UNIVERSITY OF DERBY

5CM510 – Networks System Development
Lecture 2: Concepts

derby.ac.uk

Sensitivity: Internal

Instructor Contact & Resources

Ioannis Tsioulis

i.tsioulis@mc-class.gr

github.com/giannis00

Overview

- Some basic concepts about the Internet and internets
- Overview of Internet protocols
    - UDP
    - TCP
- IP addresses and addressing
- Routing concepts
- Client/server concepts
- Python network programming

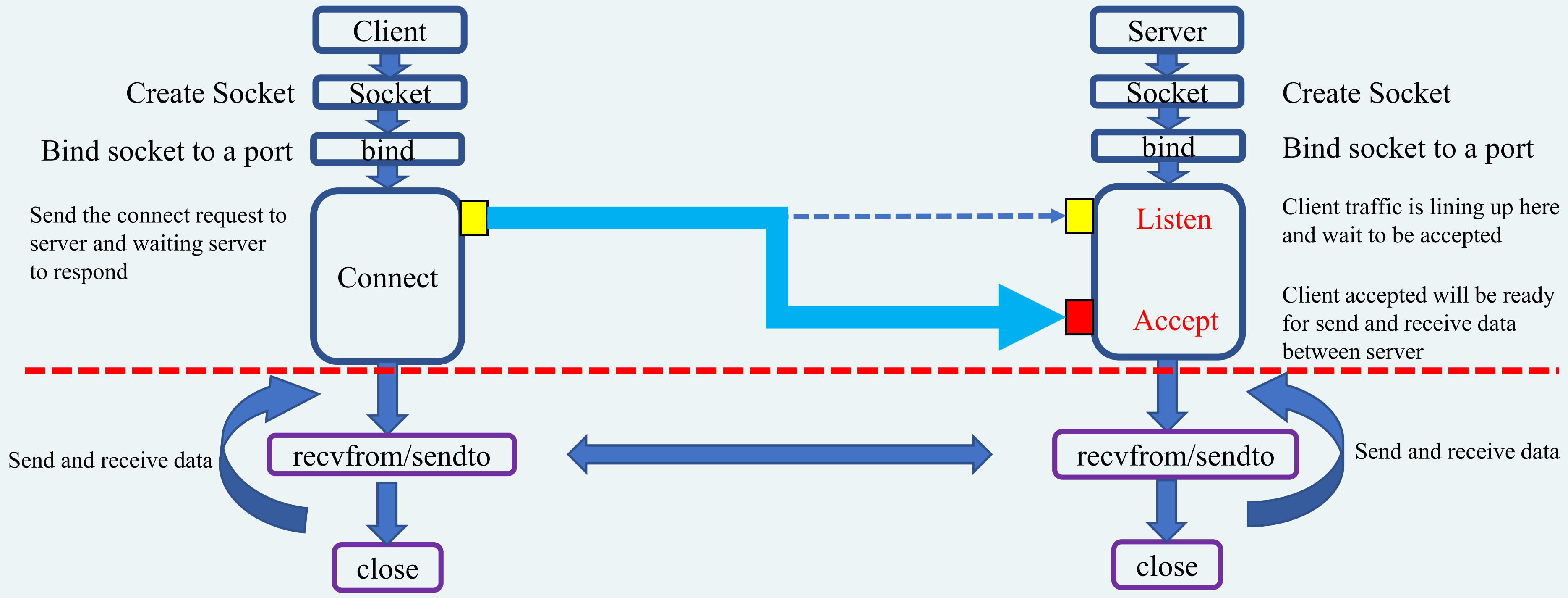
Part 1

1. Client/server concepts
2. Circuit-switching versus Packet switching
3. Basic concepts about the Internet and internets
4. Routing concepts (from a network programmer’s perspective)
5. Overview of Internet protocols: TCP, UDP

Client/Server concepts

- Server opens a specific port
  - The one associated with its service
  - Then just waits for requests
  - Server is the passive opener
- Clients get ephemeral ports
  - Guaranteed unique, 1024 or greater
  - Uses them to communicate with server
  - Client is the active opener

