if } \omega = H \\ 0, & \text{if } \omega = T \end{cases}$$

For a sample space Ω, and a random variable X, we have

$$\mathbb{P}(X = x) = \mathbb{P}(\{\omega \subseteq \Omega : X(\omega) = x\})$$

10 Expected values

The **expected value** is the "averaged" value of a series of experiments:

$$\mathbb{E}[X] = \sum_i x_i \cdot \mathbb{P}(X = x_i)$$

It has some useful properties:

$$\begin{aligned} \mathbb{E}\left[\sum_i a_i X_i\right] &= \sum_i a_i \mathbb{E}[X_i] \\ Var[X] = \mathbb{E}\left[\left[X - \mathbb{E}[X]^2\right]\right] &= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 \end{aligned}$$

There are some useful random variables, which one should know, since they come up a lot:

	Bernoulli	Binomial	Geometric	Poisson
Intuition	Can either fail or succeed	n independent Bernoulli experiments, with X returning the sum	We do independent Bernoulli, until we succeed, X is the number of tries	Counting the number of events that occurred in some period of time
Probability mass function	$\mathbb{P}(X = 1) = p \wedge \mathbb{P}(X = 0) = 1 - p$	$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$	$\mathbb{P}(X = k) = (1-p)^{k-1} p$	$\mathbb{P}(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$
Notation	$X \sim Ber(p)$	$X \sim Bin(n, p)$	$X \sim Geo(p)$	$X \sim Pois(\lambda)$
Exp	$\mathbb{E}[X] = p$	$\mathbb{E}[X] = np$	$\mathbb{E}[X] = \frac{1}{p}$	$\mathbb{E}[X] = \lambda$
Var	$Var[X] = p(1-p)$	$Var[X] = np(1-p)$	$Var[X] = \frac{1-p}{p^2}$	$Var[X] = \lambda$

Table 1:

10.1 Examples

We want to send a message from node S to node D which are connected by a chain of n links. The probability of any link to fail is p (independently). At time $T = 0$, S sends a message to D.

Example 2. *What is the probability that node D got the message?*

Solution. This is the probability that no link would fail. p is the probability of a given link failing, so $1 - p$ is the probability of the link succeeding. To send a message, all n links successfully send the message, the probability of which is independent of each other, so the probability is the union of them all succeeding, ie $(1-p)^n$ □

Example 3. *Assume that if the message does not reach D, then S will send it again until it succeeds. What is the expected number of packets we need to send until node D gets the packet?*

Solution. We define X as "the number of packets to send from S until a packet reaches D successfully" – we note that $X \sim Geo\left((1-p)^n\right)$ and so:

$$\mathbb{E}[X] = \frac{1}{(1-p)^n}$$

□

Example 4. *We now add the following mechanism: Every node keeps on sending the message to its next hop until it reaches it, except the first node that sends only once. What is the probability that node D got the message?*

Solution. All the other hops will eventually succeed, so just the probability that the first succeeds $(1p)$. □

Example 5. *What is the expected number of packets that a single node will send until success?*

Solution. Let us define $X \sim Geo(1-p)$, and so $\mathbb{E}[X] = \frac{1}{1-p}$ □

10.2 Continuous random variables

In the case that the results of our experiment are continuous (for example, measuring time), we need continuous random variables. This is similar to the area under a curve, and the probability of any singleton is 0. We are only interested in probabilities such as $\mathbb{P}(a \leq X \leq b)$. It behaves like discrete random variables, however:

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f(x)\,dx$$

$$\int_{-\infty}^{\infty} f(x)\,dx = 1$$

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x)\,dx$$

$$Var[X] = \int_{-\infty}^{\infty} (x - \mathbb{E}[X])^2\,dx$$

	Uniform	Exponential	Gaussian (Normal)
Notation	$X \sim Uni(a,b)$	$X \sim Exp(\lambda)$	$X \sim N(\mu, \sigma)$
Probability distribution function	$f(x) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a,b] \\ 0, & \text{else} \end{cases}$	$f(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x > 0 \\ 0, & \text{else} \end{cases}$	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Exp	$\mathbb{E}[X] = \frac{a+b}{2}$	$\mathbb{E}[X] = \frac{1}{\lambda}$	$\mathbb{E}[X] = \mu$
Var	$Var[X] = \frac{(b-a)^2}{12}$	$Var[X] = \frac{1}{\lambda^2}$	$Var[X] = \sigma$

Table 2:

11 Memorylessness

Some random variables have the following property:

$$\mathbb{P}(x > S + t | x > t) = \mathbb{P}(x > S)$$

Examples of this are geometric, and exponential random variables. Intuitively, it means that our placement in time does not matter. For example, the time it takes for the first customer to arrive, vs the time it takes for the next customer after the 9th customer.

11.1 Stochastic processes

The random (Stochastic) process: We would like to model a series of events in a non deterministic way, so observing the same process twice might (and should) give us different results (even if the initial starting point is the same). The process models the evolution of a system through time, where we assume that the system is represented by some random variable. In this course, we only care about discrete processes. The total number of successes after n steps in a stochastic process of i.i.d. Bernoulli trials has a binomial distribution.
In an actual network the time a packet leaves an intermediate node (switch/router) is not pre-determined. Hence, we will usually model the packets going in and out of nodes as a stochastic process. We could also model power/traffic demand and available resources to account for them.

Formally: Suppose we have random variables $X_i > 0$, that represent the time it takes for the ith packet to arrive. Let us define:

$$\begin{aligned} S_1 &= X_1 \\ S_2 &= S_1 + X_2 \\ &\vdots \\ S_i &= S_{i-1} + X_i \end{aligned}$$

We call S_i the **arrival epoch** (the specific time at which an event occurs), $N(t) = n$ for $S_n \leq t < S_{n+1}$.

A **counting process** is a random process that counts the number of arrival epochs until time t. A counting process is defined by $\{N_t | t \geq 0\}$, where we have the following:

- $N_0 = 0$
- $N_t \geq 0$
- $s \leq t \implies N_s \leq N_t$ (monotonicity), and if $s < t$, then $N_t - N_s$ is the number of events that occurred in $(s, t]$

Consider as an example, the number of customers that arrive in a store.

A **Poisson process** $\{N_t | t \geq 0\}$, with a rate of λ is a counting process, where in addition:

- Independent increments: Number of events in any two disjoint intervals is independent
- Stationary increments: The number of events in any interval of length t is a Poisson random variable with parameter λt (depends on the interval's length and not on its timing). ie. $\mathbb{E}[N_t] = \lambda t$
- No bunching: The probability of > 1 arrivals in a tiny interval is negligible

Given this, is the number of people boarding the bus a Poisson process?
No. The passengers board in groups (at bus stops), and during rush hour more people board.

The inter-arrival times between events in a Poisson counting process are I.I.D and have an exponential distribution with parameter λ.

Example 6 (Poisson process)**.** *In each interval of length T, the number of transmitted packets is a Poisson random variable with parameter gT. What is the probability to have only a single packet transmitted?*

Solution. Recall $X_o \sim Poi(gt)$, and $\mathbb{P}_{t=T}(X = i) = \frac{(gT)^i}{i!} e^{-gT}$. Let us define a random variable X that represents the number of packets transmitted at some interval. Then, $X \sim Poi(gT)$ and we get:

$$\mathbb{P}_{t=T}(X = 1) = gT \cdot e^{-gT}$$

□

Part IV
Lecture 3 — 2025-11-02

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

12 Protocols

The *protocol* handles all the communication, but what is a protocol? It is an agreement on how to communicate, handling the exchange of data, and coordinating sharing resources. Protocols specify *syntax* and *semantics*. The *syntax* is how the protocol is structured, handling the format, and message order, where the *semantics* are how to respond to various messages and events.
We have various examples of protocols in real life, such as how we ask questions in class, or having a conversation, or perhaps answering the phone.

Since we want many machines to work together, it is very important to **standardise** the protocol, and for everyone to follow the same protocol. Very small modifications can make very significant differences, and potentially even prevent it from working. Standards allow us to have multiple implementations of the same protocol. To create this standardisation, we have the Internet Engineering Task Force (IETF).

Below is an example for the HTTP application level protocol. This is built upon the TCP packet protocol, and each layer of the image represents that layer's abstraction. The HTTP message, and the TCP message are both simply that, but the TCP segment is made up of IP packets sent between routers, which are built upon different physical interfaces.

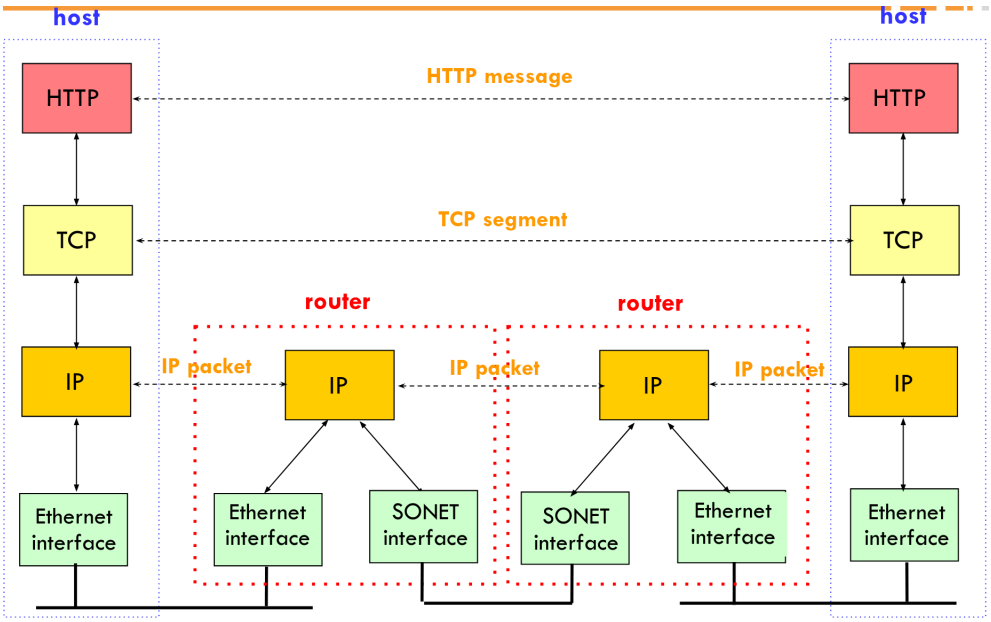


Figure 3: HTTP message

Each layer has to add its own header. The common case is that we add 20 bytes for the TCP header, 20 for the IP header, and 14 for the Ethernet header, resulting in an overall overhead of **54 bytes**.

13 OSI model

The OSI model is a model through which we describe the layers. It is comprised of 5:

1. Physical
2. (Data) Link
3. (Inter) Network
4. Transport
5. Application

Each layer involves discussing its *service*, ie what it does, the *service interface*, which is how to access the service, and the *protocol*, or how the peers communicate. A layer can have many implementations, allowing innovation. For example, HTTP in the application layer, vs SMTP.

13.1 Physical layer

Service: Move bits between 2 systems, connected by a physical link. This handles low level physics, that is scary to us computer scientists.
Interface: Specifies how to send, and receive bits, including things such as timing, and how to avoid both transmitting at the same time, thus losing data.
Protocols: A coding scheme that is used to represent bits, voltage levels, the duration of a bit, and so on.
Examples: Optical, wireless, Ethernet cables...

13.2 (Data) link layer

Service: Enable hosts to exchange message "frames", using abstract (but local) addresses, rather than direct physical connections. Here there are many hosts on the same LAN, and they send messages to their MAC addresses, rather than down the physical wire.
Interface: Send frames to other local hosts (ie, hosts on the same LAN), and receive messages addressed to this host.
Protocols responsible for: Media Access Control (MAC)
Examples: Ethernet, 802.11 (wireless protocol), Frame Relay, ATM

13.3 (Inter) Network

Service: Deliver packets to specified destinations. This is between local networks, not just on *our* local network. This uses abstract global addresses, across multiple layer 2 networks. For example, this may be from Ethernet, to 802.11, to Frame Relay, to ATM.
Interface: Send packets to specified internetwork destinations, and receive packets destined for the end host.
Protocols responsible for: Routing
Examples: IP (currently on IPv4, but about to move for IPv6! It's only taken us 20+ years...)

13.4 Transport layer

Service: Communication between *processes*. This performs multiplexing of communication between hosts, and perhaps includes ensuring reliability of transportation, and rate adaptation for when the network is too busy.
Interface: Send a message to a specific process, at a given destination. A local process will receive messages sent to it.
Protocols responsible for: Reliability, flow control, packetisation of large messages.
Examples: TCP, UDP

13.5 Application layer

Service: Any service provided to the end user
Interface: Depends on the application
Protocol: Depends on the application
Examples: Zoom, SMTP (email), HTTP (websites), Halo, BitTorrent, and many, many, more

13.6 Narrow waist

This returns us to the narrow waste problem. As we saw, all the layers have multiple possibilities, and protocols, aside from the Network layer, which is just IP. This is considered the narrow waist. This provides massive interoperability, but has massively limited innovation. This only really occurs at this layer (there are also some problems at the transport layer) since this is the lowest level layer, where each package is required to get to many other pieces of hardware. A new application layer protocol is fine, since it's handled purely in the application software, and in the link, and physical layers it's less of an issue, since it only matters as far as getting the packet to the next piece of hardware. However, the network layer sends a packet end to end, often with hardware optimisations, thus providing severe restrictions on our ability to update the protocol involved.

14 Building the internet from the bottom up

As discussed, the OSI model has 5 layers. We will ignore the application, and physical layers, since they are not the focus of this course, we will return to the transport layer, and the network layer first requires interconnected LANs, so:

14.1 Challenge 1: communicating in a LAN

14.1.1 Terminology

End hosts, and relays, are called *nodes*. Communication channels that connect adjacent nodes along communication paths are called *links*. There are wired, and wireless links, and also logical links (for broadcasting). A *layer 2 packet*

is a *frame*, and it encapsulates a datagram. The data link layer has the responsibility of transmitting datagrams from one node, to an adjacent node, over a link.

The links may all be different, for example a first link over Ethernet, the second 802.11, and so on. Each link provides different services, a first may provide reliability, where the second does not.

14.1.2 Single link: Services

The channel is a shared medium, and "MAC" addresses are used in frame headers to identify the source, and destination. Note that this is **distinct** from IP addresses. It would also be logical for this to provide reliable delivery between adjacent nodes. For example, perhaps it should provide error detection / correction, and retransmit when necessary. This should be seldom used when we have a low bit error link (such as fibre, or twisted pairs). Contrast this to wireless links, which would have exceedingly high error rates, and would thus have to handle this.

We implement the link layer at each and every host, in its "adaptor", often a network card. This could be an Ethernet card, an 802.11 card. The network card implements the link, and physical layers. It is attached into the host system's buses, and is a combination of hardware, software, and firmware.

When communicating, the sending side encapsulates datagrams in frames, and adds error checking bits, along with ensuring reliable data transfer. The receiving side then looks for errors (for example, in the error checking bits), and extracts the datagram, passing it on to the upper layers, through the system buses, at the receiving side.

This all seems fairly simple, but consider. What happens when there are 2 or more simultaneous transmissions by the nodes, on the same link? This same link is a single, shared, broadcast channel, and a node receiving two or more signals simultaneously will result in a **collision**. To handle this we have the *Multiple Access Protocol*. This is a distributed algorithm, that determines how nodes share a channel. That is to say, it determines when a node can transmit. It is important to note that the communication about channel sharing must use the channel itself, since we do not have a second channel in which to coordinate. Throughout the 90s, the channel was often a bus topology. This means that all nodes are effectively connected to the same wire. This also means that all nodes are in the same collision domain, and their messages may all collide with each other. Today, we tend to use the **star** topology. There is an **active switch** in the centre, which connects individually over a separate Ethernet protocol link to the nodes. This way, nodes do not collide with each other. Everything only needs to handle communication with 2 nodes on the same link, which is much simpler than many more, and enables much higher data transfer rates. Switches will be discussed more later.

14.1.3 Multiple Access Protocols

Let us consider an ideal MAP, of rate R bps:

1. When one node wants to transmit, it can send at rate R
2. When M nodes want to transmit, each can send at average rate $\frac{R}{M}$
3. It is fully decentralised, so there is no special node to coordinate transmissions
4. There is no synchronisation of clocks, or slots
5. Since we are having very pretty dreams, let's make it simple as well, while we are at it

A *Random Access Protocol* is when a node has a packet to send, it can transmit at full channel data rate R, with no a priori coordination among nodes. Two nodes transmitting at once causes a collision, so a random MAC protocol specifies:

- How to detect collisions
- How to recover from collisions (eg, via delayed retransmission)

There are some examples of these random MAC protocols, such as slotted ALOHA, ALOHA, CSMA, CSMA/CD, CSMA/CA.

Part V
Tutorial 2 - ALOHA — 2025-11-06

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

15 Introduction

16 Open-Systems Interconnect (OSI) model

The OSI model provides an abstract way of representing the functions needed in a communication system. It divides the networking tasks into separate layers. This model has 7, but sometimes there are 5, depending on how they are divided. Each layer uses the abstractions provided by the layer below it, and provides services to the one above it.

#	Layer Name	Data Unit	Function
1	Physical	Bit	Transmit and receive raw bits over some physical medium
2	Data link	Frame	Transmission of data frames between 2 nodes
3	Network	Packet	Structure and manage multi node network (addressing, routing, etc.)
4	Transport	Segment	Transmission (reliable or not) of segments between points in the network (ack'ing, etc.)
5	Session	Data	Manage a continuous communication session
6	Presentation	Data	Translation of data between a networking and an application (e.g. encryption / decryption)
7	application	Data	High level APIs between network applications (e.g. GET, POST)

Table 3: OSI 7 layer model

In this tutorial, and the next, we are going to focus on layer 2, but in the future, we will discuss all the layers (aside from 1, we can leave that for the physicists and the engineers).

Examples:

- Layer 7 (Application): DNS (turn website names to addresses), HTTP (send web pages)
- Layer 4 (Transport): TCP (reliably send segments), UDP (unreliably send segments)
- Layer 3 (Network): IP (sending and receiving of packets)
- Layer 2 (Data Link): MAC

17 Layer 2 - Data link

These operate over a share medium, where all nodes in the network broadcast and listen on the same channel. When a node broadcasts, it uses the full link bandwidth. If two nodes broadcast at the same time, there is a **collision**, and the data is lost. We ignore the noise on the channel.

Consider a protest. Only one person can talk at once, since when multiple people start talking, it all devolves into noise, and no information can be gathered (other than maybe lots of people have lots of things to say).

17.1 Definitions

Name	Explanation	Notation	Units
Bandwidth	The maximum rate of data transfer on the communication channel	B	$\frac{bit}{sec}$
Throughput	The fraction of the bandwidth used to send traffic	-	None
Goodput	The fraction of the bandwidth used to successfully send traffic	η	None

Table 4: Definitions table

Note that

$$\eta = \frac{\text{total average effective bandwidth usage}}{\text{total bandwidth}} = \frac{\text{total average effective time}}{\text{total time}}$$

So, assuming transmission uses the full bandwidth, $0 \leq \eta \leq 1$. An example of throughput vs goodput would be the background noise within a class. Throughput is the amount of noise made, but the goodput is how much information is communicated.

17.1.1 Example question

Let us assume that the link bandwidth is $B = 5$. Given a packet size of 30 bits, how long does it take to send a packet?

$$T = \frac{|\text{packet}|}{B} = \frac{30}{5} \cdot \frac{\text{bit}}{\frac{\text{bit}}{\text{second}}} = 6s$$

Assume that we need to transmit 3 packets, with a size of 30 bits, within 2 seconds. What is the required bandwidth?

$$B = \frac{|\text{packet}|}{B} = \frac{3 \cdot 30}{2} = 45\frac{\text{bits}}{\text{sec}}$$

18 ALOHA

The ALOHA protocol is a protocol, originally developed in Hawaii for creating a local network that could span their islands. It is the first wireless network (undersea cabling was not good enough yet), and is the basis of the Ethernet, WiFi, and the 1G mobile network protocols. It makes the following assumptions:

- N nodes in the network. We usually analyse for $N \to \infty$
- B - link bandwidth
- L - number of bits in a frame
- Time is divided into slots of length $T = \frac{L}{B}$ T is the time required for a message to be transmitted. Successful transmissions use the entire bandwidth, and the entire time slot.
- A node can start transmitting *only* at the beginning of a slot. If a message arrives in the middle of a slot, then it will be sent at the beginning of the next slot
- **Collisions detection**: When a node *finishes* transmitting a message, it can detect collisions (if there are, it can then resend the message in the future)
- Propagation (physical delay) time is discarded

There are two main types of ALOHA that will be discussed, pure vs slotted:

- Pure ALOHA assumes that every node has its own clock (since synchronising clocks is an expensive task)
- Slotted ALOHA assumes that all nodes are *somehow* synchronised. We will not discuss how.

18.1 Binomial vs Poisson ALOHA

Let there be X_p, the number of transmission messages in a time segment T.

18.1.1 Binomial assumption

We assume that there are n end points, with probability p:

$$X_P \sim Bin\,(n, p)$$

18.1.2 Poisson assumption

There are ∞ end points, together with a rate of λ per T:

$$X_P \sim Poi\left(\lambda T\right)$$

Recall that if n is large, and p is small, then $Bin\left(n,p\right) \approx Poi\left(\lambda\right)$, where $np = \lambda$.

We're going to analyse 4 models, Binomial slotted, and pure ALOHA, and Poisson slotted, and pure ALOHA.

18.2 Binomial slotted

We assume that at any given time, all nodes have something to transmit, and each node tries to transmit in each slot with probability p_{send}. What is the goodput of this protocol? Note that each successful message effectively uses the entire bandwidth.

Since all the slots are the same, let us consider a single time slot. We need to find how much time (the expectation), was spent on *successful transmission* in this slot. If we define X_p as the number of transmissions in a **single** time slot, then we want to find

$$P_{suc} = \mathbb{P}\left[X_p = 1\right]$$

Where $X_p \sim Bin\left(n, p_{send}\right)$ The probability for a successful frame:

$$P_{bin} = \binom{n}{x} p^x \left(1-p\right)^{n-x}$$
$$P_{suc} = \mathbb{P}\left[X_p = 1\right] = np_{send}\left(1-p_{send}\right)^{n-1}$$

Let us denote T_{suc} as the time spent on successful transmission in a single slot. What is $\mathbb{E}\left[T_{suc}\right]$?

$$\mathbb{E}\left[T_{suc}\right] = T \cdot P_{suc} + 0 \cdot \left(1 - P_{suc}\right) = T \cdot P_{suc}$$

and the goodput of the network will be

$$\eta = \frac{\mathbb{E}\left[T_{suc}\right]}{T} = P_{suc} = np_{send}\left(1-p_{send}\right)^{n-1}$$

To find the relation between successful, and unsuccessful slots. We can ask how many slots on average do we have before there is a successful slot? Let us define X_p as the number of tries. We will note that

$$X_p \sim Geo\left(p_{suc}\right) \wedge \mathbb{E}\left[X_p\right] = \frac{1}{p_{suc}}$$

So the goodput of the network is

$$\eta = \frac{T}{T \cdot \left(\frac{1}{P_{suc}}\right)} = P_{suc}$$

We want to maximise the goodput of η where:

$$\eta = \mathbb{P}\left[X_p = 1\right] = np_{send}\left(1-p_{send}\right)^{n-1}$$

Let us denote $p = p_{send}$, and solve for p:

$$\begin{aligned}
\frac{d}{dp} np\left(1-p\right)^{n-1} &= n \cdot \left(1-p\right)^{n-1} + np \cdot \left(n-1\right)\left(1-p\right)^{n-2} \cdot \left(-1\right) \\
&= n \cdot \left(1-p\right)^{n-2} \cdot \left[\left(1-p\right) - p \cdot \left(n-1\right)\right] \\
&\vdots \\
\implies \frac{d}{dp}\eta &= 0 \\
\implies p &= \frac{1}{n}
\end{aligned}$$

We get that $p^* = \frac{1}{n}$ maximises η.

How good is p^*? Well, let us set p_{send} as above, and take $n \to \infty$. We want to maximise the goodput of η.

$$\begin{aligned}
\lim_{n\to\infty} \eta\left(p^*\right) &= \lim_{n\to\infty} np^*\left(1-p^*\right)^{n-1} \\
&= \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^{n-1} \\
&= \frac{1}{e} \\
&\approx 0.37
\end{aligned}$$

Slotted ALOHA gives us freedom to fully distribute our system, but 0.37 is not very efficient.

18.3 Pure ALOHA

Here, each station transmits at the beginning of when *it* thinks the slot begins. Since the clocks are not synchronised, this can be at any time. Given a fixed packet length of T, then there is a vulnerable time period (when there can be a collision) of $2T$: For T before transmission begins, and T from when the packet begins. If there is a collision, then the transmitter waits a random amount of time before retransmitting.

What is the probability of a successful frame?

$$\begin{aligned} P_{suc} &= \mathbb{P}\left[\text{some node transmitted}\right] \cap \mathbb{P}\left[\text{no other node transmitted in 2 overlapping slots}\right] \\ &= \mathbb{P}\left[\text{some node transmitted}\right] \cdot \mathbb{P}\left[\text{no other node transmitted in 2 overlapping slots}\right] \\ &= np_{send}\left(\left(1-p_{send}\right)^2\right)^{n-1} \\ &= np_{send}\left(1-p_{send}\right)^{2(n-1)} \end{aligned}$$

Goodput: Let us define a random variable

$$T_{suc} = \begin{cases} T, & \text{successful slot} \\ 0, & \text{else} \end{cases}$$

Therefore $\mathbb{E}\left[T_{suc}\right] = T \cdot P_{suc}$, and therefore

$$\eta = \frac{\mathbb{E}\left[T_{suc}\right]}{T} = P_{suc} = np_{send}\left(1-p_{send}\right)^{2(n-1)}$$

The maximum lies in $p_{send} = \dfrac{1}{2n-1}$, and the goodput is $\dfrac{1}{2e}$, which is **half** the goodput of slotted ALOHA.

19 Questions

Exercise 1. *Host C and host D are connected in 2 different networks, the upper network runs Slotted ALOHA, and contains two more nodes, where the lower network runs pure ALOHA, and contains 3 more nodes. D never sends messages, and when C has a message to send to D, it chooses which network to use, and sends it with a probability of q. C chooses the upper network with a probability of p, and in every slot, each other node X sends with a probability of q.*

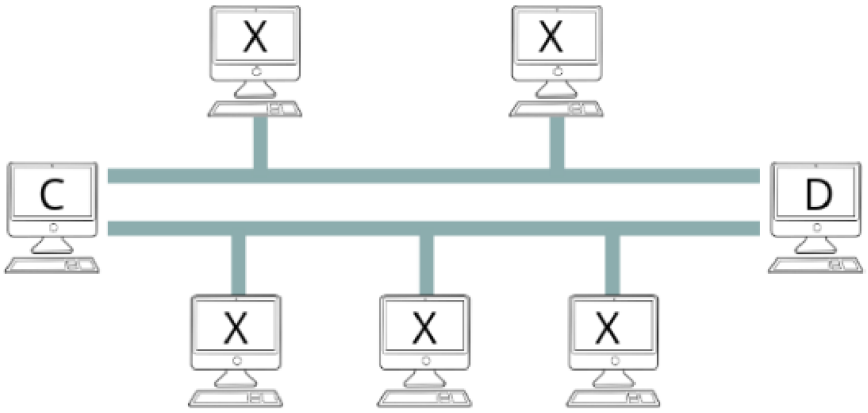


Figure 4: ALOHA network

C has one attempt to send the message to D, what is the probability that the message will arrive at D?

Solution. For the upper network, which is slotted,

$$\begin{aligned} P_{suc} &= \mathbb{P}\left[\text{C sends a message}\right] \cdot \mathbb{P}\left[\text{the others do not send}\right] \\ &= q \cdot \left(1-q\right)^2 \end{aligned}$$

For the lower network, which is pure

$$\begin{aligned} P_{suc} &= q \cdot \mathbb{P}\left[\text{No other transmission for two slots}\right] \\ &= q \cdot \mathbb{P}\left[\text{No other transmission for two slots}\right] \\ &= q \cdot \left(\left(1-q\right)^2\right)^3 \\ &= q \cdot \left(1-q\right)^6 \end{aligned}$$

So

$$\begin{aligned} p_{suc}^{total} &= p \cdot p_{suc}^{upper} + (1-p)\, p_{suc}^{lower} \\ &= q\left(p\,(1-q)^2 + (1-p)\,(1-q)^6\right) \end{aligned}$$

□

Now C tries to send at the beginning of each slot. What is the goodput of the network?

Solution.

$$\begin{aligned} P_{suc}^{lower} &= \binom{3}{1} q\,(1-q)^{2+2} \\ &= 3q\,(10q)^4 \\ P_{suc}^{upper} &= \binom{2}{1} q\,(1-q) \\ \eta &= (1-p)\,P_{suc}^{lower} + p \cdot P_{suc}^{upper} \end{aligned}$$

□

Exercise 2. *There are $2n$ nodes in the network. The first n run pure ALOHA, with slots of length $2T$. The last n run slotted ALOHA with slots of length T. Every node sends a message at the beginning of each slot with probability p, and the message that is sent takes all the slot.*

We choose a random slot. What is the probability of a node that runs Slotted ALOHA to send a message successfully in this slot?

Solution. We need the end host to send a message, which is probability of p, that the other nodes will not send messages in this slot: $(1-p)^{n-1}$, and that the pure ALOHA will not send for **three** slots: $(1-p)^{3n}$. Thus:

$$p_{suc} = p \cdot (1-p)^{n-1} \cdot (1-p)^{3n}$$

□

We choose a random slot. What is the probability of a node that runs Pure ALOHA to send a message successfully in this slot?

Solution. Let us divide into cases, where the message overlaps a single slot, and where it overlaps 2 slots.

- In the case where the message overlaps a single slot, the probabilities are as follows: the end host sends a message: p, the other slotted ALOHA nodes will not send any message at this slot: $(1-p)^n$, the other pure ALOHA nodes will not send any messages for two slots: $(1-p)^{2(n-1)}$, thus

$$P_{suc} = p \cdot (1-p)^n \cdot (1-p)^{2(n-1)}$$

- In the case where the message overlaps two slots, then the probabilities are: the end host will send a message: p, the other slotted nodes will not send any message for **two** slots: $(1-p)^{2n}$, the other pure ALOHA nodes will not send any messages for two slots $(1-p)^{2(n-1)}$, and thus

$$P_{suc} = p \cdot (1-p)^n\,(1-p)^{2(n-1)}$$

□

Let us now assume that the nodes that ran pure ALOHA start running slotted, with slots of length T, such that the slots are synchronised. What is the goodput of the network?

Solution. If we look at some slot (of $2T$), there are some options for successful transmissions within it:

1. Successful packet of length $2T$
2. Successful packet of length T, only in the first slot
3. Successful packet of length T at the second slot only
4. 2 successful packets of length T, one in each slot

Let us consider each case:

1. Successful packet of length $2T$: For this, we need exactly one of the n nodes running slotted ALOHA, will attempt to send:
$$\binom{n}{1} p\,(1-p)^{n-1} = np\,(1-p)^{n-1}$$
All the other nodes running pure ALOHA will not send: $(1-p)^{2n}$ Both these events are independent, so:
$$np\,(1-p)^{n-1}\,(1-p)^{2n}$$

2. Successful packet of length T, only in the first slot: For this we need exactly one node with length T to transmit:
$$\binom{n}{1} p\,(10p)^{n-1} = np\,(1-p)^{n-1}$$
All the other nodes will not send: $(1-p)^{n}$, and we need the second slot to fail, which is the inverse of one T node will send: $1 - np\,(1-p)^{n-1}$. We multiply the independent factors and get:
$$np\,(1p)^{n1}\,(1p)^{n}\left(1np\,(1p)^{n1}\right)$$

3. Same as the successful packet in the second slot only

4. 2 successful packets of length T, one in each slot: For this we need exactly one of the nodes transmits a packet of length T in the first slot:
$$\binom{n}{1} p \cdot (1-p)^{n-1} = np \cdot (1-p)^{n-1}$$
We also need the same thing in the second slot, and that all the nodes that transmit $2T$ will not send: $(1-p)^{n}$. Since both events are independent, we can multiply them:
$$\left(np\,(1-p)^{n-1}\right)^{2}(1-p)^{n}$$

This leaves us with an overall goodput of
$$\eta = A + \frac{1}{2}B + \frac{1}{2}C + D$$

□

Part VI
Lecture 4 — 2025-11-9

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

20 Multiple Access Domains

Consider when there is a single shared broadcast channel, with two or more simultaneous transmissions by nodes. There is a collision if a node receives two or more signals at the same time. Multiple Access Protocols (MAPs) are distributed algorithms that determine how nodes share this channel, or in other words, when a node can transmit. The communication about sharing this channel takes place over the channel itself.

