**About this Path**

Get a quick introduction to this path.

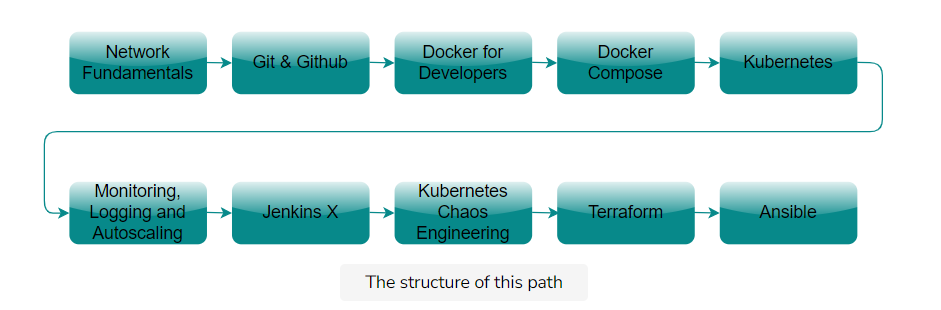
**We'll cover the following**

* [What will we cover?](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#What-will-we-cover?)
  + [Network Fundamentals](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Network-Fundamentals)
  + [Git and Github](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Git-and-Github)
  + [Docker for Developers](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Docker-for-Developers)
  + [Docker Compose for Developers](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Docker-Compose-for-Developers)
  + [A Practical Guide to Kubernetes](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#A-Practical-Guide-to-Kubernetes)
  + [Kubernetes Monitoring, Logging and Autoscaling](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Kubernetes-Monitoring,-Logging-and-Autoscaling)
  + [Jenkins X with Kubernetes](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Jenkins-X-with-Kubernetes)
  + [Kubernetes Chaos Engineering](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Kubernetes-Chaos-Engineering)
  + [Terraform: From Beginner to Master with Examples in AWS](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Terraform:-From-Beginner-to-Master-with-Examples-in-AWS)
  + [Ansible](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Ansible)

Welcome to the **DevOps for Developers** path. This path will help you’ll have cutting-edge skills and hands-on experience to excel in any DevOps role. If you want to gain in-demand DevOps skills such as at-scale application deployment, live updates, and containerization, you are at the right place.

## What will we cover?[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#What-will-we-cover?)

The path is divided into ten modules:



### Network Fundamentals[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Network-Fundamentals)

This module teaches us network fundamentals essential for any DevOps specialist.

### Git and Github**[#](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN" \l "Git-and-Github)**

This module provides hands-on experience working with Git and its advanced concepts.

### Docker for Developers[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Docker-for-Developers)

This module helps us learn everything about Docker containers to easily create, deploy, and run our applications with them.

### Docker Compose for Developers[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Docker-Compose-for-Developers)

This module explains advanced docker tools to help us simplify our workflows with Docker Compose and scale our clusters with Swarm.

### A Practical Guide to Kubernetes[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#A-Practical-Guide-to-Kubernetes)

This module explains the fundamentals of Kubernetes and the main components of a cluster. We’ll also learn how to secure our deployments and manage resources, which are crucial DevOps skills.

### Kubernetes Monitoring, Logging and Autoscaling[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Kubernetes-Monitoring,-Logging-and-Autoscaling)

This module helps us expand our Kubernetes knowledge with techniques to make our clusters dynamic and resilient.

### Jenkins X with Kubernetes[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Jenkins-X-with-Kubernetes)

This module will help us using Jenkins X to automate our pipeline and achieve CI/CD.

### Kubernetes Chaos Engineering[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Kubernetes-Chaos-Engineering)

This module helps us to level up our testing with chaos engineering. We will learn how pushing our programs to failure can help us plan the improvements.

### Terraform: From Beginner to Master with Examples in AWS[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Terraform:-From-Beginner-to-Master-with-Examples-in-AWS)

This module explains how to build large and scalable projects with Terraform.

### Ansible[**#**](https://www.educative.io/module/lesson/network-fundamentals/N7jVZ8l1qYN#Ansible)

This module teaches us the ins and outs of Ansible with the goal of managing and automating our infrastructure and code deployment.

# About this Module

This lesson discusses the intended audience and the necessary prerequisites of the module.

**We'll cover the following**

* [Who is this module for?](https://www.educative.io/module/lesson/network-fundamentals/N82AEJVj8G2#Who-is-this-module-for?)
* [Module Structure](https://www.educative.io/module/lesson/network-fundamentals/N82AEJVj8G2#Module-Structure)

## Who is this module for?[**#**](https://www.educative.io/module/lesson/network-fundamentals/N82AEJVj8G2#Who-is-this-module-for?)

This module is for developers who are looking to enhance their skills and knowledge of networks and are looking to get started with their journey to become DevOps developers.

## Module Structure[**#**](https://www.educative.io/module/lesson/network-fundamentals/N82AEJVj8G2#Module-Structure)

This module contains six chapters, namely:

1. **Internet and the OSI Model:**

This chapter introduces us to the internet and its layered architecture, as well as the OSI, and TCP/IP models.

1. **The Application Layer:**

This chapter goes through the application layer of the OSI model. It also teaches us about network application architectures, HTTP, cookies, and DNS.

1. **The Transport Layer:**

This chapter introduces us to the transport layer of the OSI model and how the protocols and methods are used by the layer. It also teaches us the concepts of the UDP and TCP protocols.

1. **An Introduction to Socket Programming with Python:**

This chapter teaches us the basics of socket programming, and how to set up sockets for both UDP and TCP.

1. **The Network Layer:**

This chapter introduces us to the network layer of the OSI model. It also teaches us the Internet Protocol (IP) and the differences between its versions.

1. **The Link Layer:**

This chapter introduced us to the data link layer and its principles. It also teaches us about ethernet.

# What is the Internet?

Let's start with the big question.

**We'll cover the following**

* [But First, What’s a Network?](https://www.educative.io/module/lesson/network-fundamentals/YVYK8KwGpEn#But-First,-What%E2%80%99s-a-Network?)
* [The Internet: A Network of Networks](https://www.educative.io/module/lesson/network-fundamentals/YVYK8KwGpEn#The-Internet:-A-Network-of-Networks)

The Internet permeates our very existence. Most of us cannot imagine life without it. We often depend on it for livelihoods, for routine commutes, and for entertainment. It has become almost like a utility. You’re accessing this course through the Internet.

But how does it actually work? What goes on behind the scenes? Well, you’ve come to the right place to learn that! The Internet is a global network of computer networks.

## But First, What’s a Network?

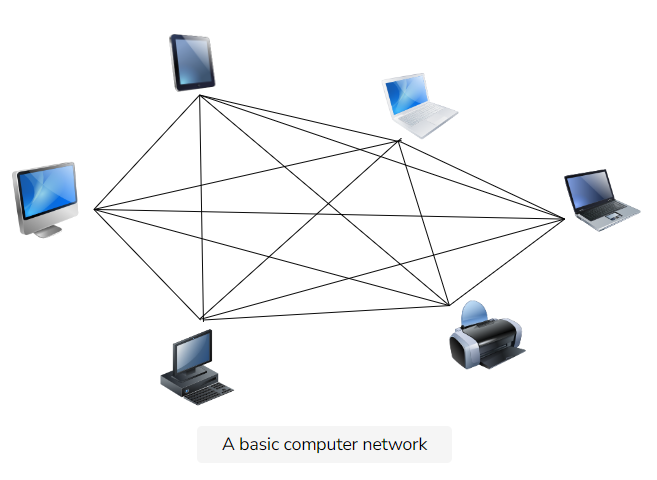
A **network is** officially defined as **a group or system of interconnected people or items**.

So, by this definition, train stations connected to each other with rail tracks make up a *railway* network. People who follow each other on Twitter make up an online *social* network.

Similarly, computers connected to each other with cable or wireless radio make up a *computer network*.

**Why Computer Networks?**

There are **two main purposes** of computer networks: **Communication** using computers and **sharing of resources**. An “internet” allows doing these two things across different computer networks.



A basic computer network

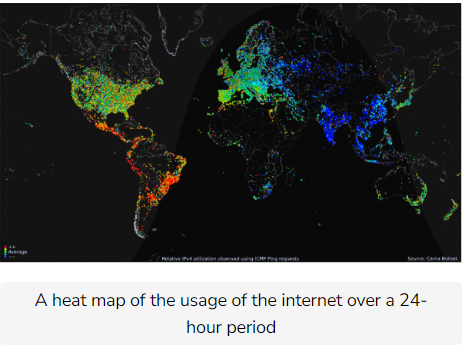
📝 **Note: internet vs. Internet** An internet with a lowercase “*i*” is *any* interconnection of computer networks. Whereas the global Internet is always spelled with a capital I.

**The Internet**

The Internet is essentially a network of computer networks.

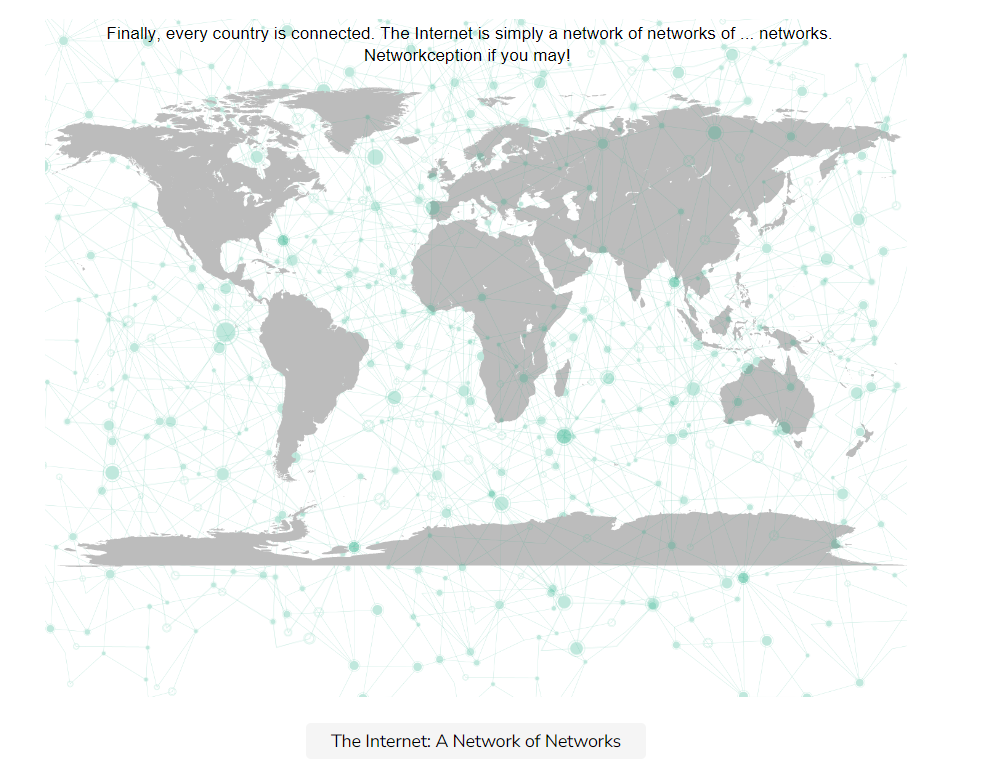
So your personal computer is connected to other computers at your house or workplace to create a small computer network, which is in turn connected to other computer networks. And so the global Internet encompasses a complex web of interconnected computer networks.

This concept can be better visualized in the illustration below:



A heat map of the usage of the internet over a 24-hour period

## The Internet: A Network of Networks



In the next lesson, we’ll take a look at a short history of the Internet and how it all began.

# Layered Architectures & Protocol Stacks

Layered architectures are a way to organize computer networks. Let's dive right in!

**We'll cover the following**

* [Introduction to Layered Architectures](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Introduction-to-Layered-Architectures)
  + [Why Layers?](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Why-Layers?)
* [An Analogy: Post](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#An-Analogy:-Post)
* [Layers As Services To Each Other: Layers Are Vertical](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-As-Services-To-Each-Other:-Layers-Are-Vertical)
  + [Vertical Layers in Post](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Vertical-Layers-in-Post)
  + [Vertical Layers in Networks](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Vertical-Layers-in-Networks)
* [Layers Communicate with Their Parallels: Layers Are Horizontal](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-Communicate-with-Their-Parallels:-Layers-Are-Horizontal)
  + [Horizontal Layers in Post](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Horizontal-Layers-in-Post)
  + [Horizontal Layers in Networks](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Horizontal-Layers-in-Networks)
* [Layers Evolve Independently](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-Evolve-Independently)
  + [Independent Evolution in Post](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Independent-Evolution-in-Post)
  + [Independent Evolution in Networks](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Independent-Evolution-in-Networks)
* [Encapsulation & Decapsulation](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Encapsulation-&-Decapsulation)

## Introduction to Layered Architectures[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Introduction-to-Layered-Architectures)

When building a large complex system, it helps to approach the problem at gradually increasing levels of abstraction. Thus, systems can be composed of **layers**, each performing a specific set of tasks.

