Réseaux, Information, et Communications — INFO-F303

Notes: Networks, a Top-Down Approach

Elliot Huet — 25/10/2025

Contents

1. Introduction 3
2. Application Layer 4
   2.1. Principles of Network Apps 4
      Architectures 4
      Interface Between Processes and the Network 5
      Identifying Hosts and Processes 6
      Transfer Protocol Guarantees 6
      TCP 7
      UDP 7
      Application-Layer Protocols 7
   2.2. The Web and HTTP 8

1. Introduction

2. Application Layer

(EX) The World Wide Web, VoIP video conferencing (Teams, Whatsapp, Discord, ...), on demand streaming (YouTube, Netflix, Spotify, ...), social media (Instagram, Twitter, ...), ...

2.1. Principles of Network Apps

Network applications run on multiple end devices and consist of two different programs: the server, and the client. These apps do not run on network-core devices within the internet itself but on the internet's edge[1].

1. Even if you wanted to, you could not write these apps to work on these devices as they run at the network level and below which is way below the application level.

Architectures

There are two main ways to build a network application:

— **Client/Server Architecture**. The app is separated into two distinct programs: the **client** running on end user machines, and the **server** running on dedicated machines usually housed inside of datacenters. It has the following properties:

1. the server is always on;
2. the server has a fixed, well known IP address;
3. clients do not communicate with each other directly, but going through the server instead.

(EX) Web, FTP, e-mail, Telnet, ...

— **Peer-to-Peer Architecture** (**P2P**). Here there are no dedicated machines running a server, instead every client can also become a server and provide service to its peers. The peers are not owned by the service provider but by end users.

One of the main advantages of the P2P architecture is **self-scalability**, each connected host puts strain on the network, but as it also creates extra capacity by contributing to the service.

(EX) BitTorrent, ...

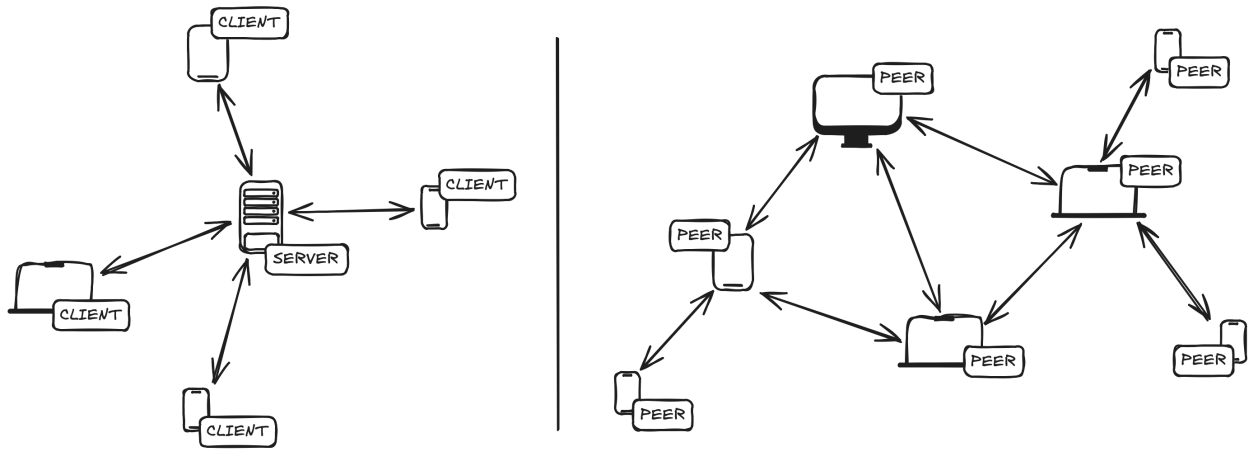

Figure 1: Server/Client architecture on the left, P2P architecture on the right.

Both architectures have their advantages and disadvantages which can be found in the table below.

Criterion	Server/Client	P2P
Cost	↑	↓
Scalability	↓	↑
Security	↑	↓
Performance	↑	↓
Reliability	↑	↓

Table 1: Advantages and disadvantages of Server/Client and P2P architectures.

Figure 1 makes the slight mistake of displaying the various nodes as connecting to each other, instead **processes** running on those devices connect to each other via the network.

In both architectures, in the context of a communication session between two processes, it is possible to label one as the **client** and one as the **server**. The process that initiates the communication is called the client, while the process waiting for a communication session is called the server.

Interface Between Processes and the Network

Processes that want to communicate with each other can use an **API** (**Application Programming Interface**) called **socket** to send **messages** to each other. Using it is quite simple, to the application programmer a socket is a black box where they can push messages and they magically appear at the other end intact. Two processes communicating via a socket is represented on Figure 2.

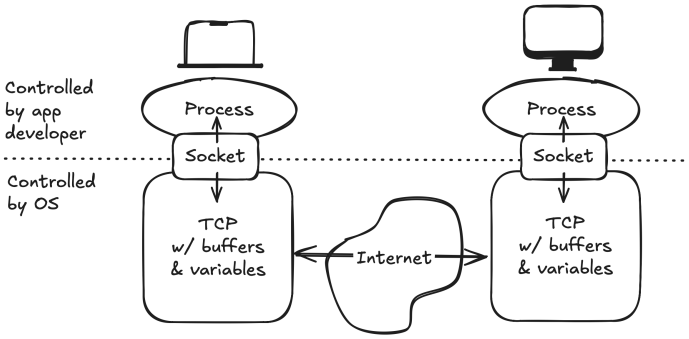

Figure 2: Processes communicating via a socket and the underlying transfer protocol.