20.1 Broadcast domains

20.1.1 Wired

It used to be that all nodes would be effectively connected to a single wire, potentially indirectly through a *hub*. We do not especially use this today, since this had all nodes in the same collision domain, and massively reduced the amount of data able to be transmitted. Today we mainly use a star topology, with an active *switch* in the centre. Each node runs a separate connection to the switch, massively increasing available bandwidth.

20.1.2 Wireless

Wireless networks suffer from the same problems with collisions, with the additional issues of potentially some nodes not receiving the messages, due to signal attenuation over distance, or potential geographic features blocking the signal.

20.2 MAPs

An ideal MAP is as follows:

1. When one node wants to transmit, it can send at rate R
2. When M nodes want to transmit, each can send at an average rate $\frac{R}{M}$
3. It is fully decentralised, there is no node that coordinates transmissions
4. There is no clock / slot synchronisation
5. It is as simple as possible

There are also **Random Access Protocols**, where when a node has a packet to send, it transmits at full channel data rate R, with no a priori coordination between the nodes. Two or more transmitting nodes result in collisions. Therefore, the RAC specifies how to detect collisions, and how to recover from them. Some examples include slotted ALOHA, and ALOHA.

20.2.1 Goodput and throughput

Definition 20.1 (Goodput)**.** *The proportion of successful slots (as in, in how many there was a transmission, without a collision)*

Definition 20.2 (Throughput)**.** *The proportion of slots in which there was a transmission (not necessarily successful)*

20.2.2 Slotted ALOHA

We assume that all frames are the same size, and that time is divided into equal size slots (each slot is the time to transmit one frame). Nodes start to transmit only at the beginning of the slot, and they are synchronised. If 2 or more nodes transmit in a slot, all nodes detect the collision.
They operate as follows: When a node needs to transmit a frame, it transmits it in the next slot. If there is no collision, great! If there is a collision, then the node retransmits the frame in the next slot with a probability p, repeated until it succeeds.

This comes with the benefits that a single active node can continuously transmit at the full rate of the channel. It is also highly decentralised, since only the slot times need to be in sync across the nodes. It's also beautifully simple. However, there are inevitably collisions, which result in wasted slots. Additionally, there are often many idle slots, and nodes may be able to detect a collision in less than a slot time. Finally, how are we meant to synchronise the clocks?

Suppose that there are N nodes, each with many frames to send, each transmitting in a given slot with probability p. The probability that a given node succeeds in a given slot is

$$p\,(1-p)^{N-1}$$

and the probability that *some* node succeeds is

$$Np\,(1-p)^{N-1}$$

To find the maximum efficiency, we want to find the p^* that maximises $Np\,(1-p)^{N-1}$. For many nodes, taking the limit of $Np\,(1-p)^{N-1}$ as $N \to \infty$ gives a maximum efficiency of $\frac{1}{e} \approx 0.37$, or the channel only sends useful transmissions 37% of the time, which is *not great*.

20.2.3 Pure ALOHA

Pure, or unslotted ALOHA, is simpler, with no synchronisation. Each host behaves as it if it running slotted ALOHA, but the slots are local to itself, rather than shared over the network. It sends a frame with a probability of p, at the beginning of a slot. This increase the probability of a collision, each frame that is sent at time t_0 will collide with frames sent in slots that start anywhere from $(t_0 - 1, t_0 + 1)$.

The goodput is given

$$\begin{aligned}
\mathbb{P}\left[\text{success by given node}\right] &= \mathbb{P}\left[\text{node transmits}\right] \cdot \mathbb{P}\left[\text{no other node transmits in interacting times}\right] \\
&= \mathbb{P}\left[\text{node transmits}\right] \\
&\cdot \mathbb{P}\left[\text{no other node transmits in } (t_0 - 1, t_0)\right] \\
&\cdot \mathbb{P}\left[\text{no other node transmits in } (t_0, t_0 + 1)\right] \\
&= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1} \\
&= p \cdot (1-p)^{2(N-1)}
\end{aligned}$$

Choosing the optimal p, and $n \to \infty$ results in even **worse** than slotted ALOHA, with a result of $\frac{1}{2e} \approx 0.18$.

20.3 CSMA

Carrier Sense Multiple Access type protocols *listen* before they transmit. If a channel is sensed as idle, transmit the entire frame, and if it is busy, defer the transmission. This is like hearing someone else is talking, and waiting for them to finish before you speak.

20.3.1 CSMA/CD

Collisions can still occur, the propagation delay means that two nodes may not hear each others transmission. If there is a collision, then the entire packet transmission time is wasted. It should be noted that distance and propagation delay both play a significant role in determining collision probability.

So how do we add collision detection? In a wired LAN we measure the signal strength, and compare the strengths of transmitted vs received signals. This is harder in wireless, since the received signal strength is overwhelmed by local transmission strength. Should we detect a collision, then the colliding transmissions are aborted, reducing the channel wastage.

Adding this in, when a host has a packet to transmit, it checks that the line is quiet before transmitting. If it detects a collision, then it should stop transmitting, wait a random time, and return to the beginning. This together is called **CSMA/CD**.

To ensure that a packet is transmitted without a collision, a host must be able to detect a collision *before* it finishes transmitting a packet. As a result, we can see that for a host to detect a collision before it finishes transmitting a packet, we require the transport time to be

$$\text{Transport time} > 2 \cdot \text{ propagation delay}$$

In other words, there is a minimum length packet for CSMA/CD networks, and that is at least 2 times the propagation delay.

Let us consider the goodput. The time is slotted, and all packets are the same length. A time slot is equal to 2· propagation time. In any given time slot, a host will decide to transmit (or not) with the probability p. So:

$$\eta = \frac{\text{Time taken to send data}}{\text{Time taken to send data + overhead}}$$

Therefore, to find the goodput $\alpha(p)$, we take the probability that exactly one node transmits in a given slot

$$\begin{aligned}\alpha(p) &= \binom{N}{1} p(1-p)^{N-1} \\ \frac{d\alpha}{dp} &= N(1-p)^{N-1} - pN(N-1)(1-p)^{N-2} \\ \implies p = \frac{1}{N} &\implies \alpha_{max} \approx 37\%\end{aligned}$$

We will define A to be the expected number of time slots **wasted** before a packet is successfully transmitted. Consider for example a coin X, such that

$$\mathbb{P}[X = H] = 0.4 \stackrel{def}{=} a$$

Therefore, the expected number of coin tosses before the first head is $\frac{1}{0.4} = 2.5$, and therefore $A = 1.5$ unsuccessful attempts, which would then be followed by one successful one.
Let us call the minimum length packet TRANSP. So

$$\begin{aligned}\eta_{CSMA/CD} &= \frac{TRANSP}{TRANSP + \mathbb{E}[\text{Number wasted slots per packet}]} \\ &= \frac{TRANSP}{TRANSP + A(2 \cdot PROP)} \\ &= \frac{TRANSP}{TRANSP + (3 \cdot PROP)}\end{aligned}$$

21 Single (Logical) Link in practice

21.1 MAC Addresses

Devices connected to a LAN have a MAC address, whose function is to get frames from one interface, to another physically connected interface on the same network. Most LANs use a 48 bit MAC address, burned into the NIC ROM. Despite this, it is sometimes settable in software. Each adaptor on the LAN has a unique MAC address. Their allocations are administered by IEEE. Each manufacturer buys a portion of the MAC address space (which ensures uniqueness). Consider a MAC address like a personal ID number. This uniqueness ensures that we can move network cards between LANs without issue (not necessarily true for every address type, consider IP which are hierarchical, and thus not portable).

21.2 Ethernet

Ethernet is the dominant wired LAN technology. It has nice cheap NICs, and is the first widely used LAN technology. Over the years, new versions have been released to keep up with speed improvements.

Ethernet is a CSMA/CD algorithm:

1. NIC receives a datagram from the network layer, and creates a frame
2. If the NIC senses that the channel is idle, it starts frame transmission, and if it senses that it is busy, then it waits until the channel is idle, and then transmits.
3. If the NIC transmits an entire frame without detecting another transmission, then it is done with the frame.
4. If the NIC detects another transmission *while* transmitting, then it aborts, and sends the **jam** signal
5. After aborting, the NIC enters **exponential backoff**. After the mth collision, it chooses k at random from $\{0, 1, \ldots, 2^m - 1\}$, and waits $k \cdot 512$ bit times, before returning to step 2

Part VII
Tutorial 3 - ALOHA continued — 2025-11-13

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

22 Reminder

A RAP is an algorithm that is distributed across the network, and each node uses the same protocol. This is contrasted to protocols where there is a single central node that controls the communication. This is very widely used in networking protocols, since it makes adding and removing nodes quickly rather easy. This occurs at layer 2 - the data link layer. This all occurs over the shared communication channel, so a single collision domain. Should there be a collision, then all data is lost. This protocol also states that when a node broadcasts, it uses the *full* link bandwidth.

Definition 22.1 (Bandwidth)**.** *The maximum rate of data transfer on the communication channel* $\frac{bit}{sec}$

Note, bandwidth $\neq$ propagation speed, and sometimes it refers to the width of a frequency range

Definition 22.2 (Goodput)**.** *The fraction of the bandwidth used to send traffic* ***successfully****:* $\eta \in [0, 1]$

22.1 ALOHA

In general, we have $N \to \infty$ nodes in the network, a link bandwidth of B, sending L bits in a frame, with time divided into slots of length $T = \frac{L}{B}$. T is the time required for a message to be transmitted, and as we said before, successful transmissions use the entire bandwidth, and the entire time slot.

Note that a node can start transmitting *only* at the beginning of a slot. In pure ALOHA, every node is running its own time slots, they are **not** shared across the network. When a node *finishes* transmitting a message, it can detect collisions. We will discard propagation time (also known as physical delay), along with the noise on a channel.

Slotted ALOHA assumes that all node slots are synchronised, where pure ALOHA assumes that the nodes all have their own clocks, that are not necessarily synchronised.

Until now we have been considering the random variable X_p to be the number of transmission messages in a time segment T, under the binomial assumption:

$$X_p \sim Bin\left(n, p\right)$$

Today we will extend to the Poisson assumption:

$$X_p \sim Poi\left(gT\right)$$

We examined slotted and pure ALOHA under the binomial assumption last tutorial, and will be examining them both under the Poisson assumption.

23 ALOHA - Poisson Approach

A Poisson process $\{N_t | t \geq 0\}$ with a rate of g is a counting process where

- The number of events in any two disjoint intervals is independent
- The number of events in any interval of length T is a Poisson random variable with parameter gT. This depends on the interval's *length* and not its *timing*
- The sum of independent Poisson random variables is itself a Poisson random variable:

$$X \sim Poi\left(\lambda_1\right),\ Y \sim Poi\left(\lambda_2\right) \implies X + Y \sim Poi\left(\lambda_1 + \lambda_2\right)$$

Notation	Explanation	Units
g	Rate of total messages (both new and failed) that are up for transmission	$\frac{1}{sec}$
$G = gT$	Average number of transmitted (successful and failed) messages in time interval T	None
P_{suc}	Probability of a (single) successful transmission (ie, no collision occurred)	None

Table 5: Definitions

If a message arrives in the middle of a slot, it will be sent at the beginning of the next slot.

23.1 Slotted ALOHA (Poisson) - Analysis

What is the probability for a successful frame? Recall that

$$X_p \sim Poi\left(gT\right)$$

So, we have

$$\begin{aligned} P_{t=T}\left(X=i\right) &= \frac{\left(gT\right)^i}{i!}e^{-gT} \\ P_{suc} &= P_{t=T}\left(k=1\right) \\ &= gT \cdot e^{-gT} \\ &= G \cdot e^{-G} \end{aligned}$$

As for the goodput, as in the binomial model, P_{suc} is also the goodput.

The previous calculation is the perspective of the slot. Let us consider the perspective of the packet: What is the probability if a message has already been sent:

$$P_0 = P_{t=T}\left(k=0\right) = e^{-gT} = e^{-G}$$

Here, to calculate the goodput, we should consider the average number of packets transmitted in an interval T, which is by definition G:

$$\eta = G \cdot P_0 = G \cdot e^{-G}$$

We should actually calculate the conditional probability of no transmissions in period $t = T$, given that there is a transmission. However, under the assumption that $N \to \infty$, the difference between N, and $N1$ is negligible.

So, what is the maximum goodput?

$$\begin{aligned} \frac{d}{dG}\left(Ge^{-G}\right) &= e^{-G} - Ge^{-G} \\ &= e^{-G}\left(1-G\right) \\ \implies \frac{d}{dG}\left(\eta\right) &= 0 \\ \implies G^* &= 1 \\ \eta_{G=1} &= e^{-1} \\ &\approx 0.37 \end{aligned}$$

23.2 Pure ALOHA (Poisson) - Analysis

What is the probability for a successful packet?
Recall that given $X_p \sim Poi\left(gT\right)$, it holds that

$$\begin{aligned} P_{t=T}\left(X=i\right) &= \frac{\left(gT\right)^i}{i!}e^{-gT} \\ P_{suc} &= P_{k=0}\left(t=T\right) \cdot P_{k=1}\left(t=T\right) \\ &= e^{-gT} \cdot gTe^{-gT} \\ &= gT \cdot e^{-2gT} \\ \eta &= P_{suc} \\ &= gTe^{-2gT} \\ &= Ge^{-2G} \end{aligned}$$

From the packets perspective, what is the probability of successful packet? We will assume that a packet was sent, and calculate the probability that no other packet was sent in those 2 frames.

$$\begin{aligned} P_{suc} &= P_{k=0}\left(t=T\right) \cdot P_{k=0}\left(t=T\right) \\ &= e^{-gT}e^{-gT} \\ &= e^{-2gT} \end{aligned}$$

So, the goodput is Ge^{-2G}, and maximising for G we get

$$\begin{aligned}\frac{d}{dG}Ge^{-2G} &= \frac{d}{dG}(G)\cdot e^{-2G} + G\cdot\frac{d}{dG}e^{-2G}\\ &= e^{-2G} - 2e^{-2G}G\\ &= e^{-2G}(1-2G)\\ e^{-2G}(1-2G) &= 0\\ &\implies \begin{cases} e^{-2G}=0, & \text{never}\\ 1-2G=0, & \text{if } G=\frac{1}{2}\end{cases}\end{aligned}$$

The maximum goodput is

$$\eta_{max} = \frac{1}{2}e^{-\frac{2}{2}} = \frac{1}{2e}$$

24 Questions

24.1 Question 1

Messages arrive as a Poisson process with rate g. There are two types of messages in the network:

1. Messages of length s bits, with probability p
2. Message of length l bits, with probability $1-p$

It is given that $l > s$. Every node can send only at the beginning of a slot (for both message types). The link rate is $1bit/sec$, and a slot is of length l seconds ($T = l$). There are an infinite number of nodes.

What is the probability of a message being sent successfully? Recall that for $X_p \sim Poi(gT)$, it holds that

$$P_{t=T}(X=i) = \frac{(gT)^i}{i!}e^{-gT}$$

A successful l message will be with the probability

$$P_{suc,l} = (1-p)\cdot P_{t=l}(k=1) = (1-p)\,gle^{-gl}$$

and a successful s message will be the probability

$$P_{suc,s} = p\cdot P_{t=l}(k=1) = pgle^{-gl}$$

What is the goodput of the network?

$$\begin{aligned}\text{Define RV: } T_{suc} &= \begin{cases} l, & \text{if successful } l \text{ packet}\\ s, & \text{if successful } s \text{ packet}\\ 0, & \text{else}\end{cases}\\ \mathbb{E}[T_{suc}] &= l\cdot P_{suc,l} + s\cdot P_{suc,s}\\ \eta &= \frac{\mathbb{E}[T_{suc}]}{l}\\ &= \frac{l}{l}\cdot P_{suc,l} + \frac{s}{l}\cdot P_{suc,l}\\ &= gle^{-gL}\left(\frac{s}{l}p + \frac{l}{l}(1-p)\right)\\ &= gle^{-gL}\left(\frac{s}{l}p + 1 - p\right)\end{aligned}$$

24.2 Question 2

We now decide to use Pure instead of Slotted ALOHA, and we ask the same questions: Probability of a successful message, and goodput of the network.

Recall that $l > s$, and the slot length is l seconds. Other l have an unsafe start time of $(-l, l)$, where s frames have an unsafe start time of $(-s, l)$. The arrival rate of an s bit message is

$$g_s = p\cdot g$$

and the arrival rate of an l bit message is

$$g_l = (1-p)\, g$$

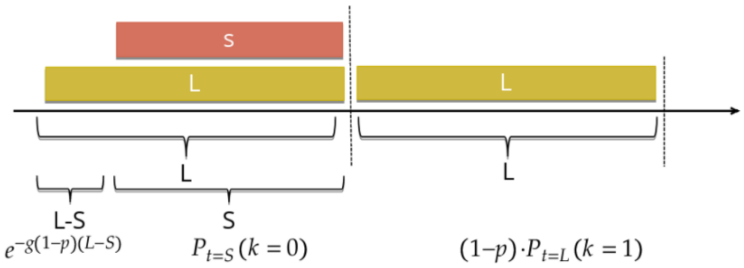


Figure 5: Occurrences

What is the probability that we successfully send an l message $P_{suc,l}$? The probability that no l message is sent between $-l, -s$ is

$$e^{-g_l(l-s)} = e^{-g(1-p)(l-s)}$$

The probability that no l, nor s message is sent between $(-s, 0)$ is

$$P_{t=s}(k=0) = e^{-(g_l+g_s)s} = e^{-gs}$$

The probability that exactly one l message is sent at 0 is

$$(1-p)\cdot P_{t=l}(k=1) = (1-p)\, e^{-gl}$$

When we multiply these three independent factors, we get that

$$P_{suc,l} = (1-p)\, e^{-g(2l-p(l-s))}$$

What is the probability that we successfully send an s message $P_{suc,s}$? The probability that no l message is sent between $-l, -s$ is

$$e^{-g_l(l-s)} = e^{-(1-p)(l-s)}$$

The probability that no l, nor s message is sent between $(-l, 0)$ is

$$P_t(k=0) = e^{-(g_l+g_s)s} = e^{-gs}$$

The probability that exactly one s message is sent at 0 is

$$p\cdot P_{t=s}(k=1) = (1-p)\, e^{-gs}$$

When we multiply these three independent factors, we get that

$$P_{suc,s} = pe^{-g(l+s-p(l-s))}$$

What about the goodput? Let us define RV:

$$T_{suc} = \begin{cases} L, & \text{if successful } L \text{ packet} \\ S, & \text{if successful } S \text{ packet} \\ 0, & \text{else} \end{cases}$$

$$\begin{aligned} \mathbb{E}[T_{suc}] &= L\cdot P_{suc,L} + S\cdot P_{suc,S} \\ \eta &= \frac{L\cdot P_{suc,L}}{L} + \frac{S\cdot P_{suc,S}}{L} \\ &= P_{suc,L} + \frac{S}{L}P_{suc,S} \\ &= (1-p)\, e^{-g(2L-p(L-S))} + \frac{S}{L}pe^{-g(L+S-p(L-S))} \end{aligned}$$

24.3 Question 3

There are N nodes in the network, divided into 2 groups:

1. X - contains $p\cdot N$ nodes that send messages of size S, using $\frac{1}{4}$ of the bandwidth
2. Y - contains $(1-p)\, N$ nodes that send messages of size $S\cdot\alpha$ such that $\alpha > 1$, using $\frac{3}{4}$ of the bandwidth

There is no overlap between the frequencies of X and Y. We will begin by discussing slotted ALOHA, every node can send only at the beginning of a slot, and $T = S$.

The arrival rate of S_α messages is

$$(1-p)\cdot g \implies S_p^\alpha \sim Poi\left((1-p)\,g\right)$$

and the arrival rate of S messages is

$$p\cdot g \implies S_p \sim Poi\left(pg\right)$$

What is the probability of a successful message? We will split into a successful S_α, and S. Recall that S_α messages are longer than a slot.

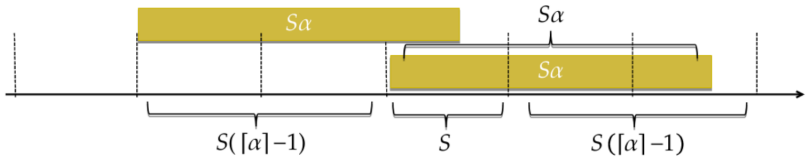


Figure 6:

$$\begin{aligned}
P_{suc,S_\alpha} &= P_{k=0}^{S_\alpha}\left(t = S\left(\lceil\alpha\rceil - 1\right)\right)\cdot P_{k=1}^{S_\alpha}\left(t=S\right)\cdot P_{k=0}^{S_\alpha}\left(t = S\left(\lceil\alpha\rceil-1\right)\right)\\
&= e^{-g(1-p)S(\lceil\alpha\rceil-1)}\cdot g\left(10-p\right)Se^{-g(1-p)S}\cdot e^{-g(1-p)S(\lceil\alpha\rceil-1)}\\
&= g\left(1-p\right)Se^{-g(1-p)S(2\lceil\alpha\rceil-1)}
\end{aligned}$$

and

$$\begin{aligned}
P_{suc,S} &= P_{k=1}^{S}\left(t=S\right)\\
&= gpSe^{-gpS}
\end{aligned}$$

What about the goodput? We are looking at some time window (of length $S\lceil\alpha\rceil$). During this window, there are some cases:

- A successful packet that starts at t, and contributes $S\alpha$
- A successful packet that starts at $t-1$, and contributes $S\left(\alpha-1\right)$
- $\vdots$
- A successful packet that starts at $t+S$, and contributes $S\left(\lceil\alpha\rceil-1\right)$

So we can construct a very large random variable

$$T_{suc,S\alpha} = \begin{cases}
S\alpha, & \text{if successful packet that starts at } t\\
S\left(\alpha-1\right), & \text{if successful packet that starts at } t-S\\
S\left(\alpha-2\right), & \text{if successful packet that starts at } t-2S\\
\vdots & \\
S\left(\lceil\alpha\rceil-1\right), & \text{if successful packet that starts at } t+S\\
S\left(\lceil\alpha\rceil-2\right), & \text{if successful packet that starts at } t+2S\\
\vdots & \\
S\cdot 1, & \text{if successful packet that starts at } t+S\left(\lceil\alpha\rceil-1\right)\\
0, & \text{else}
\end{cases}$$

So

$$\begin{aligned}
\mathbb{E}\left[T_{suc,S\alpha}\right] &= P_{suc,S\alpha}\left(S\alpha + \sum_{k=1}^{\lceil\alpha\rceil-1} S\left(\alpha-k\right) + \sum_{k=1}^{\lceil\alpha\rceil S(\lceil\alpha\rceil-k)}\right)\\
&= P_{suc,S\alpha}S\left(\alpha + \sum_{k=1}^{\lceil\alpha\rceil-1}\alpha - \sum_{k=1}^{\lceil\alpha\rceil-1}k + \sum_{k=1}^{\lceil\alpha\rceil-1}k\right)\\
&= P_{suc,S\alpha}S\left(\alpha + \left(\lceil\alpha\rceil-1\right)\alpha\right)\\
\mathbb{E}\left[T_{suc,S\alpha}\right] &= P_{suc,S\alpha}S\alpha\left\lceil\alpha\right\rceil
\end{aligned}$$

We will then do the same thing with S.

In total, we get

$$T_{suc,S\alpha} = \begin{cases} S, & \text{if successful } S \text{ packet} \\ 0, & \text{else} \end{cases} \implies \mathbb{E}\left[T_{suc,S}\right] = S \cdot P_{suc,S}$$

So the goodput of the network comes out to be

$$\begin{aligned} \eta &= \frac{3}{4}\eta_{S\alpha} + \frac{1}{4}\eta_S \\ &= \frac{3}{4}\frac{\mathbb{E}\left[T_{suc,S\alpha}\right]}{S\left\lceil\alpha\right\rceil} + \frac{1}{4}\frac{\mathbb{E}\left[T_{suc,S}\right]}{S} \\ &= \frac{3}{4}\frac{S\alpha\left\lceil\alpha\right\rceil \cdot P_{suc,S\alpha}}{S\left\lceil\alpha\right\rceil} + \frac{1}{4}\frac{S \cdot P_{suc,S}}{S} \\ &= \frac{3}{4}\alpha P_{suc,S\alpha} + \frac{1}{4}P_{suc,S} \end{aligned}$$

Part VIII
Lecture 5 — 2025-11-16

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

25 IEEE 802.11

In CSMA/CD, we ran the idea that we listen first, then transmit, and detect collisions. This does not work so well in WiFi, since it is difficult to detect collisions when transmitting, due to weak receiving signals. We cannot sense all collisions in any case, there may be terminals who transmit too weak for a particular node to notice. Our new goal is to *avoid* collisions (CSMA/C(ollision) A(voidance)).

The sender senses if a channel is idle for time period DIFS, and then transmits an entire frame with no collisiion detection (CD). If it senses that the channel is busy, then it starts a random backoff time, which only runs while the channel is idle (ie, when the channel is not idle, the timer pauses), and then transmits when the timer expires. If it does not receive an ACK (acknowledgement), then it increase the random backoff interval, and repeats.
The receiver returns ACK after gap SIFS, if the frame was received successfully.

The idea behind avoiding collisions is we allow the sender to "reserve" the channel, rather than random access of the data frames. This way, we avoid collisions of long data frames. The sender first transmits a small request to send (RTS) packet to AP/BS, using CSMA. These can still collide with each other, but are less likely so to do, due to the fact that they are short. AP/BS broadcasts the clear to send (CTS) response to RTS. CTS is heard by all nodes, and the sender transmits a frame, while the other stations defer transmissions.

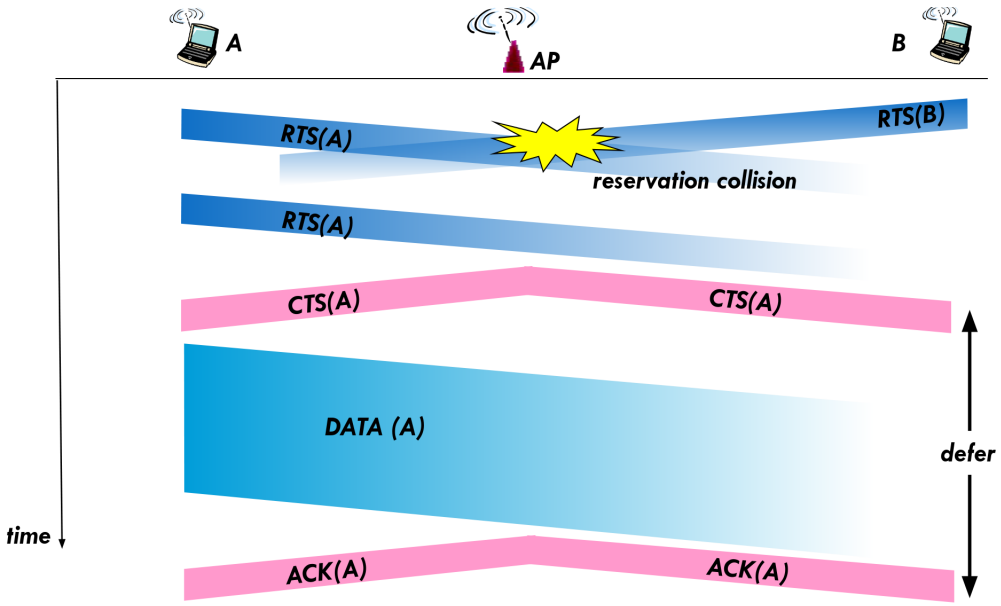


Figure 7: RTS-CTS exchange

In summary for MAC protocols, we have two broad classes:

1. Centralised coordination, which uses channel partitioning (eg time / frequency modulation)
2. Random Access: ALOHA, S-ALOHA, CSMA, CSMA/CD. Carrier sensing is easy for some (wired), and hard for others (wireless). We use CSMA/CD in ethernet, and CSMA/CA in 802.11

26 Interconnecting broadcast domains

26.1 Switch - definition

A switch is a link-layer device that "breaks" apart broadcast domains. It stores, and forwards Ethernet frames, and examines the incoming frame's MAC address, selectively forwarding the frame to one or more outgoing links. It uses CSMA/CD to access segments. It is important that these switches are *transparent* to the hosts, the hosts should be unaware of the presence of switches. Furthermore, switches need not be configured, they teach themselves about the state of the network.

Since the switch breaks apart broadcast domains, it allows *multiple* simultaneous transmissions. Since each host has a dedicated, and direct connection to the switch, and switches can store buffers, the Ethernet protocol is used on each incoming link, with no collision issues, since each link is its own collision domain. The switch then connects

messages between each node **simultaneously**, without collisions. This may be contrasted with a hub, where every node was on the same broadcast domain.

Now we are left with a question, how does the switch know which interface is connected to which node? The switch maintains a *switching table*, which stores this data, along with a timestamp, which allows it to delete old entries. Since we do not configure this table ourselves, it must learn it by itself. Every time the switch receives a frame, it records in the switching table the MAC, the interface upon which it arrived, and assigns a TTL (Time to Live) for this entry. This way, it builds the switching table, and can then begin to use it to forward on packets. It uses a fairly simple algorithm:

```
if (entry found for destination) {
    if (destination is interface from which frame arrived) {
        drop the frame
    } else {
        forward the frame on to the indicated interface
    }
} else {
    flood (forward on all interfaces, aside from the one from which the frame arrived)
}
```

Switches can thus be connected, and chained, together.

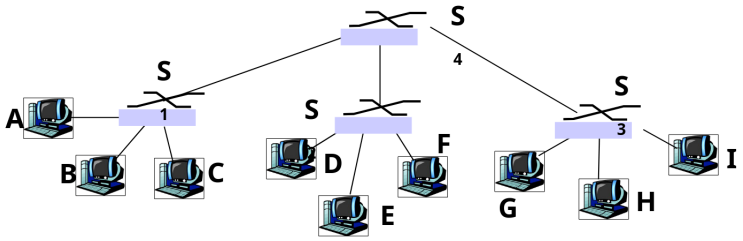


Figure 8: Chained switches

Remember, since switches are invisible in the network, even S_4 (the topmost) when receiving a message that came originally from A, will make note of it being on that interface, even though the message actually came from S_1.

26.2 Loops

What if we have a loop? This will cause mistakes in learning, surely. Consider if we have A, connected to two separate switches, both of which are connected to B. These switches are also connected together. When A sends the message, both switches learn that A is located on their first interface. However, when switch 2 receives the message from switch 1, it also updates its table stating that A is located on 2. The reverse happens as well, resulting in the first switch updating its table to say that A is connected to 2. This results in an unstable switch table, and we waste network space.

In order to avoid this, we want our network topology to be stored in a tree. Now, this could be done with a very careful network admin, who ensures that he never connects a cable that creates a cycle. This is a lovely idea, but firstly we have the problem that people are stupid, and make mistakes. Secondly, this means that our network has no *redundancy*. If a port fails, then that entire section of the tree will be lost to the network. Instead, we will have our protocol state that the switches will organise themselves in a spanning tree by **disabling a subset of the interfaces**.

The protocol that does this is called the Spanning Tree Protocol (STP). Switches communicate with special configuration messages (BPDUs, Bridge Protocol Data Units). It is standardised by IEEE 802.1D.

26.3 STP

We need to solve the following, in order:

1. How to find a root switch?
2. How to compute a spanning tree of the switches?
3. How to compute a spanning tree of broadcast domains?

This protocol is part of a family of protocols, called self stabilising protocols. The objective of these is to stabilise the network to some form, which will not have it constantly changing. In our context, we want every switch to use the same tree.

26.3.1 Choosing a root switch

Assume each switch has a root identifier. Each switch remembers the lowest ID that it has seen so far, and periodically floods its root ID to all its neighbours. When it receives a flooded ID from its neighbours, it updates its root ID if necessary.