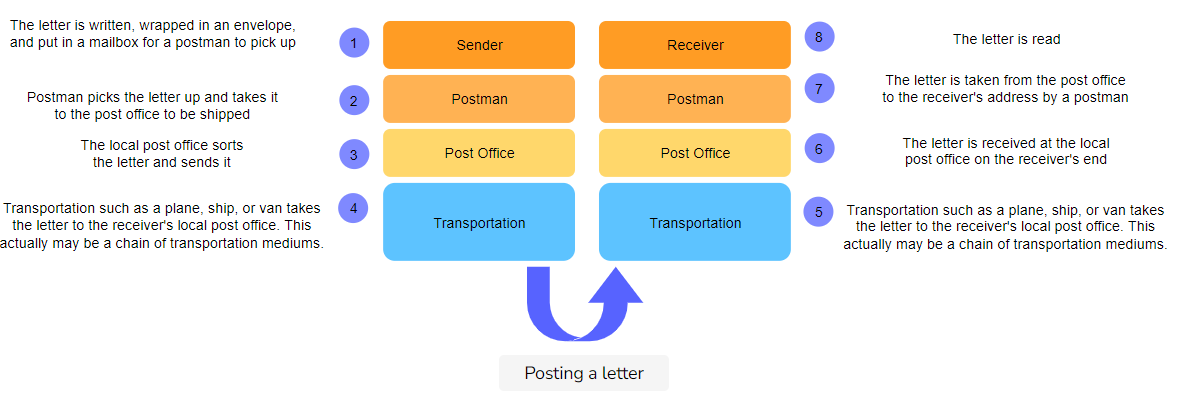
### Why Layers?[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Why-Layers?)

Layered architectures give us modularity by allowing us to discuss **specific, well-defined parts of larger systems**. This makes **changing implementation-level details** and **identifying bugs easier**.

## An Analogy: Post[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#An-Analogy:-Post)

Before we dive deep into different models of the network layer stack, let’s look at an interesting analogy.

Think about posting a letter or a package. The general steps to doing so are as follows,



Notice that a few things are in **parallel with computer networking** here. Here are some examples of how that is the case:

## Layers As Services To Each Other: Layers Are Vertical[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-As-Services-To-Each-Other:-Layers-Are-Vertical)

**Each layer provides some services to the layer above it**. Furthermore, the layer above is **not concerned with the details of how the layer below performs its services**. This is called **abstraction**. So in this way, the layers communicate with each other in a vertical fashion.

### Vertical Layers in Post[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Vertical-Layers-in-Post)

In our letter analogy, each layer is servicing the layer above it. For instance, the postman provides services to the senders and receivers. They collect dropped letters from mailboxes and deliver mail to the houses.

Furthermore, all a sender knows and cares about is that once they write a letter, put it in an envelope, stick a stamp on it and drop it in a letterbox, it will eventually be delivered at the destination. Whether it’s transported on pickup trucks, on railway trains or by air is irrelevant and immaterial to senders. So, how layer 4 does its job is irrelevant to the layers above, and that’s called **abstraction**.

### Vertical Layers in Networks[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Vertical-Layers-in-Networks)

Similarly, computer networks are conceptually divided into layers that each serves the layer above and below it.

* For example, the top layer in most layered models is called the **application layer**. End-user applications live in the application layer, which includes the web and email and are almost always implemented in software. The application layer is also where an outgoing message starts its journey.
* The application needs an underlying service that can get application messages delivered from source to destination and bring back replies which is what the layer(s) after do(es).

Since the underlying layer collects messages from the upper layer for delivery to the destination and hands over messages destined for the upper layer, it **serves the application layer**. Furthermore, the application layer **abstracts**, and hence is not concerned with any implementation details of the layers below.

## Layers Communicate with Their Parallels: Layers Are Horizontal[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-Communicate-with-Their-Parallels:-Layers-Are-Horizontal)

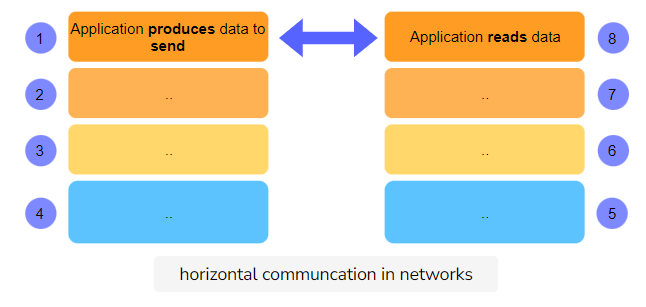
Note that each layer at the sending end has a parallel in the receiving end.

### Horizontal Layers in Post[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Horizontal-Layers-in-Post)

In the post analogy, the letter writer and receiver appear to be in direct communication with each other. The writer writes, the reader reads, oblivious to the man-hours spent in the lower layers. Similarly, the post office at the sender’s side is in communication with some other post office. They cooperate in getting the letter delivered. At the lowest layer, there could be multiple hops. For instance, there is a bicycle pickup of letters from a box, delivered to the post office. Then, the letters are bundled and sent by pickup truck to an airport. The airport flies the postage to a different airport. The airport sends the postage to a post office by a pickup truck and the delivery ensues. Sometimes there are multiple entities horizontally, but we only see the sender and the receiver.

### Horizontal Layers in Networks[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Horizontal-Layers-in-Networks)

This makes more sense in the case of computer networks. For example, applications in the **application layer** send and receive data from the network. The application layer on one end system has a parallel on another end system, i.e., a chat app on one end system could be communicating with a chat app on another. **These applications in the application layer are seemingly communicating with each other directly or horizontally.** They are not aware of the layer below.



## Layers Evolve Independently[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Layers-Evolve-Independently)

Any lower layer in this model provides certain services that the upper layer can build other services upon. The upper layer can evolve to build different kinds of services, like going from text-based email to attachments, to the world wide web, to dynamic websites, interactive gaming, interactive video conferencing and so on, all happening in the top layer over the same infrastructure.

### Independent Evolution in Post[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Independent-Evolution-in-Post)

1. For instance, the item being sent does not necessarily have to be a **letter** – It can also be a **package**.
2. It can be put in an **envelope or a box**.
3. It can be taken to the **post office, dropped off into a post box, or picked up**.
4. The receiver’s end can be a **P.O. box, a home or an office**.

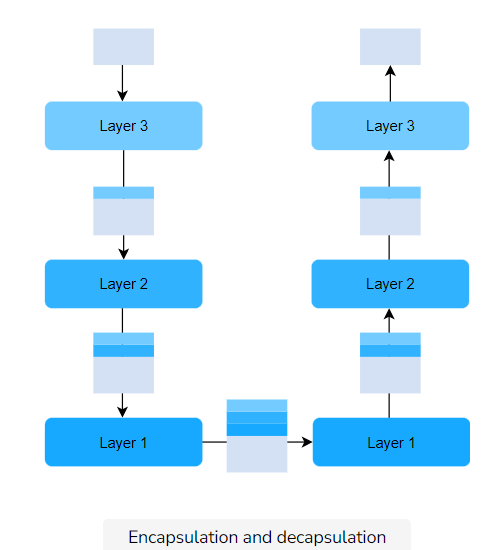
### Independent Evolution in Networks[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Independent-Evolution-in-Networks)

The applications in the application layer can send and receive almost any form of data, be it an **mp3 file or a word document**.

## Encapsulation & Decapsulation[**#**](https://www.educative.io/module/lesson/network-fundamentals/N72J4Dz26Qm#Encapsulation-&-Decapsulation)

Each layer adds its own header to the message coming from above and the receiving entity on the other end removes it. The information in each header is useful for transmitting the message to the layer above. Adding the header is called encapsulation and removing it is called decapsulation. Have a look at the following drawing to see how this works.

We have not given names to these layers because we have not introduced them yet, but the general idea is depicted.



Now that we have an introduction to layered architectures, let’s discuss a popular network reference model in the next lesson.

# The Open Systems Interconnection (OSI) Model

The OSI layer model will help us to understand the overall picture of how computer networks work without getting into too many low-level details.

**We'll cover the following**

* [Common Models](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Common-Models)
* [The OSI Model](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#The-OSI-Model)
* [The Layers of the OSI Model](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#The-Layers-of-the-OSI-Model)
  + [Mnemonic](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Mnemonic)
  + [Application Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Application-Layer)
  + [Presentation Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Presentation-Layer)
  + [Session Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Session-Layer)
  + [Transport Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Transport-Layer)
  + [Network Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Network-Layer)
  + [Data Link Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Data-Link-Layer)
  + [Physical Layer](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Physical-Layer)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Quick-Quiz!)

## Common Models[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Common-Models)

There are several models along which computer networks are organized. The two most common ones are the **Open Systems Interconnection (OSI)** model and the **Transmission Control Protocol/Internet Protocol (TCP/IP)** model.

We will discuss each model and the differences between the two in detail starting with the OSI model.

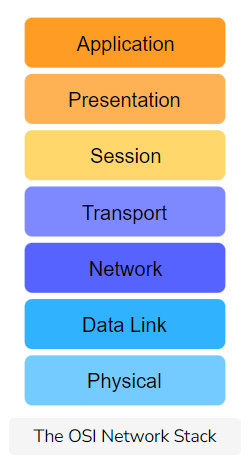
## The OSI Model[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#The-OSI-Model)

The OSI Model was developed in the '70s by the Organization for Standardization (ISO). At this time, the Internet was in its infancy and its protocols had not fully matured. The OSI model **provides a standard** for different computer systems to be able to communicate with each other.

## The Layers of the OSI Model[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#The-Layers-of-the-OSI-Model)

The model splits up a communication system into **7 abstract layers**, stacked upon each other.

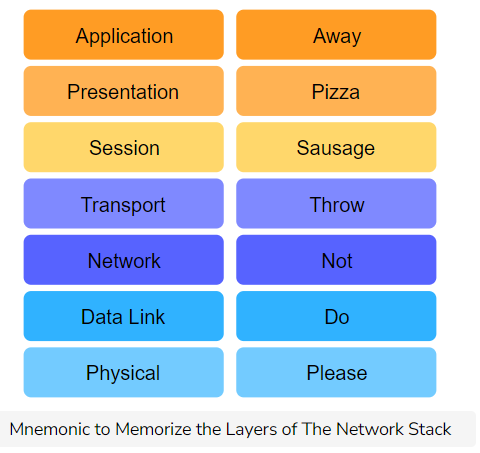
Here are the seven layers of the OSI Model.



### Mnemonic[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Mnemonic)

A good mnemonic device to help remember these layers is:

**Please Do Not Throw Sausage Pizza Away**



Network protocols are implemented in software, hardware or a combination of both, and their hardware and software components are organized into these layers. So the **main purpose** of this ‘network stack’ is to **understand how the components of these protocols fit into the stack and work with each other**.

Here are some key responsibilities of each layer. Note that we are listing only some of the responsibilities of each layer. The exhaustive discussion is deferred to later chapters.

### Application Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Application-Layer)

* These applications or protocols are almost **always implemented in software**.
* **End-users interact** with the application layer.
* The application layer is where most **end-user applications** such as web browsing and email **live**.
* The application layer is where an outgoing message starts its journey so it **provides data for the layer below**.

🔎 **Did You Know?** Ascribing to the OSI model, **Layer 8** is a pseudo-layer that consists of the **end-users**! A lot of IT support humor has involved telling unsuspecting non-tech-savvy users that the issue is a “Layer 8 issue.” So the application layer can technically be said to be sitting in between layers 8 and 6!

### Presentation Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Presentation-Layer)

* **Presents data** in a way that can be easily understood and displayed by the application layer.
  + **Encoding** is an example of such presentation. The underlying layers might use a different character encoding compared to the one used by the application layer. The presentation layer is responsible for the translation.
* **Encryption** (changing the data so that it is only readable by the parties it was intended for) is also usually done at this layer.
* **Abstracts**: the presentation layer assumes that a user session is being maintained by the lower layers and transforms content presentation to suit the application.
* **End-to-end Compression**: The presentation layer might also implement end to end compression to reduce the traffic in the network.

### Session Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Session-Layer)

* The session layer’s responsibility is to take the services of the transport layer and build a service on top of it that **manages user sessions**.
  + As we will see shortly, the transport layer is responsible for transporting session layer messages across the network to the destination. The session layer must manage the mapping of messages delivered by the transport layer to the sessions.
* A session is an exchange of information between local applications and remote services on other end systems.
  + For example, one session spans a customer’s interaction with an e-commerce site whereby they search, browse and select products, then make the payment and logout.
* **Abstracts:** the session layer assumes that connections establishment and packet transportation is handled by the layers below it.

