Foundations

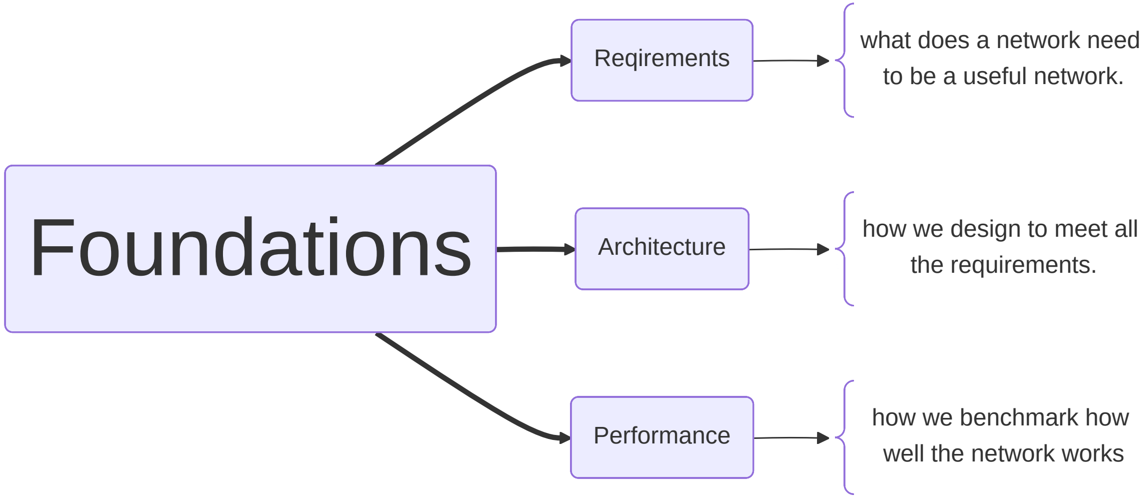
of data communication networks

Objectives

1. Reflect on and collect **requirements** that are the basis of our definition of a *network*.

2. Introduce the ideas of **architecture** (layers and protocols) and apply structured problem solving.

3. Define **performance** metrics for how we measure success.



Doing that thing Computer Scientists do...

“How did I get here?”

- Talking Heads

Applications

The Web

Based on Hypertext Transfer Protocol (HTTP)

Designed for marked up text, Hypertext Markup Language (HTML)

Includes linking, descriptive formatting, style, images, video, ...

Incredibly flexible, often used by higher-level applications

Applications

The Web

The first killer app for the Internet

Email

- Simple Mail Transport Protocol (SMTP)
- Post Office Protocol (POP)
- Internet Message Access Protocol (IMAP)

Now more common to access email via web (HTTP).

Applications

The Web

Email

File Storage

Even before the cloud there was network storage

- Network File System (NFS)
- Common Internet File System (CIFS)
- IP Small Computer System Interconnect (iSCSI)

...and of course, over HTTP these days

Applications

The Web

Email

File Storage

Multimedia

Often hidden behind web browser or media player

Motion Picture Experts Group (MPEG) is most common

- Audio - Layers 1, 2, 3 (mp3)
- Images - JPEG
- Video - MPEG-2, ..., MPEG-4

Applications

The Web

Email

File Storage

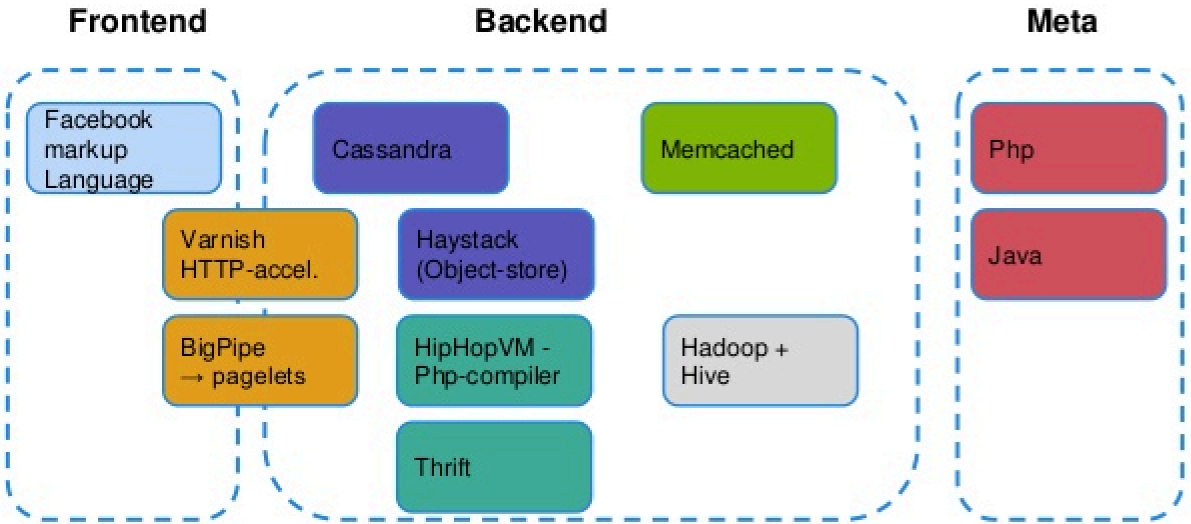
Multimedia

More...

There are too many other applications to enumerate

- Social Media (facebook, X/twitter, instagram)
- Chat & Teams (discord, slack, ...)
- News (reddit, feedly)
- Document and Spreadsheet editing (Google, Microsoft, ...)
- Slide presentations (Google, Microsoft, ...)

Facebook



Source: Gerald Maduabuchi(Quora), Phillip Webber

Tech Stacks

In groups of 2-3, Google the "tech stack" for a website or web based application. In your investigation, answer these two questions:

> **Question**
>
> **What technologies do they use?**

> **Question**
>
> **Which are *network* technologies?**

05:00

▶ Start

Tech Stacks

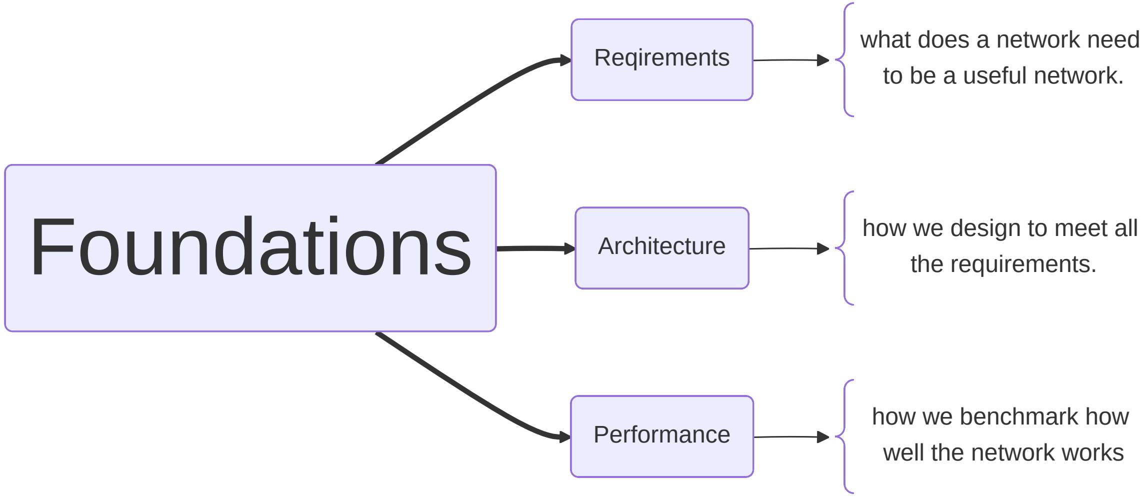
- What technologies did you find?
- Which ones "cross the network"?
- Were there any you found as *standards* (e.g. `http`)
- Were there any that were *unique* to that website or application?
- What else did you find?