26.3.2 Compute a ST given a root

The idea is that each node finds its shortest path to the root. At each node, output the parent pointer, and distance. This is done through the distributed Bellman Ford algorithm:
Assume that there is a unique root node s. Each node will, periodically, tell all of its neighbours what is its *distance* from s. They can tell since at s, $dist\,(s) = 0$. At node v:

$$dist_v = \min_{u:(v,u)\in E} \{dist_u + 1\}$$

Mark the neighbour with the lowest distance as the parent.

Bellman Ford has some important properties:

- It works for any assignment of **non negative** link weights $w\,(u, v)$
- It works when the system operates asynchronously
- It works regardless of the initial distances

To compute the spanning tree of LAN segments, we assume that we are given a spanning tree of the switches. The idea is that each broadcast domain has at least one switch attached, but only *one* of them should forward packets. We choose the switch closest to the root, and break ties by switch ID. If then they are still tied, by the interface / port ID. Switches all listen to all distances announcements on each port. They mark the port as "designated port" **if and only if** the switch in question is the best on that port's associated broadcast domain.

Part IX
Tutorial 4 - CSMA/CD — 2025-11-20

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

27 CSMA / Collision Detection

27.1 Introduction

We saw 2 variations of the ALOHA protocol, slotted which is (magically) synchronised, which had the better goodput of around 0.37, and pure, which is not synchronised, and had a goodput of 0.18. They have poor performance since they do not deal well with collisions. They can be improved by having a central manager, but in this course we are focusing on distributed protocols.

To improve these protocols, we can add some features, such as a node checking if it can transmit a message or if this would case a collision. If a node observes another transmission at the same time, then it can react. Consider a conversation, you do not just talk with probability p, but rather listen first, and speak if no one else is. We would like to add these two features.

When more than one node is using the same broadcast channel, then collisions are bound to occur, so to avoid these we can potentially check if the channel is free, schedule time for each node, or even add more channels (not in this course). For now, let us focus on checking the availability of the channel.

27.2 Carrier Sense Multiple Access (CSMA)

Let us suppose we simply test if the channel is occupied before we transmit. This is insufficient, since transmissions upon it are **not** instantaneous. There is a propagation time, so node A could transmit, and at the same time node B could test, and observe that the channel is unoccupied. Since it is unoccupied (from B's perspective), B begins to transmit, causing a collision with A's transmission.

Let us define T_{prop} to be the maximum propagation time in the channel, between any two nodes. A transmission may collide with another within the first T_{prop} seconds of it beginning to transmit. A node should decide what to do if the channel is occupied, and we will discuss the following 3 approaches:

Approach	What to do if free	What to do if occupied	Intuition
1-persistent	Send	Wait until the channel becomes idle, and then send immediately	High chance of collisions in loaded channels
Non-persistent	Send	Backoff and try again	Lower utilisation on low load
p-persistent	Send with probability p	Wait until the channel becomes idle, and then send with probability p	A compromise between the other two approaches

Table 6: Approaches

When detecting a collision, we want **all** the nodes to know about it, but then we have a few problems:

- How long does it take to detect a collision?
- How can we notify? Nodes A and B know that there was another transmission when they were transmitting, but node C just saw information on the channel, and does not know that this was not a legitimate transmission
- When should a node try and retransmit?

27.2.1 How long

If the network propagation time is T_{prop}, how long does it take to notice a collision?

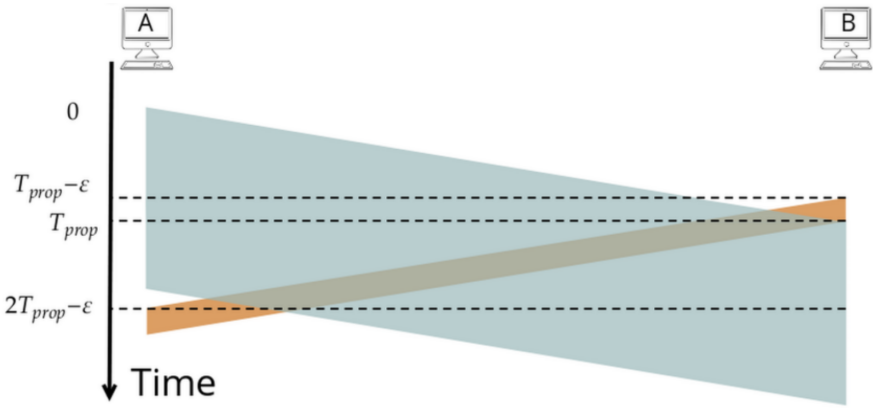


Figure 9: Collision time

Let us assume that the maximum one way propagation time of the network is between nodes A and B, and is T_{prop}. Node A begins transmitting at time 0, and B begins at $T_{prop} - \varepsilon$. Node A will only see the collision at time $2T_{prop} - \varepsilon$, so the amount of time it takes to notice a collision is $2T_{prop}$.
In conclusion, we need the frames of our CSMA/CD network to be long enough such that

$$T_{transmission} > 2T_{prop}$$

27.2.2 Jam signal

Other nodes listening will hear the collided signals, but not know if that was an intended transmission. They are notified that a collision occurred by the node that detected said collision sending a jam signal, after which all the nodes will be aware of the collision, and will now ignore the previous (corrupted) data.

27.2.3 Non-persistent

In the case of collisions, we want to wait before trying again. There are a few ways to decide how long to wait, but the standard method is called "exponential backoff".
Exponential backoff depends on a parameter c, which we usually set to 2. After the kth attempt to transmit a frame, we will uniformly select j from $\{0, 1, \ldots, c^k - 1\}$, and wait $j \cdot T$ time. When a node transmits a frame successfully, then it resets its number of attempts counter (k) to 0. The attempt counters of other nodes remain unaffected.

27.2.4 p-persistent

Here, we have **no** exponential backoff. In the case of a collision, then the entire frame is wasted. The nodes instead simply wait until the channel is idle, and then send with probability p.

27.3 CSMA/CD p-persistent

27.3.1 Definitions

For networking, let:

- End to end propagation time (E2EP) - T_{prop}
- Number of stations - N
- Frame transmission time - T_{trans}

For the protocol, let:

- Time is divided into slots of size $2T_{prop}$ time units
- Nodes always have data to transmit
- The probability of a node transmitting is p

27.3.2 Analysis

So, our p-persistent protocol comes down to:

1. Sense: If busy, wait another time slot, and if free, send with probability p. Once sent, start the detection phase
2. Detecting: If detected a collision, stop transmitting, and try sending again in the next slot (with probability p)

The jam signal is still sent, and relevant, just not so interesting, and not mention here.

So the network may be in the following states:

- Idle - No node is using the network
- Transmitting - Some node succeeded in sending data, without a collision
- Contention - 2 nodes (or more) started sending at a difference of less that T_{prop}, and nodes will know of this only after at most $2T_{prop}$ time units have passed

Let us begin from the goodput of p-persistent: The probabilithy of a successful transmission is the same as in slotted ALOHA:

$$P_{suc}(p) = P(X_p = 1) = np(1-p)^{n-1}$$

The maximum of this function is at $p = \frac{1}{n}$, and so

$$\max_{p\in[0,1]} \{P_{suc}(p)\} = \left(1 - \frac{1}{n}\right)^{n-1} = S$$

This is insufficient for calculating the goodput, since we need to account for the lost time of contention, and idle slots. Since each slot is of size $2T_{prop}$, the result is different from "regular" slotted ALOHA. This is like asking for the probability of k idle / contended intervals, followed by a successful transmission. This is a geometric distribution, with a parameter S. If the contended intervals is a geometric random variable, with parameter S, then its expectation is $\frac{1}{S}$, and so there are $\frac{1}{S} - 1$ wasted slots. Each wasted slot costs $2T_{prop}$ time, so the expected wasted time is

$$2T_{prop}\left(\frac{1}{S} - 1\right)$$

We can then calculate the goodput as

$$\frac{\text{Transmission time}}{\text{Transmission time + Wasted time}} = \frac{T_{trans}}{T_{trans} + 2T_{prop}\left(\frac{1}{S} - 1\right)}$$

So as $n \to \infty$, the maximal goodput is

$$\frac{T_{trans}}{T_{trans} + 2T_{prop}\left(\frac{1}{S} - 1\right)} = \frac{1}{1 + \frac{2T_{prop}}{T_{trans}}}(e-1)$$

If our goodput is given as

$$\frac{1}{1 + \frac{2T_{prop}}{T_{trans}}}(e-1)$$

Then the goodput will tend to 1 when either the transmission time tends to infinity, meaning that after the risky T_{prop}, no nodes will transmit, or when the propagation time tends to 0, meaning that the risky T_{prop} time will be insignificant.

Whereas in slotted ALOHA, without CSMA/CD, we calculated

$$S_{max} = \frac{1}{e}$$

28 Questions

Consider a CSMA/CD network with the following parameters:

- Link bandwidth: $10Mbps$
- Link propagation speed: $2 \cdot 10^8 m/s$

We want to add a similar network, distanced $20Km$ from this one, connected by a medium of same properties The distance between two nodes within the same network is $<< 20Km$.

28.1 Question 1

Can a message of size 64Bytes be used in this type of network?

The time it takes for a single bit to cross between the two networks is

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{\text{propagation speed}} \\ &= \frac{20Km}{2 \cdot 10^8 m/s} \\ &= 10^{-4}s \\ &= 100\mu s \end{aligned}$$

Where the time it takes to transmit 64B is

$$\begin{aligned} T_{trans} &= \frac{\text{bits}}{\text{bits per second}} \\ &= \frac{8 \cdot 64}{10Mb/s} \\ &= \frac{8 \cdot 64}{10^7} \\ &= 51.2\mu s \end{aligned}$$

Since $T_{trans} < 2T_{prop}$, we cannot use such a message in this network, and implement CSMA/CD. A station could finish sending a $64B$ frame before a worst-case collision has time to propagate back and be detected.

28.2 Question 2

What is the smallest change we need to make to solve the above problem?

We need to make the packet size large enough so that the time it takes to send is larger than $2T_{prop} = 200\mu s$. If we use $256B$ messages (4 times larger), this would give us a transmission time of $204.8\mu s$, which satisfies this.

28.3 Question 3

What would change if the link bandwidth was $100Mbs$?

Increasing the link bandwidth allows us to transmit at a higher rate, so T_{trans} is a factor of 10 smaller. We need to increase the packet size for the $T_{trans} \geq 2T_{prop}$ to hold, so since we have increased the link rate by 10, we would need a packet size of at least $2560B$.

28.4 Question 4

Figure 10: Network architecture

Let the propagation speed be

$$v_{prop} = 6 \cdot 10^7 m/s \tag{1}$$
$$B = 3Mb/s \tag{2}$$

For the CSMA/CD protocol to work properly, what is the minimum message size (x) that A can send C?

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{v_{prop}} \\ &= \frac{10Km}{6 \cdot 10^7 m/s} \\ &= \frac{1}{6} \cdot 10^{-3} s \\ T_{trans} &= \frac{x}{B} \\ &\geq 2T_{prop} \\ \Leftrightarrow \frac{x}{3 \cdot 10^6} &\geq \frac{1}{3} \cdot 10^{-3} \\ \Leftrightarrow x &\geq 10^3 bit \end{aligned}$$

For the CSMA/CD protocol to work properly, what is the minimum message size (x) that A can send B?

We must take into account the worst case scenario, so as above:

$$x = 10^3 bit$$

28.5 Question 5

Consider a network that runs **slotted ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA** improve the goodput? No, since if nodes sense the channel at the *beginning* of each slot, they will always sense that the medium is free, and will then transmit.

Consider a network that runs **slotted ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA/CD** improve the goodput? No, since even if a collision is detected, then the entire slot is already wasted.

Consider a network that runs **pure ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA** improve the goodput? Yes, this is a trivial CSMA improvement.

Consider a network that runs **pure ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA/CD** improve the goodput? Yes, since if collided transmissions will stop earlier, this will free the medium for other transmissions.

28.6 Conclusions of CSMA/CD

In CSMA/CD, we wait for the channel to become idle before sending, detect collisions and notify everyone of them when they happen (the jam signal), and wait a random time period before retrying (backoff). It makes sense to use CSMA/CD when the E2EP time is significantly shorter than the transmission time.

29 Errors Detection

When sending information, there is every chance (cosmic rays, noisy channels, etc.) that same information will not arrive in the end, some bits may be changed. We need to find a way to **detect** this corruption, and **recover** from them if possible.

Consider for example the Israeli ID number. The final digit is a checksum, computed by the Luhn algorithm:

1. Firstly, it alternates multiplying digits by 1 or 2
2. If a digit is multiplied by 2, and the result is greater than 9, then the digits of the doubled number are summed (e.g. $2 \cdot 7 = 14 \rightarrow 1 + 4 = 5$)
3. All the resulting digits (from the original digits, or their doubles) are then added together
4. The checksum digit is calculated as the difference between 10, and the last digit of the sum (sum modulo 10).

This is equivalent to the following Python:

```python
def luhn(tz_without_last_digit: int) -> int:
    digits = str(tz_without_last_digit)
    total = 0
    for index, string_digit in enumerate(digits):
        # Alternate multiplying by 1 and 2
        mult = 1 if index % 2 == 0 else 2
        digit = mult * int(string_digit)

        # Sum digits if result of multiplication is more than 1 digit long
        if digit > 9:
            digit = sum([int(sub_digit) for sub_digit in str(digit)])

        total += digit

    # Generate checksum from total
    last_digit = total % 10
    checksum = 10 - last_digit

    return checksum
```

So one can very quickly check if an entered value fits this formula, and verify that it is a legitimate ID number.

Similarly, to identify if some error occurred to our packet, since it is just a collection of bits, we can add additional, redundant data / bits that would allow us to observe changes in the original data. This would naturally have some cost, since more data is now being sent on the network.

29.1 Repetition code

Before we start with clever methods, consider the basic one of just repeating the packet. If one triples the packet, this would increase the packet size from n to $3n$, but for each bit, one may consider which bit occurs the most. If one suffered some corruption, and a bit was swapped, but the other 2 did not for this bit, it may be seen that two thirds agree that it is the other bit, and choose that bit. Essentially, each bit is chosen by majority vote of the repetitions. However, this comes with significant overhead:

$$\begin{aligned}\frac{\text{Actual - Original}}{\text{Original}} &= \frac{\text{Actual}}{\text{Original}} - 1 \\ &= \frac{\text{\# Redundant bits}}{\text{\# Original bits}}\end{aligned}$$

So repeating 3 times has the overhead of

$$\frac{3N}{N} - 1 = 2$$

29.2 1D parity

To reduce the additional of $2n$ bits, there are things to consider. If we were guaranteed that there could be at most a single bit being corrupted, then we can identify it through bit arithmetic. We can add a single bit indicating if there are an odd, or even number of 1s in the packet. That way, we can identify if there is an error, but not correct it. This has the overhead of

$$\frac{N+1}{N} - 1 = \frac{1}{N}$$

The cost of bit parity is a single bit per packet, which is not very much, but it cannot recover from failure. Additionally, the assumption that only one bit can change does not hold in practice. 2 bits changing can affect the corruption detection, so to overcome this we use 2D parity.

29.3 2D parity

Here, we convert the data into a rectangular form of a specific length. Parity bits are then added for both the rows, and the columns, for example, if there is a length of j, and data of size $i \cdot j$ bits, then the actual length of the packet is $i \cdot j + i + j + 1$ bits.

$$\begin{bmatrix} d_{1,1} & \dots & d_{1,j} \\ d_{2,1} & \dots & d_{2,j} \\ \vdots & \dots & \vdots \\ d_{i,1} & \dots & d_{i,j} \end{bmatrix} \implies \left[\begin{array}{ccc|c} d_{1,1} & \dots & d_{1,j} & d_{1,j+1} \\ d_{2,1} & \dots & d_{2,j} & d_{1,j+1} \\ \vdots & \dots & \dots & \vdots \\ d_{i,1} & \dots & d_{i,j} & d_{i,j+1} \\ \hline d_{i+1,1} & \dots & d_{i+1,j} & d_{i+1,j+1} \end{array}\right]$$

This way, we may identify errors from both the rows, and the columns, and thus correct them (with limitations). This approach can **detect** 1 bit errors, 2 bit, and 3 bit errors. Some 4 bit errors may be corrected, but others may

not. For example, when a 4 bit 2x2 block is all swapped, this will not affect the overall sums in the rows and columns, and is thus a silent failure.

We may use this approach to recover lost data, and may recover a maximum of $\frac{\text{dimension}}{2}$ bits, so in this case: 1.

What would be the overhead of 2D parity for a 100bit packet with $j, i = 10$?

$$\begin{aligned}\frac{i \cdot j + i + j + 1}{i \cdot j} - 1 &= \frac{100 + 10 + 10 + 1}{100} - 1 \\ &= \frac{121}{100} - 1 \\ &= 0.21\end{aligned}$$

So in summary:

Code	Overhead	Ability to correct	Ability to detect
Repetition (3 times)	2	1	2
1D Parity	$\frac{1}{N}$	0	1
2D parity	$\frac{i+j+1}{N}$	1	3

Table 7: Error detection and recovery summary

29.4 Cyclic redundancy checks (CRC)

Cyclic redundancy checks are commonly used, and types are referred to as crc-r, for example, Ethernet uses crc-32. This adds r bits to the overhead, and only fails to detect at probability $\approx 2^{-r}$. We treat the packet bits as a long binary number (or a polynomial over $GF(2)$), and divide it by a fixed generator polynomial. The remainder of this division is the CRC value, which will be added to the packet. It is used as follows:

1. The sender computes the CRC, and appends it to the frame
2. The receiver divides the received bits (data + CRC) but the same polynomial
3. If the remainder is 0, we assume no error; otherwise, we detected corruption

Consider the following example of crc-3: We choose a simple generator, of degree 3:

$$G(x) = x^3 + x + 1 \Leftrightarrow G = 1011$$

We have the data to send $D = 1101$. We append $r = 3$ zero bits (the degree of G):

$$D \cdot x^3 = 1101000$$

and divide 1101000 by 1011 by using binary (XOR) long division, getting a remainder $R = 001$. The sender appends the CRC:

$$\text{coded message} = D \| R = 1101001$$

The receiver divides 1101001 by 1011, and gets a remainder of 000, meaning no error was detected. If there was a bit flip, then division by 1011 results in a non-zero remainder, and an error is detected.

Part X
Lecture 6 — 2025-11-23

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

30 Interconnecting Broadcast Domains

30.1 STP

We need to solve the following, in order:

1. How to find a root switch?
2. How to compute a spanning tree of the switches?
3. How to compute a spanning tree of broadcast domains?

This protocol is part of a family of protocols, called self stabilising protocols. The objective of these is to stabilise the network to some form, which will not have it constantly changing. In our context, we want every switch to use the same tree.

30.1.1 Choosing a root switch

Assume each switch has a root identifier. Each switch remembers the lowest ID that it has seen so far, and periodically floods its root ID to all its neighbours. When it receives a flooded ID from its neighbours, it updates its root ID if necessary.

30.1.2 Compute a ST given a root

The idea is that each node finds its shortest path to the root. At each node, output the parent pointer, and distance. This is done through the distributed Bellman Ford algorithm:
Assume that there is a unique root node s. Each node will, periodically, tell all of its neighbours what is its *distance* from s. They can tell since at s, $dist\,(s) = 0$. At node v:

$$dist_v = \min_{u:(v,u)\in E} \{dist_u + 1\}$$

Mark the neighbour with the lowest distance as the parent.

Bellman Ford has some important properties:

- It works for any assignment of **non negative** link weights $w\,(u, v)$
- It works when the system operates asynchronously
- It works regardless of the initial distances

To compute the spanning tree of LAN segments, we assume that we are given a spanning tree of the switches. The idea is that each broadcast domain has at least one switch attached, but only *one* of them should forward packets. We choose the switch closest to the root, and break ties by switch ID. If then they are still tied, by the interface / port ID. Switches all listen to all distances announcements on each port. They mark the port as "designated port" **if and only if** the switch in question is the best on that port's associated broadcast domain.

Definition 30.1 (Link cost)**.** *The **link cost** is a cost associated with each port on each switch ("weight" of connection to broadcast domain). We default to 1. This cost can be either manually, or automatically assigned, and can be used to alter the path to the root switch.*

Definition 30.2 (Root port)**.** *Each non-root switch has a **root port**, which is the port on the path towards the root switch.*

I think this would be better called parent port, it is not the root of the tree, but rather the parent of this specific node in the tree. Consider it a parent pointer in a linked list. The root port is part of the lowest cost path towards the root switch of the tree. If port costs are equal on a switch, the port with the lowest ID becomes root port.

So from this, we see that the spanning tree has a few requirements:

- Each switch has a unique identifier
- There is a unique port identifier for all ports, on all switches

So the algorithm for the spanning tree is as follows:

- Keep sending a single message, which contains this switches' root ID, its cost to root, and its ID

- Save my_root_ID to be the smallest seen root_id so far, and save the lowest cost to root to be the smallest transmitted root_cost, plus the link cost

In short, every node publishes to its immediate connections who it is, how far it is from its root, and its ID. Every node receives this, and updates (if necessary) its parent switch to be the node with the shortest path to the root switch.

Only the root, and designated ports are active for data forwarding. All the other ports are in the blocking state, and do not forward packets. If a switch has no designated port, then it does not forward anything. This includes on the root port.
All ports always send BPDU (Bridge Protocol Data Unit, the above combination of data) messages, in order to update the tree regularly.

31 IP Networks - Interconnecting LANs

31.1 Introduction to IP addressing

We will mostly consider IPv4 in this lecture. In this scheme, an IP address is a 32 bit (split across 4 values) identifier for a host, router, and interface. An *interface* is the connection between a host / router, and the physical link. A router typically has multiple interfaces, where a host typically only has one or two interfaces. They each will have IP addresses associated with each interface. An example would be 192.168.1.178. The IP address is split into the subnets, and host, where the subnets are indicated by the higher order bits, and the host by the lower order bits. So in our example, our host 178 is on the subnet 192.168.1. The router can have many subnets, across different ports. Traffic may be passed within a single subnet trivially, but to communicate across subnets, one must involve the router.

For this kind of addressing, we use CIDR: Classless InterDomain Routing. The subnet portion of the address is of an arbitrary length, with an address format a.b.c.d/x. Here, x is the number of bits in the subnet portion of the address. In the above example of an IP address, it would be given as 192.168.1.178/24, indicating that the first 24 bits indicate the subnet, and the last 8 (the number 178) indicates the host. This x is also called the subnet mask, because that is the size of the bitwise mask we apply to get the host.

So this builds into why the internet is not just a single, huge LAN. Each switch stores a table of every MAC it knows, which given the size of the internet, would not be manageable. Instead, we use **hierarchical addressing**. This way, each ISP has a block of IP addresses, advertising to the rest of the world that any datagram whose x first bits match its blocks should be sent to it, and then handles routing from the rest of the internet into its IP addresses itself. An organisation may well rent a large block of these from an ISP. Should an organisation renting from an ISP change ISP, then the new ISP will begin advertising this address as well, rather than the organisation changing all its IPs. The method used here is longest prefix matching, and everyone routes towards the IP that matches the longest part of the prefix. ISPs get assigned blocks of IP addresses by ICANN (Internet Corporation for Assigned Names and Numbers), which handles allocating IP addresses, DNS, and assigns domains.

The IP also handles demultiplexing, where it needs to be decided where to send the packet in the end user. There is a field in the packet header called the "upper level" field. If it's TCP, then it is sent to the process handling TCP packets. Similarly for UDP, ICMP, and so on. There is a similar demultiplexing for layers 2 and 3, using the type field in Ethernet, and between layers 4 and 5 with ports. There are 2^{16} ports, allowing a lot of choice.

31.2 DHCP

One needs to *acquire* an IP address from somewhere. This can be hard-coded into a system, by its administrator, but this is liable to cause errors. What if it changes networks / subnets? What if someone else is already using this IP address? The IP will no longer be valid. To resolve this we have **DHCP**: Dynamic Host Configuration Protocol. This dynamically gets IP addresses from serves, and behaves "plug and play", like adding more switches to a network.

31.2.1 Overview

The goal is to allow a host to dynamically obtain an IP address from the network when it joins. This IP address is leased to the host, since the IP may be handed to *any* host. It can renew its lease on this address later, should it still be connected. This allows reusing addresses, since addresses should only be held when the device is connected, and active.

Upon connection, the host broadcasts the DHCP discover message. The DHCP server responds with an offer. The host requests an IP address with a "DHCP request", and the server sends it an address using a "DHCP ack" .

Part XI
Tutorial 5 - Spanning Trees — 2025-11-27

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

32 Spanning Tree

Until now we have worked on a network where we have a single, large, shared channel (think like a bus from computer architecture), into which every computer on the network connects. This shared channel was a single shared collision domain. As we have seen, as the number of computers increases, so too do the number of collisions, reducing the goodput significantly. To resolve this we generally split networks into one or more *broadcast domains*. A **broadcast domain** is the set of all nodes that receive each others layer 2 broadcast frames. Each node in the domain could reach every other node via broadcast messages. A **collision domain** is who can interfere with whom on the wire. It is important to note that from now, a broadcast domain will *not* be the same as a collision domain.

Splitting the network into larger physical topologies, split by switches. Consider the following image:

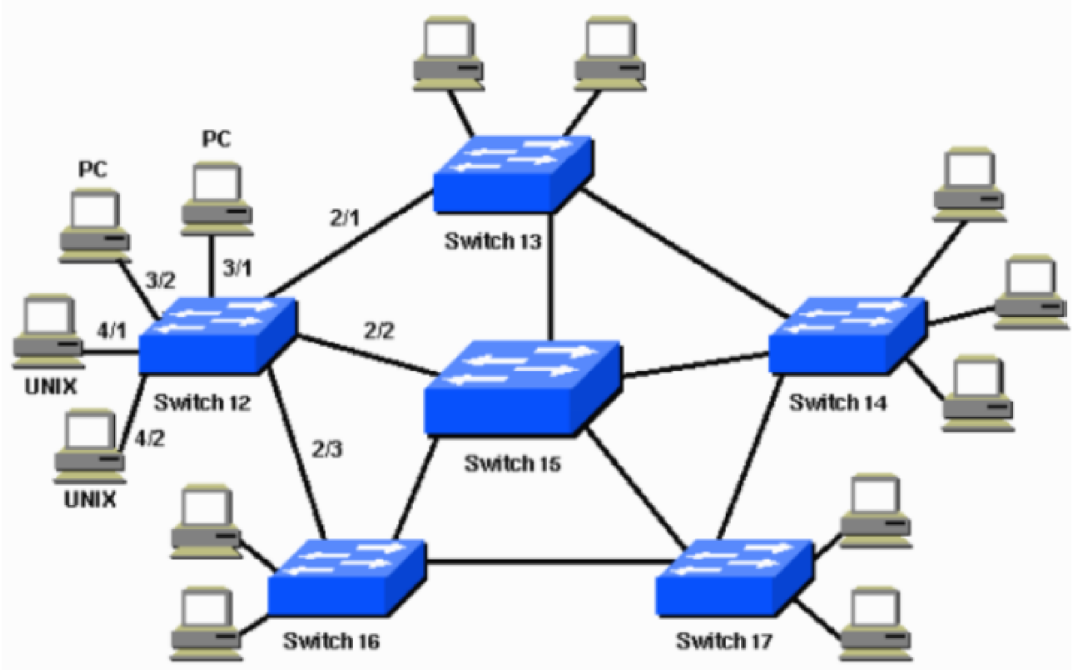


Figure 11: An example complex topology

So here, the collision domain is *only* between each switch, and either switch or node, across a single link, but every node connected to the same switch will be on the same broadcast domain.

There are 2 main ways to connect network elements, switches and hubs. Hubs are stupid. They are boxes, with many ports, and you connect a computer to each port. Whenever they receive a packet on a given port, they rebroadcast it on all other ports. They do not keep state, they do not cache messages to be sent when they will not collide. They are thus one very large collision domain.

On the other hand, switches will retain state, cache messages to avoid collisions, and move packets in a smarter way, such that packets only intended for one recipient will only go to that recipient. They thus split their connected components to many individual collision domains.

32.1 Switches

A switch allows any two connected nodes to communicate with each other, usually without collisions. It can store some packets in case of contention, and it learns the output ports upon which to send each incoming packet (based on previous incoming MAC address). When a node sends a frame, it contains the source, and destination MAC addresses. Whenever a switch receives a packet on one of its ports, it saves the pair (SRC MAC, IN PORT) in a table, and if there is an entry (DST MAC, PORT k) in the table, then it sends it through port k, unless k is the incoming port. If it does not have that entry, then it sends it on all the ports (aside from the incoming one, since that computer clearly already has the packet).

Since we do not manually update the network topology on all connected devices every time we connect / disconnect a device, switch entries (the above table) have a timeout. If a certain MAC has not been seen for this timeout, then the entry is removed, since it may be assumed that the device has been disconnected.

So, we have a clever method for our switches to learn a topology, but this has its limitations, for example, when there is a loop in the network. When there is a loop, then the switches will be constantly relearning the origin port

for packets, which is obviously *not good*. We do not want to only have one patch between 2 LANs, since having more than one path is important for redundancy. If a device / link fails, we do not want the entire network to fail as a result. However, the problems only begin when there is more than one *active* path between 2 LANs, so to fix loops we can:

1. Add a counter packets that is reduced at each switch (this counter acts as a lifetime)
2. Create a single path between any two LAN segments

We will try option 2: We would like to create a single active path between any 2 LAN segments, but we should account for the existence of multiple physical paths. To avoid loops, we will create an active topology of a tree, placed virtually on top of our physical network, such that if there is a failure, the tree can regrow a connection.

33 Spanning Tree Protocol

We ignore the hosts in the network while running this protocol (as in, the end computers, remember, switches are not considered hosts). After convergence (the tree is connected), we want:

- There to be a single root, acknowledged by all
- Each LAN segment has one **and only one** active link on the path to the root (through a designated switch)
- Each switch port not taking part in the active tree will be disabled (but listening for changes to allow healing disconnecting/failing parts of the tree)

We will assume (for this course) that each switch has its own unique ID (this should always be the case in real life, but sometimes some companies are [REDACTED]). We use special data frames called HELLO (or BPDUs - Bridge Protocol Data Units) which contain:

- Root ID (RID) - What this switch currently thinks is the root ID of the tree (initially, its own ID)
- The distance to root (DTR) / also known as the root path cost, which is the cost of the path from this switch, to the root
- The switch ID (SID) of this switch

Every node performs the following algorithm **independently** of other nodes:

- Send HELLO frames through all ports. If it gets a HELLO frame, update the RID and DTR if necessary
- If it was expecting a HELLO, but did not receive during a defined time period, compute a new RID

A node will select the switch with the lowest RID as its root. Based on a new HELLO packet, the switch will update its path to the root (i.e., which ports will remain active, and which ones will be disabled) using the following logic in descending order of priority:

1. Root ID (which root it believes in)
2. Shortest distance to the root
3. Lower SID
4. Lower port number (this is a tie breaker for when there are 2 physical links of the same cost to the same location)

If everything proceeds as it should (no failures), then every switch will have the same root port.

From this protocol, we have 3 port types:

- Root Port (RP) / Parent Port (PP): This connects a switch towards the root, is active, and there is only **one** per switch
- Forwarding port (FP) / Designated Port: A port that forwards traffic for its LAN segment, as in, a port that is connected to the PP for another switch (there is one designated port per segment)
- Blocked Port (BP): The switch will not forward any data from/to those ports. However, the switch will listen to new HELLO messages on those ports in order to potentially heal the network

RPs, and FPs will be part of the spanning tree. It is important to note that a HELLO packet never leaves the LAN segment on which it was transmitted, since it is only important for the two most directly connected switches.