### Transport Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Transport-Layer)

* The **transport layer** also has protocols implemented largely in software.
* Since the application, presentation and session layers may be handing off large chunks of data, the transport layer segments it into smaller chunks.
  + These chunks are called **datagrams or segments** depending on the protocol used.
* Furthermore, sometimes some **additional information** is required to transmit the segment/datagram reliably. The transport layer **adds this information to the segment/datagram**.
  + An example of this would be the **checksum**, which helps ensure that the message is correctly delivered to the destination, i.e. that it’s not corrupted and changed to something else on the way.
  + When additional information is added to the **start** of a segment/datagram, it’s called a **header**.
  + When additional information is appended to the **end** it’s called a **trailer**.

### Network Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Network-Layer)

* Network layer messages are termed as **packets**.
* They facilitate the **transportation of packets** from one end system to another and help to **determine the best routes** that messages should take from one end system to another.
* **Routing protocols** are applications that run on the network layer and exchange messages with each other to develop information that helps them route transport layer messages.
* **Load Balancing** There are many links (copper wire, optical fiber, wireless) in a given network and one objective of the network layer is to keep them all roughly equally utilized. Otherwise, if some links are under-utilized, there will be concerns about the economic sense of deploying and managing them.

### Data Link Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Data-Link-Layer)

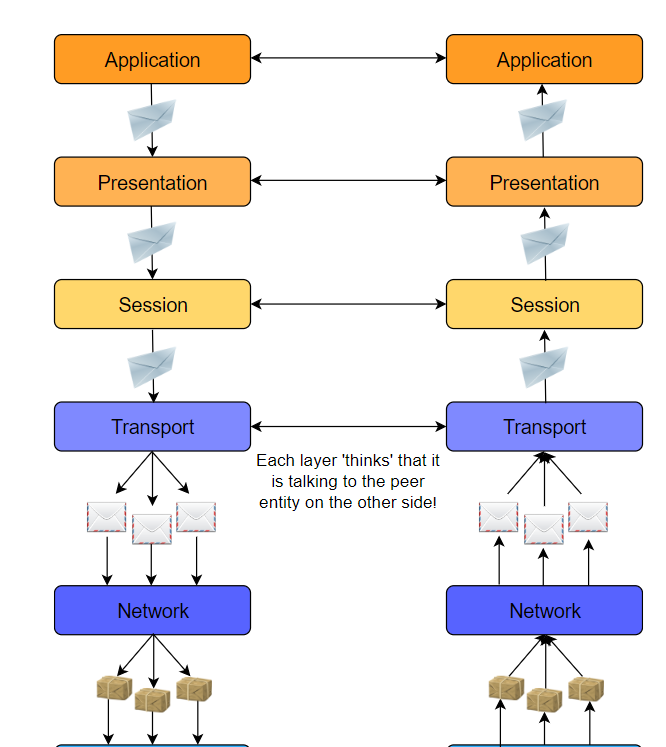
* Allows directly connected hosts to communicate. Sometimes these hosts are the only two things on a physical medium. In that case, the challenges that this layer addresses include **flow control** and **error detection/correction**.
* **Encapsulates packets** for transmission across a single link.
* **Resolves transmission conflicts** i.e., when two end systems send a message at the same time across one singular link.
* **Handles addressing** If the data link is a broadcast medium, addressing is another data link layer problem,
* **Multiplexing & Demultiplexing**:
  + Multiple data links can be multiplexed into something that appears like one, to integrate their bandwidths.
  + Likewise, sometimes we disaggregate a single data link into virtual data links which appear like separate network interfaces.

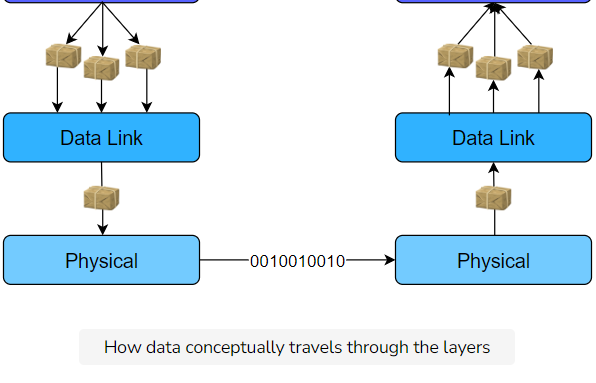
### Physical Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/7AAqwx78YJr#Physical-Layer)

* Consists largely of hardware.
* Provides a solid electrical and mechanical medium to transmit the data.
* Transmits bits. Not logical packets, datagrams, or segments.
* Also has to deal with mechanical specifications about the makeup of the cables and the design of the connectors.

We’ve mostly already studied what constitutes the physical layer. We don’t need to know more than what we’ve looked at in the [Phyical Communication Media](https://www.educative.io/collection/page/10370001/6105520698032128/5697465422446592" \t "_blank) chapter.

Have a look at the following slides to understand how data would conceptually travel through the layers.





# The TCP/IP Model

Let's now have a look at the TCP/IP Model

**We'll cover the following**

* [Introduction](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Introduction)
* [The Layers of The TCP/IP Stack](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#The-Layers-of-The-TCP/IP-Stack)
* [TCP/IP vs OSI](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#TCP/IP-vs-OSI)
  + [Key Differences](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Key-Differences)
  + [Differences in Layer Functionality](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Differences-in-Layer-Functionality)
* [There is No Unanimous Stack](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#There-is-No-Unanimous-Stack)
* [The End-To-End Argument in System Design](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#The-End-To-End-Argument-in-System-Design)
  + [Packet Switched Core](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Packet-Switched-Core)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Quick-Quiz!)

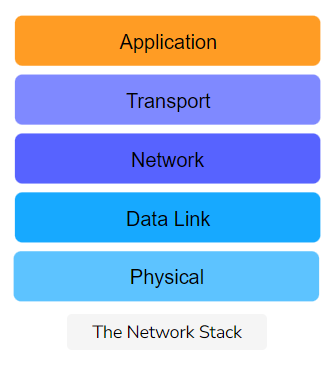
## Introduction[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Introduction)

* The TCP/IP Model, also known as the **Internet protocol suite**, was **developed in 1989**.
* Its **development was funded by DARPA** (Advanced Research Projects Agency (ARPA) was renamed to the Defense Advanced Research Projects Agency (DARPA)!)
* Its technical specifications are detailed in [RFC 1122](https://tools.ietf.org/html/rfc1122).
* This model is primarily based upon the most protocols of the Internet, namely the **Internet Protocol (IP)** and the **Transmission Control Protocol (TCP)**.
* The protocols in each layer are **clearly defined**, unlike in the OSI model. In this course, we’ll largely adhere to the TCP/IP model and take a protocol-oriented approach.

## The Layers of The TCP/IP Stack[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#The-Layers-of-The-TCP/IP-Stack)

The TCP/IP model splits up a communication system into **5 abstract layers**, stacked upon each other. Each layer performs a particular service and communicates with the layers above and below itself.

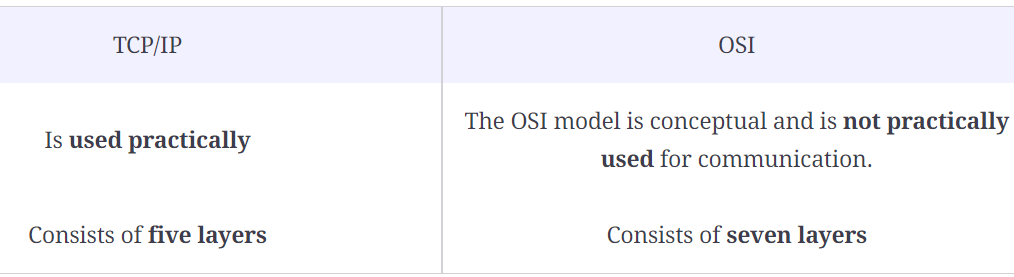
Here are the five layers of the TCP/IP model:



## TCP/IP vs OSI[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#TCP/IP-vs-OSI)

### Key Differences[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Key-Differences)

Here are some main differences between TCP/IP and OSI.

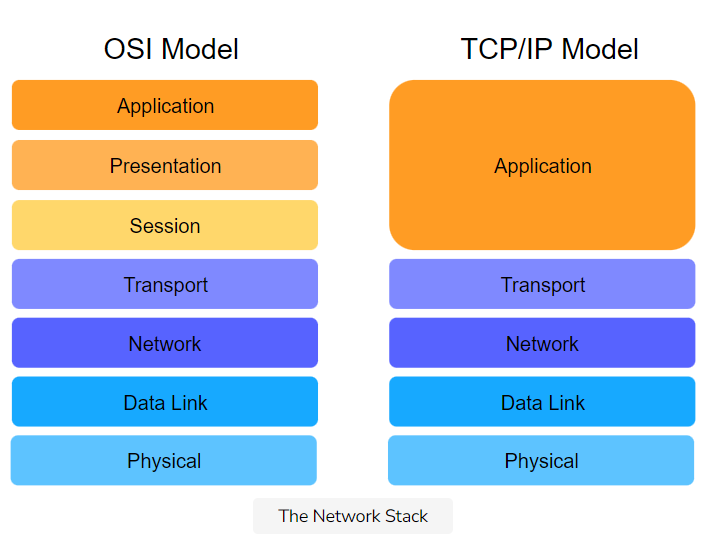


* Elaborating further on the first point, OSI is a **theoretical model** and works very well for teaching purposes, but it’s far too complex for anyone to implement.
* TCP/IP, on the other hand, wasn’t really a model. People just implemented it and got it to work. Then, people **reverse-engineered a reference model** out of it for theoretical and pedagogical purposes. So, something that “sounds like” a great idea might not be the eventual winner. It’s de facto vs de jure standards.

### Differences in Layer Functionality[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Differences-in-Layer-Functionality)

The layers in the TCP/IP stack largely perform the same functions as their counterparts in the OSI model, except that the application layer in the TCP/IP model encompasses the functionalities of the top three layers of the OSI model.

Have a look at the following diagram for a more concrete view.



## There is No Unanimous Stack[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#There-is-No-Unanimous-Stack)

This is an example of where primary sources like RFCs clash with secondary sources like textbooks. There is, in fact, an entire [table on Wikipedia](https://en.wikipedia.org/wiki/Internet_protocol_suite#Layer_names_and_number_of_layers_in_the_literature) (<https://en.wikipedia.org/wiki/Internet_protocol_suite#Layer_names_and_number_of_layers_in_the_literature> )dedicated to the prominent layer stacks! Regardless, we’ll be sticking to the TCP/IP model described above.

## The End-To-End Argument in System Design[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#The-End-To-End-Argument-in-System-Design)

The TCP/IP protocol suite is heavily influenced by the following design choice, also known as the **end-to-end argument**.

Implementing intelligence in the core was too expensive, therefore, intelligence was implemented at edge devices. So, the Internet’s design was of **intelligent end devices** and a **dumb and fast core network**.

# What Is the Application Layer?

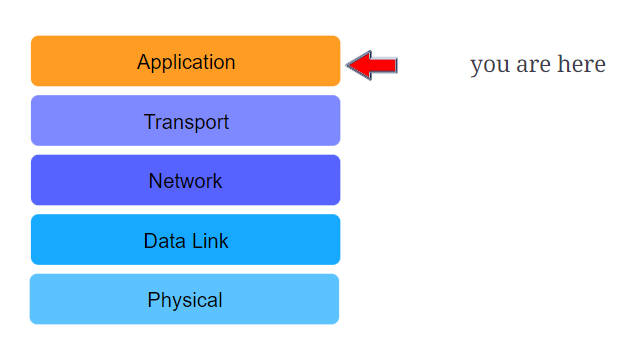
Here's an introduction to the application layer!

**We'll cover the following**

* [You Are Here!](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#You-Are-Here!)
* [Key Responsibilities of the Application Layer](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Key-Responsibilities-of-the-Application-Layer)
  + [The Post Analogy](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#The-Post-Analogy)
* [Where It Exists](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Where-It-Exists)
* [Application Layer Protocols](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Application-Layer-Protocols)

## You Are Here![**#**](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#You-Are-Here!)

We’re starting our study of the TCP/IP layers with the application layer.



### Packet Switched Core[**#**](https://www.educative.io/module/lesson/network-fundamentals/RM7A42REMqz#Packet-Switched-Core)

Furthermore, the core was made **packet-switched**, which means that packets are routed **per-hop**, so they can circumvent failures because the requirement was for resilience.

With **circuit-switched networks**, however, torn connections have to be re-established, if there is still a path.