Objectives

1. Reflect on and collect **requirements** that are the basis of our definition of a *network*.

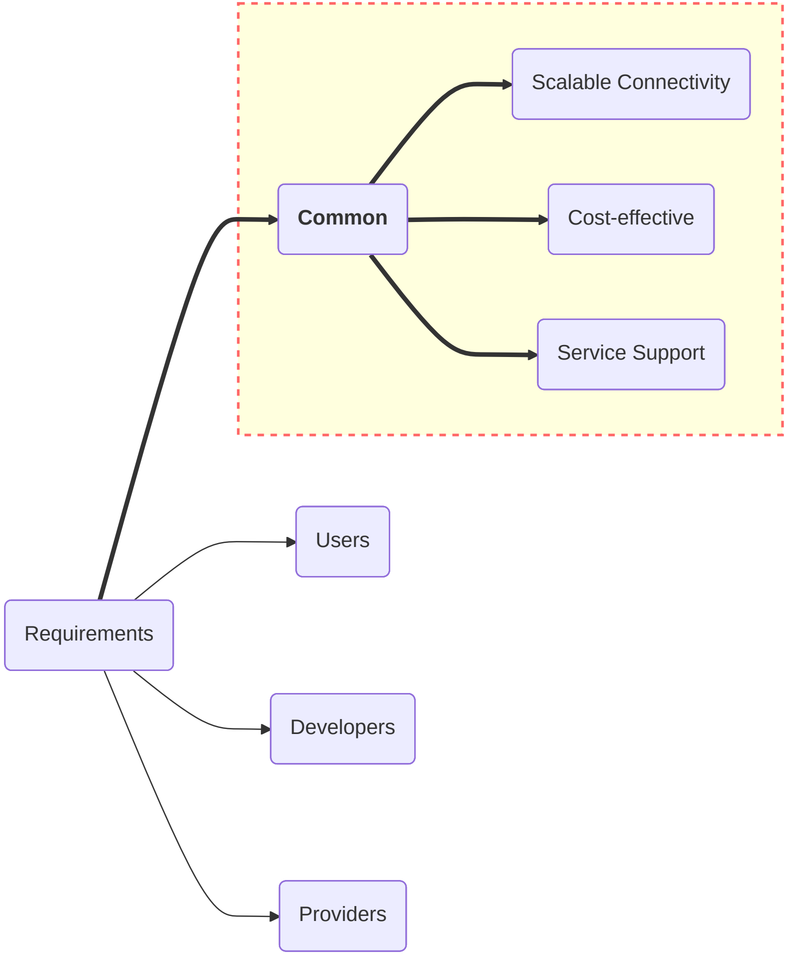
2. Introduce the ideas of **architecture** (layers and protocols) and apply structured problem solving.

3. Define **performance** metrics for how we measure success.



Let's start looking at *requirements* first...

Requirements



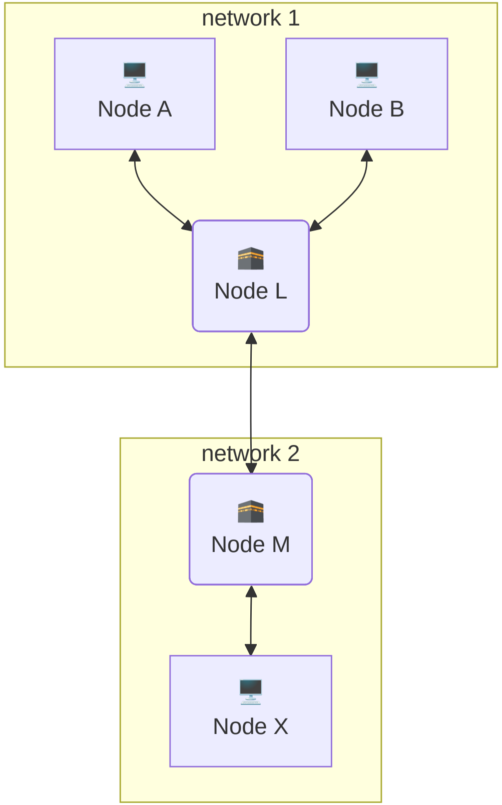
Common Requirements

These are basic requirements across all perspectives.

- The network needs to be **connected** and **scaleable**.
- The network needs to be **cost-effective**
- The network needs to **support the services** we want to use.

Common Requirements

Scalable Connectivity



> ⓘ **Definition**
>
> ***Network***: Two or more **nodes** connected by a physical **link**, or two or more **networks** connected by a node.

- **node** is a device connected to the network.
- **link** is what connects nodes together, the physical connection.

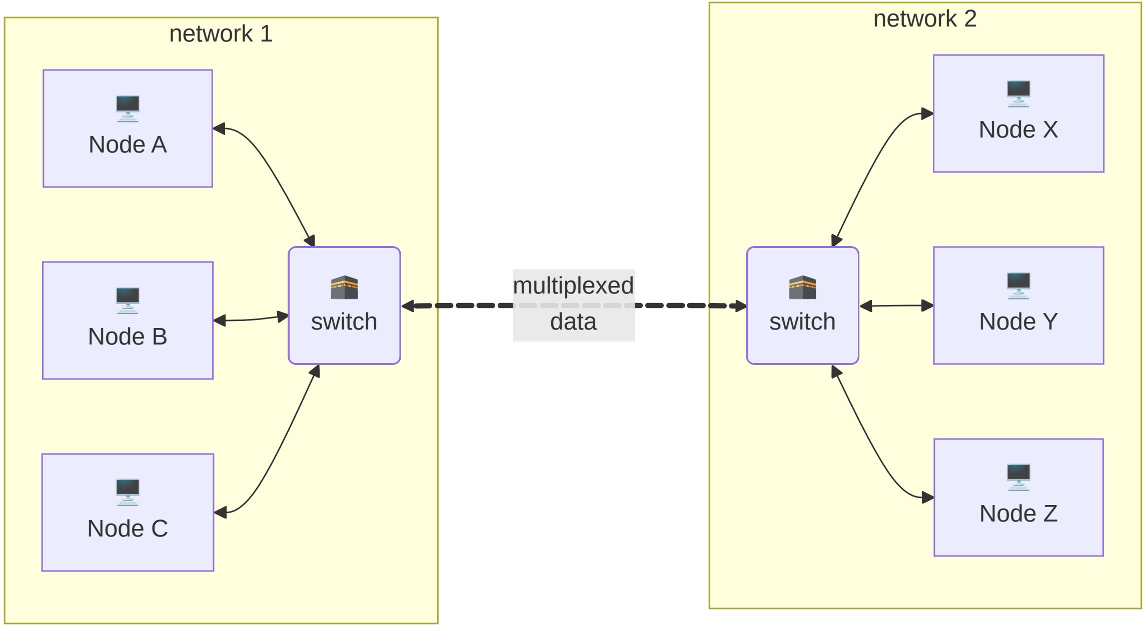
Some other relevant terms; a **host** is a type of node, a **switch** or **router** are internally a node and a link.

Common Requirements

Scalable Connectivity

Cost-effective

To be **cost-effective** data needs to be **multiplexed** over the links among nodes and networks.



There are a few common ways we can accomplish this.

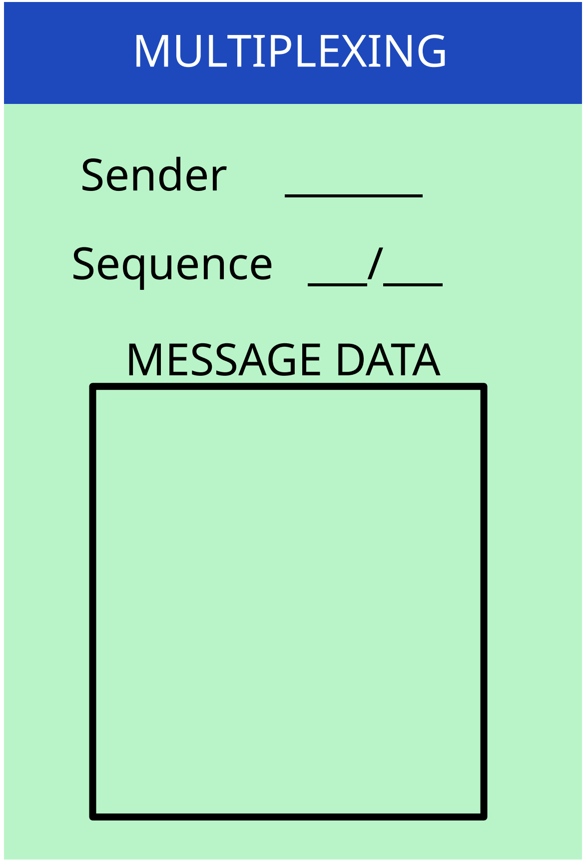
Sharing the Wire
A Mutiplexing Simulation

Let’s play a card game to explore different ways to **multiplex** data over a single link.