Behold, an example of a spanning tree:

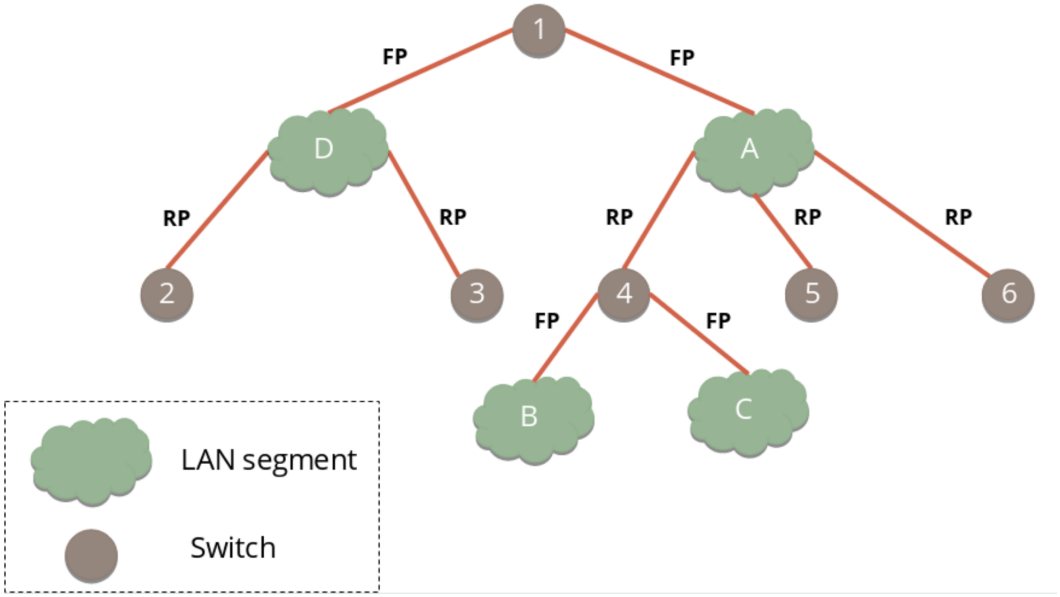


Figure 12: Spanning tree example

What happens if different links have different (known) bandwidths? What needs to change?

- We would prefer using high bandwidth links over low bandwidth ones
- We would need to change the algorithm into a weighted version
- Would need to select high bandwidth links before low bandwidth ones

This modification represents an interesting thought experiment, and a reasonable example of a modification that may be made for an exam question.

33.1 Example

Let us create the spanning tree for a given network (very common exam question). Consider the following network:

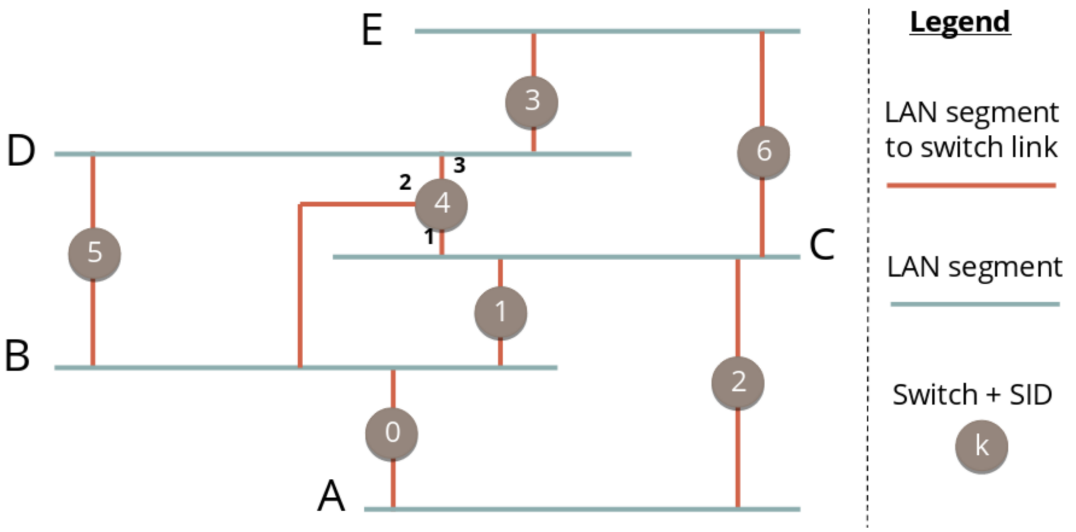


Figure 13: Example network

We begin with switch 0 sending a HELLO packet on segments A, and B. Switch 2 will see RID 0 (since 0 sent its own SID as the RID), and since this is lower than its SID (and currently RID) of 2, so it updates its RID to 0, its DTR to 1, and the root port to point towards switch 0.
This may also be seen in tabular form:

SID	RID	DTR
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0

Table 8:

Everyone thinks that they are the root at the beginning. The above steps update the table

SID	RID	DTR
0	0	0
1	1	0
2	0	1
3	3	0
4	4	0
5	5	0
6	6	0

Table 9:

Following on from this, switches 1, 4, and 5 do the same as 2 did above:

SID	RID	DTR
0	0	0
1	0	1
2	0	1
3	3	0
4	0	1
5	0	1
6	6	0

Table 10:

So in order for segments A, and B to send to the root, they will send to switch 0. Those connections will now be the designated / forwarding port. Remember, the root port is where a **switch** sends towards the root, and the forwarding port is where a **LAN** sends towards a root.

This has left us with the following network:

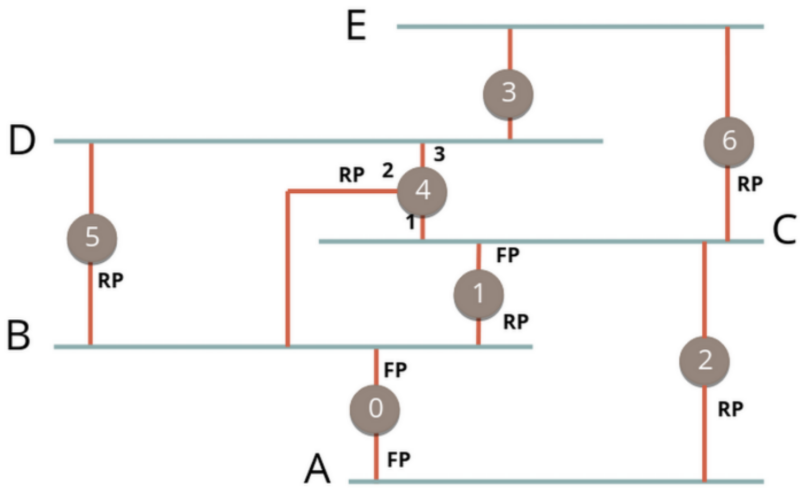


Figure 14:

Now, switches 1, 2, 4, and 6 send HELLO packets on C (in reality, all LANs would have these packets sent on them in parallel, but it is easier to calculate 1 at a time). Switches 1, 2, and 4 have already learned that 0 is the root, with a cost of 1, so they send on C (RID=0, DTR=1, SID $\in \{1, 2, 4\}$). So now, 6 updates its RID from 6 to 0, and DTR to 2:

SID	RID	DTR
0	0	0
1	0	1
2	0	1
3	3	0
4	0	1
5	0	1
6	0	2

Table 11:

On LAN C, to pick the designated port it evaluates the cost to root, the switch ID, and the port ID. Since switches 1, 2, and 4 tie with the lowest RID, we move to the next level which is the SID. SID 1 is the lowest, so switch 1 is the forwarding port for LAN C.

Now, switches 3, 4, and 5 send HELLO on LAN D. 3 updates its RID to 0, and DTR to 2:

SID	RID	DTR
0	0	0
1	0	1
2	0	1
3	0	2
4	0	1
5	0	1
6	0	2

Table 12:

As a result, switch 4 is the designated on D. It is designated instead of 5, despite them having the same DTR, because 4 has a lower SID.

Now, switches 3 and 6 send HELLO on LAN E. There are no switch updates to happen, aside from switch 3 being the designated on LAN E, since its SID is lower than that of switch 6.

We can now disable all the links that do not contain a Root Port, or a Forwarding Port, and then all the switches that do not connect at least 2 LANs (as in, the switch needs at least 1 Forwarding Port, and at least 1 Root Port). We are now left with the final (virtual) network topology tree:

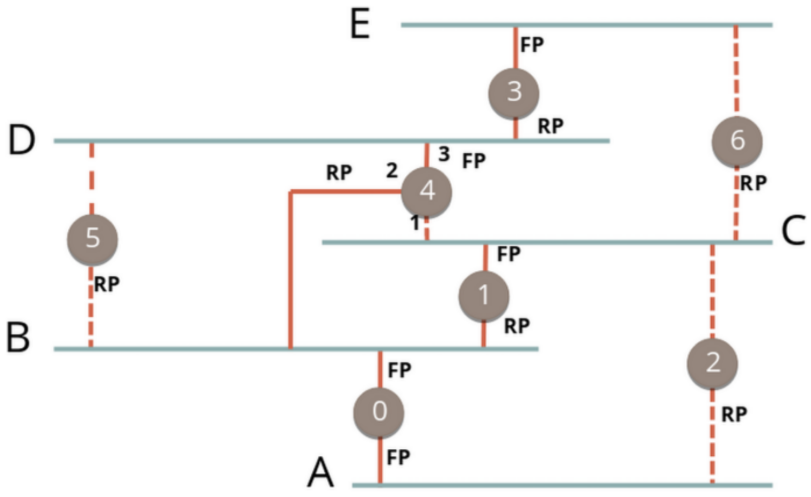


Figure 15:

We may also try an alternative method, which is less demonstrative of the realities, but is easier for us humans. Let us begin just with the LANs:

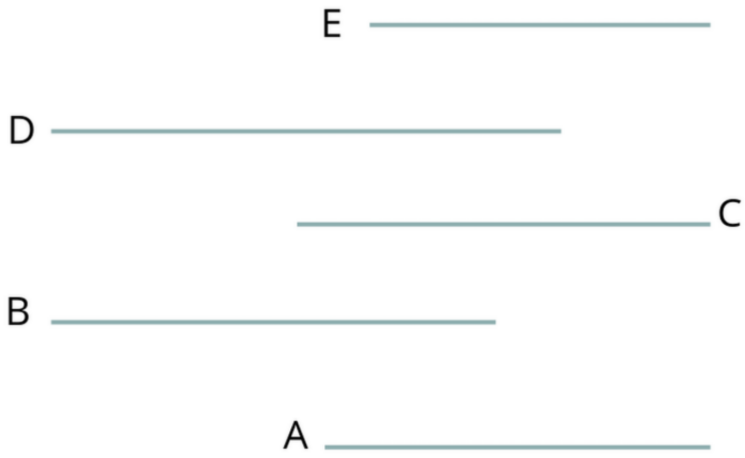


Figure 16:

We now add the root switch, and its connections. In this case, switch 0, with FPs from both A, and B. We may now go to the next unconnected LAN (C), and find its lowest DTR. Since it is connected to 0 through both switches 1, and 2, both with a DTR of 1, we connect in switch 1 with the corresponding FP, and RP, and disconnect 2:

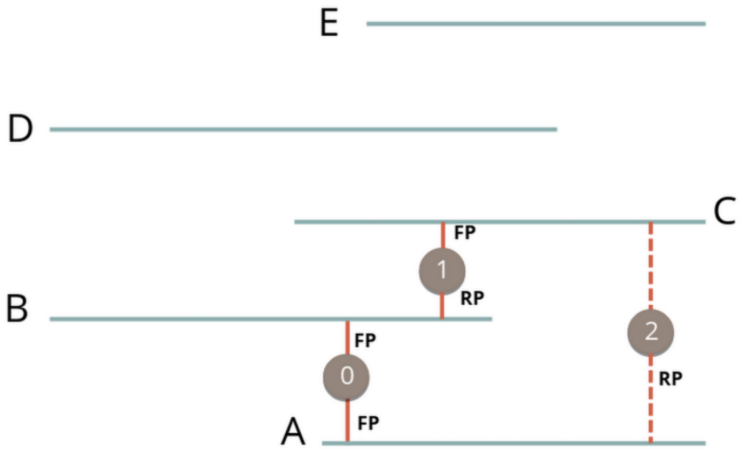


Figure 17:

We may repeat this this by D, and then E by the same path, and will wind up with the same result. Either method may be used in an exam / homework, depending on what the individual finds the easiest.

Let us now consider, what happens if switch 0 crashes?

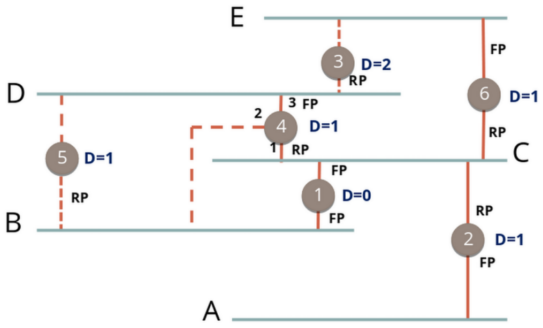


Figure 18:

Now, switch 1 will become the root of the tree, switch 2 replaces switch 0 on LAN A, switch 6 replaces 3 on LAN E, and switch 4 now uses port 1 as its RP, instead of port 2, since it is the same distance, but a lower port number.

If instead switch 1 crashes, then switch 2 will replace it, as no HELLO was observed on LAN C. These kinds of changes are also common questions, and the best way to resolve them is by intuitive understanding, probably gained form doing lots of examples.

34 Questions

34.1 Part 1

The network is composed from:

- LAN segments (horizontal lines): marked with capital letters
- End-nodes (triangles): marked with small letters
- Switches (squares): marked with numbers (those are their unique IDs)

MAC addresses of end nodes are named by their ID (e.g., station a: MAC(a)). MAC addresses of nodes that are connected to more than one LAN are marked by the node's ID and the PORT number (e.g., MAC(b,3)).

Run the STP algorithm when the dotted lines (connected to d) are disconnected. Mark the roots, disabled ports / switches, and parent (root) ports, if they exist.

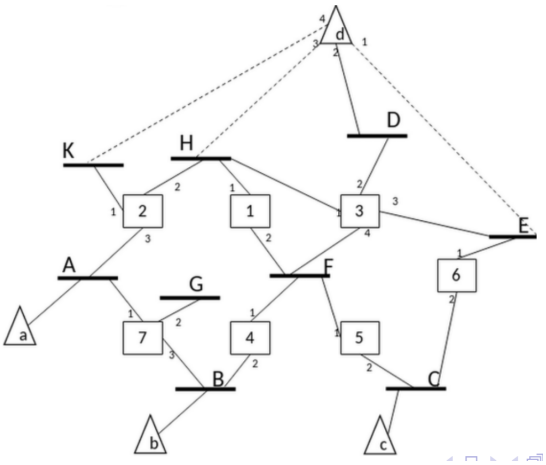


Figure 19:

I am not going to include here the step by step drawing of a diagram, because frankly, drawing in LaTeX is quite a lot of effort. I will instead describe the steps, and you may draw your own diagram as you follow along. The final answer will be included below. If we begin from the LANs, let us pick the root switch to be 1 (it has the lowest ID), which is between LANs H and F. If we now connect D, then switch 3 will be connected to the root through LAN H, since this is the lowest port ID. To now connect in LAN 3, we see that it is also connected through switch E, since this is its shortest path to the root. LAN K is simply connected through switch 2, to LAN H, and A will also connect through switch 2. LAN G only has one connection, to switch 7, so it is connected there. Switch 7 will now connect to LAN A (and thus switch 2). LAN C will connect to switch 5, and finally LAN B will connect to switch 4, since this is its shortest path to the root. So below is the resultant network topology:

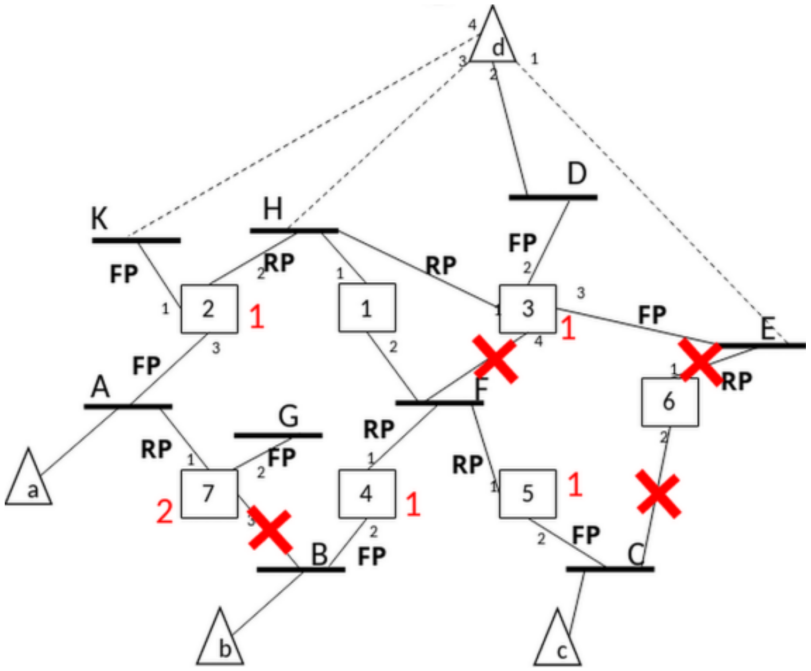


Figure 20:

34.2 Part 2

Should we run the STP algorithm again, but this time where the dotted lines are connected, then there will be no change.

34.3 Part 3

Assume that the network has been working for a long time and that all end-nodes have sent a message to all other end-nodes over all ports.

Fill in the switching table of switch 1:

MAC Address	Port
MAC(a)	1
MAC(b)	2
MAC(c)	2
MAC(d, 1)	1
MAC(d, 2)	1
MAC(d, 3)	1
MAC(d, 4)	1

Table 13:

Fill in the switching table of switch 2:

MAC Address	Port
MAC(a)	3
MAC(b)	2
MAC(c)	2
MAC(d, 1)	2
MAC(d, 2)	2
MAC(d, 3)	2
MAC(d, 4)	1

Table 14:

Part XII
Lecture 7 — 2025-11-30

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

35 IP Networks - Interconnecting LANs

35.1 Introduction to IP addressing

We will mostly consider IPv4 in this lecture. In this scheme, an IP address is a 32 bit (split across 4 values) identifier for a host, router, and interface. An *interface* is the connection between a host / router, and the physical link. A router typically has multiple interfaces, where a host typically only has one or two interfaces. They each will have IP addresses associated with each interface. An example would be 192.168.1.178. The IP address is split into the subnets, and host, where the subnets are indicated by the higher order bits, and the host by the lower order bits. So in our example, our host 178 is on the subnet 192.168.1. The router can have many subnets, across different ports. Traffic may be passed within a single subnet trivially, but to communicate across subnets, one must involve the router.

For this kind of addressing, we use CIDR: Classless InterDomain Routing. The subnet portion of the address is of an arbitrary length, with an address format a.b.c.d/x. Here, x is the number of bits in the subnet portion of the address. In the above example of an IP address, it would be given as 192.168.1.178/24, indicating that the first 24 bits indicate the subnet, and the last 8 (the number 178) indicates the host. This x is also called the subnet mask, because that is the size of the bitwise mask we apply to get the host.

So this builds into why the internet is not just a single, huge LAN. Each switch stores a table of every MAC it knows, which given the size of the internet, would not be manageable. Instead, we use **hierarchical addressing**. This way, each ISP has a block of IP addresses, advertising to the rest of the world that any datagram whose x first bits match its blocks should be sent to it, and then handles routing from the rest of the internet into its IP addresses itself. An organisation may well rent a large block of these from an ISP. Should an organisation renting from an ISP change ISP, then the new ISP will begin advertising this address as well, rather than the organisation changing all its IPs. The method used here is longest prefix matching, and everyone routes towards the IP that matches the longest part of the prefix. ISPs get assigned blocks of IP addresses by ICANN (Internet Corporation for Assigned Names and Numbers), which handles allocating IP addresses, DNS, and assigns domains.

The IP also handles demultiplexing, where it needs to be decided where to send the packet in the end user. There is a field in the packet header called the "upper level" field. If it's TCP, then it is sent to the process handling TCP packets. Similarly for UDP, ICMP, and so on. There is a similar demultiplexing for layers 2 and 3, using the type field in Ethernet, and between layers 4 and 5 with ports. There are 2^{16} ports, allowing a lot of choice.

35.2 DHCP

One needs to *acquire* an IP address from somewhere. This can be hard-coded into a system, by its administrator, but this is liable to cause errors. What if it changes networks / subnets? What if someone else is already using this IP address? The IP will no longer be valid. To resolve this we have **DHCP**: Dynamic Host Configuration Protocol. This dynamically gets IP addresses from serves, and behaves "plug and play", like adding more switches to a network.

35.2.1 Overview

The goal is to allow a host to dynamically obtain an IP address from the network when it joins. This IP address is leased to the host, since the IP may be handed to *any* host. It can renew its lease on this address later, should it still be connected. This allows reusing addresses, since addresses should only be held when the device is connected, and active.

Upon connection, the host broadcasts the DHCP discover message. The DHCP server responds with an offer. The host requests an IP address with a "DHCP request", and the server sends it an address using a "DHCP ack" .

Despite this, DHCP is more than just the IP address. It also contains the address of the first hop router for the client, the name and IP address of the DNS server, and the network mask (indicating network vs host portion of the address).

35.2.2 Example

A laptop connects to a network, and requires an IP address, the address of the first hop router, and the DNS server. To achieve this, it makes use of DHCP. It sends a DHCP request, which is encapsulated in UDP, which is in turn encapsulated in IP, within the 802.1 Ethernet protocol. This is sent over Ethernet frame broadcast to FFFFFFFFFFFF on the LAN, and is thus received at the router running the DHCP server. This Ethernet frame is then demultiplexed into IP, into UDP, and into DHCP. The DHCP server then creates a **DHCP ACK** which contains the client's IP address, the IP address of the first hop router for the client, the name, and the IP address of the DNS server. The client now knows its IP address, the name and IP address of the DNS server, and the IP address of its first-hop router.

35.2.3 Destination IP address - Naming and addressing

If you want to call someone, then you need their phone number. One cannot just dial "Alice" without it. Similarly, to send them mail, you need their address. Similarly, in the internet, to reach Google, one would need their IP address. Humans are really bad at remembering endless series of numbers, so instead we need a method that can map human memorable names onto IP addresses. These names provide little (if any) information about the destination location, in contrast to the IP addresses which are hierarchical, and related to the host location.

Names are easier to remember, and provide an abstraction to the addresses. One can migrate ones website to a different IP address, but maintain the name, renumbering the address to which it resolves, thus making this invisible to the users. Additionally, the name may map to multiple IP addresses, enabling load balancing, reducing latency by using the nearest server, and tailoring the content based on the requester's location / identity. Additionally, one may have multiple names for the same address, a website may buy all the possible similar domains for their site, such as the .com, .org, .co.il, and so on endings, for security, and reachability reasons.

35.3 DNS

The Domain Name System (DNS) is a system to solve this name resolution problem, first proposed in 1983 by Paul Mockapetris. DNS is distributed database, containing a hierarchy of name servers. It contains a simple client / server architecture, and the protocol runs in the application layer. It adds some amount of hierarchy to the namespace, as opposed to the original, flat namespace. For example, many domains are contained within the ".com" top level domain (TLD), and Google owns all the domains that end "google.com", such as mail.google.com, and so on. Data is organised as a tree structure, where each zone is authoritative for its own local data.

Most DNS servers do not actually know the address for a given URL (name). A DNS query often follows the following path:

1. Ask local DNS server
2. Local DNS server does not know, so it checks the root DNS server
3. If the root does not know, then it checks the first level down DNS (for example, edu, com, etc.)
4. If this does not know, then it asks the next level down, the authoritative DNS server
5. And so on down the chain

For those curious for more information, I recommend reviewing the documentation at `pihole` (it is also just generally a wonderful resource).

35.3.1 DNS caching

As we can see, there may be many queries that take place before the address is actually found. All these queries take place before actual communication, which could cause a massive amount of latency before starting the operation that you want to carry out.
To try and resolve this, we use **caching**. Since top level domains rarely change, and popular sites are visited often, local DNS servers often have the addresses of these domains cached, such that when your computer requests it, it does not need to go ask the root, but can rather immediately respond from its cache.
These cache fields include Time To Live (TTL) fields, and the entry is deleted after this time expires. This helps reduce the risk of outdated caches returning bad data.

35.3.2 Resource Records

The records stored in the DNS database is known as Resource Records (RR). Their format is (name, value, type, ttl). There are a few different types:

- A: name is hostname, and value is IP address
- NS: Name is a domain (such as foo.com), and value is the hostname of the authoritative name server for this domain
- PTR: The name is the reversed IP quads (78.56.34.12.in-addr.arpa), and the value is the corresponding hostname
- CNAME: Name is an alias name for some canonical name (www.cs.mit.edu is really eecsweb.mit.edu), the value is the canonical name
- MX: Value is the name of the mailserver associated with name. This also includes a weight / preference

With this, we can reconsider the DNS request:

1. Ask the resolver for www.google.com. Assume that no previous results are cached at the recursive resolver

2. Query the root servers. The answer is downward delegation, sending an NS request to find the TLD server for com:
   com NS a.gtldservers.net
   A response is then received:
   a.gtld-servers.net A 75.292.124.59

3. A query is then sent to the .com zone authority servers, and results in another downwards delegation, with an NS request sent to find the google zone authority:
   google.com NS ns1.google.com
   And a response received:
   ns1.google.com A 122.45.212.57

4. Finally, based off the previous response, query the google.com zone authority servers, and get a response of the desired location: www.google.com A 24.122.49.76

Resource records need to be entered into DNS, for example, whenever one creates a new startup. Let us say you create the startup "FooBar" (by the time you finish reading this, hopefully Google will have made you an offer for your company, in exchange for an astounding amount of money). You acquire a block of adress space from an ISP, let us say you get 212.44.9.128/24. You now register `foobar.com`, and provide a registrar with names and IP addresses of your authoritative name server. This registrar then inserts RR pairs into the com TLD server, and whenever people request your site, they will first be directed to your authoritative DNS server.

35.3.3 Security - vulnerabilities and solutions

DNS is massively important. Without it, you cannot access any websites. Furthermore, your online identity is inextricably connected to your email. Someone could hijack the DNS for your mail server, and pretend to be you. DNS is the root of trust for the web, when navigating to one's bank website, one expects to be taken to said website, but if the DNS record is compromised, a bad actor could phish all their information.

A first major problem we may discuss is DoS, or Denial of Service. Here, one floods DNS servers with request until they fail. In 2002 and 2015 there were massive Distributed DoS, or DDoS attacks against the root name servers. However, thanks to caches, most users did not even notice, since the root zone file is cached almost everywhere. More targeted attacks can be more effective though, blocking off a local DNS server means that that locality cannot access DNS, and blocking an authoritative server would mean that that domain is inaccessible.

When sitting at a café, and surfing the web, ones laptop can access a site like google.com by asking the local name server. This is run by the café (or their contractor), and can return you *any answer that they please*. This can encourage **man in the middle attacks**, where there is a site that forwards your query to Google, and forwards back the reply to you, changing anything they like in either direction. You cannot know that you are getting the correct data (though using TLS enabled websites, those that run HTTPS helps prevent this).

Part XIII
Tutorial 6 - Layer 3 — 2025-12-04

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

36 Introduction

We are now moving over to layer 3. For the first half of the semester we have been dealing with level 2, handling multiple hosts sharing a collision domain, and most of the probability problems (there will still be some in TCP, but we have sorted most of the probability now). When sharing a collision domain, every node connected will hear every transmitted message. We also saw adding switches, such that there can be messages sent to single hosts, rather than broadcast to everyone. These switches handled connecting many broadcast domains, using the STP (Spanning Tree Protocol).

We will begin discussing today transmitting outside of a broadcast domain, by adding routers (remember, we cannot have the entire internet on a single broadcast domain, as discussed in the lecture this week).
Every node in the network needs to have a unique address. This is called a MAC address. There is a unique MAC address for every network interface card (NIC), it is 48 bits (6B) long. Since it is dependent on NICs, since a single host can have many NICs, it can also have many MAC addresses. MAC addresses are known as flat, so we cannot infer the location of a device from its address, and they are vendor specific (vendors buy blocks of MAC addresses, to prevent people from having the same address). These flat addresses are great, but we want to have hierarchical addressing, such that it also encodes location.

We will discuss today what happens behind the scenes when one plugs a host into a layer 2 network, and what happens to connect between a set of layer 2 networks.

37 Layer 3 Network

37.1 Internet Protocol (IP)

The IP address represents a device within the internet. It is 32 bits long, and usually presented as a combination of 4, 3 digit numbers in the 8 bit range. Localhost is a special IP, local to your machine, and is 127.0.0.1. Your home router is almost always something in the options of 192.168.1.1, 10.0.0.1, or 172.17.x.x.

IP is used by most internet applications, so therefore to use the internet you must have one. It also has an expiration time, to allow its reuse when devices disconnect from the network (consider TTL in STP).

37.1.1 Packets

Messages in layer 3 are called *packets*, unlike the layer 2 frames. Below is a description of the packet header:

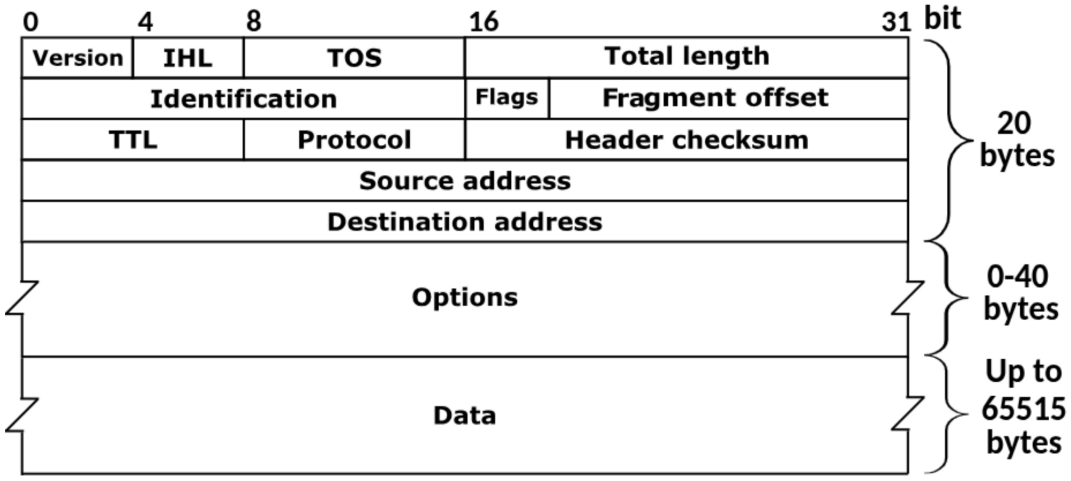


Figure 21: IP packet description

We will note that the concept of security is not built into the internet packets, since bad actors were not particularly considered when the original internet was implemented. In order to prevent this in the modern day, we have added security on top of it (such as TLS).

In layer 3, a packet is consisted of

SRC IP	DST IP	DATA

Table 15: Packet

This is in contrast to a frame in layer 2:

SRC MAC	DST MAC	DATA

Table 16: Frame

However, this has not thrown away the above data. If you open up the data portion, you will see the source and destination IP. Layer 2 will simply ignore these parts, since they are not relevant to it.

37.1.2 Subnets

Given a host A, and a host B, it is possible to know whether or not they share a LAN based off the IP subnet information. An IP subnet is a segmented piece of a larger IP network, allowing efficient routing, and management of IP addresses. The internet is composed of many subnets. A subnet usually contains a range of IP addresses, we define a subnet using the IP prefix, in the form of IP/<int>. For example, 211.12.3.0/24. The prefix is the number before the slash, where the number after the slash is the prefix size. This is also known as the subnet mask. Here, the 24 means that the first 24 bits are constant within this subnet, and only the last 8 may change. The greater the prefix size, the **fewer** addresses the subnet contains. In this example, the subnet mask is 255.255.255.0 (since this way there are 24 1s, followed by 8 0s when converting the address into binary).
In IPv4, there are insufficient IP addresses for the modern internet. We are not discussing IPv6 extensively in this course, but in contrast to the 4 groups of 3 decimal numbers, and IPv6 address is comprised of 8 groups of 4 hexadecimal digits.

A router is a networking device that forwards packets between computer networks. Each of the router's connections belong to different IP subnet. Much like how switches mapped MAC's to ports, routers map an IP subnet to a port. In the below example, there are 3 subnets:

- 223.1.1.0/24
- 223.1.2.0/24
- 223.1.3.0/24

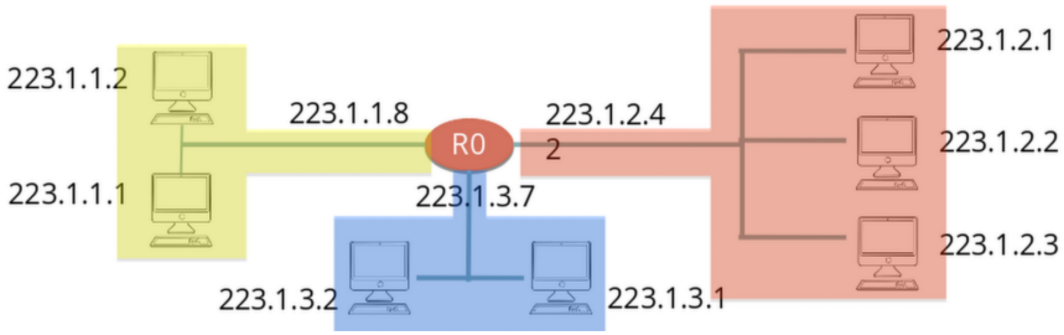


Figure 22: Subnet example

37.2 Address Discovery Protocols

Getting an IP address is an important part of using the internet, since otherwise, you *cannot*. When you first connect to a network (be it WiFi, Ethernet, or anything else), something needs to give you an IP address for the local network. There are several ways on which you can get an IP address. The network admin can provide one (human resource expensive, and slow), the ISP can provide one (unnecessary, they should not be involved in a local network), or a service within the network can provide one. We will focus on that option, it seems to be the best.