## Key Responsibilities of the Application Layer[**#**](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Key-Responsibilities-of-the-Application-Layer)

The main job of the application layer is to enable end-users to access the Internet via a number of applications. This involves:

* **Writing data off to the network** in a format that is compliant with the protocol in use.
* **Reading data** from the end-user.
* **Providing useful applications** to end users.
* Some applications also ensure that the data from the end-user is in the correct format.
* Error handling and recovery is also done by some applications.

### The Post Analogy[**#**](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#The-Post-Analogy)

* Imagine you post a package across the world.
* Presumably, the post system would hand it off to an airplane or ship to transport it across the world.
* However, you would take it to the post office first to be shipped off. **Carrying the package to the post office** is what the application layer does in networks, except that **it carries messages to the transport layer**!

## Where It Exists[**#**](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Where-It-Exists)

The application layer resides entirely on end-systems. These end-systems can be any Internet-enabled device, be it a refrigerator or a tower PC.

## Application Layer Protocols[**#**](https://www.educative.io/module/lesson/network-fundamentals/JQOmBM8AZzv#Application-Layer-Protocols)

Most would argue that **user applications are the true purpose of the Internet. If useful applications did not exist**, the Internet would not be what it is today.

* The development of the Internet in the last century started with text-based network apps such as **e-mail**.
* Then came **the** app: the **World Wide Web** which revolutionized everything.
* **Instant messaging** came at the end of the millennium, which has changed the way we communicate.
* Since then, we have come up with **voice over IP**, (WhatsApp calls), **video chat** (Skype), and **video streaming** (YouTube).
* **Social media** has also taken the world by storm resulting in complex human social networks and businesses building on top of these websites.

All of these applications **run on application layer protocols**. Due to the presence of these standard protocols, client applications developed by various vendors can talk to server applications developed by others!

Let’s uncover some of the underlying application layer protocols, in the next few lessons.

# Network Application Architectures

In this lesson, we'll learn about network application architectures.

**We'll cover the following**

* [Client-Server Architecture](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Client-Server-Architecture)
  + [Servers](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Servers)
  + [Clients](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Clients)
  + [An Example](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#An-Example)
* [Data Centers](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Data-Centers)
* [Peer-to-Peer Architecture (P2P)](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Peer-to-Peer-Architecture-(P2P))
  + [An Example](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#An-Example)
* [Hybrid](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Hybrid)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Quick-Quiz!)

Before we start off with application layer protocols, **it’s important to understand how applications are structured across end systems**. This is called the network application’s **architecture** and it’s designed by application developers. The architecture lays out **how the application communicates and with what**.

Let’s discuss some common application architectures.

## Client-Server Architecture[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Client-Server-Architecture)

In this architecture, a network application consists of two parts: **client-side** software and **server-side** software. These pieces of software are generally called **processes**, and they communicate with each other through **messages**.

### Servers[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Servers)

The server process controls access to a centralized resource or service such as a website.

Servers have two important characteristics:

1. Generally, an attempt is made to keep servers online all the time, although 100% availability is impossible to achieve. Furthermore, servers set up as a hobby or as an experiment may not need to be kept online. Nevertheless, the client must be able to find the server online when needed, otherwise, communication wouldn’t take place.
2. They have at least one reliable IP address with which they can be reached.

A good analogy is a 24/7 pizza delivery place. They are always open and have a phone number with which they can always be reached.

### Clients[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Clients)

Client processes use the Internet to consume content and use the services. Client processes almost always initiate connections to servers, while server processes wait for requests from clients.

### An Example[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#An-Example)

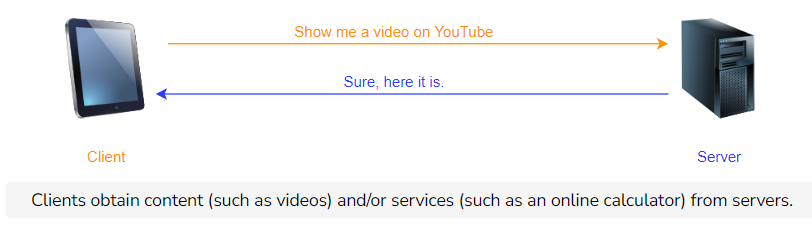
A good example of the client-server architecture is the **web**.

Take **Google** for instance. Google has several servers that control access to videos. So when a [google.com](http://google.com/) is accessed, a client process (a browser) requests Google’s homepage from one of Google’s servers. That server was presumably online, got the request, and granted access to the page by sending it.

## Data Centers**[#](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73" \l "Data-Centers)**

Now, you might have noticed that we mentioned that Google has servers and not one server. That’s because, as mentioned previously, when client-server applications scale, one or even two servers can’t handle the requests from a large number of clients. Additionally, servers may crash due to any reason and might stop working. Most applications have several servers in case one fails. Therefore, several machines host server processes (these machines are called servers too), and they reside in **data centers**.

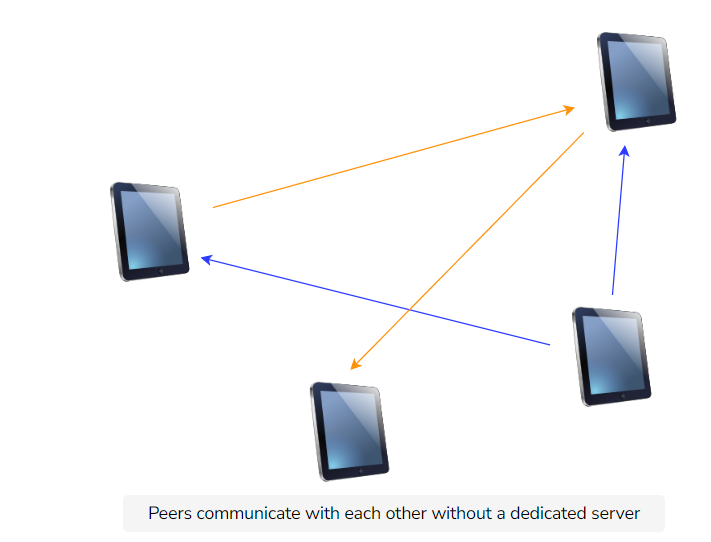
Data centers are buildings that house servers. [Facebook](https://datacenterfrontier.com/facebooks-accelerates-data-center-expansion/), for example, has “nearly 15 million square feet of data center space completed or under construction, with several million more feet in the planning stages” as of **2018**.



## Peer-to-Peer Architecture (P2P)[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Peer-to-Peer-Architecture-(P2P))

In this architecture, applications on end-systems called ‘peers’ communicate with each other. No dedicated server or large data center is involved. Peers mostly reside on PCs like laptops and desktops in homes, offices, and universities.

The key advantage of the P2P architecture is that it can scale rapidly – without the need of spending large amounts of money, time or effort.



Regardless of P2P’s decentralized nature, each peer can be categorized as servers or clients i.e., every machine is capable of being a client as well as a server. Strictly speaking, the peer that initiates a connection is the client, and the other one is called the server.

### An Example[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#An-Example)

A lot of popular applications today, like **BitTorrent**, are based on P2P architectures.

When a file is downloaded via BitTorrent, the downloading party accesses **bits** of the file on several other users’ computers and puts them together on its end. No traditional ‘server’ is involved in this scenario.

📝 **Note: P2P Is Not the Same as File Sharing!** Some early P2P applications were used for file sharing. For example, [Napster](https://en.wikipedia.org/wiki/Napster) and [Gnutella](https://en.wikipedia.org/wiki/Gnutella). Because of the massive impact of these P2P applications, a lot of people associate file sharing exclusively with P2P.  
  
**File sharing** is a specific application. Whereas **P2P** is a design principle for distributed systems and an application architecture.  
  
Also, file sharing is not the only application of P2P. Other examples include: streaming media, telephony, content distribution, routing, and volunteer computing.

## Hybrid[**#**](https://www.educative.io/module/lesson/network-fundamentals/q2Q12zklY73#Hybrid)

The hybrid architecture involves server involvement to some degree. It’s essentially a combination of the P2P and client-server architectures.

# P2P vs. Client-Server

Before we move on with the details of BitTorrent, it's useful to do a quantitative comparison of the P2P architecture with the client-server architecture.

**We'll cover the following**

* [Quantitative Comparison of P2P with Client-Server](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#Quantitative-Comparison-of-P2P-with-Client-Server)
  + [Client-Server](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#Client-Server)
  + [P2P](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#P2P)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#Quick-Quiz!)

## Quantitative Comparison of P2P with Client-Server[**#**](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#Quantitative-Comparison-of-P2P-with-Client-Server)

Let’s calculate how long it will take to transmit a file from one server to a number of clients based on both the P2P and server-client architectures. The calculations will be performed based on the following givens.

* A **server** that can upload at a rate of up\_s*ups*​ where up\_s*ups*​ is the upload speed in bits/second.
* There are **N*N* clients** all wanting to download the same file from the server. Client i*i* can upload at a rate of up\_i*upi*​ bits/second and download at a rate of dwn\_i*dwni*​ bits/second.
* The size of the file that all the peers want is S*S*.

### Client-Server[**#**](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#Client-Server)

Let’s start with the **client-server** architecture. The following can be observed.

* Since N*N* clients each want a file of size S*S*, the server will have to upload NS*NS* bits. The upload rate of the server is up\_s*ups*​ so the server will take at least \frac{NS}{up\_s}*ups*​*NS*​ time to transmit the file to all N*N* clients.
* The client with the lowest download rate (dwn\_{min} = min(dwn\_i)*dwnmin*​=*min*(*dwni*​)) will take at least \frac{S}{dwn\_{min}}*dwnmin*​*S*​ time to download the full file.

So, in total the time taken to transmit the file will be the maximum of both of the times above, i.e.:

\max{\{\frac{NS}{up\_s},\frac{S}{dwn\_{min}} \}}max{*ups*​*NS*​,*dwnmin*​*S*​}

### P2P[**#**](https://www.educative.io/module/lesson/network-fundamentals/mymP13vRg19#P2P)

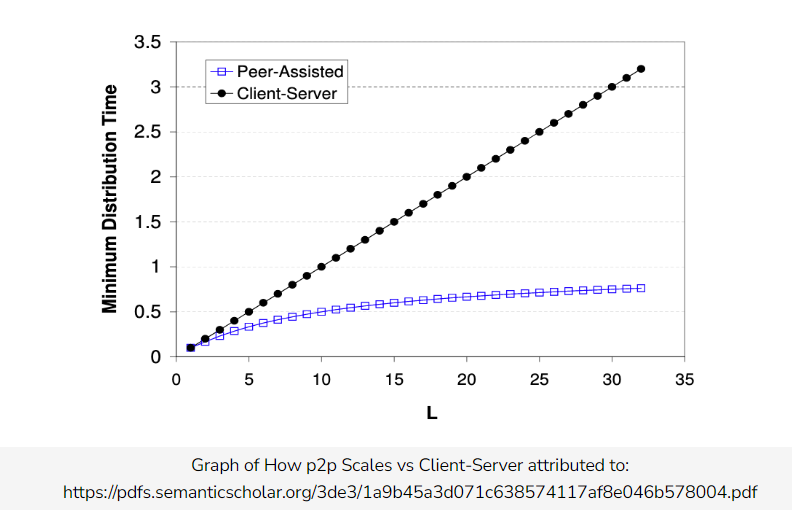
We can make the following observations:

* Initially, only the server has the file. It has to throw the file out into the network and to do that, it will take at least \frac{S}{up\_s}*ups*​*S*​ time. While the file is being sent out into the network of peers, they start to distribute it amongst themselves.
* The peer with the lowest download rate (dwn\_{min}*dwnmin*​) will take at least \frac{S}{dwn\_{min}}*dwnmin*​*S*​ time to download the full file.
* The file cannot be transmitted faster than the total upload speed of the entire network: (up\_{sum} = \{up\_1+up\_2+up\_3+...+up\_N\}*upsum*​={*up*1​+*up*2​+*up*3​+...+*upN*​}). Since the file has to be distributed to all N*N* peers, NS*NS* bits have to be transmitted, that will take \frac{NS}{up\_{sum}}*upsum*​*NS*​ time.

Therefore, the time taken in total to distribute a file of size S*S* to N*N* peers is:

\max{\{\frac{S}{up\_s},\frac{S}{dwn\_{min}} ,\frac{NS}{up\_{sum}}\}}max{*ups*​*S*​,*dwnmin*​*S*​,*upsum*​*NS*​}

Note that as the number of clients/peers, N*N*, grows, the time taken by the client-server architecture also grows. Here is a graph of how the distribution time grows for each architecture as the number of clients/peers grow:



**P2P networks are extremely mathematically scalable**. The resources of a P2P system grows with the number of peers in the system. Thus, applications with P2P architecture are self-scaling.