Work in groups of 4. Each group gets an **Instruction Sheet** and Game Area and **15 message cards** (3 red, 3 green, 3 orange)

Setup

- 3 **Senders** take colored cards and write a (less than 5 letter) word, one letter per card in the message data area. Include the position in the word for the letter as the sequence number.
- 1 **Receiver** setup the game board next to them where messages will be delivered.



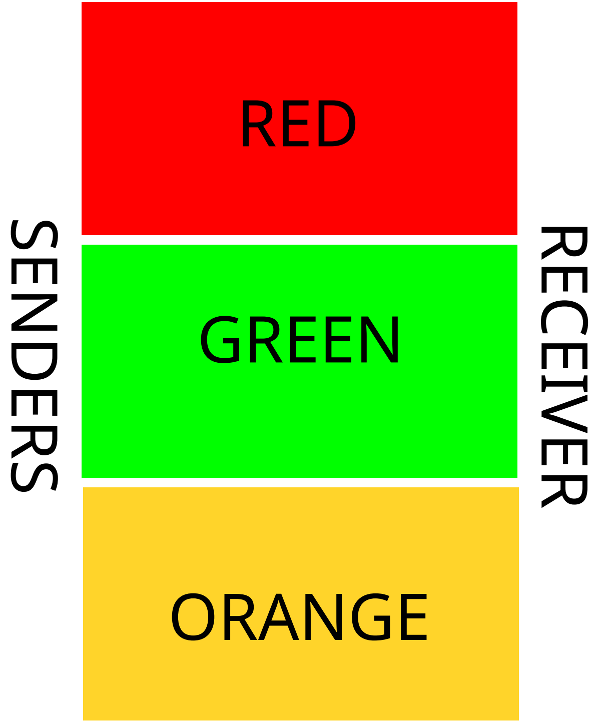
e.x. Sequence 1="D", 2="o", 3="g"

Sharing the Wire
Round 1

The **Time Keeper** (Instructor) will be the system clock. Players can only act when the time keeper says to act and they can only take one action!

- Each sender has their own lane (colored area).
- All senders place one packet in their lane at the same time.
- Receiver collects one packet from each lane, adds the new packets to pile of packets for each sender.

After time ends, the receiver decodes the messages sent by the senders.

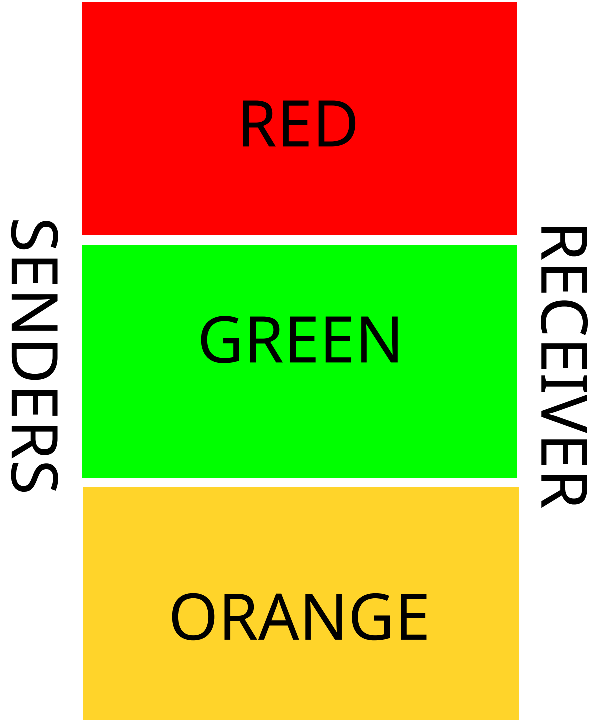


Sharing the Wire
Round 2

The **Time Keeper** (Instructor) will be the system clock. Senders can only act when the time keeper says to act and they can only take one action!

- All senders share **one lane** (center lane).
- Senders take turns in fixed order placing their next packet.
- Receiver picks up packets one at a time and puts it into their correct received pile.
- If it is a senders turn and they have no more data, the turn is not used, the sender and reciever do nothing.

After time ends, the receiver decodes the messages sent by the senders.

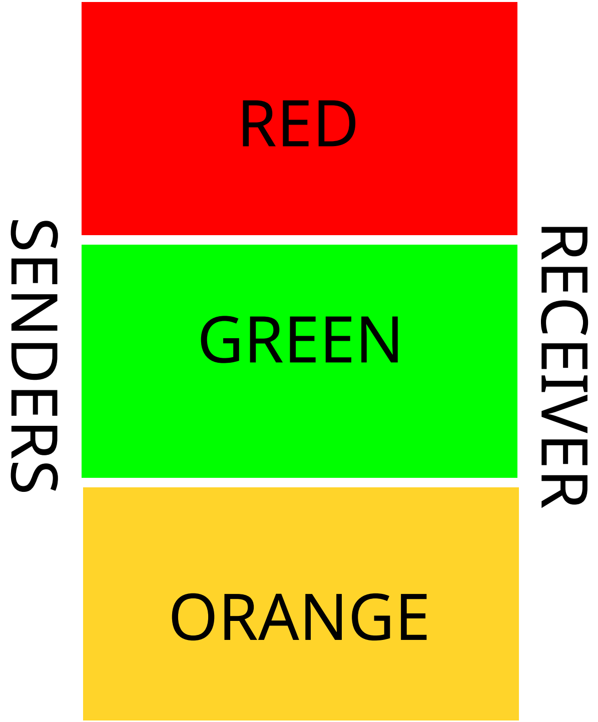


Sharing the Wire
Round 3

The **Time Keeper** (Instructor) will be the system clock. Senders can only act when the time keeper says to act and they can only take one action!

- All senders share **one lane** (center lane).
- If there are no packets in the lane, senders place a packet in any order, whoever gets their packet there first wins.
- Receiver picks up packet and puts into their correct received pile.
- If a sender has no more packets to send, they do nothing.

After time ends, the receiver decodes the messages sent by the senders.



Sharing the Wire – Discussion

- **Round 1** - Frequency Division
  - Each sender has $1/3$ the total **bandwidth** and sends $1/3$ of a packet at a time.
  - Wasted lane when sender has no more data to send. **not efficient**
- **Round 2** - Time Division
  - Each sender has full bandwidth for $1/3$ of the time.
  - Wasted time slots when sender has no more data to send. **not efficient**
- **Round 3** - Packet Switching
  - No wasted lane or time slots
  - Senders collide when sending.
  - Receiver may need to put packets back in order.

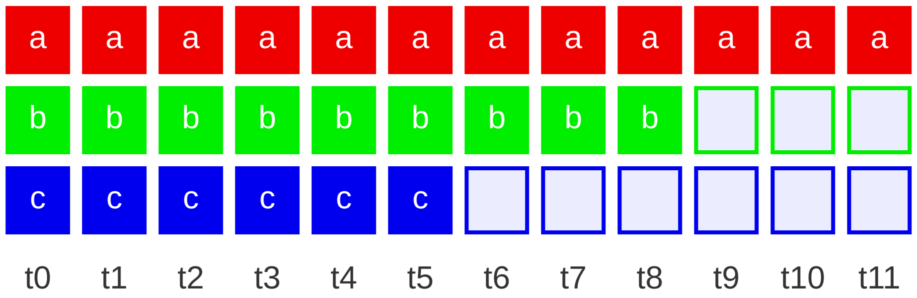
Common Requirements

Scalable Connectivity

Cost-effective

Frequency Division Multiplexing

Each sender gets a unique channel that is $1/n$ the entire band.



The empty boxes above represent when no sender has any more data to send, their frequency band (or lane) goes unused.

This is how broadcast radio and television work. Each station gets a sub-band of the entire band. They broadcast continuiously and thus always have "data" to send.

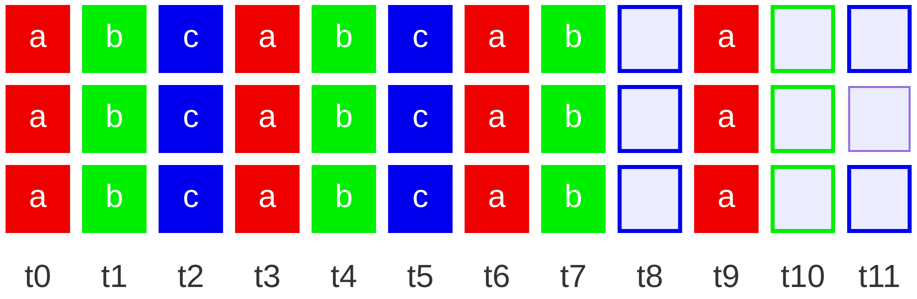
Common Requirements

Scalable Connectivity

Cost-effective

Time Division Multiplexing

Each sender gets a unique time slot, it goes unused if they have no data to send.