37.3 DHCP

Each host needs to dynamically obtain its IP address when it joins the network. To do this, it needs to use some central entity which handles the IP distribution within the local network. This service needs to support mobile users who want to join the network. It needs to support renewing the lease of addresses in use, and allow the reuse of addresses for devices that are no longer connected (in short, an address is not for an individual host, but can be given to other hosts when the former host is not connected).

In summary, the DHCP protocol is as follows:

1. Host: sends *DHCP Discover* [optional]
2. Server: sends *DHCP Offer* [optional]
3. Host: Request IP address *DHCP Request*
4. Server: Send IP address *DHCP ACK*

Since one does not initially know from where to request an IP address, all the messages here are broadcast to the entire network, rather than unicast to a single host.

A DHCP packet contains the following fields:

- Source IP
- Destination IP
- YIADDR (your IP address)
- Transaction ID

When first connecting, when the DHCP Discover is sent, it needs to include a source IP. This is a problem, since the host has not yet been *given* an IP address. As a result, it sends from 0.0.0.0. It is transmitted to 255.255.255.255, the broadcast IP, so everyone on the network receives it. This is because it does not know the IP of the router, so it cannot transmit directly to it. The DHCP Offer will also be broadcast to the entire network (since again, the host has no IP address). The Offer will contain a suggested IP address for the host. The host can now send (still from 0.0.0.0, and still broadcast) a DHCP Request, requesting the IP that the router suggested, and finally the router will respond with a DHCP ACK (over broadcast) informing the host that it has been assigned said address.

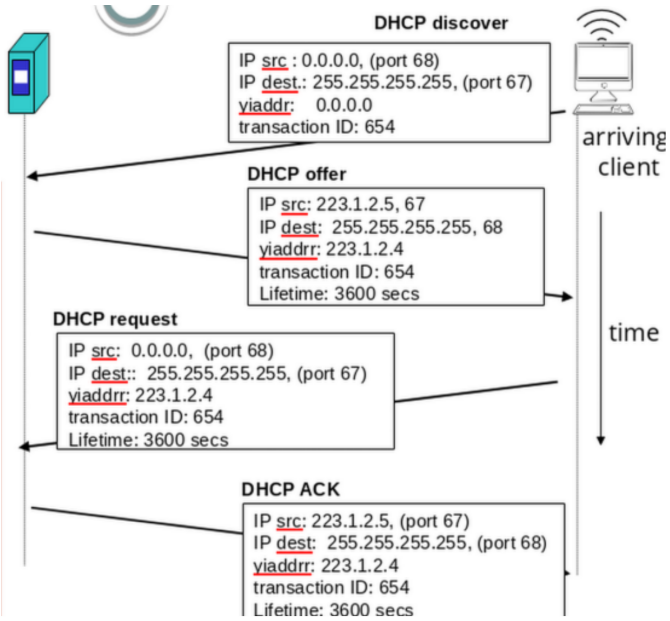


Figure 23: DHCP assignment

In many cases, the DHCP server will provide the client with more than just its IP address. It can also provide:

- The IP of a *DNS server* (often the router itself, which then forwards on requests to another DNS server)
- The IP address of a *default gateway*: The point at which messages leave the network, not necessarily the same as the DHCP server. That being said, when both are handled by the router, it will be the same.
- The *subnet mask* of the LAN

Once allocated, the IP address is valid for only a limited time. If you remain connected to the network when this time elapses, then your computer will most likely attempt to renew its current address, rather than switching to a new one. There can be multiple DHCP providers within a network. In this case, the first to offer an IP address is usually the one that assigns the IP address. This implies a race condition, which creates security problems (like DHCP spoofing), so it is not recommended to do this.

Once the host has an IP address, it can begin to communicate with hosts within, and without, the LAN. To commuicate with a service / host in the LAN, the host needs to know its MAC, for layer 2 communication. So, finding a MAC based off an IP leads us neatly onto the next topic, ARP.

37.4 ARP

ARP is the **Address Resolution Protocol**. Let there be Host 1, which wants to transmit to Host 2. ARP runs as follows: Host 1 will broadcast to the entire network an *ARP request*, which contains the IP of the Host 2, asking for its MAC. Host 2 will then send an *ARP Response*, containing its MAC.
ARP messages are on layer 2. Each node contains an ARP table, which maps IP addresses to MAC addresses. Much as with every table discussed so far, each entry has a TTL. Remember, IP addresses get remapped. If a valid mapping exists in the table, then the host has no need to send an ARP request. Note that there are once again possible race condition, enabling ARP spoofing. This will not be extensively discussed in this course.

37.4.1 Question 1

Can you verify if your existing IP address is active using ARP?

We could send an ARP request for our own IP, if we got no reply then it's free.

37.4.2 Question 2

Suggest a way to use more than one DHCP server within a LAN.

Divide the range between the two servers.

37.4.3 Question 3

Let there be the following network:

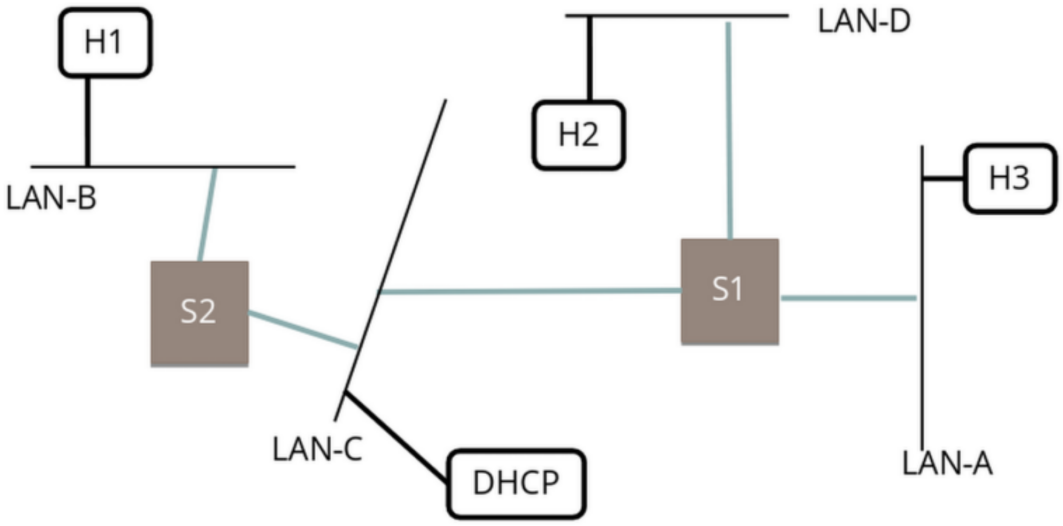


Figure 24: Network

H1 wants to communicate with another host, with IP address H3, **within** the same network. In order for this to happen, it needs to ask for an IP address, and find the MAC of host H3. What messages are heard over the network?

In phase 1, it needs to talk with the DHCP server to get an IP address. All the hosts will hear **all** the DHCP messages, since they are all sent on broadcast. Once this stage has completed, H1 has all of SRC IP, DST IP, SRC MAC, and the DATA, but it is still missing the DST MAC. It now moves on to phase 2, using ARP to getting the DST MAC. It broadcasts the ARP Request (so everyone hears it), and H3 responds with the ARP Response on **unicast**, so only H1 hears it.

37.4.4 Hierarchical questions

So far we have assumed that the network is a LAN, but usually, a network is comprised of multiple LANs. Let us consider how things change when combining multiple LANs together, and how we transfer LAN specific messages between them.

37.4.5 Question 4

Consider the following network:

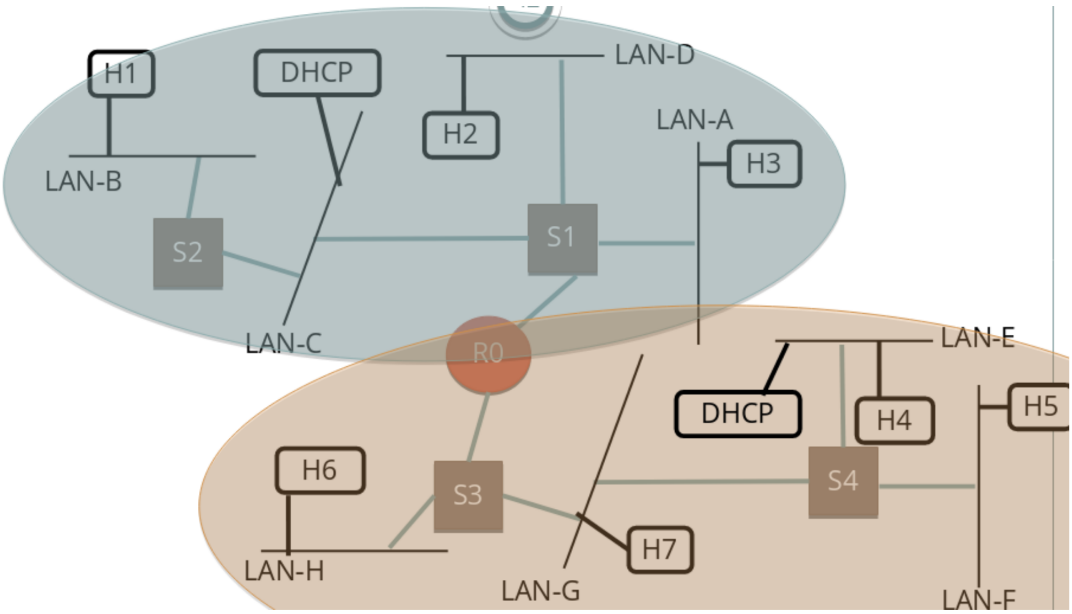


Figure 25: Multiple LANs

If H1 wants to communicate with another host, with IP address H3:
Question 1: What messages are heard over network 1?
The same messages
Question 2: What messages are heard over network 2?
None of them.

37.4.6 Broadcast domains

Each router blocks a single broadcast domain. This means that broadcast packets are usually dropped at the router. Until now, everything was a single broadcast domain composed of a single or multiple LAN segments. To combine multiple broadcast domains, each leg of the router corresponds to a different broadcast domain. So in the previous network, there exist 2 broadcast domains, one for each side of the router.

Consider two hosts, A (MAC a, IP A), and B (MAC b, IP B), connected to different legs of a router R_0. In order for A to communicate with B over layer 3 (IP, and A already knows B's IP), we need to:

Step	SRC MAC	DST MAC	SRC IP	DST IP
$A \to R_0$	a	c	A	B
$R_0 \to B$	d	b	A	B
$B \to R_0$	b	b	B	A
$R_0 \to A$	c	a	B	A

Table 17: Steps

If A does not know R's MAC, then it begins with sending an ARP request. Note that router do **not** forward ARP messages between different networks.

37.4.7 DHCP Relaying

We saw what happens to a packet when it transitions from one L2 network, to another, via some router. Mainly the router changes the MAC address, and will perform relevant ARP requests. When the DHCP server is outside of the local LAN, to get an IP address, the local router (R0) needs to transfer DHCP requests to the other network. This is called **DHCP Relaying**, and R0 is called the **relay agent**. In this case, R0 will forward requests and replies between DHCP server and client, meaning it will forward the DHCP Discover/Request messages and the Offer/Ack messages. Messages between the client and the relay agent are sent in broadcast, and messages between the relay agent and the DHCP server are sent in unicast. The DHCP server allocates an IP address according to a field which is updated by the relay agent. Usually, we would want every LAN to have its own DHCP server (will distribute IPs within the subnet of router).

Consider a network, with H1 on LAN-A, connected to R0 over the leg with the IP 223.1.1.3. R0 is connected to LAN-B, with the address 223.1.2.2, and LAN-B has the DHCP server. For H1 to get an IP address, the following will occur:

Message	SRC MAC	SRC IP	DST MAC	DST IP
DHCP discover	MAC(H1)	0.0.0.0	broadcast	255.255.255.255
disc. (GIADDR=223.1.1.3)	MAC(R0, 2)	IP(R0,2)	MAC(DHCP)	IP(DHCP)
DHCP offer (223.1.1.x)	MAC(DHCP)	IP(DHCP)	MAC(R0, 2)	IP(223.1.1.3)
DHCP offer (223.1.1.x)	MAC(R0, 1)	IP(R0, 1)	Broadcast	255.255.255.255

Table 18: Steps

After the DHCP offer, the protocol will continue the same way with Request and ACK.

37.5 DNS

In order to communicate with a host, we need to know its IP address. Over the internet, we usually assign names for making our lives easier, so we need to resolve a name into an IP address. This uses the **Domain Name Service** (DNS) protocol. DNS is a distributed database. Queries are recursively sent to name servers (NS) from top to bottom (starting with the last part, hierarchically down the chain until it reaches the end, so www.google.com will begin at com, and then google). The Authoritative Name Servers are assigned responsibility for a specific domain.

Let us suppose that we want to find the IP address of www.kernel.org. It is first split into the TLD (top level domain) "org", and the subdomains. We begin by asking our local DNS resolver, and if it knows the address (in the cache or something), then it responds. Otherwise, it begins resolving recursively as follows in this simplified example (with www.google.com):

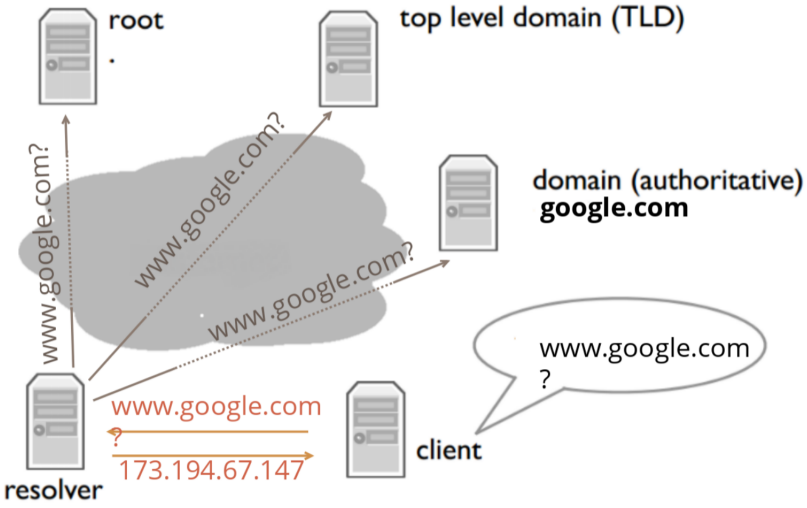


Figure 26: Simplified DNS lookup

We begin by asking the root for the entire address. The root responds with the address of the TLD, which resolves the sub parts of the com TLD. We are then sent to the authoritative DNS server of google, which will return the IP to the resolver, which may then be transferred onto the host.

There are two main types of DNS records:

- Hostname A IP Address, of the style **www.kernel.org A 172.105.4.254**. This maps a hostname to an IP
- Hostname NS Server: **www.kernel.org NS ns21.constellix.com/**. This specifies an authoritative name server for the domain.

There are also additional types, not discussed here.

There are currently 13 root servers, called A to M. There are various TLD servers (.com, .net, .edu, .org, etc.). Each DNS response (a RR - Resource Record), contains a TTL value for cache storage time. Since name servers are identified by name (e.g. ns.google.com), there could be circular dependencies, one is directed to the authoritative name server which is contained within the authoritative name server. A Name Server might add an IP address as a "glued RR" to help the process.

37.5.1 Full DNS example

Our client wants to navigate to www.google.com. It first sends a message to the resolver, asking for the IP address. Since, in this mildly contrived example, this name is not in the cache, the resolver begins by turning to the root. The root responds with an NS RR: **ns.com A 63.156.206.38**, directing the resolver to the .com TLD.
The resolver then turns to said TLD, and asks once again to resolve the full URL. It then receives an A response:
ns.google.com A 216.239.32.10
This directs it to the Google authoritative domain server. From there it asks Google's authoritative domain server for

the URL, and finally receives an A record with the requested IP:
www.google.com A 173.194.67.147. This IP is then returned to the client, which may now get on with navigating to the domain.

38 Summarising problems

38.1 Problem 1

A student just purchased a new computer and connected it to her local network (which is connected to the internet via her ISP). The student browsed www.youtube.com (via HTTP). It's the first time the computer is connected to the network. All the other network components have been active for a long time. Assume the ARP tables don't need to be refreshed (TTL $\rightarrow \infty$). The DNS and DHCP servers are located within the LAN. What messages are transferred from and to the computer from the moment it's connected to the network until the first response from YouTube's server?

Step	Message	SRC MAC	SRC IP	DST MAC	DST IP
1	DHCP discover	PC	0.0.0.0	broadcast	255.255.255.255
2	DHCP offer	DHCP server	DHCP server	broadcast	255.255.255.255
3	DHCP request	PC	0.0.0.0	broadcast	255.255.255.255
4	DHCP Ack (IP,DNS ...)	DHCP server	DHCP server	broadcast	255.255.255.255
5	ARP request – DNS server	PC	-	broadcast	-
6	ARP response – DNS server	DNS server	-	PC	-
7	DNS request – youtube.com	PC	PC	DNS server	DNS server
8	DNS response – youtube.com	DNS server	DNS server	PC	PC
9	ARP request – default gw	PC	-	broadcast	-
10	ARP response – default gw	Default gw	-	PC	-
11	HTTP request	PC	PC	Default gw	IP(youtube.com)
12	HTTP response	Default gw	IP(youtube.com)	PC	PC

Table 19: Solution

38.2 Problem 2

In this problem, all components have been active for a long time. Assume that the switching tables have been erased. Assume that the routing tables are updated and complete.

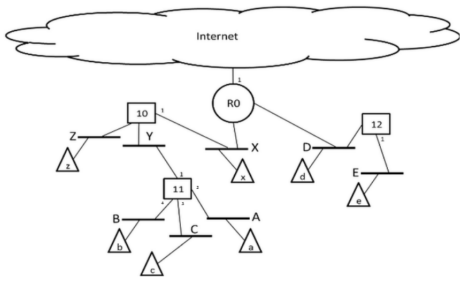


Figure 27: Network graph

The following IP messages are sent. For this question only, assume that every node knows the MAC and IP addresses of all other nodes. On which LAN segments are the following messages heard?

IP message	LAN segments
$a \rightarrow b$	A,B,C,X,Y,Z
$a \rightarrow z$	A,B,C,X,Y,Z
$e \rightarrow b$	A,B,C,X,Y,Z,D,E
$x \rightarrow z$	A,B,C,X,Y,Z
$c \rightarrow a$	A,C
$z \rightarrow c$	X,Y,Z,C

Table 20: Solution

38.3 Problem 3

For the same network, fill switch 11's table right after the last message is sent.

MAC address	Port
a	2
b	-
c	3
x	1
z	1
R0,3	1

Table 21: Solution

38.4 Problem 4

Right after the messages in problem 2, node a sends an IP message to node d. For this question only, assume that:

- Nodes a and d know the IP addresses of all other nodes
- All ARP tables are empty

Reminder, ARP responses are unicast. Describe all the transmitted messages, and in which LAN segments they are heard:

#	Message	SRC MAC	SRC IP	DST MAC	DST IP	LAN segment
1	ARP Request with IP(R0, 3)	MAC(a)	-	broadcast	-	A, B, C, X, Y, Z
2	ARP Response with MAC(R0, 3)	MAC(R0, 3)	-	MAC(a)	-	A, X, Y
3	IP message	MAC(a)	IP(a)	MAC(R0, 3)	IP(d)	A, X, Y
4	ARP Request with IP(d)	MAC(R0, 2)	-	broadcast	-	D, E
5	ARP Response with MAC(d)	MAC(d)	-	MAC(R0, 2)	-	D
6	IP message	MAC(R0, 2)	IP(a)	MAC(d)	IP(d)	D

Table 22: Solution

Determine if the following claims are true or false:

Claim	Statement	Answer
A	Node d will hear an ARP request from node z with $IP(b)$	F
B	Node d will hear an ARP request from node z with IP(e)	F
C	Router R0 has (at least) 3 IP addresses	T
D	Router R0 has (at least) 3 MAC addresses	T
E	Switch 11 has (at least) 4 IP addresses	T
F	Switch 11 has (at least) 4 MAC addresses	T

Table 23: Solution

After a long time in which the network was active, node z is moved to LAN segment X. All the routing and switching table remain like before.

Statement	Answer
Node z needs to sent a DHCP request	F
An IP packet from a to z will arrive to z	F
An IP packet from d to z will arrive to z	T

Table 24: Solution

Part XIV
Lecture 8 — 2025-12-07

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

39 DNS

39.1 Security - vulnerabilities and solutions

DNS is massively important. Without it, you cannot access any websites. Furthermore, your online identity is inextricably connected to your email. Someone could hijack the DNS for your mail server, and pretend to be you. DNS is the root of trust for the web, when navigating to one's bank website, one expects to be taken to said website, but if the DNS record is compromised, a bad actor could phish all their information.

A first major problem we may discuss is DoS, or Denial of Service. Here, one floods DNS servers with request until they fail. In 2002 and 2015 there were massive Distributed DoS, or DDoS attacks against the root name servers. However, thanks to caches, most users did not even notice, since the root zone file is cached almost everywhere. More targeted attacks can be more effective though, blocking off a local DNS server means that that locality cannot access DNS, and blocking an authoritative server would mean that that domain is inaccessible.

When sitting at a café, and surfing the web, ones laptop can access a site like google.com by asking the local name server. This is run by the café (or their contractor), and can return you *any answer that they please*. This can encourage **man in the middle attacks** (MITM), where there is a site that forwards your query to Google, and forwards back the reply to you, changing anything they like in either direction. You cannot know that you are getting the correct data (though using TLS enabled websites, those that run HTTPS helps prevent this).

This issue is made worse by caching. DNS responses are cached. This is great since it results in a quick response for repeated translations (for both the addresses, and the servers). However, even negative records are cached, since this saves time on nonexistent sites (humans make spelling mistakes). Fortunately, the cached data does eventually time out, but this is usually on the order of magnitude of 10s of minutes.

We have a 3rd problem, with a really metal name of **DNS Cache Poisoning**. Here what happens is a bad actor manages to respond to a query from a resolver, earlier than the actual DNS server. The resolver then caches the wrong result, and until the TTL expires, *every* request for this domain will receive the response of the bad actor, which has been cached. This is significantly worse than spoofing, or MITM attacks, since it can impact everyone that uses the same DNS resolver. This can be the order of an entire ISP (or even more).

So, for DNS poisoning, an attacker wants his IP address returned for a DNS query. When the resolver asks the ns1.google.com for www.google.com, the attack could simply reply first with his own IP. So, we are left with the question of why we do not see this more frequently, and how this is prevented. Firstly, we have transaction IDs: each DNS request contains a 16 bit random number, called the transaction ID. Since the adversary is normally located outside of the message route, then the adversary will not know the transaction ID, and would have to guess, with a success rate of $\frac{1}{2^{16}}$, also known as: very unlikely. Early versions of DNS deterministically incremented the ID field. This made it easy to fake, since managing to observe a single request could mean that the adversary would be able to interfere with all the future requests. Randomisation helped this, but there were also issues of using weak random number generators, so one could predict an ID from seeing a few in the past. Additionally, if a resolver sends many messages requesting the same domain, and the adversary sends many responses, then thanks to the birthday paradox, the success probability can be close to 1.

This brings us to the **Kaminsky attack**, first introduced in 2008, resulting in a great deal of patching, that could only go so far in reducing the attack's likelihood of working. Here, the adversary does not need to wait to try again. The adversary makes a request of the resolver for a site, such as google.com. He most likely loses the race, and any further attempts will be suppressed by the TTL. He may now begin iterating through 1.google.com, 2.google.com, and so on. Each one has a low probability of success, but he may well eventually succeed, say at 185.google.com. So now 185.google.com has been poisoned, but this does not appear to matter much. However, it needs to be noted that this is a *subdomain* of google.com. Subdomains almost always have the same location as the parent domain, so when the TTL runs out, then the server may well respond with the poisoned address of 185.google.com for the queries to google.com.

This is bad. There is no need to wait for the old, good, cached entries to expire, and there is no wait penalty for failing. The attack is only limited by the bandwidth, which is often very large. One can defend against this attack by randomising the UDP port used to respond to the DNS query, so the attacker has to guess that port correctly as well. This is merely a mitigation, and not a proper defence.

The long term solution to these issues is to use DNSSEC, where one adds security keys, and signatures to each response. However, this has been "on the cusp of happening" for a while now, to little effect. Perhaps in a few years it shall become the new standard. Additionally, even if the DNS server supports DNSSEC, if the resolver does not, then it will fall back to previous DNS. The roots adopted DNSSEC in 2010, and in 2016 a study by Chung et al found that 90% of TLDs registered public keys, and 83% of the recursive resolvers support DNSSEC. However, only around 1% of level 2 domains registered public keys. In 2021 almost all the TLDs registered public keys, and in 2025 we have reached around 93%.

40 ARP

This may also be considered L2 vs L3, or when we want to translate between IP addresses, and MAC addresses. ARP stands for Address Resolution Protocol, and was discussed at length in the tutorial (though it will also be discussed now).

Let us suppose that we have some devices connected to a LAN. In order to send messages between them, we need to know the MAC addresses, but have only been informed of their IP addresses. Each node (host, router) on the LAN has an ARP table. This ARP table contains IP and MAC address mappings, along with a TTL field (typically 20 minutes). Whenever a host wants to send a packet, it consults the table, and maps the destination IP address, to the destination MAC address. This neglects when the IP is not in the table. AT this point the sender broadcasts a query for who has the IP address, and the receiver (owner of that IP) responds with its MAC address. At this point, the sender caches the result in its ARP table, and there is no need for a network administrator to get involved.

Now, we also need to route data across LANs, connected by a router. If the host A, knows the destination B's IP address, but it cannot directly send it a datagram, what happens instead is that A uses ARP to get the MAC of the router's NIC. It then creates a link-layer frame, with R's MAC address as the destination, and the frame contains the IP datagram from A to B. When R receives, and decodes the IP datagram, it sees that it is destined for B, and so uses ARP to get B's MAC address, and creates a frame containing the A to B IP datagram, which is then sent to B.

41 IP address shortage

IPv4 can only handle 32 bit addresses. This is about $2^{32} \approx 4.2 \times 10^9$ devices, which is not many when we consider that there are an excess 8 billion humans. It is not even enough devices for everyone to have a unique address. This has resolutions through processes like Network Address Translation (NAT), where we have local addresses on networks, and only the router has an external IP address, but this is still pushing the problem away. To resolve this we created IPv6. There were also additional motivations, like an improved header format to help speed processing / forwarding up, changing the header to facilitate Quality of Service (QoS) levels, and a fixed 40B header.

However, transitioning from IPv4 to IPv6 is not necessarily quick or easy. Not all router can be upgraded simultaneously, so how can the network operate with a mixed collection of IPv4 and IPv6 routers? In short, it doesn't very well. We startedd trying to introduce IPv6 just before 2010, and we have now almost reached 50% adoption.

Part XV
Tutorial 7 — 2025-12-11

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

42 What is routing

We have spoken about DHCP, IP, DNS, and ARP, which all aid us in communicating in a LAN. Whenever we wanted to send information beyond the router that is located between our LAN, and the rest of the network, we have stated that this happens, but stopped at the router. Today, we will discuss what the router does.

Routing is the process of selecting a path for the traffic in a network, or between multiple networks. Sometimes packets must cross a few networks to reach the target network, so the relaying of a packet from one network interface to another is called *forwarding*.

In order to route, we need to make a more abstract version of the network. Let us consider the network as a graph $G = (V, E)$. We can turn this into a weighted graph, to represent bandwidth, and so on, but this will not be discussed in this tutorial. We can then run graph based algorithms for finding the shortest path, such as Dijkstra, and Bellman Ford. These are abstracted into two routing algorithms, that then allow us to save the routes in the routing table. This effectively stores the cost of sending information from a local node, to any other destination in the network.

There are 2 different routing algorithms:

- Distributed (decentralised): This uses a similar idea to STP, where each router computes a view of the network on its own, and we hope that everything will converge.
- Global: This assumes that every node knows the complete network topology

43 How to route between L3 networks

43.1 Distance vector

This is a distributed algorithm, based on Bellman Ford. The distance between a node s, to another node y is given as:

$$D_s(y) = \min_{v \in \Gamma(s)} \{c(s,v) + D_v(y)\}$$

Where

- $c(s,v)$ is the current value of the edge from s to v
- $D_V(y)$ is the currently known distance from v to y, based on the table that s has made
- $\Gamma(s)$ is the neighbours of node s

Every node in the network will compute a vector of distances to any other node in the network, using ∞ for distances that are unknown. The vector will continue updating until stabilisation.

The main idea (from the router's perspective): Each node keeps the DV of its neighbours, and if it gets an update from its neighbour, or one of its edges, it

1. Updates the DV
2. Distributes the updated DV to all neighbours

This is repeated until there are no further changes in this router's DV. As we can see, this breaks into two sub sections:

1. Initialisation:
   - Set the DV:
   $$D_s(v) = \begin{cases} c(s,v), & \text{if } v \in \Gamma(s) \\ \infty, & \text{else} \end{cases}$$
   - Set the neighbours DV: $\forall w \in \Gamma(s), \forall v \in V: \; D_w(v) = \infty$
   - Send the DV to all neighbours
2. Update:
   - If some neighbour's DV, **or** one of the edge's weights change:
   $$D_s(y) = \min_{v \in \Gamma(s)} \{c(s,v) + D_v(y)\}$$
   - Resend the DV if anything changed

So, we build a distance table, with our immediate information about our neighbours, and then forward this to our neighbours, who update things accordingly. This keeps happening until everyone's tables have stabilised, at which point it is transferred into a forwarding table, and all will remain stable until the weights change. The changes can be negative (links disconnected, and the like) or even positive (a new link appears in the table).

By doing examples (not shown here), one can see that after a positive change, where network weights are reduced, the algorithm will converge quickly to its stabilised state. However, when there is a negative change, then the routing loop of trying to compute the new distance vectors will continue for a long time, until the nodes establish that there is no shorter paths.

This is called *Count to Infinity*, and will continue until $c(a,b) + D_b(c) \geq c(a,c)$ (Here, a, b are routers, and a link not to a from b has been updated negatively). This can be solved with what is called *poisoned reverse*. Given the nodes a, b, c, a path $a \to b \to c$, then a can report in its DV to b that its distance to c is ∞. This means that b will not try and reroute anything to c through a, which is exactly the problem earlier that caused very slow stabilisation in the network updated with an increased weight. It should however be noted that poisoned reverse does not work in every situation, it will fail when there are loops longer than 2.

43.2 Link state

Link state is a *global* routing algorithm. This means that every node maintains a complete picture of *every* other router in the network, not just its neighbours. This means that every router knows everything about the entire network. The main idea is to iteratively find the shortest path to any destination in the network, employing Dijkstra's algorithm to do this.

Similar to DV, we keep a table of the distances from us (the node s) to every other node in the network. Starting with $c(s,v)$, for every neighbour v of s, and ∞ to every other node that is **not** a neighbour of s. We also keep the predecessor to that destination, the node one before the destination node v, marked as $p_s(v)$. This predecessor node lets us compute the path to the desired node, by recursively building the path from the predecessor nodes, until we reach a node that is a neighbour. In actuality, we would not build the entire route, but rather make note of the neighbour that begins that route. When forwarding a packet to a destination, we simply send it to the neighbour. Each node then simply handles sending the packet to its neighbour. A set of of nodes, N' is also kept, for which we definitely know the shortest paths from s, and in each step, we add a node to N', and update the values of other nodes should the need arise.