* 1. In a client-server model, the rate at which a client can download a file is limited by the (C) min( a , b )

# How Processes Communicate

Let's have a quick look at the technical aspect of how applications communicate

**We'll cover the following**

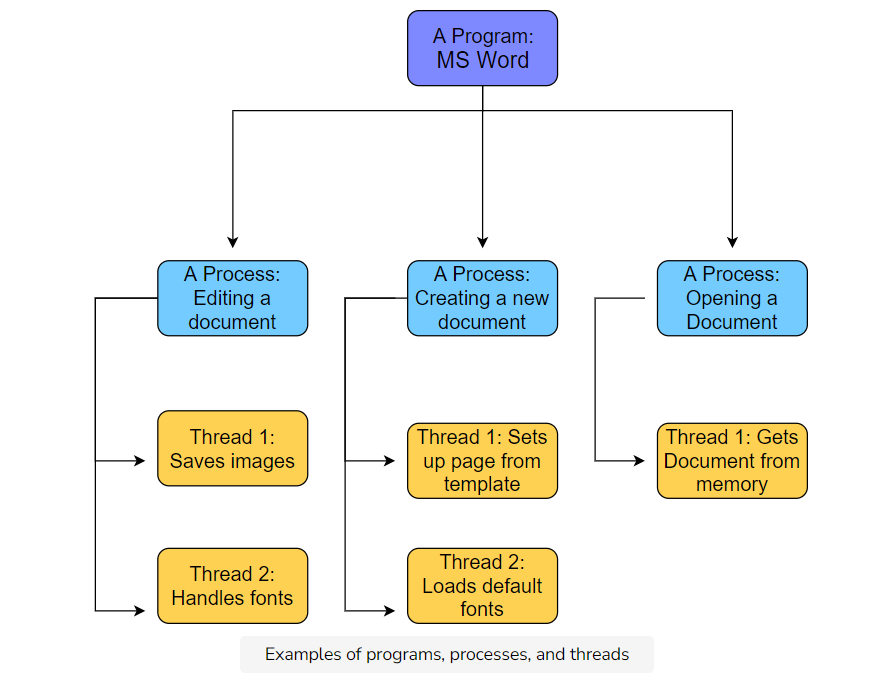
* [Program vs. Process vs. Thread](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Program-vs.-Process-vs.-Thread)
* [Sockets](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Sockets)
* [Addressing](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Addressing)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Quick-Quiz!)

## Program vs. Process vs. Thread [**#**](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Program-vs.-Process-vs.-Thread)

We’ve loosely used the term ‘process’ pretty much interchangeably with the term ‘application’ in the last few chapters. Now, let’s now get a finer definition.

* A **program** is simply an executable file. An application such as MS Word is one example.
* A **process** is any currently running instance of a program. So one program can have several copies of it running at once. One MS Word program can have multiple open windows.
* A **thread** is a lightweight process. One process can have multiple running threads. The difference between threads and processes is that threads do lightweight singular jobs.

Here’s the MS Word example illustrated:



Processes that exist on the same machine can and do regularly communicate with each other following the rules of the machine’s OS. However, we are more interested in how processes that run on different machines communicate.

## Sockets [**#**](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Sockets)

Processes on different machines send messages to each other through the computer network. The interface between a process and the computer network is called a **socket**. Note that sockets do not have anything to do with hardware – they are software interfaces.

Processes simply direct their messages to sockets and don’t worry about it after that.

## Addressing [**#**](https://www.educative.io/module/lesson/network-fundamentals/qZzBpWJgkQy#Addressing)

Messages have to be addressed to a certain application on a certain end system. How is it done with potentially millions of end systems and hundreds of applications on each of them?

Well, it’s done via addressing constructs like **IP addresses and ports**. While both were touched upon [previously](https://www.educative.io/collection/page/10370001/6105520698032128/4598191880142848/#ip-addresses), we would like to reintroduce ports a bit more in-depth.

**Ports**

Since every end-system may have a number of applications running, **ports** are used to address the packet to specific applications. As stated previously, some ports are reserved such as port 80 for HTTP and port 443 for HTTPS.

**An Analogy: Post**

Continuing with our post analogy, you can think of an end-system like a large apartment complex. Each apartment in the complex is an application.

The mailing address of the complex is like the IP address of the end-system. All running applications share it, just like all apartments share the street address. Each application running on a host has a different port number, just like each apartment has a different apartment number.

**Ephemeral Ports**

The port that an application will use is usually predefined by its application developers. So an application can have port 3000 reserved for it. But what if **several instances (processes) of an application are running at once**? How will the system address those processes?

Well, the answer lies in [Ephemeral Ports](https://en.wikipedia.org/wiki/Ephemeral_port). Different port numbers are dynamically generated for each instance of an application. The port is freed once the application is done using it.

Furthermore, **server processes need to have well defined and fixed port numbers** so that clients can connect to them in a systematic and predictable way. However, **clients don’t need to have reserved ports**. They can use ephemeral ports. Servers can also use ephemeral ports **in addition** to the reserved ones. For instance, a client makes the initial connection to the server on a well-known port and the rest of the communication is carried out by connecting to an ephemeral port on the server.

# HTTP: The Basics

Welcome to the core of this course! We are finally getting started with protocols, the first of which is HTTP.

**We'll cover the following**

* [Introduction](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Introduction)
* [Objects](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Objects)
* [The Anatomy of a URL](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#The-Anatomy-of-a-URL)
* [HTTP](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#HTTP)
* [HTTP Requires Lower Layer Reliability](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#HTTP-Requires-Lower-Layer-Reliability)
* [Types of HTTP Connections](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Types-of-HTTP-Connections)
* [Non-persistent HTTP](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Non-persistent-HTTP)
* [Persistent HTTP](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Persistent-HTTP)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Quick-Quiz!)

## Introduction[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Introduction)

The Internet was an obscure set of methods for file transfer and email used by academics and researchers. The World Wide Web was invented to allow the European research organization CERN to present documents linked by hypertexts. All of that changed though when it caught the public’s eye and popularized the Internet. The web was different from other services such as cable television, because it served content based on demand. People could watch what they wanted. **HTTP** or **HyperText Transfer Protocol** is the protocol at the core of the web.

## Objects[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Objects)

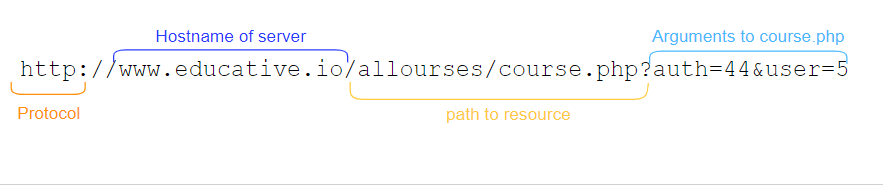
* Web pages are objects that consist of other **objects**.
* An **object is simply a file** like an HTML file, PNG file, MP3 file, etc.
* Each object has a URL
* The **base object** of a web page **is often an HTML file** that has **references to other objects** by making requests for them via their URL.

📝 **Note: HTML** or HyperText Markup Language is the standard markup language to build webpages.

## The Anatomy of a URL[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#The-Anatomy-of-a-URL)

A **URL**, or **Universal Resource Locator**, is used to locate files that exist on servers. URLs consist of the following parts:

* **Protocol** in use
* The **hostname** of the server
* The **location of the file**
* **Arguments** to the file

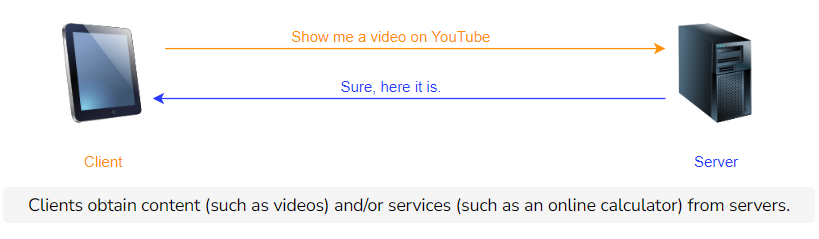


## HTTP[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#HTTP)

Let’s get back into **HTTP**. It’s a client-server protocol that specifies how Web clients request Web pages from Web servers and how Web servers send them.

Remember the following diagram from the lesson on [Network Application Architectures](https://www.educative.io/collection/page/10370001/6105520698032128/6317711061680128)? Well, it was actually outlining HTTP in general.

* The **orange** arrow represents an **HTTP request**
* The **blue** arrow represents an **HTTP response**



The first message is called an **HTTP request** and the second one an **HTTP response**. There’s a whole class of protocols that are considered **request-response protocols**. HTTP is one of them. We will look in more detail at the HTTP request message in the next lesson and response after that!

Note that HTTP is a **stateless protocol**: servers do not store any information about clients by default. So if a client requests the same object multiple times in a row, the server would send it and would not know that the same client is requesting the same object repeatedly.

## HTTP Requires Lower Layer Reliability[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#HTTP-Requires-Lower-Layer-Reliability)

* Application layer protocols rely on underlying transport layer protocols called **UDP** (User Datagram Protocol) and **TCP** (Transmission Control Protocol).
* For now, all you need to know is that **TCP ensures that messages are always delivered**. Messages get delivered in the order that they are sent.
* **UDP does not ensure that messages get delivered**. This means that some messages may get dropped and so never be received.
* **HTTP uses TCP** as its underlying transport protocol so that messages are guaranteed to get delivered in order. This allows the application to function without having to build any extra reliability as it would’ve had to with UDP.

This sort of reliance on other layers for certain jobs is one of the key advantages of a layered architecture!

* **TCP is connection-oriented**, meaning a connection has to be initiated with servers using a series of starting messages.
* Once the connection has been made, the client exchanges messages with the server until the connection is officially closed by sending a few ending messages.

## Types of HTTP Connections[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Types-of-HTTP-Connections)

There are two kinds of HTTP connections:

* **Non-persistent HTTP connections**
* **Persistent HTTP connections**

These two kinds of HTTP connections use TCP differently. Let’s discuss the key advantages and disadvantages of each.

## Non-persistent HTTP[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Non-persistent-HTTP)

**Non-persistent HTTP** connections use **one TCP connection per request**. Assume a client requests the base HTML file of a web page. Here is what happens:

1. The client initiates a TCP connection with a server
2. The client sends an HTTP request to the server
3. The server retrieves the requested object from its storage and sends it
4. The client receives the object which in this case is an HTML file. If that file has references to more objects, steps 1-4 are repeated for each of those
5. The server closes the TCP connection

For each HTTP request, more requests tend to follow, as well to fetch images, javascript files, CSS files, and other objects.

The underlying TCP connection requires three TCP messages are sent between the client and server. Similarly, when the connection is closed, three TCP messages are sent back and forth between the client and server.

## Persistent HTTP[**#**](https://www.educative.io/module/lesson/network-fundamentals/m2ALwGP88kO#Persistent-HTTP)

An HTTP session typically involves multiple HTTP request-response pairs, for which separate TCP connections are established and then torn down between the same client and server. This is inefficient. Later on, **Persistent HTTP** was developed, which used a single client-server TCP connection for all the HTTP request-responses for a session.

Typically, if there have been no requests for a while, the server closes the connection. The duration of time before the server closes the connection is configurable.

# HTTP: Request Messages

HTTP request messages are a pivotal part of the protocol. Let's have a close look at them!

**We'll cover the following**

* [Introduction](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#Introduction)
* [HTTP Request Messages](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#HTTP-Request-Messages)
* [The Anatomy of an HTTP Request Line](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#The-Anatomy-of-an-HTTP-Request-Line)
  + [HTTP Methods](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#HTTP-Methods)
  + [URL](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#URL)
  + [Version](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#Version)
  + [The Anatomy of HTTP Header Lines](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#The-Anatomy-of-HTTP-Header-Lines)

## Introduction[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#Introduction)

There are two types of HTTP messages as discussed previously:

* HTTP **request messages**
* HTTP **response messages**

We’ll study request messages in this one.

## HTTP Request Messages[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#HTTP-Request-Messages)

Let’s look at request messages first. Here is an example of a typical HTTP message:

GET /path/to/file/index.html HTTP/1.1

Host: www.educative.io

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

Accept: text/html

It should be noted that,

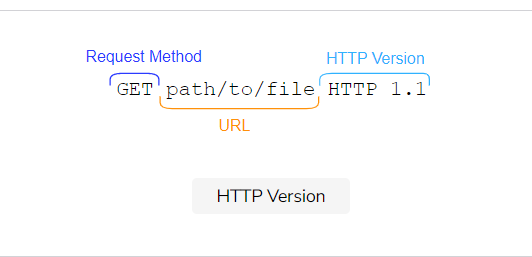
* HTTP messages are in **plain ASCII text**
* **Each line** of the message **ends with** two control characters: a **carriage return and a line feed**: \r\n.
  + The last line of the message also ends with a carriage return and a line feed!
* This particular message has 6 lines, but HTTP messages can have **one or as many lines as needed**.
* The first line is called the **request line** while the rest are called **header lines**.