Client/Server concepts

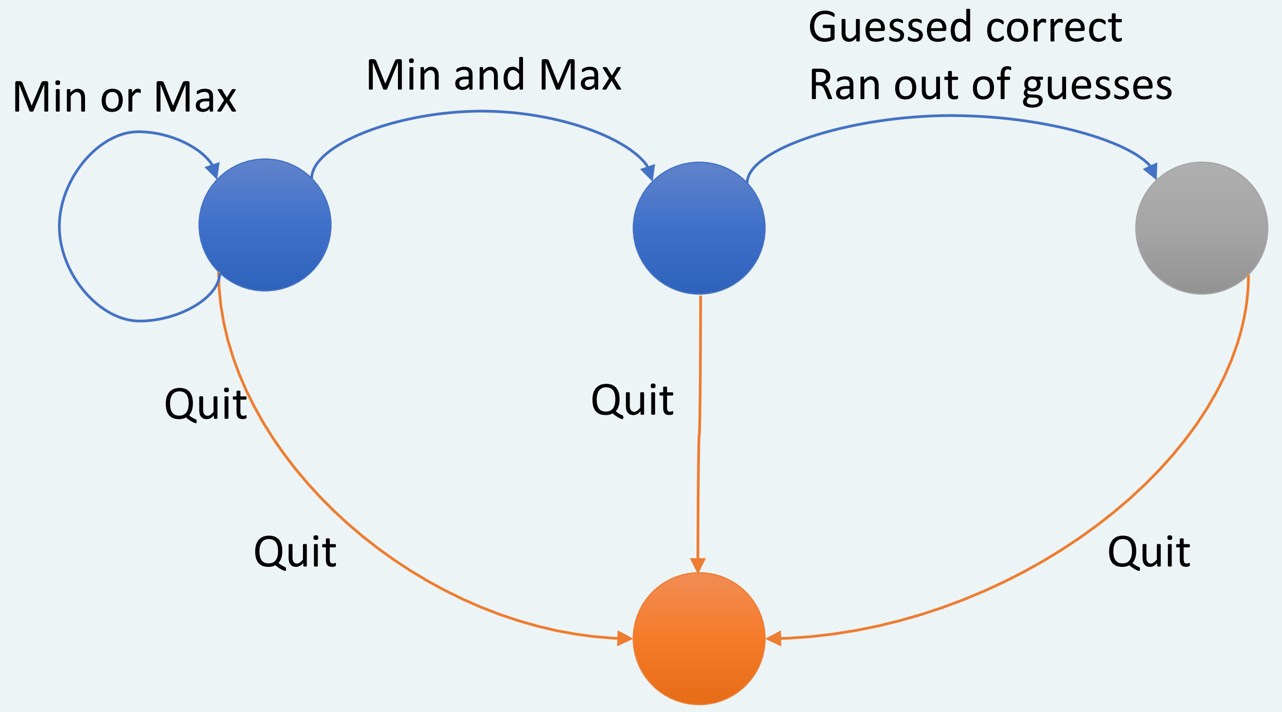


An example of data flow between client/server

Number guess on-line application:

Client	Server
←'Welcome …..'	
'Your Guess'→	
←'Assessment'	
'Your Guess'→	
←'Assessment'	
………	
←'Winner!!!'	

Adding some form of state control:

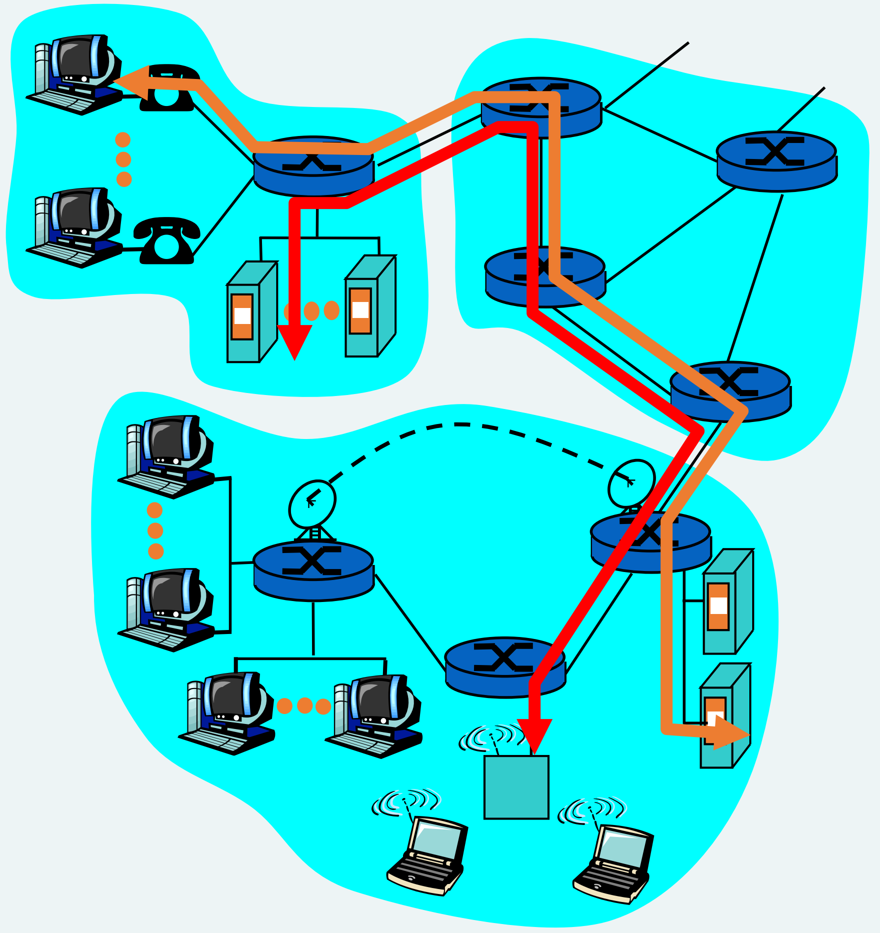


Transmission Technology

- Switched networks
  - Data is transferred from source to destination though a series of intermediate nodes
  - The nodes provide the switching facility that will move data to it’s destination
- circuit-switched network
  - A dedicated physical circuit is first established between the source and destination nodes before data transmission takes place
- packet-switched network
  - Messages are first partitioned into smaller units called packets, which are then sent to the destination node via intermediate switches
- Broadcast networks
  - There are no intermediate switching nodes. The network share a single communication medium or channel
  - Satellite Network

Circuit Switching

- End-end resources reserved for "call"
    - link bandwidth, switch capacity
- Dedicated resources
    - No sharing
    - Circuit-like (guaranteed) performance
    - Call setup required

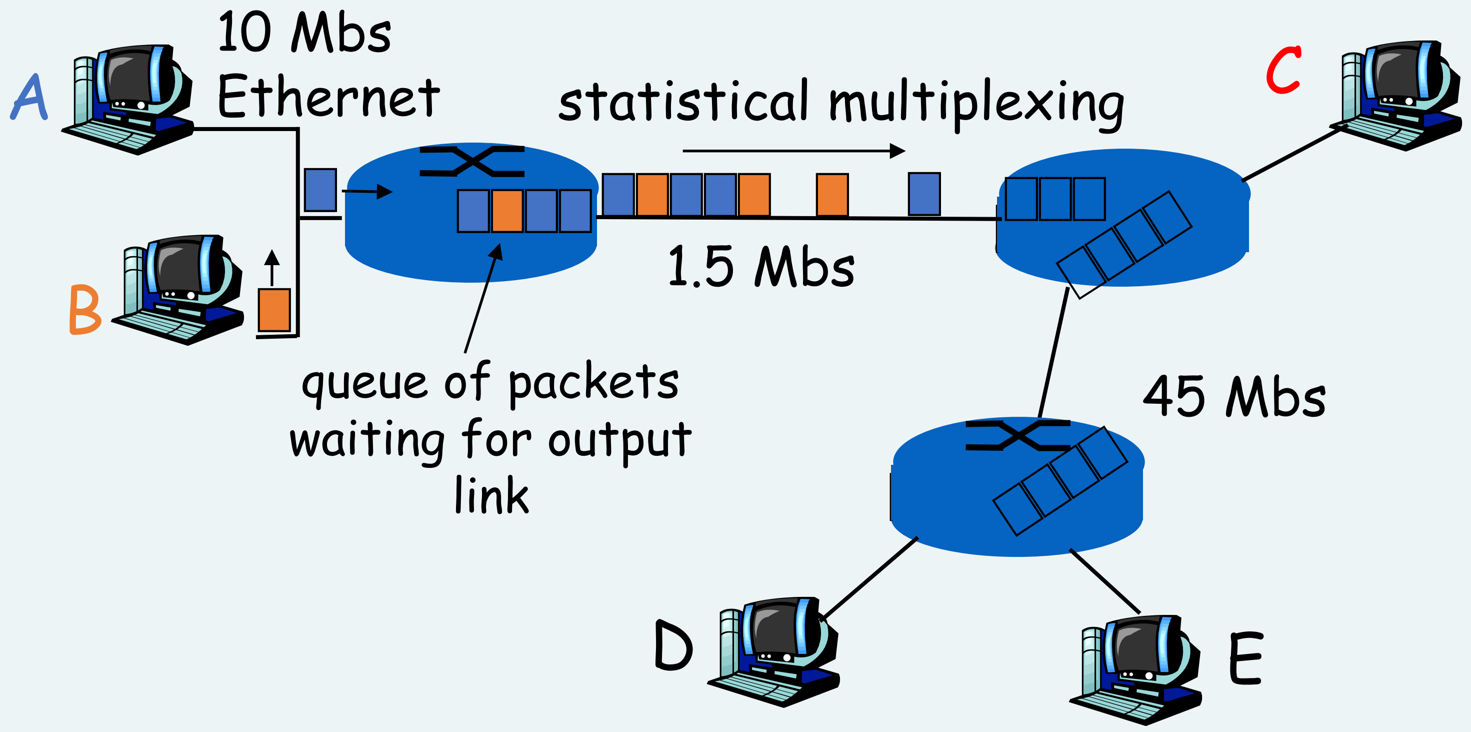


Network Core:

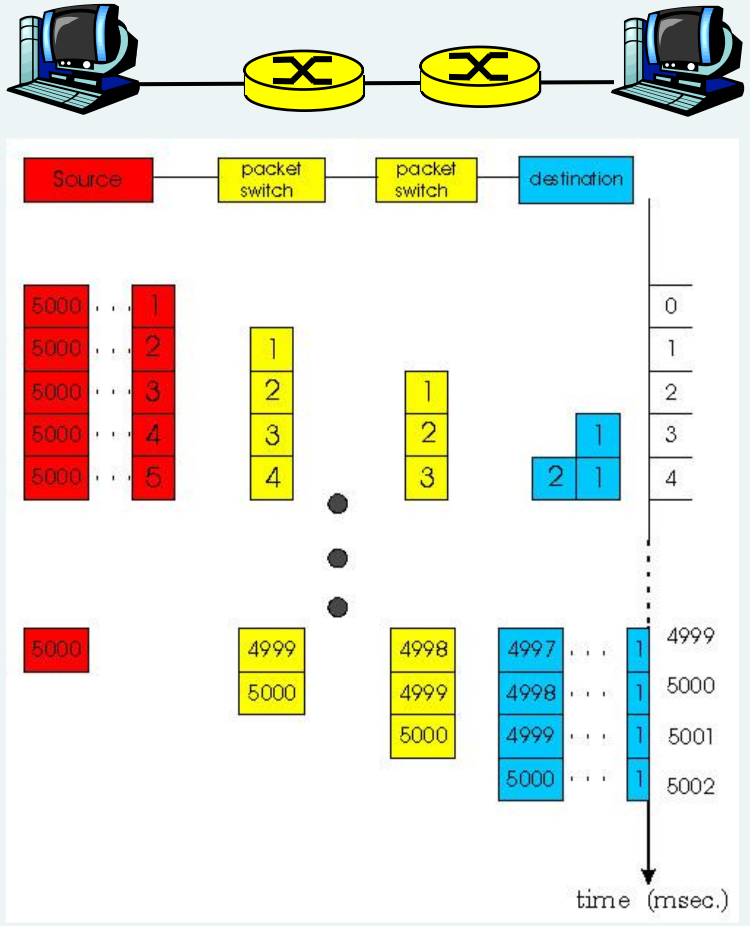
- Circuit Switching
  - Network resources (e.g., bandwidth) divided into “pieces”
  - Pieces allocated to calls
  - Resource piece idle if not used by owning call (no sharing)
  - Dividing link bandwidth into “pieces”
    - Frequency division
    - Time division

- Packet Switching
  - Each packet uses full link bandwidth
  - Resources used as needed,
  - Resource contention:
    - aggregate resource demand can exceed amount available
    - congestion: packets queue, wait for link use
    - store and forward: packets move one hop at a time
      - transmit over link
      - wait turn at next link

Packet Switching: Statistical Multiplexing



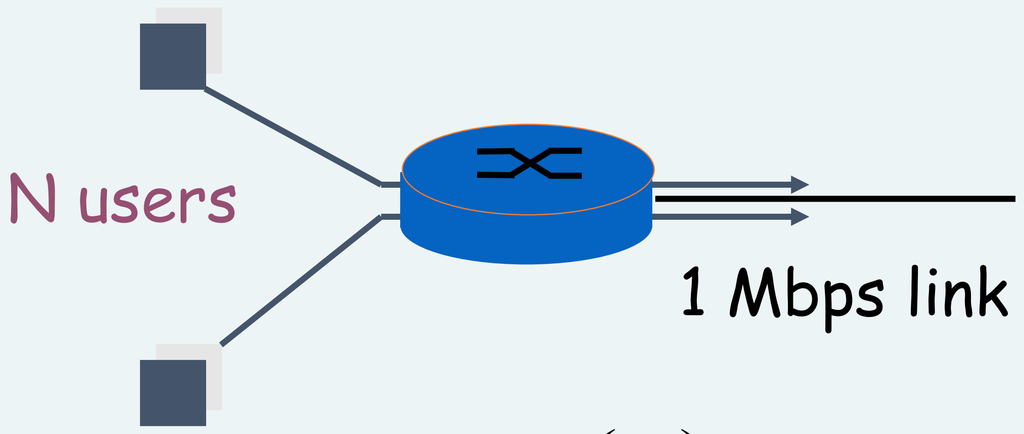
Packet Switching



- Store and Forward Behaviour:
  - break message into smaller chunks: “packets”
  - Store-and-forward: switch waits until chunk has completely arrived, then forwards/routes

Why use packet switching?