In general, the algorithm is as follows, for every node s:

1. Initialisation:
   - Set
   $$D(v) = \begin{cases} c(s,v), & \text{if } v \in \Gamma(s) \\ \infty, & \text{else} \end{cases}$$
   - Set $N' = \{s\}$ and set $\forall v \in \Gamma(s),\ p(v) = s$
2. Loop while $N' \neq N$:
   - Find $w \notin N'$ such that $D(w)$ is minimised
   - Add w into N'
   - $\forall v \in \Gamma(w) : v \notin N'$, if $D(v) > D(w) + c(w,v)$, then
   $$D(v) = D(w) + c(w,v)$$
   $$p(v) = w$$

43.3 Comparison and Real Life

Both DV and LS algorithms are implemented, and used in real life. Routing Information Protocol (RIP) is a DV implementation, and Open Shortest Paths First (OSPF) is an LS implementation. DV is usually better for smaller networks, because it takes time to stabilise, which is not a problem in smaller networks. In larger networks, one should use LS. However, DS is a simpler protocol, that is easier for each router to run, so there is some desire to use it.

RIP is based on the hop count (the number of subnets in the path), and it updates every 30 seconds. If no update is received from a neighbour for 180 seconds, then it is marked as unreachable. OSPF is an LS implementation, where each router broadcasts its LS, at least once every 30 minutes, and when a change in the topology is detected.

Part XVI
Lecture 9 — 2025-12-14

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

44 Interconnecting IP subnets

When considering layer 3, we want to connect LANs, or IP subnets. For our uses, they are separated by routers. There are 2 key characteristics to a network:

1. Topology: The physical interconnection structure of the network
2. Routing algorithm: This restricts the set of paths that the messages can follow (like STP)

44.1 Topology

This is how the components are connected. It has some important properties:

- Diameter: The maximum distance between any two nodes in the network (measured in hop count / number of links)
- Nodal degree: How many links connect to a node
- Bisection bandwidth: The lowest bandwidth between half the nodes, and the other half of the nodes, across all such partitions (consider MAX CUT from algorithms)

We want to measure what makes a *good* topology. As with everything in life, this is a loaded question, since we need to define what is good. One might think that bisection bandwidth would be a good measure, but consider a network where $\frac{2}{3}$rds of the nodes are maximally connected, and the other $\frac{1}{3}$rd are maximally connected, and they only have a single link between the two. This would not be great, but the bisection bandwidth would not consider that, since it talks of halves.
Instead, we may consider an expansion of a graph, and examine every partition:

$$\min_{S \subset V,\ 0<|S|\leq \frac{n}{2}} \left\{ \frac{\text{EdgesBetween}\,(S, V \setminus S)}{|S|} \right\}$$

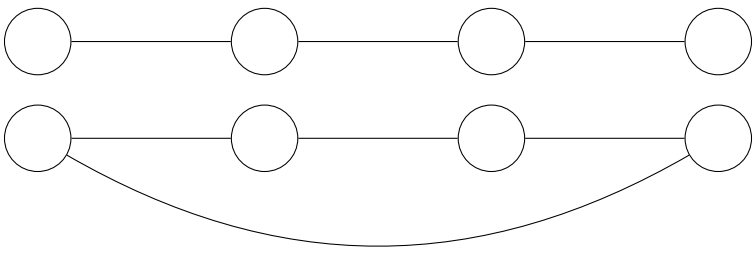
This is called the **edge expansion** of a graph. **Expanders** are graphs with a high edge expansion, i.e. close to $\frac{d}{2}$ for a d-regular graph.

There are two main tpes of topologies:

- **Explicit constructions**: Here the nodes are connected with some kind of pattern (so the graph has a structure). The nodes are identified by coordinates, and routing can usually be pre-determined by the coordinates of the nodes.
- **Non explicit constructions**: The nodes are connected arbitrarily. There is no structure to the graph, which makes it more extensible in comparison to a regular topology. These often use variations of shortest path routing.

44.1.1 Linear arrays and Rings

Consider the following two topologies, linear array, and a ring:



For the linear array topology:

- Diameter = 3 (distance between the two furthest is 3 links)
- Nodal degree = 2

- Bisection bandwidth = 1

and the ring:

- Diameter = 2
- Nodal degree = 2
- Bisection bandwidth = 2

To describe a linear array, and a ring, in an array nodes will be numbered from 0 to $n-1$, with i connencted to $i+1$ for $i < n-1$. In a ring, nodes are numbered from $0, \ldots, n-1$ as well, but $\forall i$ i is connected to $i+1 \mod n$.

44.1.2 Hypercubes

If we consider instead hypercubes (4D cubes, or greater), then there are 2^n nodes, and each node is described by its binary representation. There is a link between two nodes with the hamming distance of 1. For an n dimensional hypercube, the diameter will be n, the nodal degree will be $n-1$, and the bisection bandwidth will be 2^{n-1}.

44.1.3 Binary Trees

In a binary tree, each node has at most 2 children. They are of fixed degree, a diameter of $\log(n)$, and $O(1)$ bisection bandwidth. To route between nodes a, b, one goes up to their common ancestor, and then back down to the other node.

44.2 Routing

We focus for now on routing within a single organisation, called intradomain routing. We will discuss interdomain routing later on. We are effectively discussing a network of routers, where the links represent IP subnets.
Upon receiving a packet, our routing algorithm needs to, very quickly, decide what to do with it, or in other words, decide on which link it should send the packet. In order to achieve this, there is often a forwarding table, where one matches between header values, and output links. Forwarding is performing this action, and routing is the engine that decides where to send each packet. So the engine's output is the forwarding table.

We often work with static link weights. Here we want to compute the shortest paths to IP subnets based on the link weights. These are configured by the network operator (we will revisit this), and through this, we determine the next hop router (the next router in the path) to every IP subnet.

We have 2 main routing schemes:

- Link state routing: Each router floods information to the network in order to learn the topology, and then each router applies Dijkstra's algorithm to compute the shortest paths
- Distance vector routing: This is an iterative process, that uses Bellman Ford (similar to STP)

There are 2 ways to classify routing algorthms:

- Global: All routers have the complete topology, and link cost information. These are solved with "link state" algorithms
- Decentralised: Each router only knows its **physically** connected neighbours, and the link costs to these neighbours. Solving this involves an iterative process of computation, and information exchange with neighbours. It is solved with "distance vector" algorithms.

44.2.1 Link state routing algorithm

Here we will discuss Dijkstra's algorithm. In the network topology, link costs are known to all nodes. This is accomplished via *link state broadcast*. All the nodes have the same information. It computes the paths of the lowest cost from one node (the source), to all other nodes. This creates the forwarding table for that node. This is an iterative algorithm, where after k iterations, we will know the lowest cost path to k destinations.
Let us define some notation:

- $c(x, y)$ - The link cost from node x, to node y. This will be infinite if there is no link
- $D(v)$: The current value of the cost of the path from the source, to the destination
- $p(v)$: The predecessor node along the path from the source, to v
- V': The set of nodes for whom we definitely know the least cost path

The algorithm will run as follows:

1. Initialisation: Set $V' = u$, and for all nodes v, if they are adjacent to the starting node u,

$$D(v) = c(u, v)$$

and set $D(v) = \infty$ otherwise

2. Loop until all nodes in V': Find a $w \notin V' : D(w)$ is a minimum. Add w into V', and update $D(v)$ for every v that is adjacent to w, and **not** in V':

$$D(v) = \min(D(v), D(w) + c(w, v))$$

The new cost to v is either the old cost to c, or the known shortest path cost to w, plus the cost from w to v.

A naïve implementation for n nodes will involve $\frac{n(n+1)}{2}$ comparisons, and will be $O(n^2)$. More efficient implementations are possible (see data structures course), to be performed in $O(n \log(n) + |E|)$. The function may oscillate between solutions, and not complete, if the link weights are set dynamically. This can occur when the link cost is the amount of carried traffic (for example).

44.2.2 Distance vector algorithm

This makes use of Bellman Ford Equation (dynamic programming). Let us define

$$d_x(y) = \text{Lowest cost path cost from } x \text{ to } y$$

Then,

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$

Where the minimum is taken over all the neighbours v of x.

The basic idea is that every so often, each node will send its own distance vector estimate to its neighbours. When a node x receives this estimate from a neighbour, then it updates its own estimate using BF:

$$\forall y \in n \; d_x(y) \leftarrow \min_v \{c(x, v) + d_v(y)\}$$

So then, $d_x(y)$ converges to the actual cost.

When a link cost changes, then a node detects the local kink cost change, updates the routing information and recalculates the distance vector. If the DV changes, then it notifies its neighbours.
So, good news will travel fast (links getting faster will propagate very quickly), but bad news (link lengths getting longer) will propagate slowly.

44.2.3 Comparison of LS and DV

Complexity:

- LS with n nodes, E links, and $O(nE)$ messages sent, each node will send a broadcast message, flooded to all links
- DV: Exchange is only between neighbours

Convergence speed:

- LS: $O(n^2)$, may have oscillations
- DV: Varies, may be routing loops, and may have count to infinity problem (bad news propagates slowly)

Robustness (router malfunction):

- LS: Nodes can advertise the incorrect *link* cost, and each node computes only its own table
- DV: Nodes can advertise the incorrect *path* cost, and each node's tables are used by others (thus propagating the error through the network, blackholing traffic)

44.2.4 Real world examples

RIP - Routing Information Protocol. This is a distance vector algorithm, used by BSD-Unix in 1982. The distance metric is the number of hops.
OSPF - Open Shortest Path First. This is an open, publicly available protocol, using the LS algorithm. The OSPF advertisement carries one entry per neighbour router, and advertisements are disseminated to the entire network (via flooding). Messages are carried directly over IP, rather than over TCP, or UDP.

44.3 ARPAnet routing

ARPAnet was a wide area network that predated the internet. It originally (1969) used shortest path routing, with a dynamic setting of link weights. They were set to the instantaneous queue length, plus some constant. Each node updated its distance computation periodically.
This had some problems. There were protocol oscillations, and a high protocol overhead. Additionally, a long path would appear better than a congested path, resulting in an inefficient use of resources.

In 1979 there was a new routing protocol, which averaged the link weight over time in order to reduce fluctuations, and the frequency of updates (nicely handling some of the overhead issues). In 1987 it was revised, where traffic was shed gradually in order to prevent overreactions to congestions. Additionally, link weights were given upper bounds, in order to avoid excessively long paths.

44.4 Traffic management

Recall layering. We will cover traffic engineering, which is done in the network layer (L3), and handled within organisations, by optimising static link weights. We will soon discuss congestion control, which is done in the transport layer (L4), through protocols like TCP and UDP.

Traffic engineering is tuning the routing protocol configuration to optimise network performance (often by changing the static weights). This is done through:

- **Measurement**: Measuring the topology (as done at the beginning of the lecture), and the traffic pattern (passively inspecting the traffic)
- **Network wide models**: Making representations of the topology, and traffic
- **Network optimisation**: Algorithms to find good configurations, and operational experience (nothing beats experience) to identify constraints

44.4.1 Theory - Flow optimisation

Recall the max flow problem from algorithms. Given an undirected, weighted graph, with a source vertex, and a target vertex, output the maximum flow from s to t. This may be done through max flow - min cut.
We need to adapt this to *multicommodity flow*, since every node can send to every node. We will remove s, t, and add in the demand matrix $D = \{d_{ij}\}$. There are a few methods to solve this problem:

- Maximum multicommodity flow: maximise the total amount of sent traffic
- Minimise congestion: Minimise the load on the most congested edge
- Fairness: Distribute the traffic as evenly as possible
- Etc.

Multicommodity optimises the routes from sources to targets, and how traffic is split between routes. We need to ask if IP routing optimises this.
To optimise the (static) link weights one begins by computing the shortest paths to other routers based on the link weights to determine the next hop to every other router. The link weights are configured by the network operator.

44.4.2 ECMP

In Equal Cost Multipath (ECMP), each router computes the shortest paths to each other router, based on the configured link weights, and splits traffic designated for said router evenly between all the shortest paths.

So, our objective given the multicommodity input is to output link weights such that ECMP flow is the optimal solution (with respect to the specific objective function). We have used the word "optimal" here, which assumes that there are a set of link weights such that the ECMP flow is optimal. We need to consider if this is in fact always the case, and if it is not, if there is always a set of link weights that approximate the optimal solution.

In short, we cannot always get the optimal solution, sometimes there is not one. Instead, we will change our objective to the ECMP flow that is "closest" to a specific objective function. Turns out that this problem is NP-hard, even for simple objective functions. As a result, network operators use (non optimal, but sufficiently good in some real life environments) heuristics.

Let us now discuss the implementation, how ECMP splits traffic evenly between its next hops. Recall hash functions, which map a large datum into a small datum. This small datum can be an integer, which indexes an array. This may be expressed mathematically as mapping n bit data into k buckets, where $k \ll 2^n$. This provides time and space saving data structures for when we want to look up data. We aim for this mapping to be a low cost, deterministic, and uniform method of spreading out keys across hashes. Naturally, from the pidgeonhole principle, there will be at least 1 pair of keys that map to the same bucket. This was resolved in the data structures course.
We select paths by hashing the source, and the destination addresses, which returns a number in the range of the number of links attached. This chooses, deterministically a link on which to send data.

Part XVII
Tutorial 8 - Traffic engineering — 2025-12-18

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

45 Introduction to Traffic Engineering

Last tutorial we discussed ways to find the fastest route to a destination. However, we do not necessarily always want to send all the data through there, since that will increase the traffic, limiting how much information can be sent that way, resulting in slower transmission. We want to share the transmission across more possible routes in order to reduce network traffic.
We do not normally touch this, networks usually mostly manage themselves. Routing protocols adapt to changes, in order to manage traffic. TCP (to be discussed in the future) can identify congestion, and adapt to reduce the traffic on a network. However, they do not do that as efficiently as we might have originally desired, TCP cannot send data over a less congested path if one exists.

We want to monitor our resources (like bandwidth) in order to distribute load in ways that will reduce congestion, delay, and comply with application specific requirements. This can be achieved by fine tuning the routing protocol parameters, which are for the most part, the weights of the graph. This is not necessarily an easy problem.

46 Modelling networks as linear programs

There are many scenarios where we can model our world using functions. This modelling can help us find solutions to problems in (hopefully) polynomial, or even linear time. We will return to linear programming, as discussed in algorithms. The standard form is for a given set of constraints $\{c_i\}_{i=1}^{n}$, and constraints

$$\{a_{i,j}\}_{i,j=1}^{m,n}$$
$$\{b_i\}_{i=1}^{m}$$

Then we can maximise:

$$\max_x \left\{ \sum_{j=1}^{n} c_j x_j \right\}$$
$$\text{Subject to } \sum_{j=1}^{n} a_{ij} x_j \leq b_j \ i \in [m]$$
$$x_j \geq 0 \ j \in [n]$$

We know from algorithms that every LP problem has another representation, which provides an upper bound for the original LP, and the solution for one defines the solution for the other (duality theorem):

$$\min_y \left\{ \sum_{j=1}^{n} c_j y_j \right\}$$
$$\text{Subject to } \sum_{j=1}^{n} a_{ij} y_i \geq c_j \ j \in [n]$$
$$y_i \geq 0 \ i \in [m]$$

We also saw in algorithms the problem of MAX-FLOW, where for a given network, with a source node, and a destination (sink) node, find the maximum flow between the source and the sink.

We will not be discussing MAX-FLOW here, but rather *Multi-Commodity Flow*. This is because in a network, there are normally more than one set of active source, and destination pairs. We call the set of demands the "Demand Matrix".
So now we have a set of commodities K where $K = \{K_i = (s_i, t_i, d_i)\}_{i=1}^{n}$, and $f_i(u,v)$ which is the flow of the commodity i on the edge (u,v). Note that we allow every commodity to have its own demand (compared to just maximising the flow). The constraints become

$$\forall i, \forall v \in V \setminus \{s_i, t_i\} : \sum_{u:(u,v)\in E} f_i(u,v) = \sum_{w:(v,w)\in E} f_i(v,w)$$
$$\forall (u,v) \in E : \sum_{u:(u,v)\in E} f_i(u,v)$$
$$\forall i, \forall (u,v) \in E : f_i(u,v) \geq 0$$

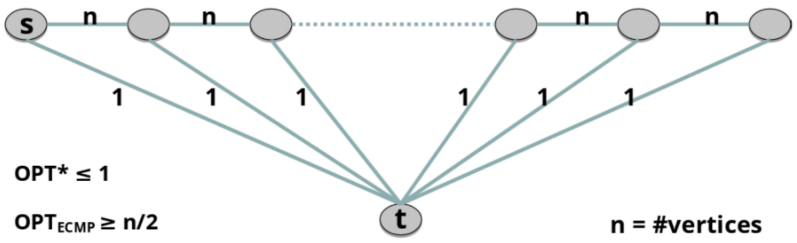
We have many things that we can optimise. For example, we can maximise the total amount of sent traffic (i.e., maximise $\sum_v |f_v|$, where $|f_v|$ is the total amount of traffic sent by v). We could also minimise congestion, where we minimise the load on the most congested edge (So minimise $\max_e \left\{ \frac{f_e}{c_e} \right\}$ where f_e is the flow along edge e). We could try and fairly allocate the resources, and even more, but those are less interesting to us right now.
So, let us consider the case of maximising the total amount of traffic sent, maximum multi commodity flow, which we will call Max-MCF. We need to satisfy the capacity constraint, in that no edge can send more than its capacity. When solving the problem, we can decide how much traffic to send from each of the commodities (nodes).
In MinCong-MCF we aim to minimise the load on the most congested edge. We need to send all the commodities, but this may possibly exceed capacities. By using MCF we optimised routes from sources to destinations (according to some optimisation goal). Most times we want to use multiple (shortest) paths between hosts. So far, we assumed that we route on a **single** shortest path. This means that our routing mechanisms need to split traffic, which brings us on to ECMP.

47 ECMP

ECMP is Equal Cost Multi Path. We want routers to route on all shortest paths between a source/destination pair. Each router keeps a set of next hops along shortest paths to a destination (i.e., the first node from the current node, on the shortest path to the destination node). Will "split" traffic evenly along those next hops (routers).

47.1 Optimising

We want to set the graphs weights so that ECMP would be the optimal solution, with respect to some optimization goal. For example: minimising the most congested link but allowing total flow to exceed the link capacity. There are not always link weights such that ECMP are optimal, but we can create link weights such that ECMP will hopefully approach the optimal solution. Consider the following graph:



Here we want to send n units of flow from s to t. This can be achieved by sending 1 along their direct link, and sending $n-1$ to the next node. This node can then send 1 along its direct link, and $n-2$ to the next node. This can keep happening until n units have been sent from s to t

Finding the optimal weights under the constraints of using ECMP and weights is a problem that is NP-hard. Even finding an approximation for the min-congestion flow with *any* constant factor is NP-hard. This is even true for a single source-destination pair. So instead of solutions, we are left with heuristics. We are not going to learn the heuristics properly, but will learn the simplest which is equally splitting among the available paths.

So, we need to establish how we know how to split. One method is to simply go over all the next hops from this node, in a "round robin" fashion. This has the problem that it could lead to a reordering of the packets, due to each of the hops having a different round trip time. So, we will add the constraint that we want all the packets of the same *flow* to have the same *path*.

47.2 ECMP Hashing

We would like all packets of the same flow to traverse the same path. To do this, we will use modulo N hash, where N is the number of next hops for the destination. The hash table maps between a flow and the next hop (router). There exists many hash functions one can use, and in practice, we don't know which hash functions vendors use.

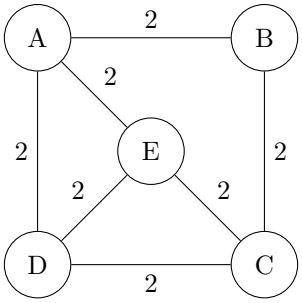
So, in order to hash, we use information from the packet header. This includes the source IP address, and port number, destination IP address and port number, and the protocol in use. These comprise the ECMP 5-tuple. This way, each packet of the same flow will have these same 5 values.

This does have some problems. Multiple "heavy" flows in the network, which share the same link, can result in poor performance. One could say that one needs to distribute the "heavy" flows differently, but this is difficult, since it is unclear to define heavy, and to identify it.

48 Questions

48.1 Definitions

Consider the following network



The capacity of each edge is 2. The weight on each edge in OSPF (uses link state) is positive/infinite On each edge, every sub-flow can be in either direction, and the total flow is the sum of all the sub-flows.

We will create the following notations:

- $Max - MCF_{OSPF/ECMP}$ is the maximisation problem of the total network flow, when using OSPF and ECMP, without restricting each flow to a single path
- $Max - MCF_{OPT}$ is the maximisation problem of the total network flow without restricting each flow to a single path, and without restricting us to equal division, any division is possible between nodes that are part of the shortest path

Similarly:

- $MinCong - MCF_{OSPF/ECMP}$ is the minimisation problem of the maximal congestion (over an edge), when using OSPF and ECMP, without restricting each flow to a single path
- $MinCong - MCF_{OPT}$ is the minimisation problem of the maximal congestion without restricting each flow to a single path, and without restricting us to equal division, any division is possible between nodes that are part of the shortest path

Recall that in both MinCong-MCF problems, **all** commodities must be sent, even if one surpasses the edge capacities, and in Max-MCF, the solution is restricted such that the edges capacities are not surpassed.

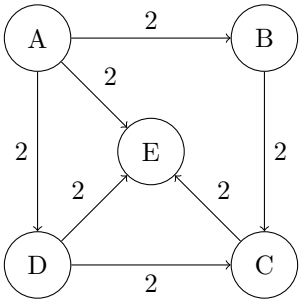
48.2 Question 1

Assume that there are two commodities (source, destination, demand):

$$(A, C, 5)$$
$$(C, E, 3)$$

What is the optimal solution of $Max - MCF_{OPT}$?

48.2.1 Solution



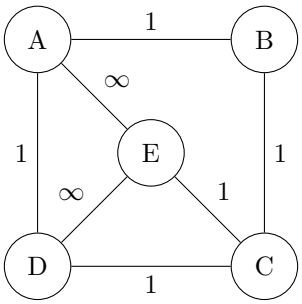
The optimal solution is 6, A sends 4 units, and C sends 2. Let us assume by contradiction that there is a better solution $s' > 6$. Let us now consider node C, for whom all the edges are saturated. Since C is in both commodities, then in s', the flow on one of the edges connected to C will increase, which violates the capacity constraints.

48.3 Question 2

Is there a feasible assignment of edges weights s.t. the solution to $Max - MCF_{OSPF/ECMP}$ yields the optimal solution from the previous question?

48.3.1 Solution

Such an assignment exists, and is shown below:



So, the packets from A to C will be equally routed through B and D, and packets from C to E will be routed directly. Solving this optimisation problem will (in this case) then imply the same flows as in the solution to $Max - MCF_{OPT}$

48.4 Question 3

Consider the following commodities:

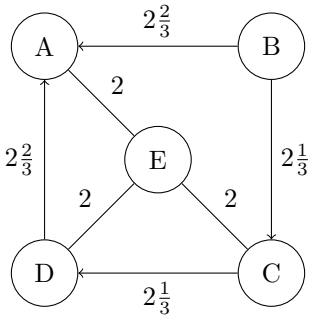
$$(B, D, 5) \tag{3}$$
$$(C, E, 3) \tag{4}$$
$$\tag{5}$$

What's the optimal solution for the $MinCong - MCF_{OPT}$ problem in this scenario? Show a flow that achieves that solution, and explain your answer.

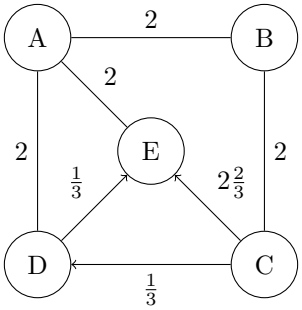
48.4.1 Solution

Let us analyse the maximum load by looking at the cut $(V, V \setminus \{B, C\})$. We will note that **all** the commodities must flow through this cut. Therefore, the minimal maximum load is lower bounded by the minimal load on this cut. The total commodities sun to 8 units, and the cut has 3 edges, with a total capacity of 6 units. Therefore, the minimal maximum load is $\frac{8}{6} = \frac{4}{3}$. So, we want a maximum of $\frac{f}{c} = \frac{4}{3} \implies f = \frac{8}{3}$ for each edge.

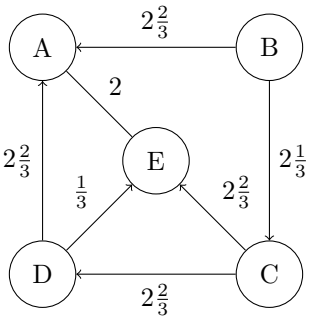
Here is a possible solution. Firstly, the flow for $(B, D, 5)$:



Now the flow for $(C, E, 3)$:



Resulting in an overall flow of:



48.5 Question 4

Suggest the weights assignment such that the solution to $Min-Cong_{OSPF/ECMP}$ (the maximum load) is 1.5. Explain your answer. This is for the following commodities:

$$(B, D, 5) \tag{6}$$
$$(C, E, 3) \tag{7}$$
$$\tag{8}$$

48.5.1 Solution

The maximum must be 3, since

$$\frac{f}{c} = \frac{f}{2} = 1.5 \implies f = 3$$

There, the edges will have the loads:

$$(A, B), (A, D), (B, C), (C, D) : \frac{2.5}{2} \tag{9}$$
$$(C, E) : \frac{3}{2} \tag{10}$$

48.6 Question 5

Is there a feasible assignment of edge weights such that the solution for $MinCong - MCF_{OSPF/ECMP}$ yields the optimal solution from question 3, which had the maximum load of $\frac{4}{3}$? If there is such an assignment, show it and explain why it yields the result, and if there is not such an assignment, then explain why.

48.6.1 Solution

There is no such assignment. We will assume by contradiction that such an assignment exists. In ECMP, if a node routes a flow to multiple neighbours, then the flow is split equally between them. Consider the node B:

- If B routes all 5 units over one edge, then the load on said edge will be $\frac{5}{2} > \frac{4}{3}$ - obviously worse than the optimal solution in question 3
- If B routes all 5 units over its two edges, then the total outgoing flows from C will be $3 + 2.5$. Considering C, the minimal load on its edges will be where it routes over both (C, E) and (C, D), and the load on the edges will be $\frac{5.4}{4} > \frac{4}{3}$, so once again, worse than the optimal solution from question 3

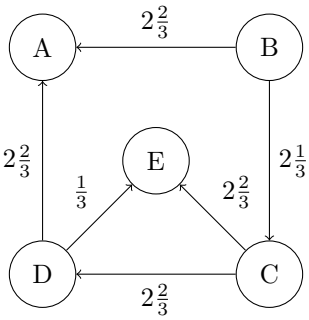
We may perform a similar analysis starting on node C which shows that C will not send on (C, B), and conclude the answer.

48.7 Question 6

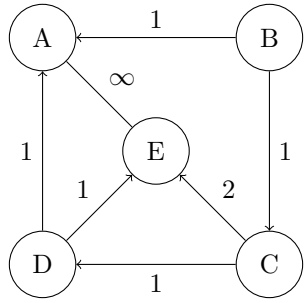
Now consider a modified version of OSPF/ECMP, in which flows can be unequally divided between neighbours with shortest paths. Is there now a feasible assignment of edges weights such that the solution for $Min-Cong_{OSPF/ECMP}$ yields the optimal solution from question 3 (maximal load= $\frac{4}{3}$)?

48.7.1 Solution

Such an assignment exists. The idea is to assign weights that will enable such a solution. For example, consider a possible solution (that yields the optimal maximal load) from question 3:



We can use the following weights assignment to yield this solution:



Part XVIII
Lecture 10 — 2025-12-28

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

49 Recap - Traffic management

We covered traffic engineering in the network layer (L3). We can also perform congestion control in the transport layer (L4), through carefully engineering the protocols. It is at this layer that concepts such as TCP, and UDP appear.

50 Transport layer

The purpose of transport services, and protocols is to provide *logical communication* between app processes running on different hosts. Transport protocols run in the end systems, the nodes on the network, rather than purely on the switches, or routers. On the sending side, the node breaks app messages into segments which it passes to the network layer, and the receiver reassembles segments received in the network layer back into messages for the application layer. There is more than 1 transport protocol available to applications: UDP and TCP.
TCP provides reliable, in order delivery. It includes congestion control, flow control, and setting up / closing connections. It ensures that when a message is sent, all of its packets will arrive at the destination host, in order. In contrast, UDP guarantees nothing, it is a minimal, no frills extension of IP.
Consider UDP to be shouting into the void. The recipient might hear, but it is not guaranteed, and if they do hear, it is possible that they will hear your words out of order, due to strange echoes. In contrast, TCP involves numbering every word you say, verifying that the recipient is ready to receive each word, and that they have heard each word every time. From this, we can intuitively see that TCP is slower than UDP. However, when one wants to receive a webpage, for example, the speed of download with a difference of potentially milliseconds is not relevant, but what is relevant is that the words on the webpage appear in the right order, and are all received.

51 Multiplexing

The transport layer also allows multiplexing messages on a single wire. We may have more than one application communication between two hosts, A and B, over the same wire. To enable this, we have the socket abstraction of the transport layer. Sockets are abstractions of IP addresses, combined with ports. Ports numbered from 0 to 65535 (16 bit number). The socket of the packet is part of the header in TCP / UDP messages. Each application sends / receives messages on a single socket, and this way the OS can separate messages that have been received from a single wire, and a single host to different applications on the computer.

Demultiplexing works as follows. When the host receives IP datagrams, each datagram has in its header the source IP address, and the destination IP address. Each datagram is responsible for 1 transport layer segment, and each segment has a source port number, and a destination port number. The host uses IP addresses, and port number to direct segments to the appropriate sockets.
UDP sockets are identified by the tuple (destination IP address, destination port number). When the host OS receives a UDP segment, it checks the destination port number in the segment, and directs the segment to the socket with that port number. It should be noted that IP datagrams with difference source IP addresses / source sockets, but the same destination port number will be directed to the same socket.
In contrast, TCP sockets are identified by 4-tuples:

- Source IP address
- Source port number
- Destination IP address
- Destination port number

The receiving host OS then directs the segments to the appropriate socket, ensuring that different transmitting hosts do not transmit to the same socket. Note that this means for web servers, they will have a different socket for every connecting client. In fact, if the clients do not use persistent HTTP connections, then there will be a different socket for every request.

52 UDP

This is a bare bones wrapper around IP. The segments sent over UDP may be lost, and delivered out of order. There is no handshake between the sender, and the receiver, and each segment is handled independently. Since TCP seems so much more stable, we are left wondering why UDP exists at all. Well, sometimes it is necessary, you cannot really have a TCP handshake in DHCP. Furthermore, the handshake can induce a (relatively) large delay before the start

of transmission, which is not always desirable.
It is also incredibly simple, with no connection state at the sender, or the receiver, it has a very small segment header, and has no congestion control, so can send messages as fast as desired. This is particularly useful in things like video / voice transmission when in a video / phone call. Since each packet contains a small amount of the data stream, losing a single packet here and there will not significantly, or even noticeably impact the experience. However, if the video stream was being sent by TCP, then should a packet be lost, then the call would freeze until that packet had been received, or worse, groups of frames would start to be played out of order, along with their attached audio, making the entire thing completely incomprehensible. Simply losing a couple of frames in the middle is greatly preferable.

53 TCP

53.1 Overview

TCP is point to point, with a single sender / receiver pair, sending a reliable, in order byte stream, and is pipelined, with congestion and flow control, according to a set window size. It sends full duplex, so data can flow in both directions in the same connection, with a defined Maximum Segment Size (MSS).

54 Opening and closing

Since we need a sender, and a receiver to establish a TCP connection before exchanging data segments, it follows that some variables need to be initialised first: The sequence numbers, the buffers into which data may be received, and the flow control information. The client is the connection initiator, sending a request to open a connection on a socket, and the server is contacted by the client, where it will accept a TCP connection on a socket. This process may be summarised by the following three way handshake:

1. The client host sends a TCP SYN segment to the server, which specifies the initial sequence number, but contains no data
2. The server host receives the SYN, and replies with the SYNACK segment. The server allocates buffers, and specifies the server initial sequence number
3. The client receives the SYNACK, to which it responds with an ACK segment, which may contain data

To close a connection, the client closes the socket. This involves two steps.

1. The client sends the TCP FIN control segment to the server
2. The server responds with ACK, and then its own FIN
3. (The client sends an ACK, but keeps the socket open for "timed wait", a period of time where it waits to see if the server will send another ACK, FIN combination, because it did not receive the clients ACK)

After the tied wait is over, the client is confident that the server has received its own ACK, and closes the socket.

