Internet Architecture Overview

Instructor: Venkat Arun

Fall 2024

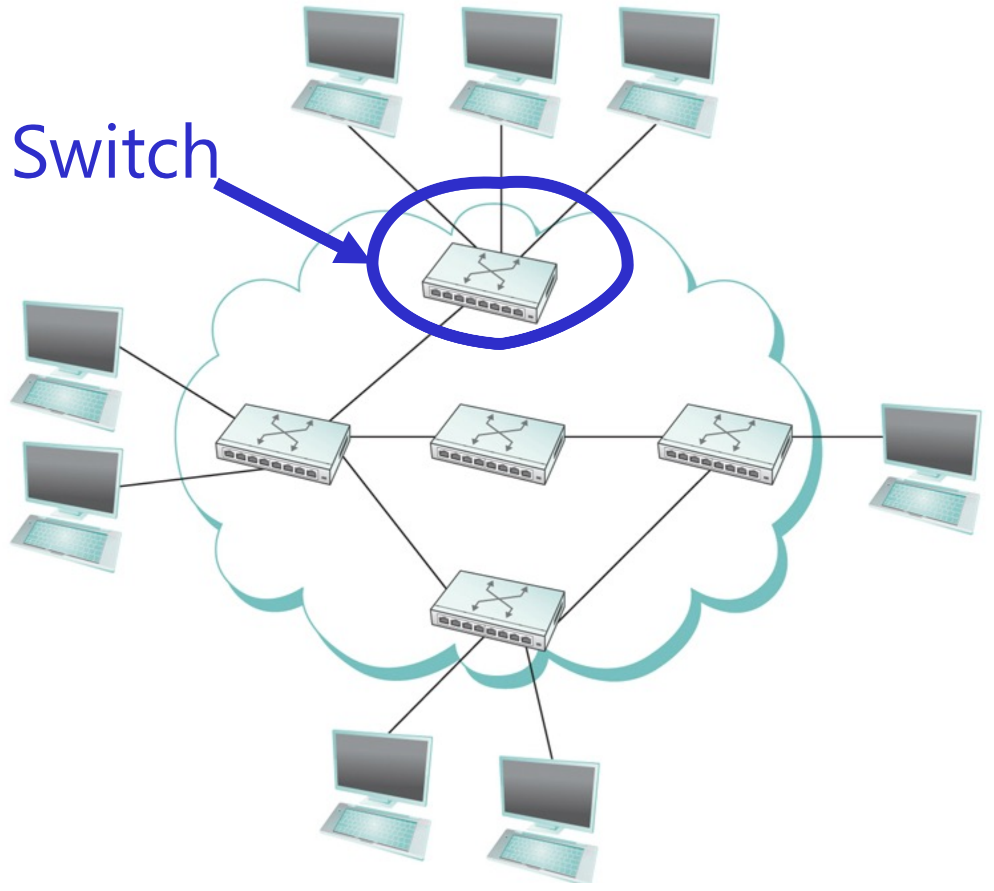
Some graphics are borrowed from the Peterson & Davie book (referred to as P&D)

Logistics

- Assignment 1 has been posted
- Canvas and Ed Discussion are published

- Today, we shall take a high-level overview

Recap: How do we make the internet scalable?



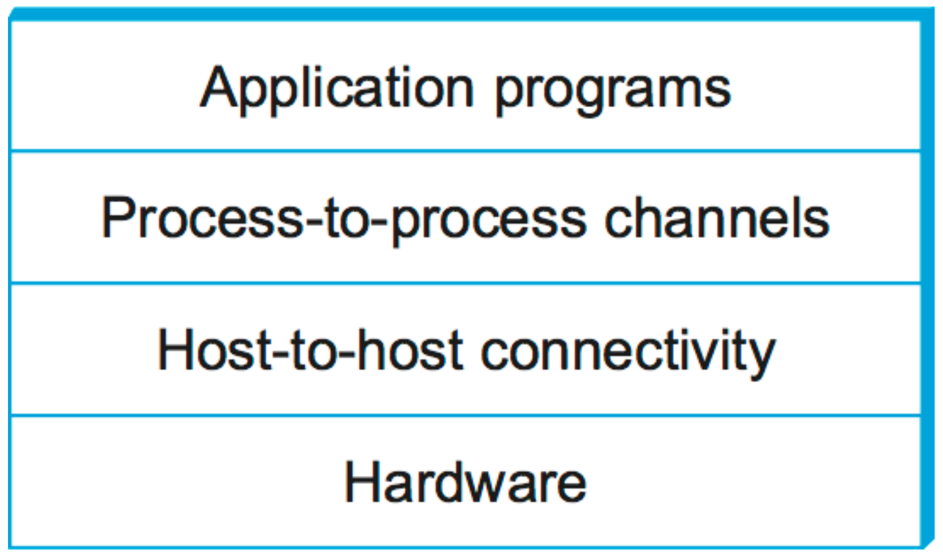
- Divide data into small chunks called packets
- End hosts create packets containing the destination address
- The network tries its “best” to get the packet to the destination
- Routers in the network store and forward packets to the (hopefully) correct next hop