The empty boxes above represent when a sender has no more data to send, their time slot goes unused.

During their time slot they use the *entire band* so can send $3\times$ the amount of data during the time slot as with frequency division.

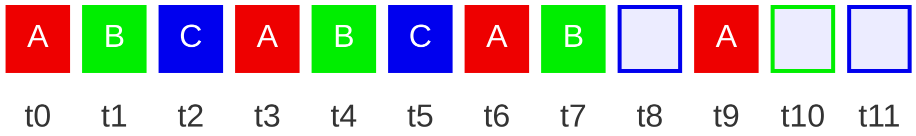
Common Requirements

Scalable Connectivity

Cost-effective

Time Division Multiplexing

Each sender gets a unique time slot, it goes unused if they have no data to send.



The empty boxes above represent when a sender has no more data to send, their time slot goes unused.

During their time slot they use the *entire band* so can send $3\times$ the amount of data during the time slot as with frequency division.

This collapsed version is a bit more compact and easy to look at. Note the "A" instead of "a" to represent $3\times$ the data in the packet.

Common Requirements

Scalable Connectivity

Cost-effective

Packet Switching

Like **Time Division** senders use the entire band to send data during fixed size time slots.

A	B	C	A	B	C	A	B	A			
t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11

With **Packet Switching** senders can send **whenever they have data and the link is idle**.

The empty boxes above represent when **no sender** has data to send. The time slots do not go unused. If there were **any data** from **any sender** they could use those slots to send their data.

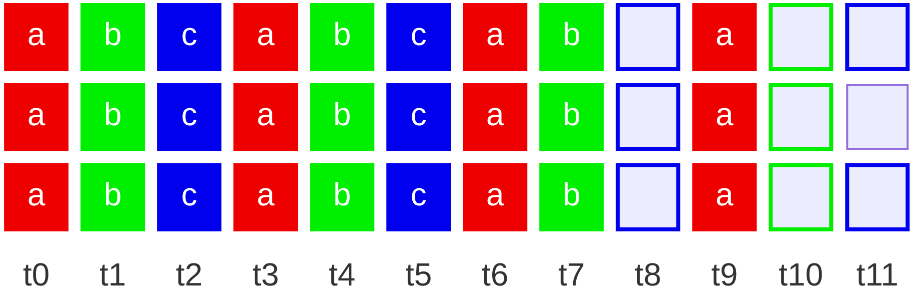
As you likely discovered, there can be **collisions** when multiple senders try to send. We will deal with those later.

Common Requirements

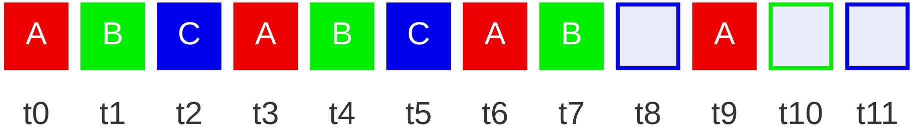
Scalable Connectivity

Cost-effective

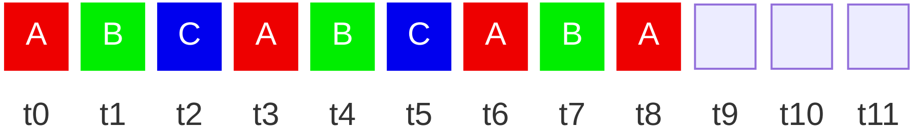
Frequency Division Multiplexing



Time Division Multiplexing



Packet Switching (most effective)



Common Requirements

Scalable Connectivity

Cost-effective

Support Common Services

The network should support common services.

It needs to define **useful channels** that understand the **application needs** and the **network’s ability**.

- File Transfer (upload/download)
- Multimedia Streaming (watch a movie)
- Web Browsing
- Interactive Videoconference
- Data Reporting (e.g. weather conditions)
- Command and Control (e.g. "turn on light")

Common Requirements

Scalable Connectivity

Cost-effective

Support Common Services

Reliable and Manageable

Reliable

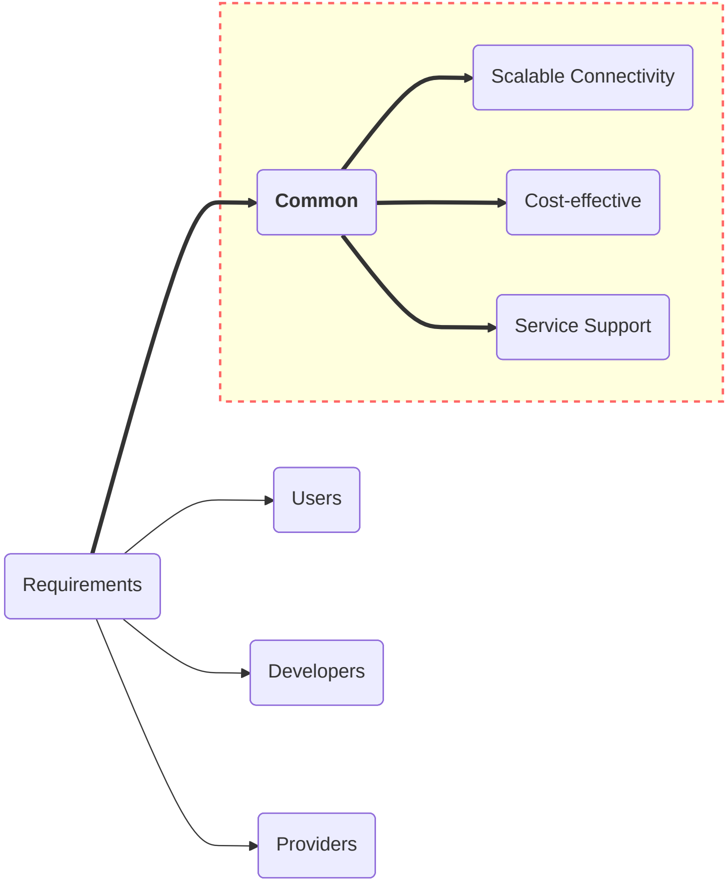
Fill in the gaps between what an application expects and what the underlying technology can provide.

- Handles interruptions in service and equipment malfunctions

Manageable

The network can continue to work with new services and parameters.

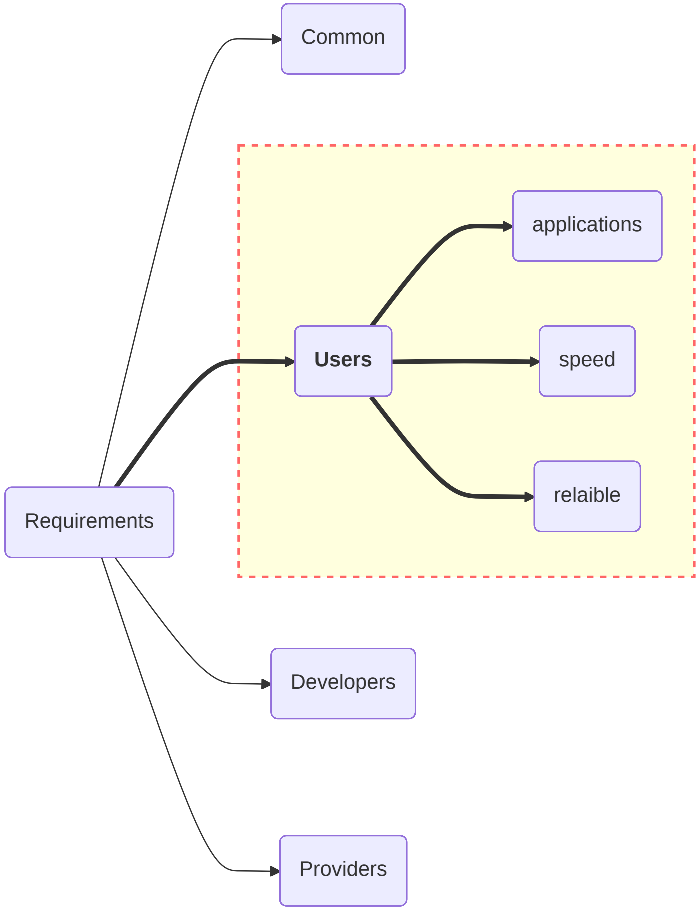
- Upgrades are easy, Billing is accurate
- Supports new applications



Common Requirements

These are basic requirements across all perspectives.