54.1 Sequences and ACKs

Each segment begins with a number, known as the sequence number, which indicates the number of this segment in the stream of data. Let us consider the transmitting host A, and receiving host B. Both hosts are maintaining their own sequence numbers, which are mirrored in the other host's ACK numbers. So, for a simplified example, when A sends a segment, then along with the data it sends it with the next sequence number in its own stream, and the ACK that corresponds to the next sequence number that B will transmit. B then responds to this message with an ACK message, containing the next sequence number in its stream, and an ACK that corresponds to A's next sequence number.

Direction	Seq	ACK	Time
A $\rightarrow$ B	42	79	1
B $\rightarrow$ A	79	43	2
A $\rightarrow$ B	43	80	3

Table 25: Simplified example

As we can see, TCP creates reliable service on top of IP's unreliable service. It provides pipelined segments, with cumulative ACKs, and a single retransmission timer. Retransmission is triggered by timeout events (too much time has passed without an ACK), and duplicate ACKs.

54.2 Simplified sender

Let us consider a simplified sender, that ignores duplicate ACKs, flow control, and congestion control. Upon receiving data, create a segment with a sequence number. The sequence number is a byte stream number of the first data byte in the segment. Start the ACK timer, if it is not already running (since if there is already a segment that has not been ACKed, the timer will already be running). If the timer reaches the timeout interval, then retransmit the segment that started the timer, and restart it. When an ACK is received, if it acknowledges previously unACKed segments, then update which segments are known to be ACKed, and start the timer if there remain outstanding segments.

Part XIX
Tutorial 9 - Reliable Transport Protocols — 2026-01-01

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

55 Reliable Transport

The concept is to provide the upper layer a service abstraction of a reliable channel, where no bits are corrupted or lost, and all bits are delivered in order. In order for this to hold, we need to make the network reliable, and ensure reliable communication over an unreliable / noisy channel. This naturally has costs, and other requirements, which will be discussed.

56 First attempt

In order for the network to be reliable, we could add the feature that when a packet is seen by a switch, we check that it reaches the next hop without corruptions, and if it is corrupted, then we resend. This has a few questions, what if 2 switches fail? Where must we buffer packets, and for how long? What is the resultant goodput, throughput, and packet delay? Since the network is unreliable, the packets may be delayed, lost, or corrupted. We thus need a mechanism to monitor any of these.

For the purposes of today, we are going to assume that we can always detect errors, that each sender always has packets to transmit, and packets are received in the order in which they are sent, without bypassing.

57 Stop and Wait

This is split into two sides, the sender, and receiver.
The **sender** sends a packet, adn starts a timer. It waits for T_{out} (timeout) units, to receive an ACK / NACK response. If it receives NACK (not acknowledged), or the timeout is reached, then it resends, and resets the timer. If it receives ACK, then it sends the next packet, and resets the timeout.
The **receiver** will send an ACK if it receives a packet, without an error, and NACK (or ignore) if the packet is corrupted. For our purposes, we will probably ignore NACKs, since it only mildly improves the speeds offered by the sender, letting the sender reach timeout is just as effective.

This is not quite sufficient, since the receiver cannot distinguish between a new message, and a resent message. We can try and resolve this, by adding a binary counter in the sent packets.
This is also not sufficient, since now the sender does not know which packet was ACKed. To resolve this, we also add a binary counter in the ACKs sent by the receiver.

The timer is initialised after sending the last bit of the packet, but we need to decide what is the minimal timeout T_{out} needed for this protocol to work. We need to consider:

- Packet siize
- ACK / NACK size
- Transmission rate (bandwidth)
- Distance between stations
- Propagation speed
- Processing time (T_{pt}) (unless told otherwise, only the receiver can have $T_{pt} > 0$)

In order to calculate the timout, we will first note that we only care about the time from the last sent bit, and the *worst* case ACK / packet size.

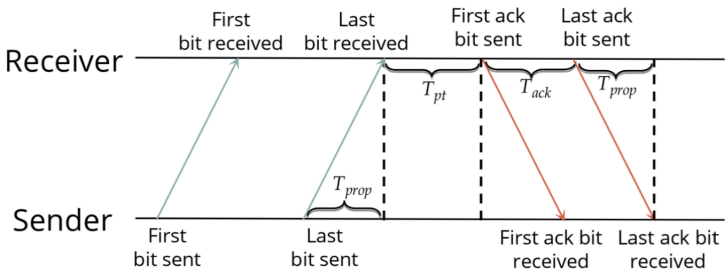


Figure 28: Timeout size

So we can see that

$$T_{out} \geq 2T_{prop} + T_{ack} + T_{pt}$$

Example 7. *Let:*

- *Packet size: 2000 bit*
- *ACK/NACK size: 200 bit*
- *Transmission rate (BW): 10 Kbps*
- *Distance between stations: 1000 km*
- *Propagation speed: 2 · 108m/s*
- *Processing time (Tpt): 0 sec*
- *What is the minimal timeout, needed for this protocol?*

Solution. We will note that the propagation speed is

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{\text{propagation speed}} \\ &= \frac{10^6 m}{2 \cdot 10^8 \frac{m}{s}} \\ &= 5 \cdot 10^{-3} s \end{aligned}$$

The ACK time is

$$\begin{aligned} T_{ack} &= \frac{\text{ACK size}}{\text{Transmission rate}} \\ &= \frac{200 bits}{10^4 \frac{b}{s}} \\ &= 2 \cdot 10^{-2} s \end{aligned}$$

So

$$\begin{aligned} T_{prop} &= 5 \cdot 10^{-3} s \\ T_{ack} &= 2 \cdot 10^{-2} s \\ T_{pt} &= 0 \\ T_{out} &\geq 2T_{prop} + T_{ack} + T_{pt} \\ &= 2 \cdot 5 \cdot 10^{-3} + 2 \cdot 10^{-2} + 0 \\ &= 3 \cdot 10^{-2} s \end{aligned}$$

□

Let us now consider the goodput. Let us assume that we use the whole bandwidth, and on failure, the receiver does not send NACK, but just ignores the packet. Packets /ACKs fail with probability

$$p \implies X \sim Geo\left((1-p)^2\right)$$

The expected number of tries until success (i.e. all attempts except the last fail):

$$\mathbb{E}[X] = \frac{1}{(1-p)^2}$$

The goodput is the ratio of effective transmission time, and total transmission time. Let T_{packet} be the transmission time of a data packet. So:

$$\text{Goodput} = \frac{T_{packet}}{\left(\frac{1}{(1-p)^2} - 1\right)(T_{packet} + T_{out}) + (T_{packet} + 2T_{prop} + T_{ack} + T_{pt})}$$

If T_{out} is set to the minimal value that we previously calculated $T_{out} = 2T_{prop} + T_{ack} + T_{pt}$ then

$$\text{Goodput} = \frac{T_{packet}}{\left(\frac{1}{(1-p)^2} - 1\right)(T_{packet} + T_{out})} = \frac{T_{packet}(1-p)^2}{T_{packet} + T_{out}}$$

We will note that doing this for *every packet* is expensive for the goodput, neatly bringing us on to the next section.

58 Go Back N (GBN)

In Stop and Wait we waste time during the timeout period. We can use this time to send additional packets. We will assume that the sender has a sliding window of size N. The sender will send N packets one after the other, and if it reaches a timeout, then it resends from the last known ACK. The receiver only sends ACKs for the packets within the sequence.

Sender: Window size of N packets, and send at most N unacknowledged packets. If a timeout occurs, resend all packets that have been sent but not yet acknowledged. $ACK(i)$ indicates that packets $1, 2 \ldots, i$ have been received correctly (meaning that it is cumulative). When an ACK is received, then the beginning of the window is moved to the next unacknowledged sequence number.

Receiver: Has a window size of a *single* packet, and if it receives a packet with sequence number i correctly, and in order (which is to say, that the last packet delivered to the upper layer has a sequence number $i-1$), then send $ACK(i)$, and deliver the data to the layer above. Otherwise the packet is corrupted, or not in order, so discard it, and send ACK for the most recently received in order packet.

The sender has a timer for each transmitted packet, which is started after sending the final bit of a packet.

Consider if we have a window size of 8, and use 3 bits to represent it. We can then desynchronise if all 8 packets are received, but all 8 ACKs are lost, since the sender will return to the beginning of the window, but the receiver will assume that this is new data. We can resolve this if for a window size of n, we use $\lceil \log_2(n) \rceil + 1$ bits to represent it.

58.1 Questions

Given a network with 2 nodes, A and B, we assume:

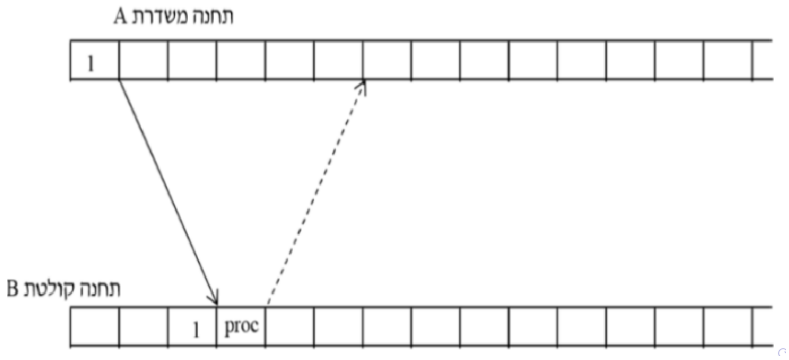
- Node A only sends data to B, B only ACKs to A
- No limit on buffers or sequence numbers
- The packet transmission (T_{packet}), propagation time (T_{prop}) and packet processing time (T_{pt}) are constant
- ACKs are delivered without any errors and $T_{ack} = 0$
- T_{out} is set to the minimal value that avoids unnecessary retransmissions
- $T_{out} = 2T_{prop} + T_{ack} + T_{pt}$
- Window size is equal to $T_{out} + 1$

58.1.1 Question 1

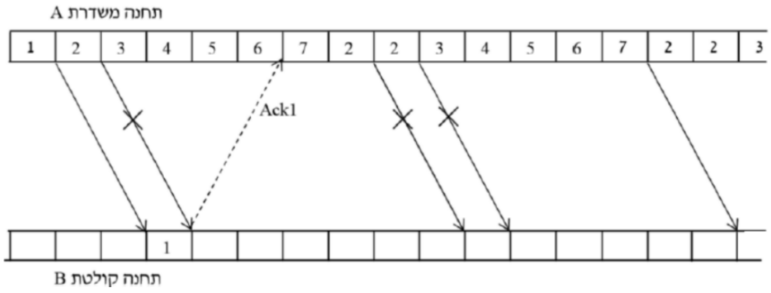
We will change the protocol as follows:

- When node A detects a failure, it re-transmits the failed packet L times
- Node A activates the timeout only after the last packet was sent
- Node A will keep transmitting the L packets even if got an ACK for one of them
- Node B will ACK the first packet and ignore all duplicates

Complete the diagram when $L = 2,\ T_{prop} = 2,\ T_{packet} = 1,\ T_{pt} = 1$, the transmissions of packet number 2 fail 3 times, and all other packets arrive successfully:



Sol. We will firstly compute $T_{out} = 2T_{prop} + T_{ack} + T_{pt} = 2 \cdot 2 + 1 + 0 = 5$. This results in a window size of 6. As can be seen, we wind up with the following answer



58.1.2 Question 2

For a given L, assume that packets fail with probability p. What is the probability to succeed with exactly k retries, where each retry consists of L packets?

Sol.

$$\text{First try: } p_0 = 1 - p \tag{11}$$
$$\text{First retry: } p_1 = p\left(1 - p^L\right) \tag{12}$$
$$\text{Second retry: } p_2 = p \cdot p^L\left(1 - p^L\right) \tag{13}$$
$$k \text{ retries: } p_k = p \cdot \left(p^L\right)^{k-1}\left(1 - p^L\right) \tag{14}$$
$$\tag{15}$$

58.1.3 Question 3

Assuming that $T_{pt} = T_{ack} = 0$, and the receiver is ready to accept the packet, what is the expected time to successfully transmit a packet?

Sol.

$$\text{First try: } t_0 = T_{packet} \tag{16}$$
$$\text{One retry: } t_1 = T_{packet} + (T_{out} + L \cdot T_{packet}) \tag{17}$$
$$k \text{ retries: } t_k = T_{packet} + (T_{out} + L \cdot T_{packet}) \cdot k \tag{18}$$
$$\tag{19}$$

The expected time is

$$\begin{aligned}
t_{avg} &= \sum_{k=0}^{\infty} t_k p_k \\
&= T_{packet} + \sum_{k=1}^{\infty} \left(T_{out} + L \cdot T_{packet}\right) k \cdot p_k \\
&= T_{packet} + pp\left(1-p^L\right)\left(T_{out} + L \cdot T_{packet}\right) \sum_{k=1}^{\infty} k\left(p^L\right)^{k-1} \\
\sum_{k=1}^{\infty} k\left(p^L\right)^{k-1} &= \frac{1}{1-p^L} \sum_{k=1}^{\infty} k\left(p^L\right)^{k-1}\left(1-p^L\right) \\
&= \frac{1}{\left(1-p^L\right)^2} \\
\implies t_{avg} &= T_{packet} + \frac{p\left(T_{out} + L \cdot T_{packet}\right)}{1-p^L}
\end{aligned}$$

58.1.4 Question 4

What is the expected goodput of the protocol?

Sol. Let $a = \dfrac{T_{out} + T_{packet}}{T_{packet}}$

$$\begin{aligned}
\text{Goodput} &= \frac{T_{packet}}{T_{avg}} \\
&= \frac{T_{packet}}{T_{packet} + \frac{p\left(T_{out} + L \cdot T_{packet}\right)}{1-p^L}} \\
&= \frac{\left(1-p^L\right) T_{packet}}{\left(1-p^L\right) T_{packet} + p\left(T_{out} + L \cdot T_{packet}\right)} \\
&= \frac{\left(1-p^L\right) T_{packet}}{\left(1-p^L\right) T_{packet} + p\left(T_{out} + T_{packet}\right) + p \cdot T_{packet}\left(L-1\right)} \\
&= \frac{1-p^L}{1-p^L + \frac{p\left(T_{out} + T_{packet}\right)}{T_{packet}} + p\left(L-1\right)} \\
&= \frac{1-p^L}{1-p^L + p \cdot a + p\left(L-1\right)} \\
&= \frac{T_{packet}}{T_{avg}} \\
&= \frac{1-p^L}{1-p^L + p\left(a + L - 1\right)}
\end{aligned}$$

In contrast, in GBN we have $L = 1$, therefore the goodput (under this question's assumptions) is

$$\text{Goodput}_{GBN} = \frac{1-p}{1-p+pa}$$

59 Selective Repeat

In GBN, we potentially retransmit many packets that were in fact received should a single packet fail. If packet 2 fails, but 3, 4, 5 all succeeded, we will still retransmit all of 3, 4, and 5, along with 2. Instead, we may retransmit selectively, and only resend the packets that failed. However, this comes at the cost of the receiver needing to store the window of messages, not just the transmitter. In the end, a minor cost.

Sender: Window size of N packets, which are sent one by one, in order. A timer for each packet is started after the transmission of the last bit of each packet, and a timeout for packet i indicates retransmitting packet i. $ACK\left(i\right)$ indicates reception of packet i *only*. If an ACK is received for i, mark it as received. If this is the first packet of the window, then shift the beginning of the window to the first unacknowledged packet.

Receiver: Stores a window size of N packets, and sends an ACK for every correctly received packet (even if it is not in order) in the window. Out of order packets with a sequence number *within* the window range are buffered until any missing packets with lower sequence numbers are received. When all missing packets are received, then deliver all in order packets to the upper layer and move the beginning of the window to the next expected (in order) sequence number.

We once again have the problem of number of bits / window size as we discussed earlier. For a window size N, we need $\geq 2N$ sequence numbers, i.e. $\lceil \log_2 2N \rceil$ bits.

Criteria	Selective repeat	Go back N
Bandwidth utilisation	Only retransmits lost packets	Might retransmit packets that have already successfully arrived
Window sizes	Sender: N, Receiver: N	Sender: N, Receiver: 1
Sequence numbers	$2N$	$N+1$
Out of order handling	Track which packets were received (out of order is possible)	No sorting, receiver only accepts packets in order
Receiver storage	Store packets until missing / damaged packets are retransmitted	No need to keep undamaged packets, since the sender may resend

Table 26:

59.1 Questions

59.1.1 Question 1

Assume an ideal SR protocol:

- Infinite sized windows on both ends
- Infinite bits for sequence numbers
- Infinite sized buffers on both ends
- ACKs do not fail and T_{ack} is negligible
- Packets fail with probability p

What is the protocol's goodput?

$$\text{Goodput} = \frac{T_{packet}\,(1-p) + 0 \cdot p}{T_{packet}} = 1 - p$$

59.1.2 Question 2

A sender (A) and a receiver (B) works with S&W with the following modification: station A transmits a window of L packets, starts a timer, and waits for all ACKs (for every packet in the transmitted window) before transmitting the next window.
In case any of the packets are not ACKed then the entire window is considered failed and station A retransmits the entire window (after the timeout)
If all packets are ACKed, station A will transmit the next window.

T_{prop} is given, T_{pt}, T_{ack} are negligible. What is the optimal T_{out}? (optimal in the sense of avoiding unnecessary retransmissions)

Sol. $T_{out} = 2T_{prop}$

From now, let us assume that $T_{out} = 2T_{prop}$, $L = 4$, $T_{prop} = 2$[unit]

59.1.3 Question 2

Draw the packets, and the ACKs transmission until packet 4 is successfully received, where packet 3 fails in its first transmission, and all other packets in the window are successful (got ACKed).

Sol.

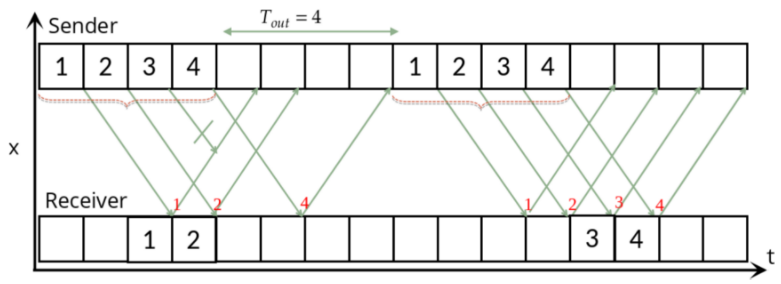


Figure 29: Solution

59.1.4 Question 3

What is the probability for an L sized window to fail?

Sol. $p' = 1 - (1-p)^L$.

59.1.5 Question 4

What is the goodput?

Sol.

$$\begin{aligned}\eta &= \frac{\mathbb{E}\left[\text{Effective sending time}\right]}{L \cdot T_{packet} + 2T_{prop}} \\ &= \frac{\Pr\left[\text{Window failed}\right] \cdot 0 + \Pr\left[\text{Window succeeded}\right] \cdot L \cdot T_{packet}}{L \cdot T_{packet} + 2T_{prop}} \\ &= \frac{L \cdot T_{packet} \left(1-p\right)^{L}}{L \cdot T_{packet} + 2T_{prop}}\end{aligned}$$

Part XX
Lecture 11 — 2026-01-04

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

60 TCP Round Trip Time / Timeout

As we have seen, a message can be lost when being sent to the recipient, or the ACK can be lost on its way to the sender. The sender cannot differentiate between these 2 scenarios, so it is very important that the sender waits long enough to receive the ACK *before* resending the message, so that it does not pointlessly transmit the same message to the recipient when there is no need.
It is important that messages are often sent before their ACK arrives, in order to save time. However, TCP uses *cumulative ACK*, meaning that if ACK(1) was lost, but ACK(2) was received, since ACK(2) implies ACK(1), it will be assumed that message 1 was in fact received as well, even without the ACK. Since the ACK values are comprised of the cumulative sum of the number of received bytes, should message 1 not arrive, but message 2 does arrive, then the ACK value for message 2 will not agree with the expected ACK value, and the sender will know that there was a missed message further back that should be resent.

We need to find a method to set the timeout value. It needs to be longer than the round trip time (RTT), but this RTT value varies. If it is too short then there will be a premature timeout, and we will have unnecessary retransmissions. However, if it is too long, then there will be a slow reaction to segment loss. As we see, we need a good estimate of the RTT. A method for estimating this is **SampleRTT**, where we measure the time from the segment transmission until the ACK receipt, ignoring retransmissions. However, this will vary, and we want the estimated RTT to be more consistent, so we take the exponentially weighted average of the most recent n measurements, rather than just the last one. This exponential weighting ensures that the older history decays over time, and a much larger focus is put on recent measurements.
To calculate the actual EstimatedRTT, we

$$\text{EstimatedRTT} = (1-\alpha)\cdot\text{EstimatedRTT} + \alpha\cdot\text{SampleRTT}$$

This gives an exponential weighted moving average, where the influence of past sample decreases exponentially. A typical value for α is 0.125.

So, now that we have the EstimatedRTT, we may set the timeout. To do this, we take the EstimatedRTT, plus some safety margin. Large variations in EstimatedRTT will result in us needing a larger safety margin. We will call this margin DevRTT, derived from the deviation of the RTT:

$$\text{DevRTT} = (1-\beta)\cdot\text{DevRTT} + \beta\cdot|\text{SampleRTT} - \text{EstimatedRTT}|$$

Where typically $\beta = 0.25$. We may then set the timeout interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4\cdot\text{DevRTT}$$

60.1 Fast retransmit

The timeout period is often relatively long, resulting in a long delay before resending the lost packet. Since the sender often sends many segments back to back, it may detect lost segments from duplicate ACKs, since if a segment is lost, there will likely be many duplicate ACKs for the preceding. If sender receives 3 ACKs for same data, it assumes that segment after ACKed data was lost, and so retransmits the segment before the timer expires. The resultant ACK for this retransmitted segment, will be the ACK of the final received segment. For example, if the sender sends packets 1, 2, 3, 4, and 5, but 2 does not arrive, then it will receive ACK(1), since that arrived, nothing for 2, and then ACK(1) 3 times in a row, for 3, 4, and 5. This tells it that 2 did not arrive, so it may retransmit 2 before the timeout expires. Upon receiving 2, the receiver will then have also received 3 - 5, and respond with ACK(5), since it is cumulative.

61 Congestion Control

We may informally define congestion as too many sources sending too much data for the network to handle. Note that this is distinct and *different* from flow control. It manifests as lost packets, from buffer overflows in routers, and long delays, since there is a large queue of messages to be sent in the router buffer.

Congestion control is important. In October 1986, the internet had its first congestion based collapse, where the link from LBL to UC Berkely, which is 365 metres, with 3 hops, and a throughput of 32Kbps, had its throughput drop to 40bps. This happened since the flow control mechanism focused only on receiver congestion, not network congestion. The large number of hosts sharing a slow and small network, resulted in the network becoming the bottleneck, rather than the receivers, but the TCP flow control *only* prevented overwhelming the receivers.

Let us take a moment to be precise about the differences between flow control, and congestion control. **Flow control** prevents the source from sending data that the recipient will not be able to handle because its buffer space

is full. This is fairly easy, and TCP handles this by having the recipient advertise its free buffer space. **Congestion control** prevents the source from sending data that will be dropped by a router, because the router's buffer is full. This is more complicated, since many different sources packets' can pass through the same router. Congestion control is handled through a combination of 2 mechanisms, protocols at the end hosts (TCP), and queuing at the routers (Droptail, RED, and so on).
There are also 2 main approaches to congestions control, **end to end**, which has no explicit feedback from the network. Congestion is inferred from the end system observed loss, and delay. This is the approach taken by TCP. There is also **network-assisted congestion control**. Here, routers provide feedback to end systems, in the form of a single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM). They also send the explicit rate at which the senders should transmit.

61.1 End to End protocols and queuing mechanisms

As we have stated, there are two families of transport layer protocols. The protocols that do not congestion control, like UDP, primarily used for time sensitive applications (VoIP, IPTV, online gaming), and servers answering small queries from many clients (DNS), and window based congestion control protocols, like TCP.

TCP sends a window of packets, in every RTT. This is to say, that it sends W packets, where W is the window size. This way, when the RTT ends, the sender receives the ACK of the first packet in the window. In order to use this, one needs to find the size of W. Now, if we limit the number of packets in the network to the window W, then the source rate is

$$\text{Source rate} = W \cdot \frac{MSS}{RTT}$$

Where

$$MSS = \text{Maximum Segment Size}$$

If W is too small, then the rate is less than the capacity, but if the rate is greater than the capacity, then that will cause congestion.

TCP congestion control is split into two main parts, Slow Start, and Congestion Avoidance. The overall concept is that we start very slowly, to prevent beginning with congestion, and slowly increase the amount of sent data, until we pass the slow start threshold (ssthresh), at which point we transition from slow start, to congestion avoidance.

61.1.1 Slow Start

Here, one starts with $W = 1$, and on each successful ACK, we increment W by 1. Whenever we reach the end of an RTT, we use exponential growth, and set $W = 2W$. When $W \geq ssthresh$, then we switch to CA.

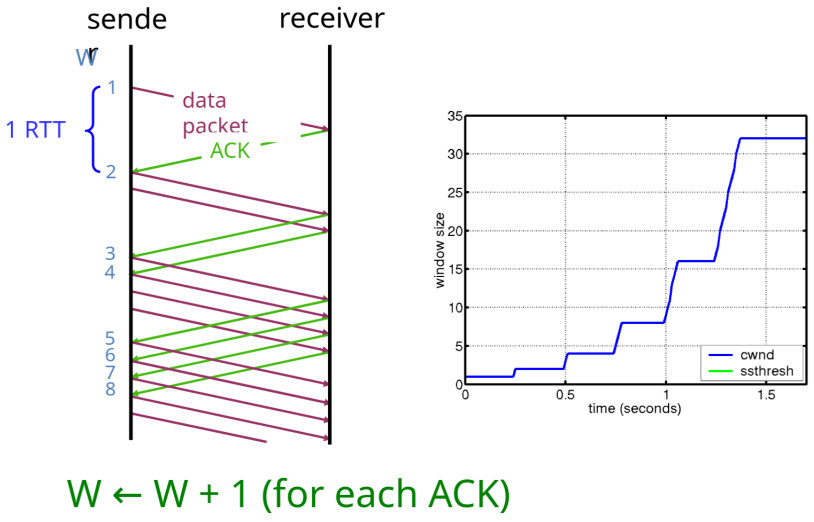


Figure 30: Slow start

61.1.2 Congestion Avoidance

On each successful ACK, set $W = W + \frac{1}{W}$. After each RTT, set $W = W + 1$.

61.1.3 TCP Tahoe

The basic idea is to gently probe the network for spare capacity, and drastically reduce the rate on congestion. Packet loss indicates congestion, and is detected by retransmission timeouts, or receipt of at least 3 duplicate ACKs. When packet loss is detected, then the slow start threshold set to $ssthresh \leftarrow \frac{W}{2}$, and then the algorithm resets to starting slow start from the beginning by setting $W = 1$.

61.1.4 TCP Reno

This avoids slow start, to prevent the network links from emptying after encountering packet loss. The basic idea is to treat timeout, and 3 duplicate ACKs differently, since every duplicate ACK indicates a successful packet transmission. It uses AIMD (Additive Increase Multiplicative Decrease). It probes the available bandwidth, and has a fast recovery to avoid a slow start. Duplicate ACKs result in a fast recovery, with a fast retransmission, but a timeout results in a retransmission, and a reset to slow start.

The basic algorithm is as follows:

Reno 1

1: **for** ACK in ACKs **do**
2: $W+ = \frac{1}{W}$
3: **end for**
4: **for** Every 3 duplicate ACKs **do**
5: ssthreshold $= \frac{W}{2}$
6: $W = \frac{W}{2}$
7: **end for**
8: **for** Every timeout **do**
9: ssthreshold $= \frac{W}{2}$
10: Enter slow start (with $W = 1$)
11: **end for**

The first for loop is the additive increase stage, and the second multiplicative decrease.

Part XXI
Tutorial 10 - Transport layer TCP and UDP — 2026-01-08

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

62 The Transport Layer

Recall that the data is encapsulated at each layer going downwards, where the application layer merely sends the data, the transport layer has provided it with an abstraction with which to do that, but adds the TCP/UDP header. Similarly, the network layer has provided the transport layer with its abstraction, and needs to add the IP header to the data, and then the data link layer adds the frame header, and trailer to all the data encapsulated in the previous layers.

We will consider two protocols, UDP, and TCP. These extend layer 3 delivery service between end hosts to a delivery service between application layer processes running on the end hosts. Recall that layer 3 (IP) only provides a "best effort" guarantee for data delivery, but data might arrived corrupted, and packets may not arrive, or arrive out of order.

To distinguish between different sockets created by application processes on our host we allocate them different addresses through the use of *logical ports*. There are 65536 (16 bits) different logical ports that may be provided to applications. In UDP, which has no consistent connection, a socket is therefore identified by the 2-tuple (destination IP, destination port), and in TCP by the 4-tuple (source IP, source port, destination IP, destination port).
When an application is ready to receive data, it allocates a logical port through a system called binding as learned in OS. Note that not all ports are available, 1024 are reserved for specific protocols, for example:

- HTTP - 80
- HTTPS - 443
- SSH - 22
- DNS - 53

63 UDP

UDP does not provide any guarantees to the user regarding the sending of data, aside from allowing multiplexing (different packets to different applications through port numbers), and packet integrity by adding a checksum to the packet. The packet will not be resent if it arrives corrupted, but the receiving application will not be sent it from the transport layer if it did arrive corrupted, it will just be dropped.

There is in fact a source port field in the UDP packet header, but it is optional. Many protocols do in fact add it to the packet, for example DNS.

64 TCP

TCP provides additional features, such as error detection and in order delivery. It makes use of checksums to detect corrupted data, and makes use of an acknowledgement system to request retransmissions in order to ensure reliable delivery. Sequence numbers are used to detect losses, and reorder transmitted data.
The main subject for this tutorial is the congestion control that TCP provides. It tries to ensure that congestion within the network is avoided if possible. Finally, it also provides flow control, to avoid overflowing the recipients buffer.

TCP makes use of sessions, one must explicitly open and close a TCP session. In this session, there will be the local sequence of ACKS. Note that this system provides no guarantees about bandwidth, or delay, since those depend on the network. Additionally, these ACKs are cumulative, similar to GBN as learnt previously.

64.1 TCP Segments

One way we could send data is byte by byte. We would need ACK for every byte sent, so this is inefficient – the overhead of headers is too high. We could instead bundle several bytes into a single segment, though this does leave us with the question about when to send the segment. There is a maximum size for a segment that we can support, which is the number of bytes in a single layer 2 frame (recall that layer 3 also adds its own header to the data, and our header).

Let us define a couple of concepts:

- MSS - maximum segment size (bytes). This is the maximum amount of application-layer data in the segment.

- MTU - maximal transfer unit (bytes). This is the largest link-layer frame that can be sent in the network.

As stated above, there is a distance between these two sizes, since we need other things to add to our packets besides the TCP segment (i.e., headers). Therefore MSS $<$ MTU.

We have three requirements for reliable transfer:

- Validate that a packet was received as it was sent (e.g., checksum)
- Notify sender that you got its message (e.g., ACK/NACK)
- Deliver the data in-order for the upper layer

There are applications that would suffer from having strictly ordered data like this, like video calls, VOIP, streaming, and so on. If a packet is lost, and then all further receiving is buffered while it waits to receive that packet, then the entire stream/conversation will lag while it waits for that packet. It is better to simply lose the packet, have a drop in quality for a moment, and then continue as normal.

64.2 Connections

64.2.1 Establishing a connection

A client initiates a connection with a server, and once it is initialised, both the client, and the server may use the connection simultaneously (meaning it is in full duplex). One of the initialisation steps is agreeing on the sequence numbers, since in order for them to have meaning, the sequence scheme needs to be established first.

The connection is established through a **three way handshake**.