How do we make it adaptable?

Principle 1: Precisely specify interfaces between different components, often arranged as layers

- Everyone can have their own implementation and yet interoperate with each other
- When possible, allow for flexibility within a component without having to change the interface (very tricky to get right)

Layering abstractions (P&D chapter 1.3)

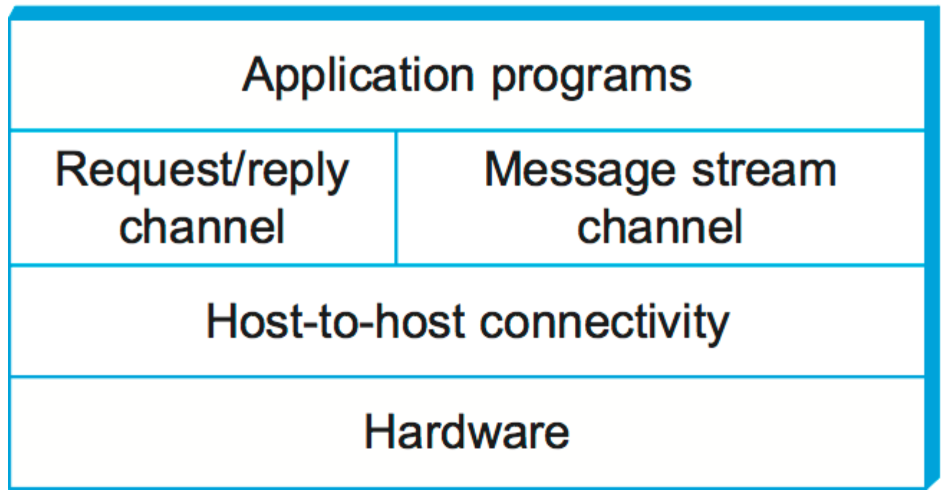


How do we make it adaptable?

Principle 1: Precisely specify interfaces between different components, often arranged as layers

- Everyone can have their own implementation and yet interoperate with each other
- When possible, allow for flexibility within a component without having to change the interface (very tricky to get right)

Layering abstractions (P&D chapter 1.3)



One layer can have many abstractions

How do we make it adaptable?