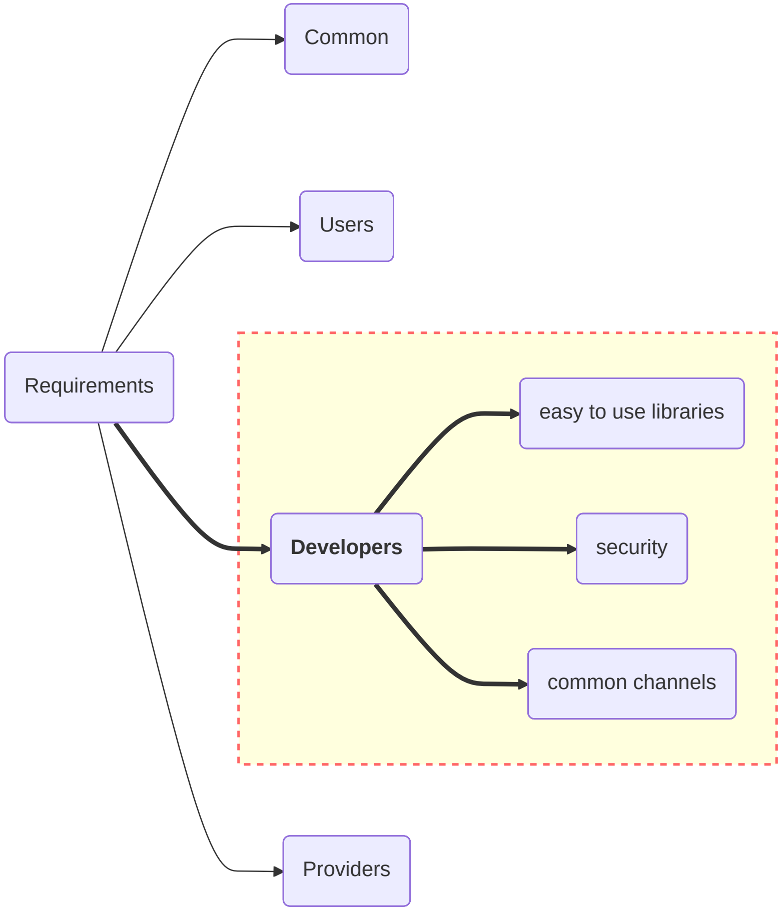
- The network needs to be **connected** and **scaleable.**
- The network needs to be **cost-effective**
- The network needs to **support the services** we want to use.



User Requirements

Users want an inexpensive and fast network that supports the applications they want to use.

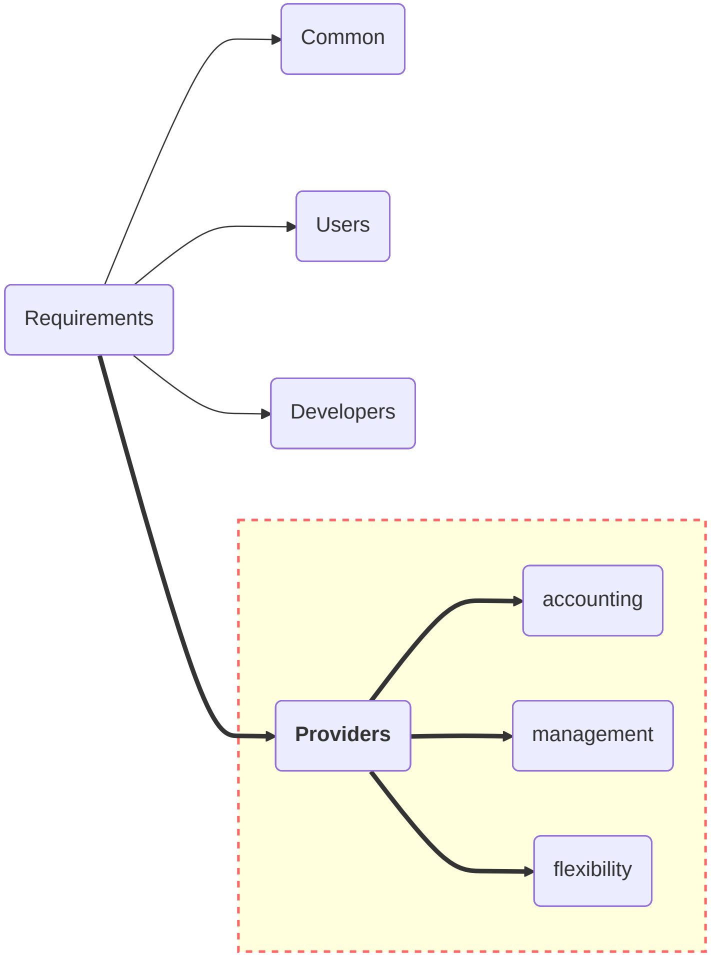
- Support common applications; web, video, audio, chat
- Fast for downloads and streaming.
- Inexpensive (or at least cost-efficient)
- Reliable and recovers quickly when failures occur



Developer Requirements

Developers want a network that provides well thought out libraries to develop against that provide the communication channels they need for their applications.

- Easy to use libraries or language constructs
- Secure and security built-in
- Supports common service paradigms; client-server, peer-to-peer
- Has good error handling and retry mechanisims



Service Provider Requirements

Service providers (ISPs) want a future-proof network that has good tools for mangement, accounting, and diagnostics.

- Provides verbose accounting tools for billing and cost recovery.
- Comprehensive management tools to pinpoint problems and recover quickly
- Flexible to make best use of costly physical infrastructure
- Reliable to avoid downtime

Requirements

Take a couple of minutes to think about and write down

Question

What requirements are we missing?

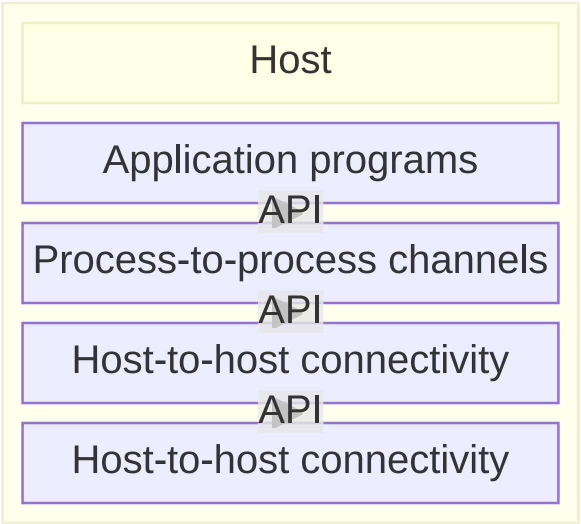
02:30

▶ Start

Architecture

Layers

We view the network as a series of **layers** in a **stack**. Each layer can call functions of the layer below it in the stack and can be called by layers above it in the stack using a well defined **Application Programming Interface (API)**.



This is a simplified view of the layers in a network stack.

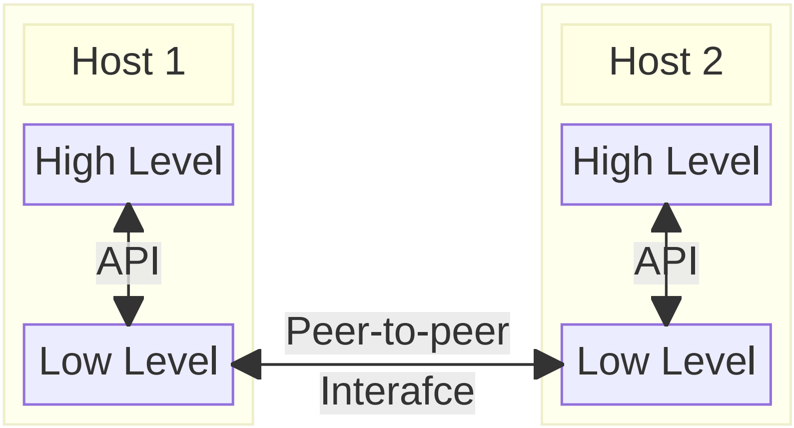
Architecture

Layers

Protocols

Layers have a **peer-to-peer** interface with the same layer on another host in the network.