- The client begins by sending **SYN**, with the field $seq = x$
- The server responds with **SYN-ACK**, with the fields $ack = x + 1, seq = y$
- The client then responds with **ACK**, with the fields $ack = y+1, seq = x+1$
- The client may now begin sending data

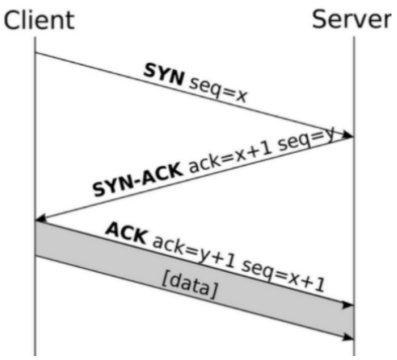


Figure 31: Three way handshake

If the SYN packets are lost, then eventually, no SYN-ACK will arrive, so the client sets a timer waiting for SYN-ACK. If it is not received, then the client will retransmit the SYN. This does leave us with the question of how long should the initial timeout be, since the client has no idea what the round trip time to the server is. Some TCPs use a default of 3 or 6 seconds, which is relatively long in comparison to the actual RTT value.

64.2.2 Closing a TCP connection

When the client wishes to close the connection:

- The client sends FIN to the server, closing its connection
- The server responds with ACK on the FIN
- The server sends a FIN message to the client, closing its side of the connection
- The client responds with ACK to the FIN

64.3 TCP sliding window

Just like in Go Back N, and Selective Repeat, TCP uses a sliding window scheme, where there will be at most N packets in flight. However, unlike GBN, and SR, the window size is **dynamic**, so we need to decide how to set it. We have two main parameters dictating the window size:

- The buffer size of the receiver (Flow control)
- The current "congestion" of the network (Congestion control)

64.3.1 Flow Control

We need to have some knowledge on the available buffer size of the receiver. The receiver sends the amount of buffer it can allocate for this connection in its ACKs, which is denoted as **rwnd** (receiver window). The sender will keep the invariant of

$$\text{Window size} = \text{LastByteSent} - \text{LastByteAcked} \leq rwnd$$

If rwnd = 0, then we want the sender to keep on sending datat to the receiver. These packets will be ACKed by the receiver, and eventually the receiver buffer will free up, and the sender could ramp up the amount of data it sends.

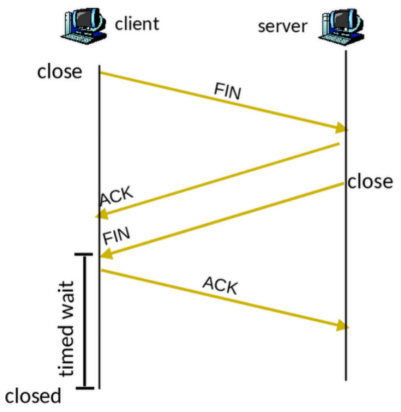


Figure 32: Close the TCP connection

64.3.2 Congestion control

the sender needs to approximate the load the network can sustain at any given moment. However, there is no feedback on congestion from the network. We may instead approximate this by the loss of ACKs, or expired timers. TCP needs to decide when to increase or decrease the congestion window size (ie, the number of packets we believe that the network can sustain).

In TCP Tahoe, and TCP Reno there are two states in which a TCP connection can be:

- Slow Start - Start sending a very small amount of data, and increase it over time
- Congestion Avoidance - Be prepared to handle congestion

There is also TCP Vegas, which is delay based, rather than loss based.

The sender starts with a pre set slow start threshold: ssthresh. This is the parameter that determines when to switch from Slow Start to Congestion Avoidance. The sender also needs to decide when to increase / decrease the congestion window size. SS works as follows:

- Start with a small window size (W = 1 MSS)
- Double the congestion window (cwnd) every RTT
- For every ACK: $W = W + MSS$
- Enter congestion avoidance when $W \geq ssthresh$

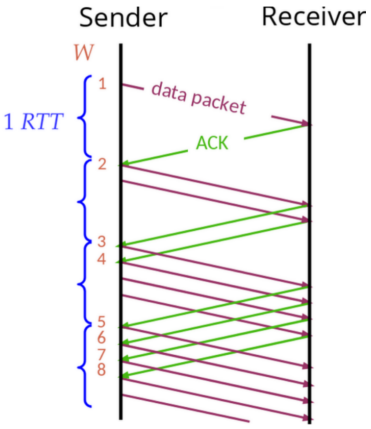


Figure 33: Slow start

Exponential increase stops either when some failure occurs, or when we are sending too much (even without failures), though this requires choosing a threshold.

When in congestion avoidance, for every N ACKs (ACKs for all packets in the window): $W := W + 1$. This results in a small additive increase – adds one MSS every RTT.

Packets should be resent after a timeout. However, timeouts can be relatively long, and waiting for that can increase latency. So instead, we can resend after getting duplicate ACKs. These are an indication that some packets failed, but afterwards some packets succeeded. In practice, we wait for 3 duplicate ACKs, and detecting them requires less time than waiting for the entire timeout. If packet 4 is lost, but 5, 6, 7 are all received, then packet 3 will be ACKed 3 times, and so we know that 4 was lost, but 5, 6, and 7 were all received, so 4 should be retransmitted.

Not every TCP variant implements this Fast Retransmission mechanism. In those that do, after receiving 3 duplicate ACKs, and retransmitting the relevant packet, depending on the TCP protocol (e.g. Reno / Tahoe), the sender may switch to SS, / CA, change the size of ssthresh, and so on. Note that those changes are not a part of fast retransmission, but a reaction to 3 duplicate ACKS.

On a packet loss, timeout, or 3 duplicate ACKs, both Tahoe, and Reno set $ssthresh = \frac{W}{2}$. Other changes depend on wheter a timeout, or 3 duplicate ACKs occurred:

	TCP Tahoe	TCP Reno
Timeout	Enter slow start, and set W = 1	Enter slow start, and set W = 1
3 Duplicate ACKs	Enter slow start, and set W = 1	Set $W = \frac{W}{2}$, and enter congestion avoidance

Table 27: Tahoe vs Reno

One may see an interactive example at this link.

So now we have the two parameters, the receiver buffer size (rwnd), and the congestion window (cwnd). TCP will select the minimal window of the two, meaning that the actual window size is:

$$\text{LastByteSentLastByteACKed} \leq \min\{rwnd, cwnd\}$$

65 Network Address Translation (NAT)

NAT is a mechanism that is optionally used by routers to reduce the number of devices using public IPv4 addresses, instead reusing a pool of private addresses. Recall that there are not many IPv4 addresses in the world, only around 4 billion, so we use NAT such that all the devices behind a single router have only the router's public IPv4 address, and it handles multiplexing the data to the relevant address within the network. This also has a possible privacy improvement.

Private IP addresses were thus constructed, these are reserved addresses that can only be used inside internal (private) LANs, they cannot be used in connections to the internet. For a device with a private IP address to communicate with devices outside of its LAN, its IP address needs to be translated to a public IP address. There are a few private ranges, such as 10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255, and 192.168.0.0 - 192.168.255.255.

All devices in the LAN have private IP addresses. The router that has access to the internet is assigned a single public IP address, which will be used for the entire LAN. The router changes the Source IP and Source Port fields of outgoing packets and Dest IP, Dest Port of incoming packets. Note: this is different from "regular" routing, where routers do not change these fields. The router keeps a translation table, where it converts the translated port, and IP to the original port and IP. This is also matched based off protocol (TCP, UDP).

Consider a LAN with 3 computers, A, B, and C:

- A has the IP 192.168.0.1, and runs two TCP applications communicating with the internet of 1024, and 2000
- B has the IP 192.168.0.2, and runs two UDP applications communicating with the internet of 1000, and 2000
- B has the IP 192.168.0.3, and runs one TCP application communicating with the internet of 1024

The network's assigned public IP is 132.58.38.2.

Below is the NAT table, where we assume that the translated port numbers start at 5001 and increase by 1 when a new row is added to the table.

L4 Protocol	Outside network		Private Network	
	Translated L4 port	Translated IP	Original L4 port	Local IP
TCP	5001	132.58.38.2	1024	192.168.0.1
TCP	5002	132.58.38.2	2000	192.168.0.1
UDP	5003	132.58.38.2	1000	192.168.0.2
UDP	5004	132.58.38.2	2000	192.168.0.2
TCP	5005	132.58.38.2	1024	192.168.0.3

Table 28: NAT table

Part XXII
Lecture 12 — 2026-01-11

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

66 TCP Reno

This avoids slow start, to prevent the network links from emptying after encountering packet loss. The basic idea is to treat timeout, and 3 duplicate ACKs differently, since every duplicate ACK indicates a successful packet transmission. It uses AIMD (Additive Increase Multiplicative Decrease). It probes the available bandwidth, and has a fast recovery to avoid a slow start. Duplicate ACKs result in a fast recovery, with a fast retransmission, but a timeout results in a retransmission, and a reset to slow start.

The basic algorithm is as follows:

Reno 2

for ACK in ACKs **do**
    $W += \frac{1}{W}$
end for
for Every 3 duplicate ACKs **do**
    ssthreshold $= \frac{W}{2}$
    $W = \frac{W}{2}$
end for
for Every timeout **do**
    ssthreshold $= \frac{W}{2}$
    Enter slow start (with $W = 1$)
end for

The first for loop is the additive increase stage, and the second multiplicative decrease.

Let us consdier the *throughput* of TCP Reno. We will calculate it on average as a function of window size, and RTT, ignoring slow start, and other buffers. Let W be the window size when a loss occurs. When the window is W, then the throughput is $\text{MSS} \times \frac{W}{RTT}$. Just after the loww, the window size drops to $\frac{W}{2}$, and the throughput to $\text{MSS} \cdot \frac{W}{2RTT}$. Thus, the average is

$$\frac{3}{4} \cdot \text{MSS} \cdot \frac{W}{RTT}$$

Let us define a *fair* protocol. If k TCP sessions share the same bottleneck link of bandwidth R, then each should have an average rate of $\frac{R}{k}$. We can see that TCP Reno is fair from the following example. Consider 2 competing sessions, and a graph where on x we have the throughput of connection 1, and on y the throughput of connection 2. There is a line connecting the R value on both axes, representing the maximum throughput. Fairness is at the point where this line meets $y = x$. We will note (from testing) that no matter where they start, both will converge on that point.

This does not happen on the internet, since many applications (such as multimedia) do not use TCP, since they do not want their rate throttled by congestion control, so handle packet loss, but transmit at a constant rate. Additionally, for all TCP may be fair, our browser can open many connections, and thus bypass each connection being "fair", since now it is taking up 10 times the bandwidth proportion.

So, we have two variants, Reno, and Vegas. They differ mostly in their congestion avoidance. Vegas is delay based, and Reno is loss based. Vegas ran into issues, where despite its model working excellently in the laboratory, when it worked with *other* TCP protocols, its transmission rate went to almost nothing, since non of the other protocols were reducing the throughput in a similar manner.

67 Queuing at routers

Congestion control is a distributed asynchronous algorithm to share bandwidth. TCP adapts the sending rate to congestion, and router queuing adjusts, and feeds back congestion information. There are 2 main ways, the naïve method is *drop tail*, which is a FIFO queue, that drops packets that arrive at a full buffer. There are also implicit feedback models, where the queueing process implicitly computes and feeds back the congestion measure. Active queue management gives **explicit** feedback, which provides congestion information by probabilistically marking packets. There are 2 ECN bits in the IP header that are allocated for active queue management. This is supported by routers, but is usually turned off.

68 Interdomain routing and BGP

We have already seen and discussed many routing methods, which all tried to find the shortest route from point A, to point B, within a single network. However, the wider internet is comprised of *many* different networks, which can range from small businesses / schools, to large multinational corporations. Every such network is called an Autonomous System (AS). For this routing, we will create the topology where each node is an AS, destinations are prefixes (such as 12.0.0.0/8) , and edges are links / business relationships. Autonomous System Numbers (ASNs) are 16 (or sometimes 32) bit values. For example, MIT has 3, Harvad 11, HUJI 378, AT&T have 7018, 6341, 5074, and more. Verizon have 701, 702, 284, 12199, and more.
In the commercial internet, ASes sign bilateral long term contracts about how much traffic to carry between them, which destinations they reach, and how much money they pair for this server. Neighbouring pairs of ASes typically have a *customer provider* relationship, or a *peering* relationship.

For the customer provider relationship, the customer needs to be reachable from everyone, so the provider ensures that all its neighbours can reach the customer
For the peering relationship, peers exchange traffic between customers. Often this relationship is settlement free (no money exchanged).

At the top of the internet hierarchy, we have the tier 1 ISPs. These have no upstream providers, and typically have a large (inter)national backbone. There are around 10-12 ASes, comprised of AT&T, Sprint, and others.
Tier 2 providers provide transit service to the downstream customers, but need at least one provider of their own. They typically have national, or regional scope, and there are a few thousand such ASes. There are also stub ASes, which do not provide transit service, and connect to upstream providers. These stubs comprise of most (about 85%) of the ASes (like HUJI).

68.1 Interdomain routing

Different ASes are connected to each other through border routers, which communicate over the protocol BGP (Border Gateway Protocol. This uses interdomain routing, performed between ASes, across different entities. This contrasts to intradomain routing, which we have discussed until now, which is within a single AS, where all the network devices belong to the same entity.
BGP is **not** shortest path routing. Different requirements may be placed on the route. For example, Google might require the cheapest route, Verizon might want the cheapest, but Comcast and AT&T might want routes that avoid each others networks.
BGP is critical for routing in the internet. For example, almost 50% of Skype disruptions are BGP related, and every year or so, a serious BGP related Internet outage makes the news. It is notoriously vulnerable to attacks.

68.1.1 Routing overview

Every node has an AS number, and each AS has an IP prefix. Each AS announces itself to its neighbours, including the AS number, and its prefix. These neighbours can then choose whether to make use of this route to the IP prefix, and distribute it to their neighbours, or make use of a different route. Routes to the destination are built hop by hop as reachability information propagates through the network, and route selection is based on local routing policies.
As we can see, this means that BGP is a distributed protocol, where for each destination IP prefix every node repeatedly executes the following process, upon receiving BGP routes from its neighbours:

- Apply *import policy*: (e.g. filter unwanted routes from neighbours)
- Select "best" route: Choose the best route according to the local preferences (distance, cost, speed, avoiding certain people)
- Apply *export policy*: Filter routes that you do not want to announce to neighbours
- Transmit the chosen BGP route to the neighbours

68.1.2 Key points

1. Independent computation for **each** IP prefix: This means that routes to **every** IP prefix are computed *independently*
2. Entire path announced: Under BGP, the **entire** AS level path to the destination is announced when a link is published (so, when AS 2 builds a link to AS 1, AS 2 will announce the path (2, 1), as opposed to AS 1 which only announced path (1)). This was originally intended for loop detection purposes
3. Local routing policies: BGP allows ASes to express complex local routing policies, such as node 2 preferring the path (2, 3, 1) over (2, 1), and node 1 may not let node 3 hear the path (1, 2).

Part XXIII
Tutorial 11 - BGP — 2026-01-15

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

69 Introduction

So far, the routing protocols that we have discussed generates forwarding tables, either using Distance Vector, or Link State. This is however, only aimed at small networks, and not appropriate for the wider internet. We need to extend this to the wider internet, which allows for different interactions between different networks, and operators, that may be unwilling to work with each other, or perhaps have different requirements over how the packets should be sent. Effectively, we have added *politics* into the picture, which complicates everything.

70 Autonomous Systems

The internet is comprised of multiple networks, and each network has its own routing policy, financial goals, and administrative unit (Google, AT&T, HUJI, Bezeq, etc,). We will call each of these networks an *Autonomous System*, or AS for short. Each AS is identified by a unique number (ASN).

ASes do not necessarily differentiate things geographically, just topologically. Packets will be sent between them according to their preferences. Google may want to send packets in the cheapest manner possible, another AS may be looking to provide its customers with the best service, and so look for the shortest route, and two providers may decide that they do not care how it affects the service / price, as long as they do not talk to each other.

Note that these different goals mean that routing will **not** necessarily be the shortest path. We do not even know the routes that packets will take *within* a different AS. As a result, a different form of routing, that incorporates these types of demands, is needed. Since these entities often want to make money, and thus *enhance shareholder value*, we need to take the financial relationship between them into account.
ASes sign bilateral, long term contracts between them, that concern themselves with how much traffic they carry which destinations can be reached from them, and how much money changes hands in order to achieve this. Neighbouring pairs of ASes typically either have customer provider relationships, or peering relationships:

- Customer/Provider: One AS pays another for reachability to some set of destinations
- Peering: Two ASes exchange traffic for free, typically only for one or two of their customers

The relationship may either be “full”, where the customer receives routes for all Internet destinations from its traffic provider, or “partial”, where the customer receives routes for some subset of all Internet endpoints.

Peering is motivated by increased redundancy, routing control, traffic management, and predictability of traffic. It also reduces latency, congestion, costs, and more.

There are 3 types of ASes:

- Tier 1 (There are roughly 10): This is the topmost level, with no providers. All ASes are fully connected to each other
- Tier 2 (Roughly 1000): Needs at least one Tier-1 provider, and provides transit service for the customers, generally on either a regional, or national scope
- Stub (Roughly 85% of all ASes, eg HUJI): Connects to one or more providers, and does not provide transit services

71 BGP

Recall that IP addresses are 4 byte numbers (32 bits), presented as ip/k: The first k most significant bits are fixed. For example, on 240.161.192.0/18, the first 18 bits are fixed for this subnet, and the remaining 14 are the available addresses.

BGP (Border Gateway Protocol) is an extension of Distance Vector routing, which supports flexible routing policies. It’s key idea is to advertise the *entire* path to an IP prefix. So, when AS 2 builds a link to AS 1, AS 2 will announce the path (2, 1), as opposed to AS 1 which only announced path (1). This was originally intended for loop detection purposes.

As we have already seen extensively in this course, BGP is a distributed protocol, meaning that every node / AS runs its computation independently. Every AS announces the IP prefixes that may be reached from it, to its neighbours, and they utilise these data, along with their existing routes, in order to update the routes that they will use. The appropriate data will then be sent to update their neighbours, and so on.
Every AS chooses routes that are based on its local routing policies. The routes to the destination are built hop by hop, and usually, an AS announces only its **chosen** routes, rather than **all** the routes of which it is aware. Note, different routes for incoming, and outgoing traffic is possible.

The BGP protocol follows the following steps:

1. Receive the BGP routes from neighbours
2. Apply the import policy (remove unwanted routes, maybe too expensive, there is a preference for AS n, and so on)
3. Select the "best" route according to the local policies
4. Apply the export policy (filter out the routes that you do not want to announce to your neighbours)
5. Transmit the chosen BGP route to neighbours

BGP occurs over TCP, on port 179. BGP connected routers establish the connection, exchange all active routes, and then exchange incremental updates while the connection is alive.
These incremental updates include **Announcement** updates, where upon selecting a new active route, the AS adds its own ASN to path and (optionally) advertises the path to each neighbour, and **Withdrawal** updates, where if the active route is no longer available, send a withdrawal message to the neighbours.

In order to avoid loops, whenever a node (router) observes a new path, it can search if it's already on the path, and if so, drop the new path. Also, note that BGP is neither a distance vector, or link state vector, since it is a path based protocol in the sense that we send the complete path, and not just the next hops. This is unlike link state, since each edge router **only** reports its routes to neighbour edge routers, and it is unlike distance vector, since the full paths are sent.

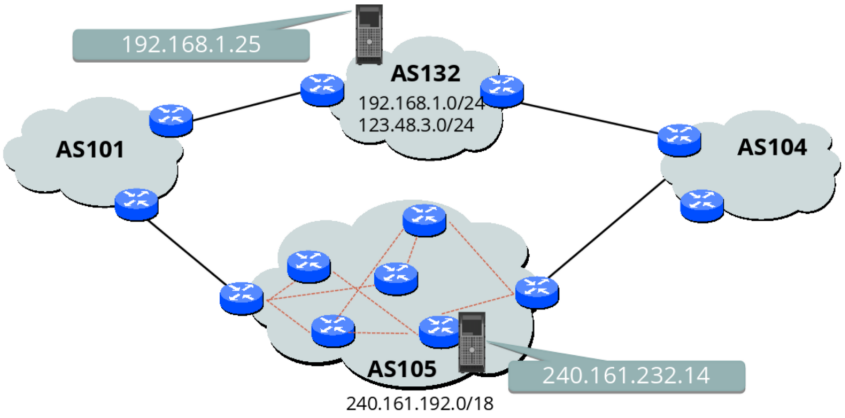


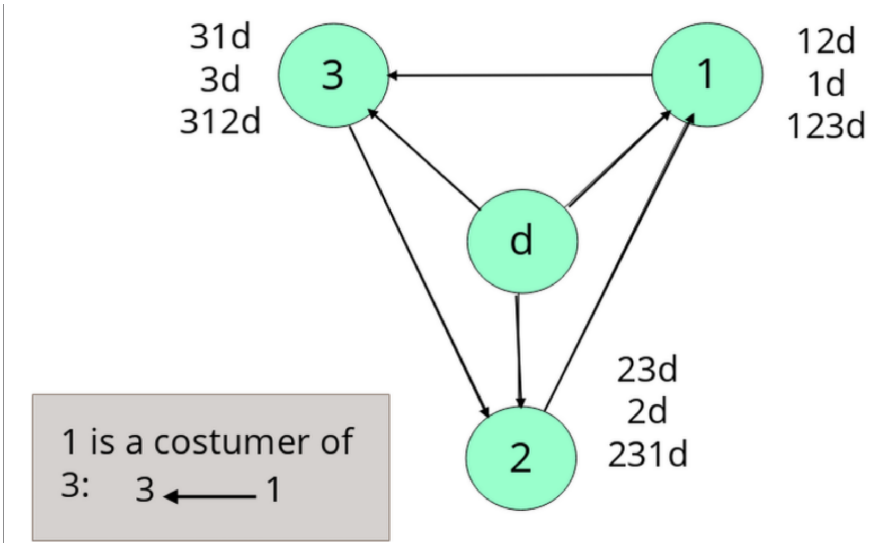
Figure 34: Routing between ASes

How will we route from 240.161.232.14 to 192.168.1.25?
First, we need to reach one of the edge routers. These are the BGP routers which are connected to other ASes' edge routers. This is followed by Internal Border Gateway Protocol (iBGP). This is a routing protocol for routers within the same AS, which announces reachability to external destinations. Finally, Interior Gateway Protocol (IGP) is used to compute paths within the AS (e.g., distance vector or link-state based protocols). In each router (edge/internal), it maps an egress point (edge router) to an outgoing link (next hop towards the egress point).

72 BGP Stability

The customers can have multiple preferences. However, this does not mean that the routes are static, there can be a hierarchy of preferences which change the selected routes, resulting in the paths potentially changing over time. This can happen when preferences change, when previously existing paths are no longer available, when new paths are available, and so on.

A stable BGP state is a state when no AS wants to change its routes. We will note that BGP is **not** always stable. Consider:



Assuming that every AS announces its chosen route to its neighbours, does a stable state exist in this network? Let us assume towards contradiction that there is. AS(1) has 3 possible routes (12d), (1d), (123d). Note that the orders of available routes are in order of preference. If AS(1) chooses (12d), then AS(2) **must** choose (2d). AS(3) cannot choose (31d), based off AS(1)'s choice, so it chooses (3d). However, in this case, AS(2) thus switches to the route (23d), and so AS(1)'s choice is no longer available.
Let us assume instead that AS(1) chooses (1d). So, AS(3) chooses (31d). Since for AS(2), the route (23d) does not exist, it chooses (2d), but then AS(1) will switch to its preferred route (12d).
Finally, should AS(1) choose (123d), then AS(2) has by definition chosen (23d), and AS(3) has chosen (3d), but then once AS(1) hears of the route (1d), it will switch to it, as it is AS(1)'s preferred route.

72.1 BGP Safety

Given a network running BGP, we say that it is **BGP Safe** if it (eventually) converges to a single, stable, BGP state, no matter from which state it started, or the message propagation time. We do not really know (yet) when a BGP network will be safe, but we do have the following theorem:

Theorem 1. *The existence of more than one stable BGP states in a network implies that it is* ***not*** *BGP safe*

In order to find stable states, we *could* try every state, but there are simpler methods.

72.1.1 Gao-Rexford Conditions

We can guarantee BGP safety in a network if **all** the following conditions hold:

1. **Topology condition**: No customer-provider cycles in the AS graph
2. **Preference condition**: Prefer customer learned routes over provider, and peer learned routes (go by preference through paying customers, than through peers)
3. **Export condition**: Prover and peer learned routes are exported *only* to customers

We will note that in the above example, the topology condition did not hold, AS(1) is a customer of AS(2), who is a customer of AS(3), who is a customer of AS(1). We can change this, by swapping the arrow from AS(1) to AS(3), to be from AS(3) to AS(1). We will note that now the preference condition does not hold, since AS(3) has a preference for sending to its provider AS(1), rather than its customer d. This may be fixed by swapping AS(3)'s preferences to be (3d), (31d), (312d).

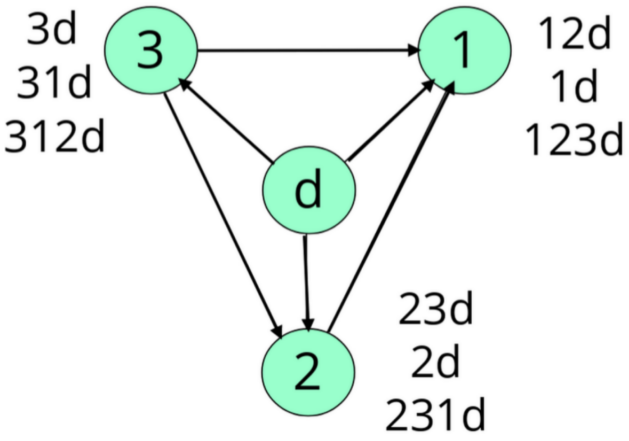


Figure 35: Stabilised version

AS(3) will choose the path (3d), and announce it to AS(1), and AS(2), since it is a customer learned path. AS(2) will choose the path (23d), ad announce it to AS(1), since it is a customer learned path. Finally, AS(1) does not know the path (12d), since it was not announced by AS(2), and so it will choose the path (1d). Everyone has their preference of paths, and so the network has stabilised.
There are not too many questions to be asked on this subject, they generally follow the format of “is this network stable”, and if it is not stable “change this network to be stable”.

Part XXIV
Lecture 13 — 2026-01-18

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

73 BGP details

Firstly, we will note that BGP takes place at the *application* layer. It takes place over TCP, on port 179. BGP connected routers establish the connection, exchange all active routes, and then exchange incremental updates while the connection is alive.
These incremental updates include **Announcement** updates, where upon selecting a new active route, the AS adds its own ASN to path and (optionally) advertises the path to each neighbour, and **Withdrawal** updates, where if the active route is no longer available, send a withdrawal message to the neighbours.

In order to avoid loops, whenever a node (router) observes a new path, it can search if it's already on the path, and if so, drop the new path. Also, note that BGP is neither a distance vector, or link state vector, since it is a path based protocol in the sense that we send the complete path, and not just the next hops. This is unlike link state, since each edge router **only** reports its routes to neighbour edge routers, and it is unlike distance vector, since the full paths are sent.

Let us consider these route update messages. They include the destination prefix (here denoted d), and the route attributes, including

- AS path (the route through the different ASes that it took to reach the recipient)
- The next hop IP address (this is the router from which the message left the current AS, such that the next AS may send messages back along the route)

74 BGP (In)Stability

There are some structures of ASes which can cause infinite oscillation, resulting in an unstable state of the network. This makes the network unpredictable, and hard to debug. It might lead to the network being constantly flooded with BGP updates, using up all the available bandwidth. This instability can also cause deteriorated instability.

We want to guarantee that BGP will always reach a stable state, and ideally this should happen quickly. This is called BGP **safety**. There are two practical mechanisms for enabling this:

1. Penalising "flaps" (route fluctuations)
2. Enforcing a minimum time interval between BGP updates (to the same neighbour).

These are only occasionally used, since they are only occasionally effective.

We will note that the internet is generally fairly stable. BGP routes to popular destinations tend to be the same for around 10 days, and most traffic in the internet travels along stable routes. Part of the reasoning here is that there are no customer, provider cycles. ASes prefer to use their customers for routing, than their providers, since that way they make money. Additionally, ASes do not announce their provider learned routes to other providers, if only from a money making standpoint.

74.1 Gao-Rexford conditions

If the following 3 conditions hold, then the BGP network is stable. We can guarantee BGP safety in a network if **all** the following conditions hold:

1. **Topology condition**: No customer-provider cycles in the AS graph
2. **Preference condition**: Prefer customer learned routes over provider, and peer learned routes (go by preference through paying customers, than through peers)
3. **Export condition**: Prover and peer learned routes are exported *only* to customers

So, to summarise BGP, it enables network operators great expressiveness at the potential cost of persistent route oscillations. Protocol divergence is bad for networks, since the network becomes unpredictable, and there is performance degradation. Gao-Rexford conditions imply BGP safety, which is a partial explanation for the Internet's relative stability.

75 Inter vs Intra

An AS is **not** a single node, but rather is comprised of multiple routers. Thus, we need to distribute BGP information within the AS. In order to achieve this, Internal BGP (iBGP) sessions are held between the routers of a single AS.

To make this happen, we need to join together information from iBGP, and IGP (Interior Gateway Protocol). iBGP announces reachability to *external* destinations, and maps a destination prefix to an egress point (we can reach 128.112.0.0/16 from 192.0.2.1), and IGP is used to compute paths *within* an AS, and masp an egress point to an outgoing link (we may reach the external IP 192.0.2.1 via the AS internal IP 10.1.1.1).
IGP is shortest path routing with static link weights. It computes the shortest paths to other routers based on the link weights, in order to determine the next hop to every other router. The link weights are configured by the network administrator.

We will note that since an AS has many edge routers, it may have numerous connections to a neighbouring AS that are equivalent. This means that multiple border routers may learn good routes with the same LocalPref and AS-path length. We are thus left with the question of how the internal routers know which egress point to use. To resolve this we use *hot potato routing*. Each router selects the *closest* egress point based on IGP link weights. So, to put it all together, a route from a router within the AS, to another AS is chosen by:

1. Highest **local preference**
2. **Shortest** AS path
3. **Closest** egress point
4. **Arbitrary** tie break

75.1 Summary

ASes are networks of routers, which combine together iBGP, and IGP routing. iBGP announces reachability to *external* destinations, and IGP computes routes *within* the AS.

76 BGP security

BGP is built on trust, and is thus exceedingly vulnerable to bad actors. Every few years, there is a serious BGP based attack that makes the news.

76.1 Session security

BGP sessions run over TCP. There is a TCP connection between neighbouring BGP routers, and they send their BGP messages over this TCP connection. This makes BGP vulnerable to attacks on TCP. One could eavesdrop by tapping the link to infer routing policies, tamper with the packets that are traversing the link, by dropping, modifying, or adding packets, thus changing, filtering, or replaying BGP routes. One may also reset the session, and congest the link.

Fortunately, one may quite easily defend a BGP session. Two end points can use known IP addresses, and ports to communicate, and agree to authenticate and encrypt their messages. Additionally, there is often very limited physical access to a link, since for most of these, one needs direct physical access, and the link is often located in a (somewhat) secured building.

76.2 Manipulating BGP

BGP is eminently vulnerable to manipulation. In 2008, Pakistan ordered their ISPs to block YouTube. To do this, their ASes started publishing that YouTube is at a different location, which led nowhere. This leaked (accidentally) to the wider internet, and brought YouTube down for a few hours. This happens easily since there is no proof that it is actually true that the person claiming to own an IP block *actually* owns that IP block, so BGP simply believes that whoever claims to own it, actually does own it.
This is called Prefix Hijacking, where two ASes are competing as to who owns a prefix. The hijacking AS needs a router with BGP sessions that is configured to originate the prefix. This could happen because a network operator makes a mistake, a network operator launches an attack, or an outsider breaks into the router, and reconfigures it.

The targeted AS may not even notice that prefix hijacking is going on. If targeted AS tries to snoop, and check, they may only experience degradation, and not the full hijack. The best way to diagnose it is to analyse updates from *many* different vantage points, by launching traceroute from many different vantage points.

76.3 Origin Authentication

To defend against prefix hijacks, there are many best common practices, such as filtering routes based off their prefix (so for example, stubs should only announce their own IP prefixes), and packet filters to avoid unwanted traffic. This

is not good enough, it depends on vigilant application of BCPs, and does not handle misconfiguration (malicious, or otherwise). It also does not address the fundamental question of who own what IP block.

The proposed solution to this is called **Origin Authentication**. Here, we use a secure database that maps IP prefixes to owner ASes. This is imperfect, since a bad actor can convince others that they have a nonexistant link, which is shorter than the standard link from it to some IP prefix. This attracts both the sources that want a shortest link, and the sources that are trying to avoid the ASes that the bad actor has cut out of its route.