- Packet switching allows more users to use network!
  - E.g., 1 Mbit link
  - each user:
    - 100Kbps when "active"
    - active 10% of time
  - circuit-switching:
    - 10 users
  - packet switching:
    - with 35 users, probability > 10 active less than .0004



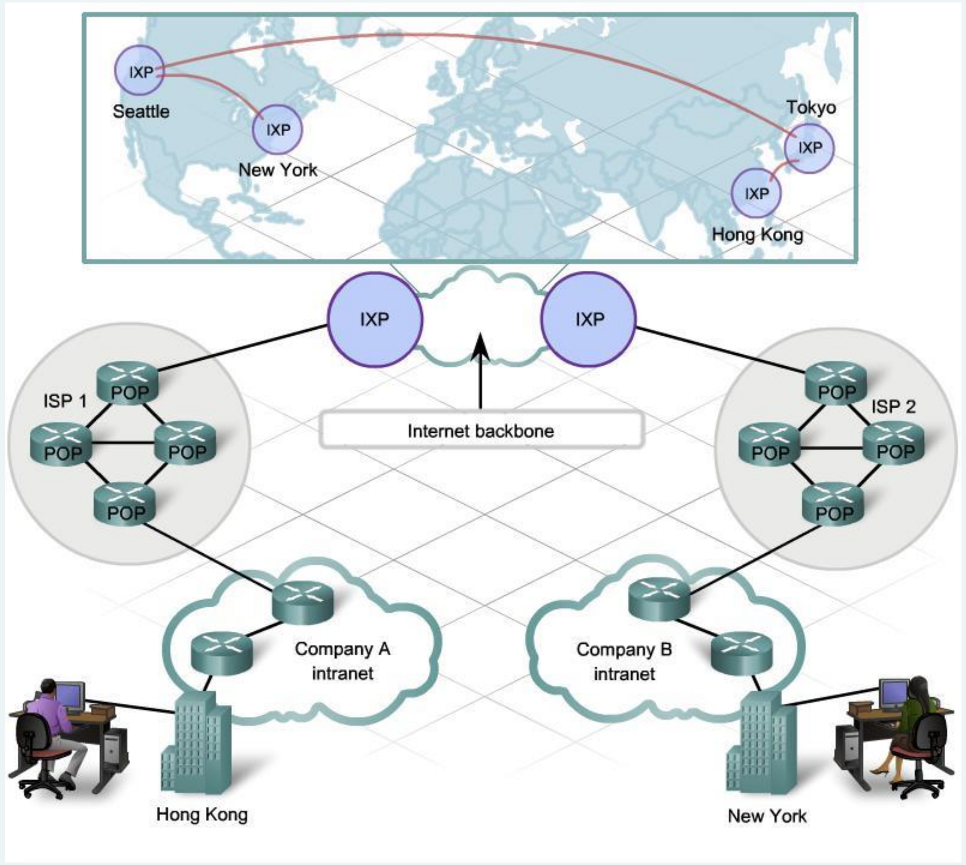
$$1-\sum_{i=0}^{10}\binom{35}{i}p^i(1-p)^{35-i}$$

Is Packet Switching the Winner?

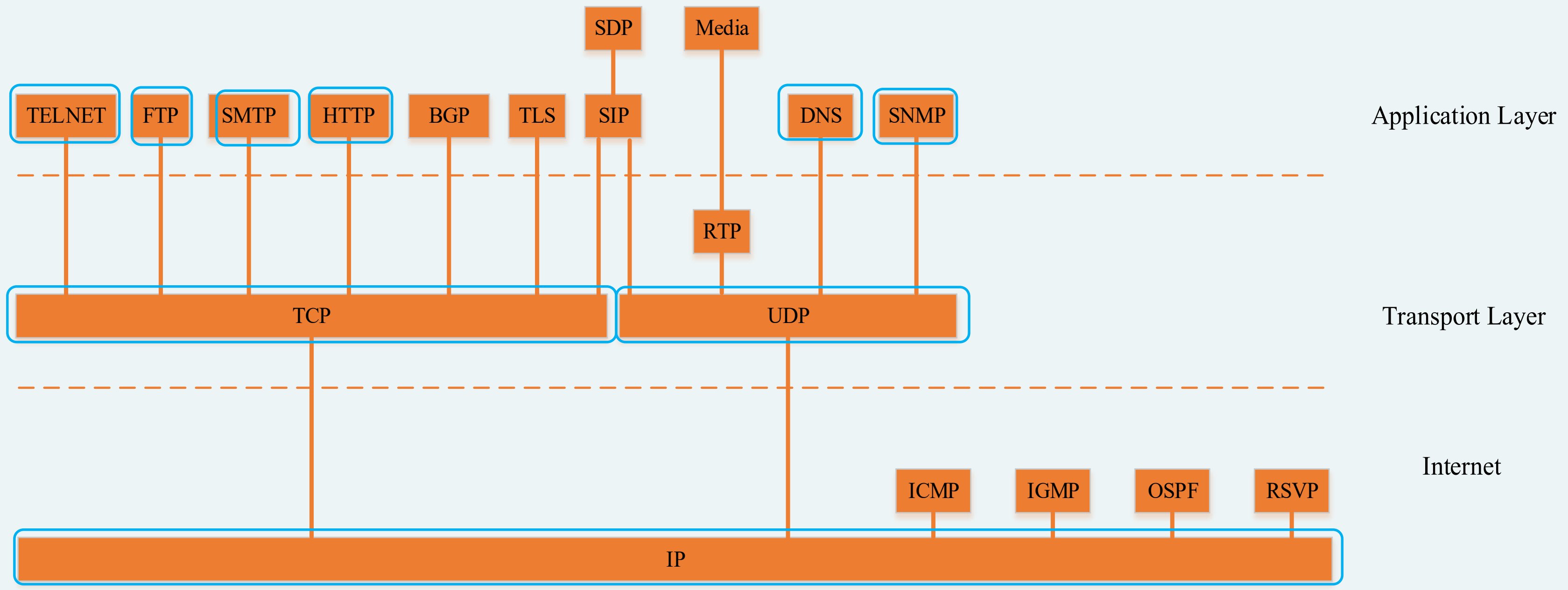
- Great for bursty data
- resource sharing
- no call setup
- Excessive congestion can cause:
  - Packet delay
  - Packet loss
- protocols needed for reliable data transfer, congestion control
- Q: How to provide circuit-like behavior?
  - Bandwidth guarantees needed for audio/video applications