Layers at the same level communicate with each other via data passed between them.



This peer-to-peer interface is called a **Protocol**.

Architecture

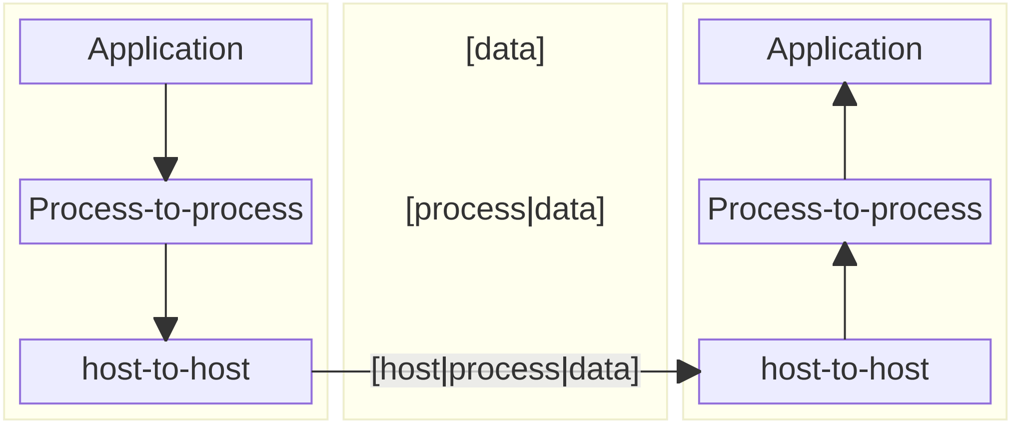
Layers

Protocols

Encapsulation

For the layers to pass control information between themselves, they **encapsulate** data from higher layers into a sub-field of the data they pass with their peer.

Layers **wrap** data from higher layers with control information and pass that the next lower layer. The data is **unwrapped** by their peer.



This peer-to-peer interface is called a **Protocol**.

Architecture

Layers

Protocols

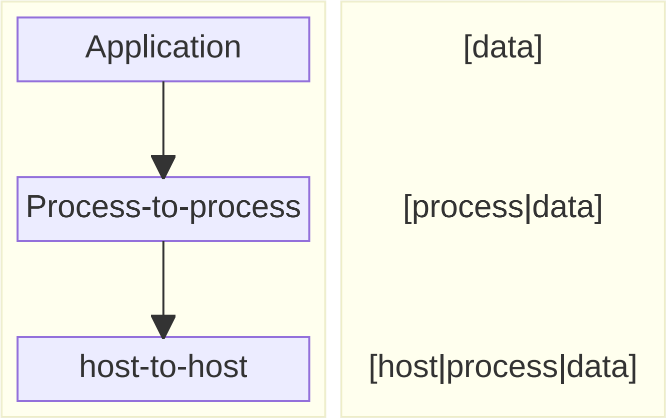
Encapsulation

- Multiplexing

When sending data, start a the top of the stack and work downwards.

The application data is passed to the *process-to-process* layer which wraps the data with the destination **process information** as determined by the protocol.

Similarly the **host-to-host** layer will add destination **host** information. Then the data will be sent over the network (by a lower layer).



Architecture

Layers

Protocols

Encapsulation

- Multiplexing

This **encapsulation** is what enables **multiplexed data**.

At the lowest layer, the **hardware** layer, the data will be sent on the network medium. It is **multiplexed** there using one of the methods we looked at earlier.

Architecture

Layers

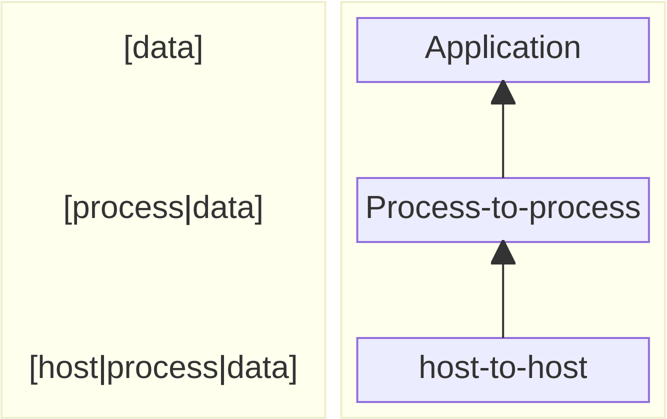
Protocols

Encapsulation

- Multiplexing
- Demultiplexing

When receiving data, start a the bottom and work upwards.

The **host** field of the *host-to-host* layer indicates which *host* the data is for. If we are the **destination host** then the unwapped data can be passed to the next layer up.



The **process** field similarly tells *which process* on the host to deliver the data to.

Architecture

Layers

Protocols

Encapsulation

Models

There are two common *layered architectures* in use today.

- *International Standards Organization's Open System Interconnect* or **OSI/ISO** model.
- *Internet* or **TCP/IP** model.

The two are interoperable but their layers do not have a direct correspondence to each other.

> **Exam Tip**
>
> These two models will be our focus throughout the course. **You should memorize them**