Identifying Hosts and Processes

On the Internet, each host is uniquely identified by an **IP Address**[2]. In addition to the IP address, each process on that host is identified by a **port** number. Any process wanting to establish a connection with another process must know its host's IP address and its port.

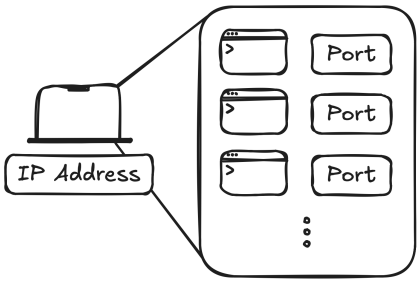

Figure 3: Processes running on a host and their identifier types.

Popular applications have been assigned specific port numbers to make things easier. Some of these are listed on Table 2.

App	Usual Port
Web Server	80
Mail Server (over SMTP)	25
⋮	⋮

Table 2: Usual ports for various internet applications.

Transfer Protocol Guarantees

Transfer protocols can offer (or not) various guarantees. There are four main ones:

1. **Reliable Data Transfer**. Guarantees that the data makes its way from one process to another without error and in the correct order. If the protocol doesn't provide this guarantee, the data may never arrive at its destination; apps that can handle this are called **loss-tolerant applications**.
2. **Throughput**. Guarantees a throughput of at least r bits/second. Applications with throughput requirements are called **bandwidth-sensitive applications**.

   (EX) VoIP conference calls, video streaming, ...

   Applications that can make use of as much or as little throughput are called **elastic applications**.

   (EX) Email, file transfers, Web transfers, ...
3. **Timing**. Guarantees that the data makes its way to the destination within n milliseconds. This would be useful in **real-time applications** where latency is critical.

   (EX) Multiplayer video games, conference calls, ...
4. **Security**. Makes sure that the data being sent is encrypted within the protocol itself, meaning that the application doesn't have to implement that functionality itself. Other security guarantees include: data integrity, end-point authentication, ...

Guarantee	TCP	UDP
Reliable	✓	×
Throughput	×	×
Timing	×	×
Security	×/(✓)	×

Table 3: Guarantees upheld by the two main transfer protocols. (The security guarantee can be upheld while using TCP if you enhance it with TLS.)

Today's internet makes two transport protocols available to network applications: **TCP** and **UDP**. These protocols cannot uphold all of these guarantees: throughput and timing cannot be guaranteed[3]. The guarantees upheld by each protocol can be found at Table 3.

2. IP adresses come in two main variants: IPv4 and IPv6. IPv4 is the universally accepted standard, but as it's encoded on a 32-bit integer, it can only have around 4.3 billion unique identifiers. We have more than 4.3 billion internet capable devices on Earth today which is a major issue. Switching to IPv6 would solve that issue as it can encode 2^{128} devices.

3. Bandwidth-sensitive and real-time apps have been built to cope with the variation in throughput and timing. This can lead to sub-optimal performance but it's good enough in most cases.

TCP

The TCP service model provides a connection-oriented, reliable transfer service. TCP connections go through the following steps:

1. **Handshaking**. The client and the server exchange transport-layer control information before the application-level information begins to flow. This prepares both the client and the server to the onslaught of packets. After this handshaking step, a **TCP connection** is established between the client and the server.
2. **Communcation**. At this point the client and the server can communicate as much as they wish in both directions. They also have full guarantee that the data exchanged is correct and in the right order.
3. **Graceful Teardown**. Once they are done communicating, the connection is torn down.

TCP also includes a **congestion-control mechanism** that works for the general welfare of the internet. It can throttle sending processes when the network is congested to not prevent other connections from going through.

UDP

UDP is connectionless, there is no handshaking before communication. It is also an unreliable data transfer service; packets can get lost or arrive in the wrong order.

No congestion-control mechanism is included, so UDP can dump data into the network layer at any rate it pleases[4].

4. Most firewalls block UDP connections by default, as it is usually unsolicited network traffic.

Application	Application-Layer Protocol	Underlying Transport Protocol
Email	SMTP	TCP
Remote Terminal Access	Telnet	TCP
Web	HTTP	TCP
File Transfer	FTP	TCP
Streaming Multimedia	HTTP, DASH	TCP
Internet Telephony	SIP, RTP, proprietary...	UDP/TCP

Table 4: Internet applications, their application-layer protocols, and their underlying transport protocols.

Application-Layer Protocols

(EX) HTTP, SMTP, Telnet, DASH, SIP, Skype, ...

Application-layer protocols defines how an application's processes running on different systems pass messages to each other. Network apps use one or more to communicate (see Table 4). They define:

1. **Types of messages exchanged**.
   (EX) request messages, response messages, ...
2. **Syntax of the various messages**.
   (EX) fields in the message, how the fields are separated, ...
3. **Semantics of the fields**. i.e. the meaning of the information in the fields.
4. **Timing rules**. When and how does a process send and/or respond to a message.

2.2. The Web and HTTP