Some basics about Internet

- The term internet is short for internetworking.
  - Interconnection of networks with different network access mechanisms, addressing, routing techniques, etc.
  - Essentially: a network of networks that is transparent to the end user
- An internet
  - Collection of communications networks interconnected by switches and/or routers
- The Internet – note the uppercase **I**
  - The global collection of individual machines and networks
  - Operates over IP (mostly, kindof, ish?)
- Internet Protocol (IP)
  - Most widely used internetworking protocol
  - Foundation of all internet-based applications



Protocols of TCP/IP Protocol Suite



Network Access

BGP = Border Gateway Protocol	RSVP = Resource ReserVation Protocol
DNS = Domain Name System	RTP = Real-time Transport Protocol
FTP = File Transfer Protocol	SDP = Session Description Protocol
HTTP = Hypertext Transfer Protocol	SIP = Session Initiation Protocol
ICMP =Internet Control Message Protocol	SMTP = Simple Mail Transfer Protocol
IGMP = Internet Group Management Protocol	SNMP = Simple Network Management Protocol
IP = Internet Protocol	TCP = Transmission Control Protocol
MIME = Multi-purpose Internet Mail Extension	TLS = Transport Layer Security
OSPF = Open Shortest Path First	UDP = User Datagram Protocol

Internet Protocol (IP)

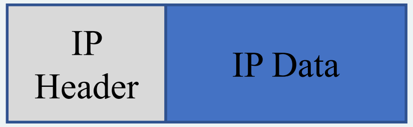
- Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries.
- Its routing function enables internetworking, and essentially establishes the Internet
- IP provides the relaying datagrams across network boundaries
- Each packet treated separately

Well not really.

- IP is the visible communications protocol for the Internet, there are many others which are hidden from you
- IP defines the visible and routable parts of the Internet – not the full thing
- Have a look at BGP and ATM… the amount of the Internet which is hidden will likely surprise you
- We can use this as well though – consider thing like onion routing!

Internet Protocol (IP) (Cont'd)

- **Functions:**
  - Datagram construction
  - Header:
    - Tagged with the source & destination IP address, and other meta-data
  - Payload:
    - The data that is transported



Internet Protocol (IP) (Cont'd)

- **Functions:**
  - IP addressing and routing
  - IP address is a numerical label assigned to each device and associated parameters to host interfaces
  - IP routing is performed by all layer 3 devices, but most importantly by routers, which transport packet across network boundaries and transition packets between different network types.

Internet Protocol (IP) (Cont'd)

- **Advantages**
  - Flexible and robust
    - Routes around damage or congestion dynamically
    - Packets can take different routes
    - Can update in near real-time
  - No unnecessary overhead for connection setup
  - Can work with different network types
    - Few demands on underlying network in terms of structure or support

- **Disadvantages**
  - Unreliable
    - Packet delivery is not guaranteed
    - Packet order is not guaranteed
    - Packets can take multiple routes through the network
    - Reliability is responsibility of next layer up (Transport Layer)
  - No separation of services, interfaces and protocols (left to us)
  - Not optimised for small networks (designed for global scale routing)

Internet Protocol (IP) (Cont’d)

Internet Protocol version 4 (IPv4): the fourth version in the development of the IP Internet, and routes most traffic on the internet. (defined in 1980)