Principle 2: Move all intelligence to the end hosts when possible

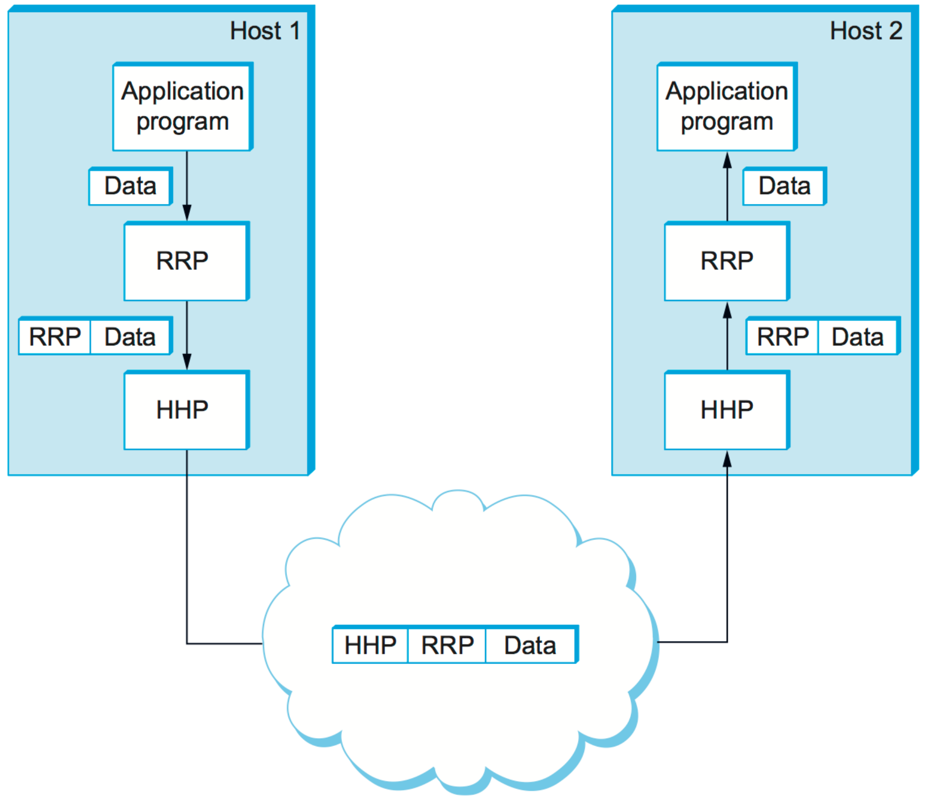
- Popularly called the “end-to-end” principle
- This way, the network is only responsible for transporting packets from one machine to the other.
- Further, we only expect it to put in its “best effort”
- Everything else is handled in the end hosts: reliability, security/encryption, assigning meaning to the bits, and application specific logic
- End hosts are easier to change. It is still difficult to get consensus from everybody though

Examples where adaptation has succeeded because of this

- Applications like zoom and slack can unilaterally change their interfaces because they are a single administrative entity
- Email, in contrast, cannot evolve as rapidly since it is run by many entities through a common protocol. However, it is much more universal
- When people realize a cryptographic technique is broken, individual software developers slowly start phasing it out (e.g. web browsers and web servers). For example, people are trying to stop using encryption mechanisms that can be broken by quantum computers
- If a company is large enough, it can unilaterally implement a new protocol. For example, Google implemented a new transport protocol called QUIC because they control a lot of browsers and servers. Now others also use it.

Implementing layers using encapsulation

Every layer adds its own header to the data. On the other end, every layer removes its header