## The Anatomy of an HTTP Request Line[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#The-Anatomy-of-an-HTTP-Request-Line)

The HTTP request line is followed by an HTTP header. We’ll look at the request line first. The request line consists of three parts:

* **Method**
* **URL**
* **Version**

Let’s discuss each.

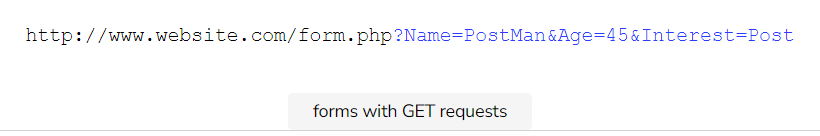


### HTTP Methods[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#HTTP-Methods)

HTTP methods tell the server what to do. There are a lot of HTTP methods but we’ll study the most common ones: GET, POST, HEAD, PUT, or DELETE.

* **GET** is the most common and **requests data**.
* **POST** **puts an object on the server**.
  + This method is generally used when the client is not sure where the new data would reside. The server responds with the location of the object.
  + The data posted can be a message for a bulletin board, newsgroup, mailing list, a command, a web form, or an item to add to a database.
  + The POST method technically requests a page but that depends on what was entered.
* **HEAD** is similar to the GET method except that **the resource requested does not get sent in response. Only the HTTP headers are sent** instead.
  + This is useful for quickly retrieving meta-information written in response headers, without having to transport the entire content. In other words, it’s useful to check with minimal traffic if a certain object still exists. This includes its meta-data, like the last modified date. The latter can be useful for caching.
  + This is also useful for testing and debugging.
* **PUT** **uploads an enclosed entity under a supplied URI**. In other words, it **puts** data at a specific location. If the URI refers to an already existing resource, it’s replaced with the new one. If the URI does not point to an existing resource, then the server creates the resource with that URI.
* **DELETE** **deletes an object** at a given URL.

Note that while most forms are sent from the POST method, the GET method is also used sometimes with the entries of the form appended to the URL, as in arguments like this:



However, sending forms with a POST request is generally better because:

1. The amount of data that can be sent via a post request is unlimited.
2. The form’s fields are not shown in the URL.

📝 **Note: URIs & URLs**

* **Uniform Resource Locators (URLs)** URLs are used to identify an object over the web. [RFC 2396](https://tools.ietf.org/html/rfc2396). A URL has the following format: protocol://hostname:port/path-and-file-name
* **Uniform Resource Identifiers (URIs)** can be more specific than URLs in such a way that they can locate fragments within objects too [RFC 3986](https://tools.ietf.org/html/rfc3986). A URI has the following format: http://host:port/path?request-parameters#nameAnchor. For instance, https://www.educative.io/collection/page/10370001/6105520698032128/6460983855808512/#http-methods is a URI.

### URL[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#URL)

This is the location that any HTTP method is referring to.

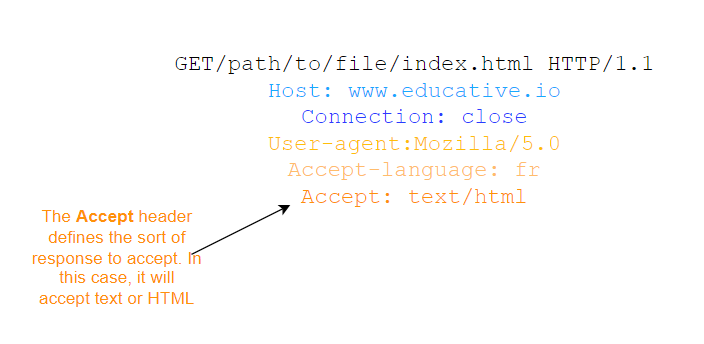
### Version[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#Version)

The HTTP version is also specified in the request line. The latest version of HTTP is [HTTP/2](https://en.wikipedia.org/wiki/HTTP/2).

### The Anatomy of HTTP Header Lines[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8rmj5AjEJm#The-Anatomy-of-HTTP-Header-Lines)

The HTTP request line is followed by an HTTP header. A lot of HTTP headers exist! We’ll be covering the most important ones in this lesson. However, if you’re interested, you can [read further](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) about all of them.

* The first header line specifies the Host that the request is for.
* The second one defines the type of HTTP Connection. It’s Non-persistent in the case of the following drawing as the connection is specified to be closed.
* The user-agent line specifies the client. This is useful when the server has different web pages that exist for different devices and browsers.
* The Accept-language header specifies the language that is preferred. The server checks if a web page in that language exists and sends it if it does, otherwise the server sends the default page.
* The Accept header defines the sort of response to accept. It can be anything like HTML files, images, and audio/video.



In the next lesson, we’ll conduct an exercise to look at real HTTP request messages!

# Exercise: Looking at a Real HTTP Request

In this lesson, you will be looking at real HTTP messages right from your browser!

**We'll cover the following**

* [Open up the Developer Tools on Your Browser](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Open-up-the-Developer-Tools-on-Your-Browser)
* [Go to the Network Tab](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Go-to-the-Network-Tab)
* [Click on Any Entry](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Click-on-Any-Entry)
* [An Example of an Entry](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#An-Example-of-an-Entry)

## Open up the Developer Tools on Your Browser[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Open-up-the-Developer-Tools-on-Your-Browser)

Have a look at this GIF. We were on **Firefox** here.

1. Navigate to any website. We picked [google.com](http://google.com/).
2. Right-click anywhere.
3. Click on ‘inspector tools’ in the drop-down menu.

## Go to the Network Tab[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Go-to-the-Network-Tab)

1. The network tab should be one of the tabs on the top-bar (or sidebar in some browsers).
2. Find it and click on it.
3. There may be a chance that your browser hasn’t logged any network calls. In that case, just reload the page.

## Click on Any Entry[**#**](https://www.educative.io/module/lesson/network-fundamentals/N8GX52APrRL#Click-on-Any-Entry)

1. You’ll see a bunch of calls. Click on any one of them.
2. You’ll see details about the HTTP message including the request headers, the kind of request, and the headers. We encourage you to spend some time exploring this.

# HTTP: Response Messages

Let's look at what HTTP response messages look like!

**We'll cover the following**

* [Introduction](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Introduction)
* [Status Line](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Status-Line)
  + [Status Code](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Status-Code)
* [Header Lines](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Header-Lines)
  + [How HTTP Headers Are Chosen](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#How-HTTP-Headers-Are-Chosen)
* [Quick Quiz on HTTP!](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Quick-Quiz-on-HTTP!)

## Introduction[**#**](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Introduction)

Let’s start with a typical example of an HTTP response message:

HTTP/1.1 200 OK

Connection: close

Date: Tue, 18 Aug 2015 15: 44 : 04 GMT

Server: Apache/2.2.3 (CentOS)

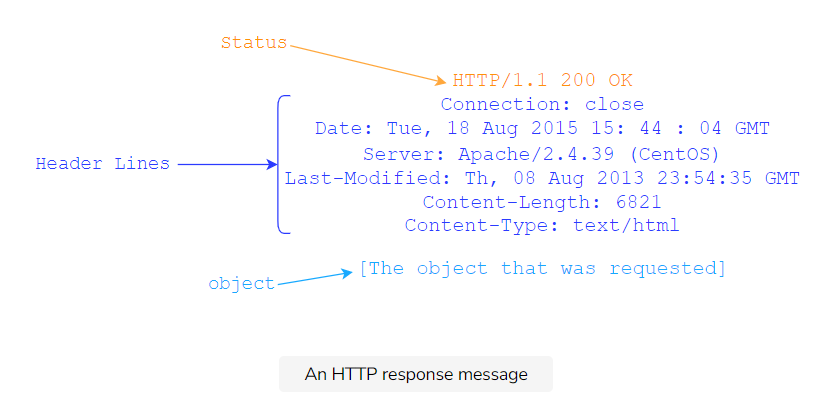
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

[The object that was requested]

It has 3 parts: an initial **status line**, some **header lines** and an **entity body**.



📝 **Note: HTTP response messages don’t have the URL or the method fields**. Those are strictly for request messages.

## Status Line[**#**](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Status-Line)

* HTTP response status lines start with the **HTTP version**.

### Status Code[**#**](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Status-Code)

* The **status code** comes next which tells the client if the request succeeded or failed.
* There are a lot of status codes:
  + 1xx codes fall in the informational category
  + 2xx codes fall in the success category
  + 3xx codes are for redirection
  + 4xx is client error
  + 5xx is server error

Here is a list of some common status codes and their meanings:

* **200 OK**: the request was successful, and the result is appended with the response message.
* **404 File Not Found**: the requested object doesn’t exist on the server.
* **400 Bad Request**: generic error code that indicates that the request was in a format that the server could not comprehend.
* **500 HTTP Internal Server Error**: the request could not be completed because the server encountered some unexpected error.
* **505 HTTP Version Not Supported**: the requested HTTP version is not supported by the server.

Have a look at pages 39 and 40 of [RFC 2616](https://www.ietf.org/rfc/rfc2616.txt) for a comprehensive list.

## Header Lines[**#**](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#Header-Lines)

Let’s study the header lines.

* **Connection type**. In this case, indicates that the server will close the TCP connection after it sends the response.
* **Date**. The date at which the response was generated.
* **Server**. Gives server software specification of the server that generated the message. Apache in this case.
* **Last-Modified.** The date on which the object being sent was last modified.
* **Content-Length.** The length of the object being sent in 8-bit bytes.
* **Content-Type.** The type of content. The type of the file is not determined by the file extension of the object, but by this header.

The **response** body contains the file requested.

### How HTTP Headers Are Chosen[**#**](https://www.educative.io/module/lesson/network-fundamentals/x1W67An4Qkn#How-HTTP-Headers-Are-Chosen)

Lastly, you must be wondering how browsers decide which HTTP headers to include in requests and how servers decide which headers to return in the response. That **depends on a complex mix of factors such as the browser, the user configurations and products**.

Which of the following are the main features of HTTP?

1. HTTP is a request and response protocol.
2. HTTP is a stateless protocol.

# Exercise: Looking at a Real HTTP Response

In this lesson, we'll look at and study real HTTP responses via cURL.

**We'll cover the following**

* [cURL](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#cURL)
* [Explanation](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#Explanation)
* [Sample Output](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#Sample-Output)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#Quick-Quiz!)

## cURL**[#](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D" \l "cURL)**

Run the following command to look at a real HTTP response.

curl http://example.org --head -silent

**cURL** (pronounced ‘curl’) is a command-line tool that transfers data to or from a server. The transfer can be based on a vast set of protocols, so we’ll be seeing cURL a lot. It’s perfect for our purposes because it doesn’t require live user interaction. cURL stands for “Client URL.” You can read more about cURL on its [manpage](https://curl.haxx.se/docs/manpage.html" \t "_blank).

## Explanation[**#**](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#Explanation)

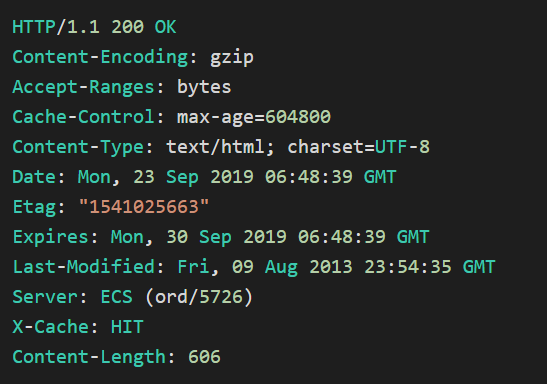
Let’s learn about all of its components.

* curl is the name of the command that tells the terminal that this is a curl command.
* The --head flag or -I in short, tells cURL to send an HTTP request with the head method. In other words, the entity-body of the HTTP message is not fetched.
* The -silent flag tells cURL to not display the progress meter. The progress meter is interpreted as an error on our platform, which is why we decided to remove it. The command is perfectly fine without this flag otherwise.

We encourage you to explore the cURL command. You can find a list of all the flags under the ‘options’ heading on cURL’s [manpage](https://curl.haxx.se/docs/manpage.html) (<https://curl.se/docs/manpage.html> ). Try different websites and different flags and see what you get!

## Sample Output[**#**](https://www.educative.io/module/lesson/network-fundamentals/qA9wXB5wj8D#Sample-Output)

The output of this command is an HTTP response such as the following. Notice the HTTP response code and the headers.



# Cookies

Let's discuss another key concept of computer networking, cookies!