Architecture

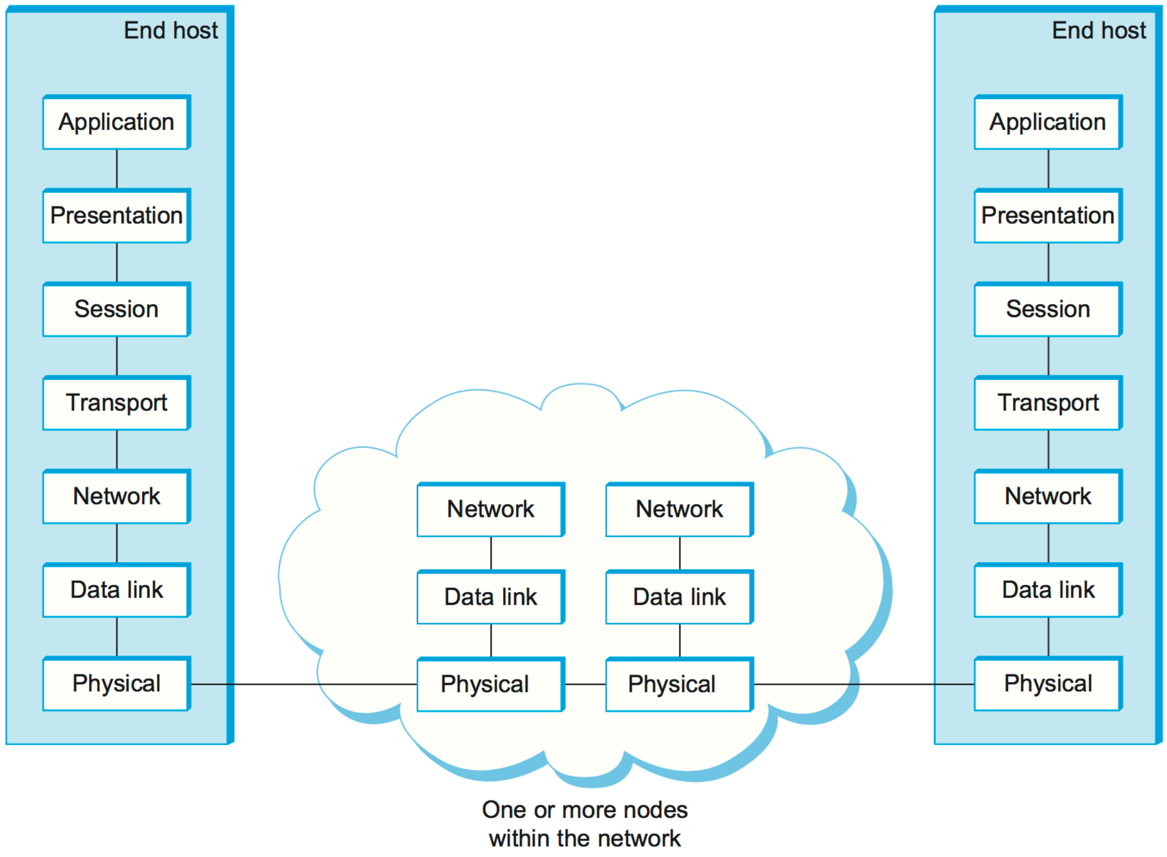
Layers

Protocols

Encapsulation

Models

- **ISO/OSI Model**



Architecture

Layers

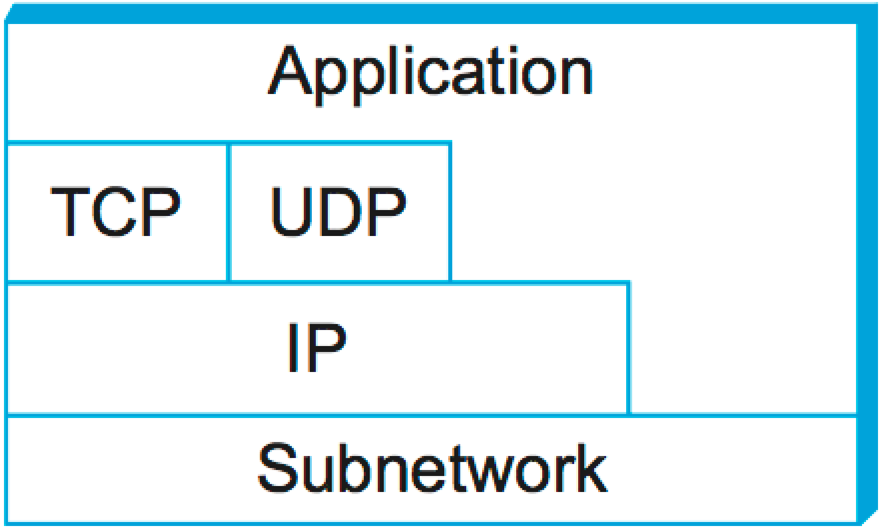
Protocols

Encapsulation

Models

- ISO/OSI Model
- **Internet Model**

The Internet model has fewer layers and allows calling any of them from any other layer.



It tends to be a more popular architecture for developers dur to its relaxed calling nature.

Architecture

Layers

Protocols

Encapsulation

Models

- ISO/OSI Model
- Internet Model
- **Comparison**

OSI Model	Internet Model
Application (7)	Application
Presentation (6)	
Session (5)	Transport TCP
Transport (4)	Transport UDP & IP
Network (3)	Internet IP & Network
Data Link (2)	Network Access
Physical (1)	

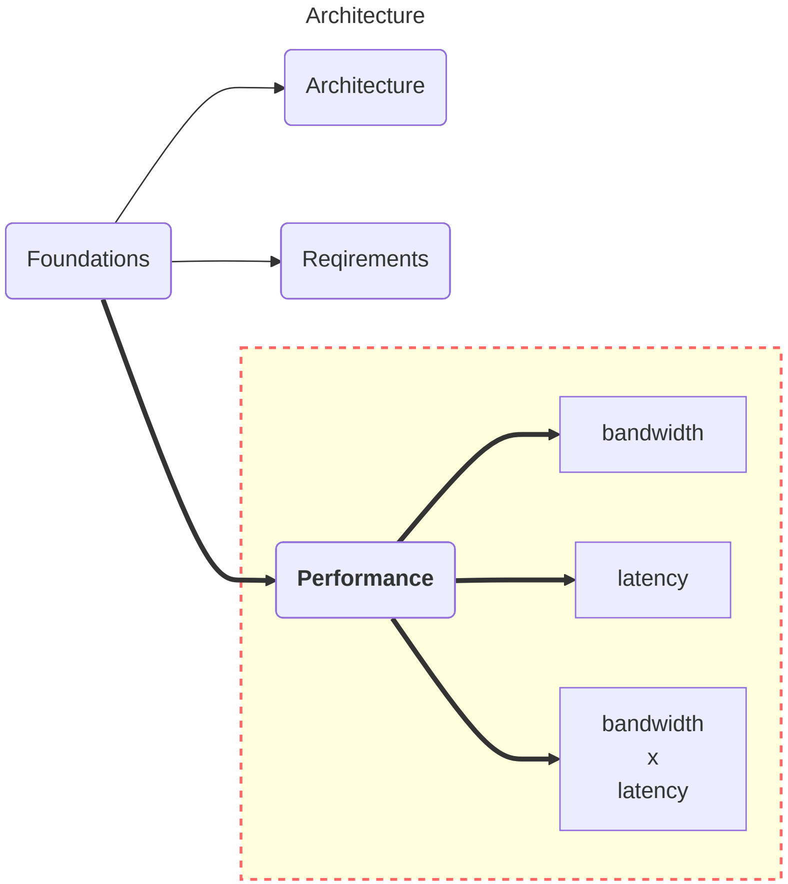
The layers in the two architectures don’t directly correspond, but we can compare them.

The Internet model does not have a *presentation* layer. It is left to the application to handle this function.

OSI *session* layer semantics are handled mostly by TCP in the Internet model.

The Internet model’s *IP* layer handles functions from the OSI’s transport and network layers.

The Internet model *network* layer performs the functions on the OSI network, data-link, and physical layers.



Performance

You may have heard the terms **bandwidth** and **latency** to determine network perofmance.

In this section we will formally define them and see how to measure the performance and the capacity of a network.

This will help us understand not only how to benchmark network performance, but also the factors that can be tuned to affect that performance.

Performance

Performance

Bandwidth

Consider these example **bandwidth** calculations:

- On a **10 Mbps** (mega bits per second) link
  - data is transmitted at **10 million bits per second**
  - each bit takes **0.1 µS** to transmit
- On a **20 Mpbs** link, each bit takes **0.05 µs**.
- On a **100 Mpbs** link, each bit takes **0.01 µs**.

> **Question**
>
> **How long does it take to transmit a single bit on a on a 1 Gbps link?**

- On a **100 Mpbs** link, each bit takes **0.01 µs**.

Performance

Bandwidth

Latency

Latency is a meausre of the **time** that it takes a message to travel from one node to another. It's also called **delay**. We consider *latency* with the following equations:

$$\begin{aligned} \textbf{latency} &= \text{propogation} + \text{transmission} + \text{queueing delay} \\ \textbf{popogation} &= \text{distance}/\text{speed of light} \\ \textbf{transmission} &= \text{size}/\text{bandwidth} \end{aligned}$$

- **distance** is the length of the medium.
- **speed of light** is the effective speed of light.
- **size** is the size of the data.
- **bandwidth** is the data rate of the transmitted data.
- **queuing delay** is the delay introduced by system components.

Performance

Bandwidth

Latency

- Speed of Light

Light travels across different mediums at different speeds!

Medium	**Speed of Light**
Vacuum	$3.0 \times 10^8 m/s$
Copper	$2.3 \times 10^8 m/s$
Fiber	$2.0 \times 10^8 m/s$

These are approximate speeds to give an idea of the differences.

> **Note**
>
> Don’t worry, the speed of light is still constant, just constant on different mediums.

Performance

Bandwidth

Latency

- Speed of Light
- Typical Latency

Latency is usually measured in the **Round Trip Time (RTT)** for data to traverse the network. This is the time to go to a remote node and return to the sender. Hence, *round trip*.

Link Type	Bandwidth	One-way Distance	RTT
Ethernet LAN	1 Gbps	50 m	0.25 µs
Wireless LAN	54 Mbps	50 m	0.33 µs
Satellite	1 Gbps	35,000 km	230 ms
Cross-country fiber	10 Gbps	4,000 km	40 ms

These are approximate times to give an idea of the differences, your measurements may vary.

Performance

Bandwidth

Latency

Bandwidth & Latency

Consider a **1 byte message** and a **1 byte response** over a **10 Mbps** link with a **10 ms RTT**.

> Question
>
> **Does bandwitdh or latency dominate the total transmission time?**

Latency will dominate the transmission time. Each single-byte transmission will take 10 ms.

> Question
>
> **When will bandwidth dominate?**

When the message size becomes large and takes more than 10 ms to transmit.

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Consider a **10 Mbps** link with a **100 ms RTT**.

A **10 KB message** ($10KB = 10,000 \times 8 = 80,000$ bits).
Will take **8 ms** to transmit.

Because the one-way latency is 50 ms, the data will finish sending before the first bit is received on the other end.

The data is completely **on the network** or **in-flight** during that time!

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Consider a **10 Mbps** link with a **100 ms RTT**.

The data is completely **on the network** or **in-flight** during that time!