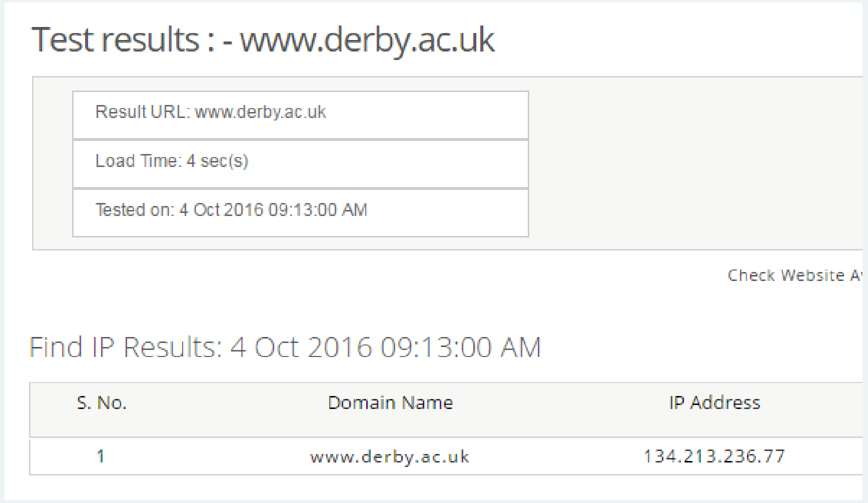
- **Main features**
  - IPv4 uses 32-bit (four-byte) addresses
  - Limits the address space to 4294967296 (2^{32}) addresses.
  - It is not enough for everyone to use their devices with unique IP address

Internet Protocol version 6 (IPv6): the most recent version of the IP, the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. (defined in 1998)

- **Main features**
  - 128-bit address, which gives approximately 3.4×10^{38} (2^{128}).
  - Improvement of network management and routing efficiency (as a large subnet space and hierarchical route aggregation algorithm is used)

IPv4 Addresses

- The IPv4 assigns a 4-byte address to every computer connected to the network.
- Such addresses are usually written as four decimal numbers, separated by periods and range from 0 to 255.
    - 134.213.236.77
- Purely numeric addresses can be difficult for humans to remember
- We generally use hostnames rather than IP addresses.
- The particular network service, called the Domain Name service (DNS)



Test results : - www.derby.ac.uk

Result URL: www.derby.ac.uk
Load Time: 4 sec(s)
Tested on: 4 Oct 2016 09:13:00 AM

Find IP Results: 4 Oct 2016 09:13:00 AM

S. No.	Domain Name	IP Address
1	www.derby.ac.uk	134.213.236.77

(www.derby.ac.uk)

Exercise: Turn a Hostname into an IP Address

How would you turn a hostname into an IP address in Python?

```
hostname = socket.gethostbyaddr("www.derby.ac.uk");
```

IPv6 Addresses

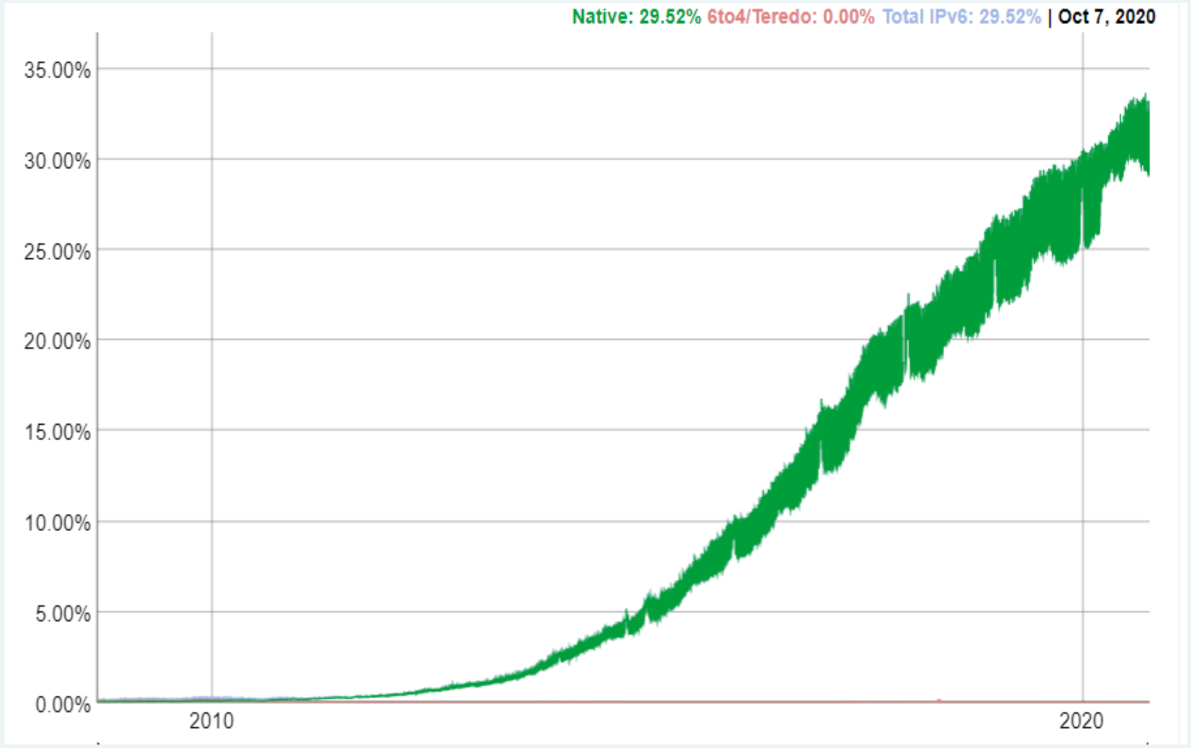
IPv6 is still slowly being deployed and reaches about 35% of the Internet

The 16-byte addresses should serve our needs for a very long time to come.