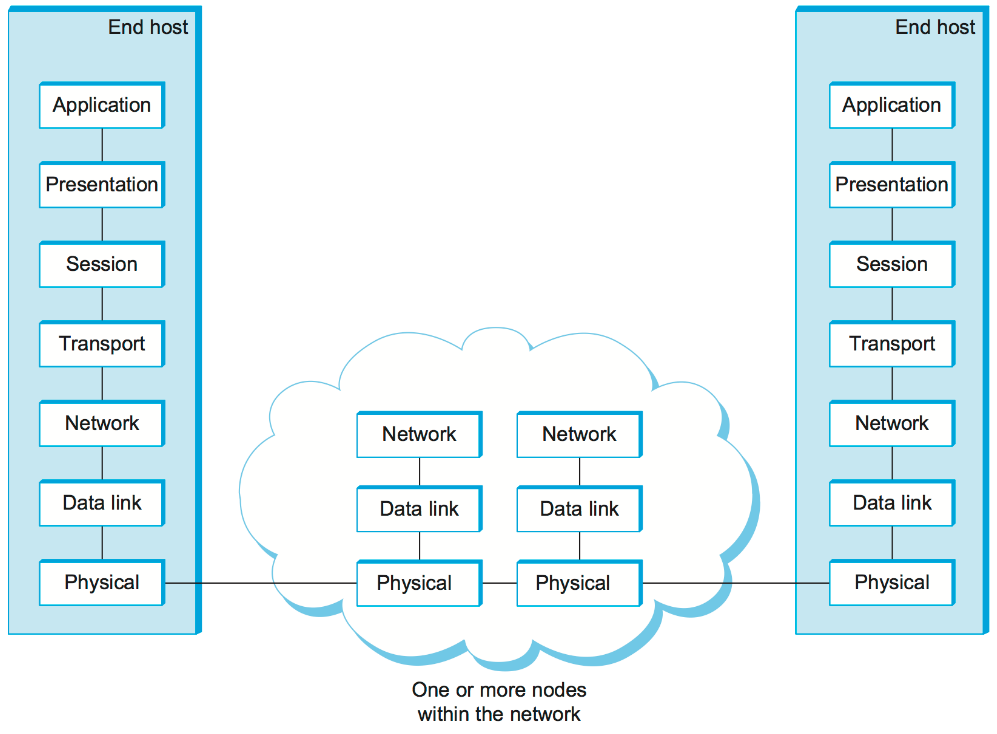


Implementing layers using encapsulation

Every layer adds its own header to the data. On the other end, every layer removes its header

Not all nodes will implement all the layers. Usually, higher layers are only implemented at the end hosts

The picture on the right is the "OSI" model. Nobody uses "Presentation" and "Session" layers anymore.



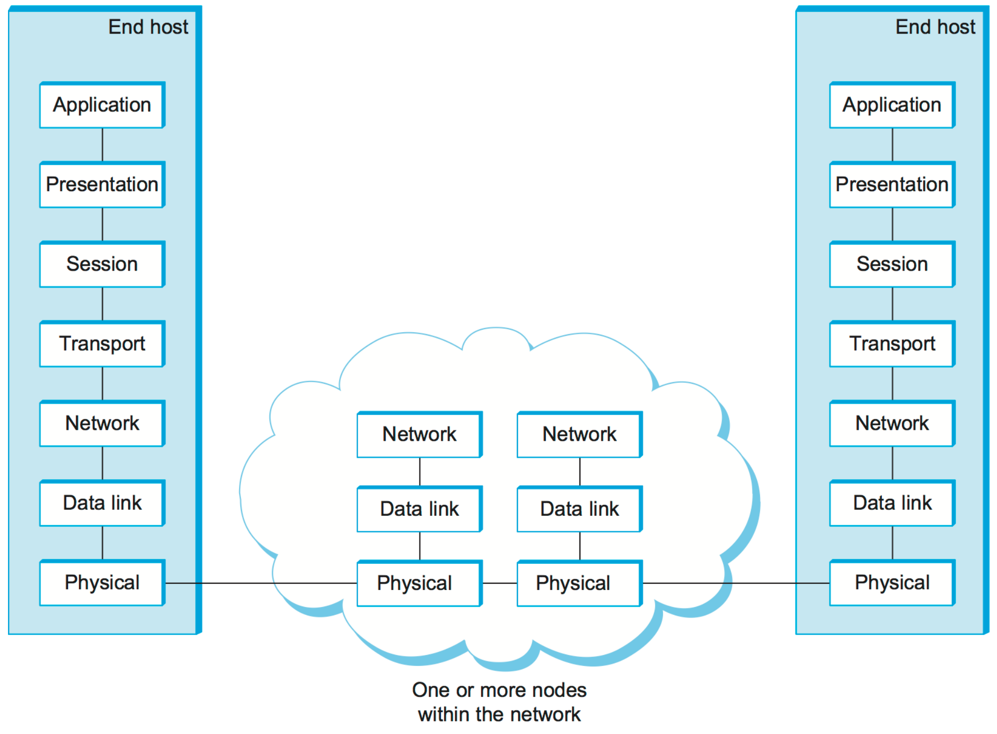
Implementing layers using encapsulation

Every layer adds its own header to the data. On the other end, every layer removes its header

Not all nodes will implement all the layers. Usually, higher layers are only implemented at the end hosts