**We'll cover the following**

* [Introduction](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Introduction)
* [Set-cookie Header](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Set-cookie-Header)
* [The Dangers of Cookies](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#The-Dangers-of-Cookies)
  + - [Example](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Example)
  + [Blocking Third-Party Cookies Is Not Enough!](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Blocking-Third-Party-Cookies-Is-Not-Enough!)
  + [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Quick-Quiz!)

## Introduction[**#**](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Introduction)

You might have heard of the term ‘cookie’ used a lot in the context of computer networks and privacy. Let’s have a closer look at what they are.

HTTP is a stateless protocol, but we often see websites where session state is needed. For instance, imagine you are browsing for products on an e-commerce website. How does the server know if you are logged in or not, or if the protocol is stateless? How does the server know what’s in your shopping cart when checking out if the protocol is stateless? Cookies allow the server to keep track of this sort of information.

**How Cookies Work**

* Cookies are **unique string identifiers** that can be stored on the client’s browser.
* These identifiers are **set by the server through HTTP headers** when the client first navigates to the website.
* After the cookie is set, it’s sent along with subsequent HTTP requests to the same server. This **allows the server to know who is contacting it** and hence serve content accordingly.

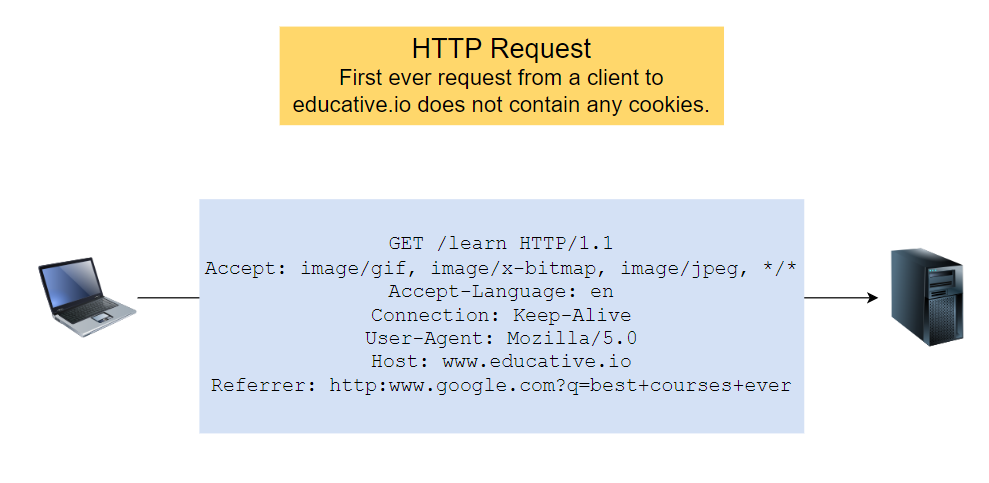
So the HTTP request, the HTTP response, the cookie file on the client’s browser, and a database of cookie-user values on the server’s end are all involved in the process of setting and using cookies.

## Set-cookie Header[**#**](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Set-cookie-Header)

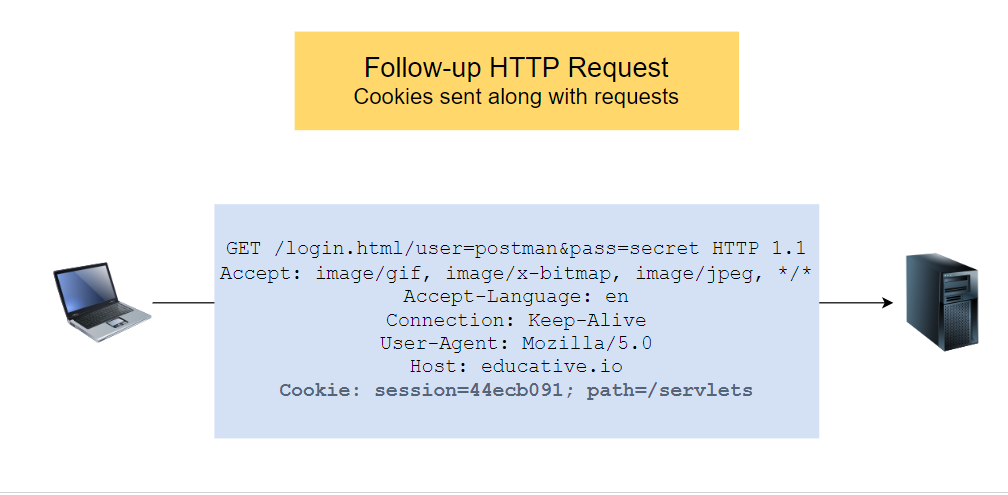
Let’s look at how cookies work in a bit more detail. When a server wants to set a cookie on the client-side, it includes the header Set-cookie: value in the HTTP response. This value is then **appended to a special cookie file stored on your browser**. The cookie file contains:

* The website’s domain
* The string value of the cookie
* The date that the cookie expires (yes, much like actual cookies, they do expire)

Have a look at the following slides to see how cookies work in practice.







## The Dangers of Cookies[**#**](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#The-Dangers-of-Cookies)

While cookies seem like a great idea to make HTTP persistent when needed, cookies have been **severely abused in the past**.

If a website has stored a cookie on your browser, **it knows exactly when you visit it, what pages you visit and in what order**. This itself makes some people uncomfortable.

**Third-party Cookies**

Also, websites may not necessarily know personally identifiable information about you such as your name (by the way, websites that require you to sign-up do know your name), and they may only know the value of your cookie. But what if **websites can track what you do on other websites**? Well, they can. Welcome to the concept of third-party cookies.

cookie monster; image attribution: https://www.flickr.com/photos/tomcrouse/23712101336

While we can’t go into too much detail, it suffices to know that **third-party cookies are cookies set for domains that are not being visited**.

#### Example[**#**](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Example)

1. A user visits [amazon.com](http://amazon.com/).
2. A cookie for [**free-stats.com**](http://free-stats.com/) is subsequently set on their browser because free-stats has placed an advertisement on Amazon. Notice that this is **a third-party cookie**!
3. Suppose, the user visits [ebay.com](http://ebay.com/), and **eBay also has placed an advertisement for**[**free-stats.com**](http://free-stats.com/).
4. The **same cookie set on the Amazon site will be reused** and sent to free-stats along in an HTTP request with the name of the host that the user is on.
5. Free-stats **can in this way track every website the user visits** that they are advertising on and create more targeted ads in order to generate greater revenue.

Also, the public has largely considered third-party cookies to be a breach of privacy and so rejected them. Most modern browsers come with the in-built option to block third-party cookies.

### Blocking Third-Party Cookies Is Not Enough![**#**](https://www.educative.io/module/lesson/network-fundamentals/7noQjM73YWO#Blocking-Third-Party-Cookies-Is-Not-Enough!)

However, firms have come up with several workarounds including but not limited to:

* [Respawning cookies](https://en.wikipedia.org/wiki/Zombie_cookie)
* [Flash cookies](https://en.wikipedia.org/wiki/Local_shared_object)
* [Entity tags](https://en.wikipedia.org/wiki/HTTP_ETag)
* [Canvas fingerprinting](https://en.wikipedia.org/wiki/Canvas_fingerprinting)

# Exercise: View and Manage Your Cookies

In this lesson, we'll go through an exercise to manage your browser's cookies.

**We'll cover the following**

* [Viewing Cookies For a Page](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Viewing-Cookies-For-a-Page)
* [Managing Cookies](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Managing-Cookies)
  + [Chrome](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Chrome)
  + [Safari](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Safari)
  + [Firefox](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Firefox)

## Viewing Cookies For a Page[**#**](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Viewing-Cookies-For-a-Page)

For most browsers, while you can’t view your entire cookie file, you can view cookies for individual websites.

1. Open up any **website of your choice**.
2. Open up the **developer tools**. All browsers have some form of developer tools.
3. Click on the **storage tab** to view details of the cookies that the website has set.

## Managing Cookies[**#**](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Managing-Cookies)

Most modern browsers allow some degree of cookie management (viewing, editing and deleting), which can be integral to protecting your privacy on the Internet. Here are some links to instructions on how to manage cookies for popular browsers. Unfortunately, for most browsers, all cookies cannot be viewed at once, but they can be managed.

### Chrome[**#**](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Chrome)

Google has provided some [instructions](https://developers.google.com/web/tools/chrome-devtools/storage/cookies) to view cookies on Chrome. Have a look!

### Safari[**#**](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Safari)

This [Apple Support page](https://support.apple.com/en-vn/guide/safari/sfri11471/mac) has some instructions on managing cookies. While you can’t view the content of the cookie files, you can view which websites have stored cookies on your browser.

### Firefox[**#**](https://www.educative.io/module/lesson/network-fundamentals/JPBMY3lrDGg#Firefox)

Here’s an [official page](https://support.mozilla.org/en-US/kb/enable-and-disable-cookies-website-preferences) to manage cookies on Firefox.

Now that we have a good understanding of HTTP, and HTTP specific constructs, let’s move on to a new protocol!

# DNS: Introduction

DNS is the Internet's directory service. Let's jump right in!

**We'll cover the following**

* [How Do We Find Things on The Internet?](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#How-Do-We-Find-Things-on-The-Internet?)
* [Distributed Hierarchical Database](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Distributed-Hierarchical-Database)
* [DNS Namespaces](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#DNS-Namespaces)
  + [Root DNS Servers](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Root-DNS-Servers)
  + [Top-level Servers](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Top-level-Servers)
  + [Authoritative Servers](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Authoritative-Servers)
  + [Local DNS Cache](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Local-DNS-Cache)
  + [Local DNS Servers](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Local-DNS-Servers)
* [Quick Quiz!](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Quick-Quiz!)

## How Do We Find Things on The Internet?[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#How-Do-We-Find-Things-on-The-Internet?)

Generally, using one of the following ways:

* **Addresses** or **locations** that specify where something is.
  + Just like with a physical address, we still may need a map to get to the address. On the Internet, the addresses are typically IP addresses, and routers know the map.
* **Names**. In particular, domain names, or the unique name that identifies a websites, are mapped into IP addresses based on lookup service that uses a database. The most well-known lookup service is the **Domain Name System (DNS)**. So when you enter the URL ‘[educative.io](http://educative.io/)’ into your browser, it uses DNS to find the actual IP address of the server that hosts it.
* [**Content-based addressing**](https://en.wikipedia.org/wiki/Content-addressable_storage).
  + The content itself is used to look up its location.

📝 **Note A Useful Analogy:** Domain names are like actual people names and IPs are like phone numbers.

In this lesson, our focus will be on **DNS**, the client-server application layer protocol that translates hostnames on the Internet to IP addresses.

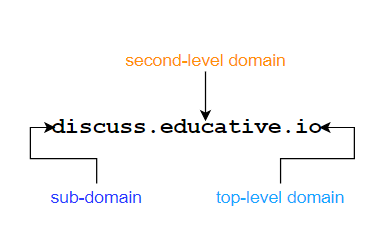
At the core, the Internet operates on IP addresses, but these are difficult to remember for humans. So, DNS names are preferably used at the application layer for which the DNS provides a mapping to IP addresses. For example, HTTP first translates the DNS hostname provided by the user in the URL to its IP address and then attempts to connect to the server. Furthermore, DNS is not just a protocol. It also consists of a distributed database of names that map to IP addresses. So essentially it’s a directory service.

## Distributed Hierarchical Database[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Distributed-Hierarchical-Database)

One single database on one single server does not scale for reasons such as:

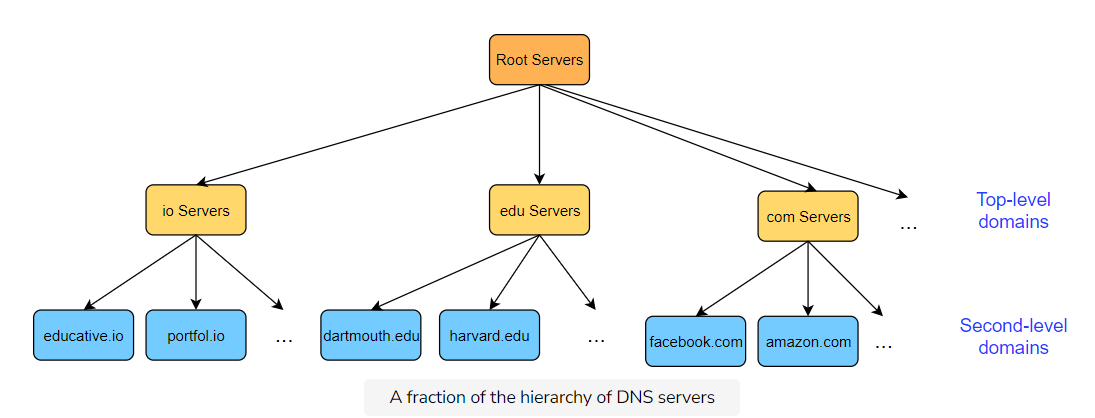
* Single point of failure. If the server that has the database crashes, DNS would stop working entirely, which is too risky.
* Massive amounts of traffic. Everyone would be querying that one server. It will not be able to handle that amount of load.
* Maintenance. Maintaining the server would become critical to the operation of DNS.
* Location. Where would the server be located?