Question

How much data can be in-flight at a time on the network?

$$\textbf{bandwidth} \times \textbf{latency} = 10 \text{ Mbps} \times 50 \text{ ms}$$
$$(10x10^6) \text{ bits/second} \times (50x10^{-3}) \text{ seconds}$$
$$500 \times 10^3 = 500Kb = \underline{\textbf{62.5 KB}}$$

62.5 KB of data can be **on the network**

The project pingfs actually makes use of this to store files "on the network" without using any local storage.

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Exercise

Lets look at some examples in a few different scenarios.

Calculate the total time required to transfer a **1000 KB file** in the following cases, assuming an **RTT of 50 ms**, a packet size of **1 KB** data, and an **initial 2 $\times$ RTT of "handshaking"** before data is sent.

1. Data can be sent continuiously.
2. The sender needs to wait 1 RTT between packets.
3. The sender can send 20 packets per RTT on an infinite-bandwidth link.
4. The sender can send 1, 2, 4, ... packets on the same infinite-bandwidth link.

> **Note**
>
> 1KB is $2^{10} = 1,024$ not $10^3 = 1,000$

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Exercise

1. The bandwidth is **10 Mbps**, and data packets can be **sent continuosly**.

$$
\begin{aligned}
\text{Handshake} &= 2 \times RTT && = 0.1 \text{ s} \\
\text{Propagation delay} &= \text{RTT}/2 && = 0.025 \text{ s} \\
\text{Packet size} &= 2^{10} \times 8 && = 8,192 \text{ bits} \\
\text{Data size} &= (1,000 \times 2^{10}) \times 8 && = 8,192,000 \text{ bits} \\
\text{Bandwidth} &= 1.5 \times 10^6 && = 10 \text{ Mpbs} \\
\text{Transmit time} &= \text{size}/\text{bandwidth} && \\
&= 8,192,000/(1.5 \times 10^6) && = 5.4613 \text{ s}
\end{aligned}
$$

$$
\begin{aligned}
&\text{Handshake} + \text{Propogation delay} + \text{Transmit} = \\
&\qquad 0.1 \text{ s} + 0.025 \text{ s} + 5.4613 \text{ s} = \underline{\mathbf{5.5863\ s}}
\end{aligned}
$$

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Exercise

2. The bandwidth is **10 Mbps**, but after we finish sending each data packet **we must wait one RTT** $(0.05\ \text{s})$ before sending the next.

$$
\begin{aligned}
\text{Transmit time} &= 8,192,000/(1.5 \times 10^6) && = 5.4613\ \text{s} \\
\text{Packet size} &= 2^{10} \times 8 && = 8,192\ \text{bits} \\
\text{Data size} &= (1,000 \times 2^{10}) \times 8 && = 8,192,000\ \text{bits} \\
\text{Packets} &= \text{Data size}/\text{Packet size} && = 1,000\ \text{packets}
\end{aligned}
$$

$$
\begin{aligned}
\text{Wait time} &= (\text{Packets} - 1) \times \text{RTT} && = 49.95\ \text{s} \\
\text{Transmit} &= \text{Transmit time} + \text{Wait time} && = 55.4113\ \text{s} \\
&= 5.4613\ \text{s} + 49.95\ \text{s} && = 55.4113\ \text{s}
\end{aligned}
$$

$$
\begin{aligned}
&\text{Handshake} + \text{Propogation delay} + \text{Transmit} = \\
&\qquad 0.1\ \text{s} + 0.025\ \text{s} + 5.4613\ \text{s} = \underline{\mathbf{55.5363\ s}}
\end{aligned}
$$

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Exercise

3. The link allows infinitely fast transmit, but limits bandwidth such that only **20 packets can be setnt per RTT**.

$$
\begin{aligned}
\text{Packets} &= \text{Data size}/\text{Packet size} && = 1,000 \text{ packets} \\
\text{Chunks} &= ceil(\text{Packets}/20) && = 50 \\
\text{RTT} &= && = 0.05 \text{ s} \\
\text{Transmit} &= \text{Chunks} \times \text{RTT} && = 2.6 \text{ s} \\
\text{One-way Propogation} &= \text{RTT}/4 && = 0.025 \text{ s}
\end{aligned}
$$

$$
\begin{aligned}
&\text{Handshake} + \text{Transmit} - \text{One-way Propogation} = \\
&\qquad 0.1 \text{ s} + 2.6 \text{ s} - 0.025 \text{ s} = \underline{\mathbf{2.575\ s}}
\end{aligned}
$$

We subtract One-way Propogation as we don't need to wait after the last chunk is sent.

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

Exercise

4. **Zero transmit time**, as in (3) but during the first RTT we can send one packet, during the second RTT we can sent two packets, during the third we can send four (2^{3-1}), etc. (A justification for such an exponential increase will be given later.)

$$\begin{aligned}
\text{Packets} &= \text{Data size}/\text{Packet size} &&= 1,000 \text{ packets} \\
\text{Chunks} &= ceil(log_2(\text{Packets})) &&= 10 \\
\text{RTT} &= &&= 0.05 \text{ s} \\
\text{Transmit} &= \text{Chunks} \times \text{RTT} &&= 0.5 \text{ s} \\
\text{One-way Propogation} &= \text{RTT}/4 &&= 0.025 \text{ s}
\end{aligned}$$

$$\text{Handshake} + \text{Transmit} - \text{One-way Propogation} =$$
$$0.1 \text{ s} + 0.5 \text{ s} - 0.025 \text{ s} = \underline{\mathbf{0.575\ s}}$$

Performance

Bandwidth

Latency

Bandwidth & Latency

Bandwidth $\times$ Latency

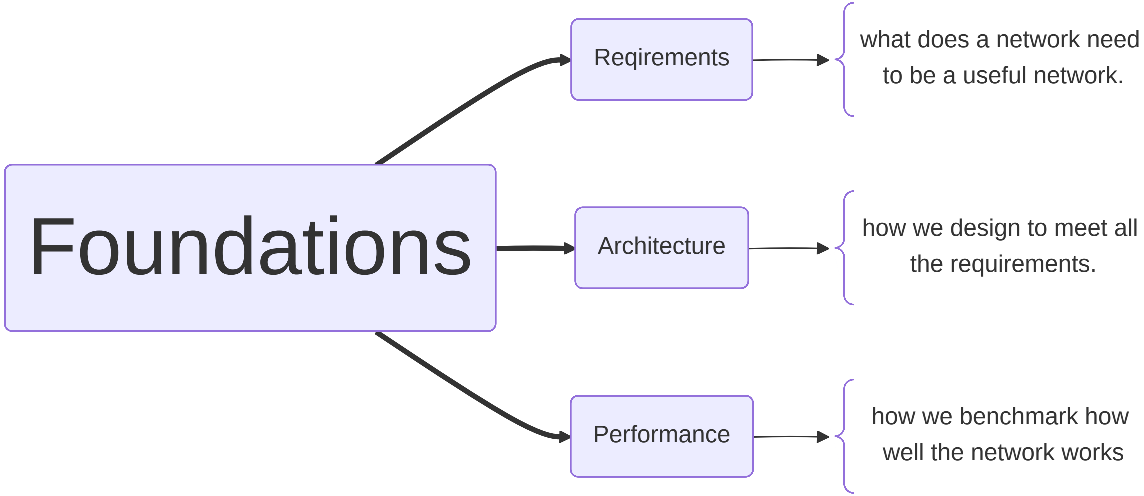
Exercise

Exercise
On your own

This exercise is very similar to the previous one, but uses a different **RTT**. Work on this one on your own for practice.

Calculate the total time required to transfer a **1.5 MB file** in the following cases, assuming an **RTT of 80 ms**, a packet size of **1 KB** data, and an **initial 2 $\times$ RTT of "handshaking"** before data is sent.

1. The bandwidth is **10 Mbps**, and data packets can be **sent continuosly**. (1.458 s)
2. The bandwidth is **10 Mbps**, but after we finish sending each data packet **we must wait one RTT** before sending the next. (124.258 s)
3. The link allows infinitely fast transmit, but limits bandwidth such that only **20 packets can be sent per RTT**. (6.32 s)
4. **Zero transmit time**, but during the first RTT we can send one packet, during the second RTT we can sent two packets, during the third we can send four (2^{3-1}), etc. (1 s)


Foundations
Reqirements
what does a network need
to be a useful network.
Architecture
how we design to meet all
the requirements.
Performance
how we benchmark how
well the network works