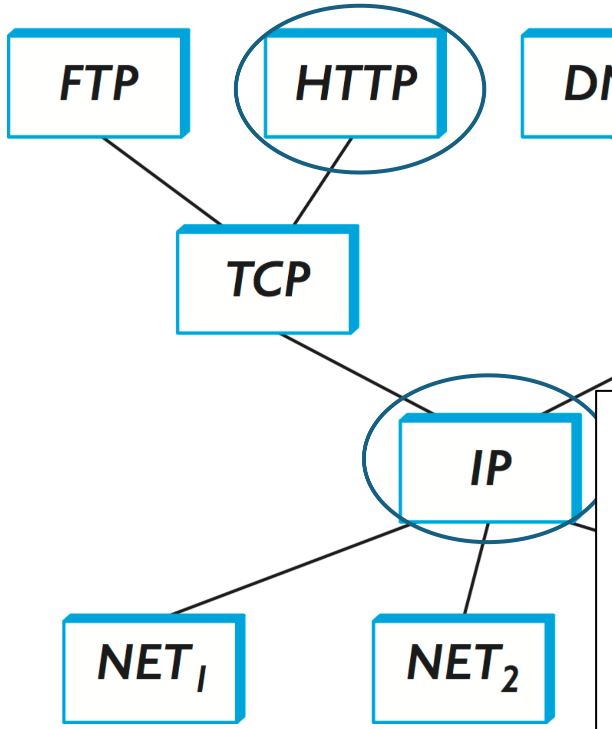
The picture on the right is the "OSI" model. Nobody uses "Presentation" and "Session" layers anymore.

Layering is not followed strictly



Todays' protocol stack

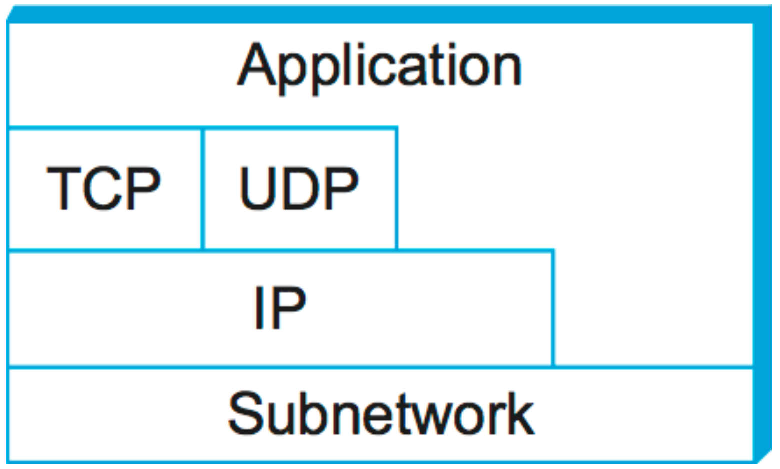
HTTP is slowly becoming another narrow waist