This is why DNS employs several servers, each with part of the database. Also, the servers exist in a hierarchy. To understand this hierarchy better, you need to understand how URLs are broken down into their hierarchies. Have a look at the following diagram.



## DNS Namespaces[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#DNS-Namespaces)

The parts of the URL above roughly map to DNS servers. These servers manage the abstract space of domains. The servers all exist in a hierarchy. Have a look at the following diagram.



### Root DNS Servers[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Root-DNS-Servers)

Root DNS servers are the first point of contact for a DNS query (after the client’s local cache of names and IP addresses). They exist at the top of the hierarchy and point to the appropriate TLD server in reply to the query. So a query for educative.io would return the IP address of a server for the top-level domain, io.

As of the writing of this course, there are 1017 instances of root servers operated by 12 different organizations. To get a full list and an interactive map, have a look at [root-servers.org](http://root-servers.org/).

### Top-level Servers[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Top-level-Servers)

Servers in the top-level domain hold mappings to DNS servers for certain domains. Each domain is meant to be used by specific organizations only. Here are some common domains:

* **com**: This TLD was initially meant for **commercial** organizations only - but it has now been opened for general use.
  + Example: [codinginterview.com](http://codinginterview.com/)
* **edu**: Used by educational institutions.
  + Example: [stanford.edu](http://stanford.edu/)
* **gov**: **Only** used by the U.S. government.
  + Example: [nasa.gov](http://nasa.gov/)
* **mil**: Used by U.S. military organizations.
  + Example: army.mil
* **net**: It was initially intended for use by organizations working in network technology such as ISPs, but it is now a general purpose domain like com.
  + Example: [doubleclick.net](http://doubleclick.net/)
* **org**: This domain was intended for non-profit organizations but has been opened for general use now.
  + [mozilla.org](http://mozilla.org/)
* **pk, uk, us,…**: Country suffixes. 244 two-letter ones exist.
* Some new and uncommon suffixes include: **name**, **mobi**, **biz**, **pro**.
* **International domains**: 中國

Today, the set of top-level domain-names is managed by the [Internet Corporation for Assigned Names and Numbers](https://www.icann.org/) (ICANN).

Once one of these servers is contacted, it points to the authoritative name server of that organization.

### Authoritative Servers[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Authoritative-Servers)

Every organization with a public website or email server provides DNS records. These records have hostname to IP address mappings stored for that organization. These records can either be stored on a dedicated DNS server for that organization or they can pay for a service provider to store the records on their server.

This is the next link in the chain. If this server has the answer that we are looking for, the IP address that it has is finally returned to the client. However, this server may not have the sought after answer if the domain has a **sub-domain**. In that case, this server may point to a server that has records of the subdomain.

For instance, if the DNS record for [cs.stanford.edu](http://cs.stanford.edu/) is being looked for, a DNS server separate from ‘[stanford.edu](http://stanford.edu/)’ may hold records for the sub-domain ‘cs.’

### Local DNS Cache[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Local-DNS-Cache)

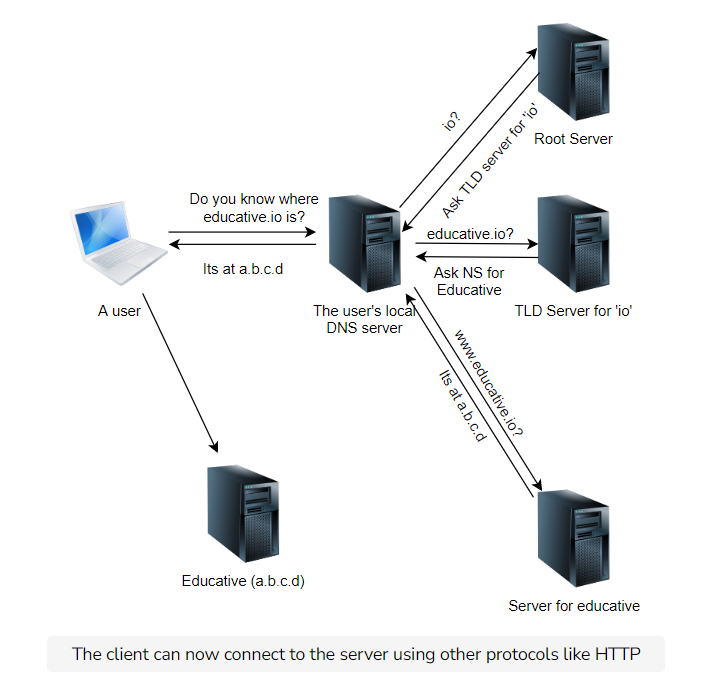
* DNS mappings are often also cached locally on the client end-system to avoid repetitive lookups and saves time for often visited websites.
* This is done via an entity called the **local resolver library**, which is part of the OS. The application makes an API call to this library. This library manages the local DNS cache.
* If the local resolver library does not have a cached answer for the client, it will contact the organization’s local DNS server.
* This local DNS server is typically configured on the client machine either using a protocol called **DHCP** or can be configured statically.
* So, if it’s configured manually, any local DNS server of the client’s choice can be chosen. A few open DNS servers are incredibly popular, such as the ones by [Google](https://en.wikipedia.org/wiki/Google_Public_DNS).

### Local DNS Servers[**#**](https://www.educative.io/module/lesson/network-fundamentals/RLYrwLNgoOz#Local-DNS-Servers)

There is one type of server that we ignored — the **local DNS Server**. Local DNS servers are usually the first point of contact after a client checks its local cache. These servers are generally hosted at the ISP and contain some mappings based on what websites users visit.

☠️ **Security Warning**: ISPs have a record of which IP address they assigned to which customers. Furthermore, their DNS server has the IP addresses of who contacts it and what hostname they were trying to resolve. So your **ISP technically has a record of all of the websites you visit. Yikes!** P.S. if this makes you uncomfortable, you can change your DNS server to any open public DNS server.

For an overview of how a query is resolved, have a look at the following slides.



# Exercise: Finding Name Servers

In this lesson, we will look at a few command-line tools to lookup domain name servers.

**We'll cover the following**

* [Finding The Authoritative Name Server Using host](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Finding-The-Authoritative-Name-Server-Using-host)
* [Checking What Your Local DNS Server Is](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Checking-What-Your-Local-DNS-Server-Is)
  + [Output](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Output)

## Finding The Authoritative Name Server Using host[**#**](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Finding-The-Authoritative-Name-Server-Using-host)

**host -t ns google.com**

host is a DNS lookup utility. It is normally used to map hostnames to IP addresses. However, with a combination of flags, it can be programmed to perform a myriad of DNS-related tasks.

The syntax of the command above, for instance, is

* host invokes the host command
* -t is the type flag. It is used to specify the type of the command. Check out [host’s manpage](https://linux.die.net/man/1/host) for a list of all the types available.
* ns specifies the type. It stands for the name server in this case.
* hostname.com can be any website of your choice.

We encourage you to experiment, explore, and get creative with the tool!

## Checking What Your Local DNS Server Is[**#**](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Checking-What-Your-Local-DNS-Server-Is)

To check the IP address of your local DNS server, run the following command on UNIX based machines. If you’re on a mobile machine, try [www.whatsmydnsserver.com](http://www.whatsmydnsserver.com/). There are a lot of instructions available for Windows machines online.

### Output[**#**](https://www.educative.io/module/lesson/network-fundamentals/B16QYQj5RQo#Output)

Here is what the output may look like

# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)  
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN  
nameserver 169.254.169.254  
search c.educative-exec-env.internal google.internal

You can safely ignore the first two lines since they are comments. On the third line, nameserver shows the IP address of the local DNS server. On the last line, search represents the default search domain that is used to resolve a query for a domain with no suffix (for example, www.facebook).

📝 **Note**: Educative’s code widgets run on a remote server. The code runs there, the output is then sent back to your machine and displayed. So, the DNS server shown here is local to the server that runs Educative’s code.

Now that we have a good idea of how DNS works, let’s study DNS records and messages in the next lesson.

# DNS: Records and Messages

Let's now get into what DNS records and messages look like.

**We'll cover the following**

* [Resource Records](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Resource-Records)
  + [Format](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Format)
  + [Types of resource records](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Types-of-resource-records)
* [DNS Messages](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#DNS-Messages)

## Resource Records[**#**](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Resource-Records)

The DNS distributed database consists of entities called **RR**s, or **Resource Records**.

### Format[**#**](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Format)

RRs contain some or all of the following values:

* **Name** of the domain.
* **Resource data (RDATA)** provides information appropriate for the type of resource record.
* **Type** of the resource record. We will discuss these shortly.
* **Time-to-live (TTL)** is how long the record should be cached by the client in seconds.
* **DNS Class**. There a many types of classes but we’re mainly concerned with IN which implies the ‘Internet’ class. That’s what all of our upcoming examples use so we won’t be discussing it again. Another common value for the DNS Class is CH for ‘CHAOS’. The CH class is mostly used for things like querying DNS server versions.

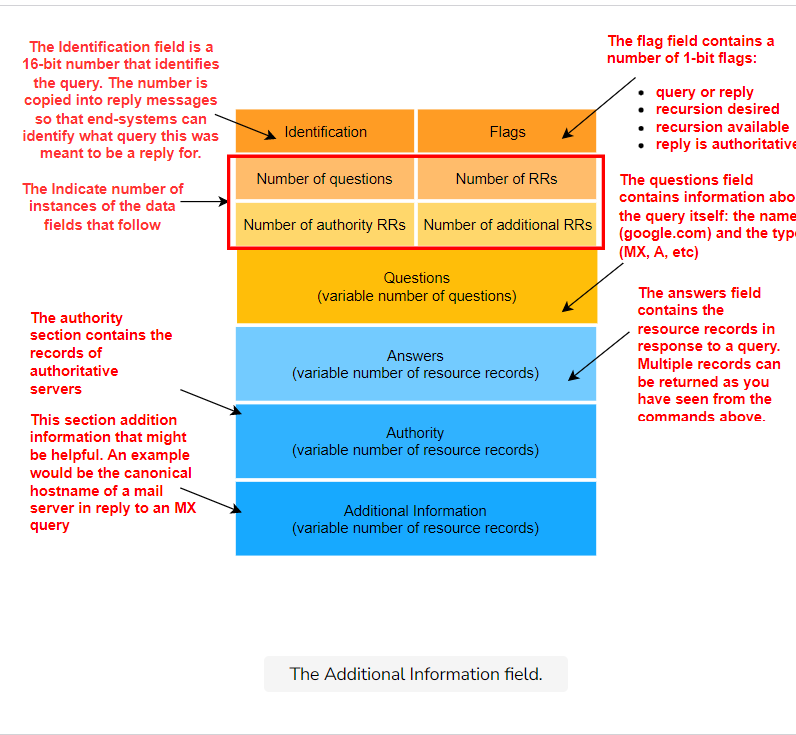
### Types of resource records[**#**](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#Types-of-resource-records)

* **Address** type or A addresses contain IPv4 address to hostname mappings. They consist of:
  + The name is the hostname in question.
  + The TTL in seconds.
  + The type which is A in this case.
  + The RDATA which in this case is the IP address of the domain.
  + **Example:** educative.io. 299 IN A 104.20.7.183 where educative.io is the name, 299 is the TTL in seconds, IN is the class, A is the type of the RR, and 104.20.7.183 is the RDATA.
* **Canonical name** or CNAME records are records of alias hostnames against actual hostnames. For example if, ibm.com is really servereast.backup2.com, then the latter is the canonical name of ibm.com.
  + The name is the alias name for the real or ‘canonical’ name of the server.
  + The RDATA is the canonical name of the server.
  + **Example:** bar.example.com. CNAME foo.example.com.
* **Mail Exchanger** or MX records are records of the server that accepts email on behalf of a certain domain. We have seen this one before!
  + The name is the name of the host.
  + The RDATA is the name of the mail server associated with the host.
  + **Example:** educative.io IN MX 10 aspmx2.googlemail.com.

These resource records are stored in text form in special files called **zone files**.

## DNS Messages[**#**](https://www.educative.io/module/lesson/network-fundamentals/qVG8KvK99Wy#DNS-Messages)

There are a few kinds of DNS messages, out of which the most common are **query** and **reply**, and both have the same format. Study the following slides for a detailed overview of a DNS message.



There are also **zone transfer request and response**. But, those are not used by common clients. Backup or secondary DNS servers use them for **zone transfers**, which are when zone files are copied from one server to another. This takes place over TCP.