Internet Protocol: The "narrow waist" of the internet

One of the few things I will say in this course that is (almost) always true

Layering is not followed strictly

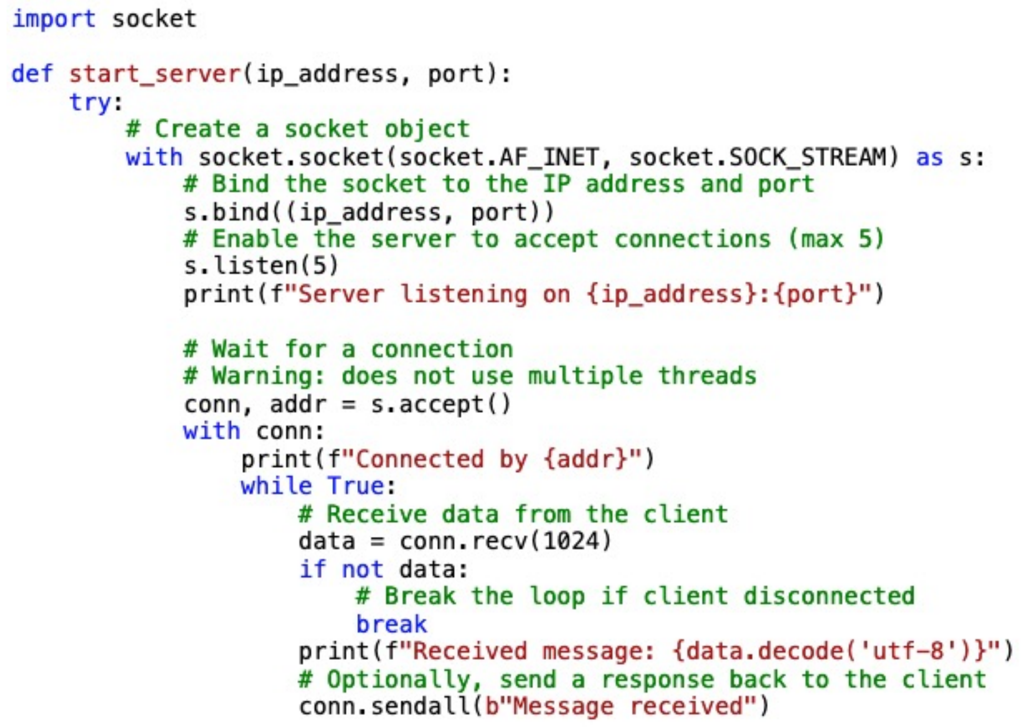


How can you use the internet? Sockets

C API is explained in P&D chapter 1.4

The client side is used in the assignment 1. The server side will be used in assignment 3

Server

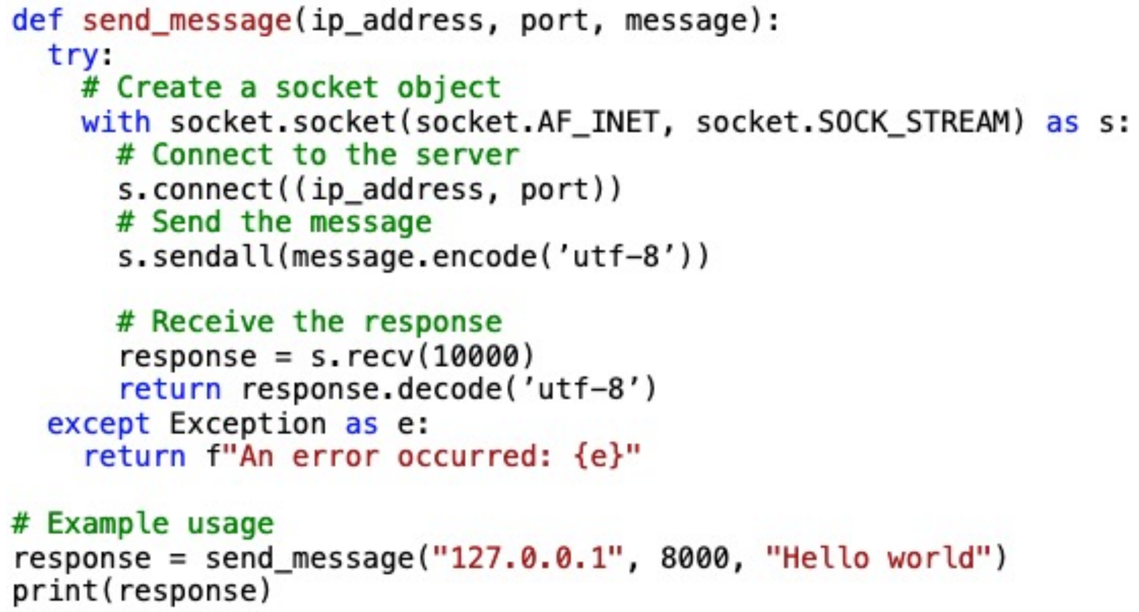


```python
import socket

def start_server(ip_address, port):
    try:
        # Create a socket object
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            # Bind the socket to the IP address and port
            s.bind((ip_address, port))
            # Enable the server to accept connections (max 5)
            s.listen(5)
            print(f"Server listening on {ip_address}:{port}")

            # Wait for a connection
            # Warning: does not use multiple threads
            conn, addr = s.accept()
            with conn:
                print(f"Connected by {addr}")
                while True:
                    # Receive data from the client
                    data = conn.recv(1024)
                    if not data:
                        # Break the loop if client disconnected
                        break
                    print(f"Received message: {data.decode('utf-8')}")
                    # Optionally, send a response back to the client
                    conn.sendall(b"Message received")
```

Client



```python
def send_message(ip_address, port, message):
  try:
    # Create a socket object
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
      # Connect to the server
      s.connect((ip_address, port))
      # Send the message
      s.sendall(message.encode('utf-8'))

      # Receive the response
      response = s.recv(10000)
      return response.decode('utf-8')
  except Exception as e:
    return f"An error occurred: {e}"

# Example usage
response = send_message("127.0.0.1", 8000, "Hello world")
print(response)
```