2001:0db8:0000:0042:0000:8a2e:0370:7334

As long as your code accepts IP addresses or host names from the user and passes them directly to a networking library for processing, you will probably never need to worry about the distinction between IPv4 and IPv6.

The operating system (OS) will know which IP version it is using and should interpret addresses accordingly.

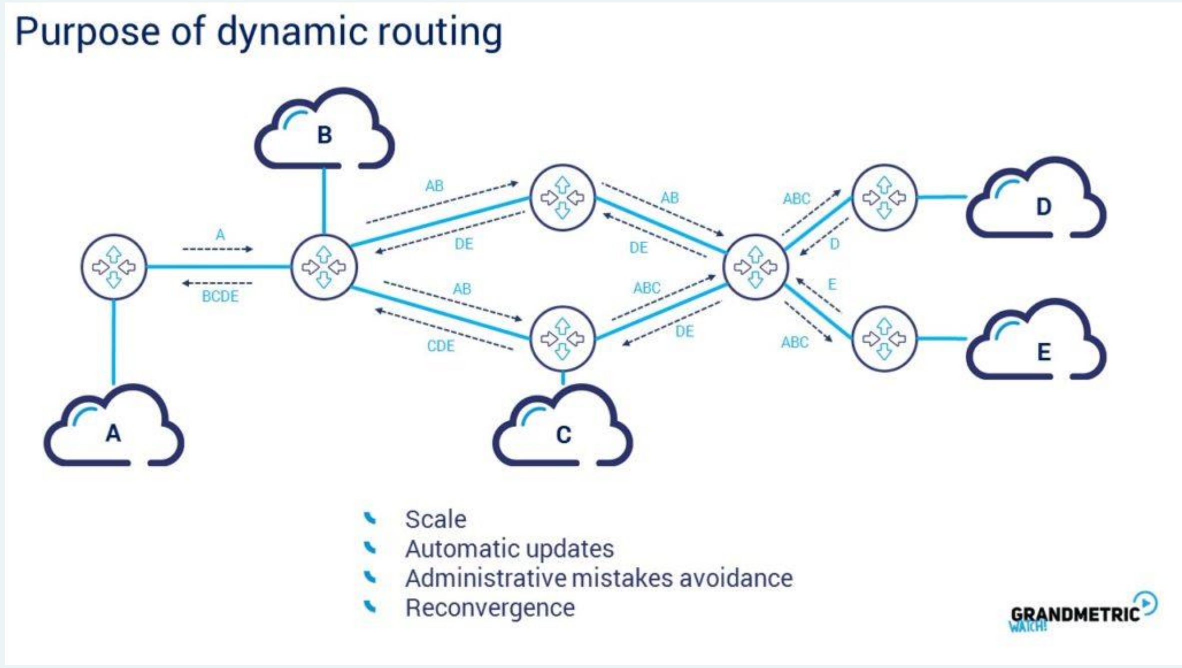


Some examples of IPv4 addresses

- IP addresses can be read from left to right
  - The first one or two bytes specify an organization
  - Then the next byte often specifies the particular subnet on which the target machine resides
  - The last byte narrows down the address to that specific machine or service.
- There are also a few special ranges of IP address have a special meaning:
  - 127.*.*.*
    - IP addresses that begin with the byte 127 are reserved range that indicates they are local to the machine on which an application is running.
    - 127.0.0.1 or ::1is used universally to mean “this machine itself that this programme is running on” (also known as the loopback address)
  - 10.*.*.*
  - 192.168.*.*
  - 172.16.31.*
- These IP ranges are reserved for what are called private subnets.
- These addresses will never be used in servers or services.
- Therefore, these addresses are free for you to use on internal networks.

Routing

- The decision of where to send each packet, based on the IP address that is its destination is called routing.
  - If the IP address look like 127.*.*.*, then the OS knows that the packet is destined for another application running on the same machine.
  - If the IP address is in the same subnet as the machine itself, then the destination host can be found by simply checking the local Ethernet segment.
  - Otherwise, your machine forwards the packet to a gateway machine that connects your local subnet to the rest of the Internet.
- Routing is only this simple at the very edge of the Internet
- The actual routing decisions are much more complex for the dedicated network devices that form the Internet's backbone!

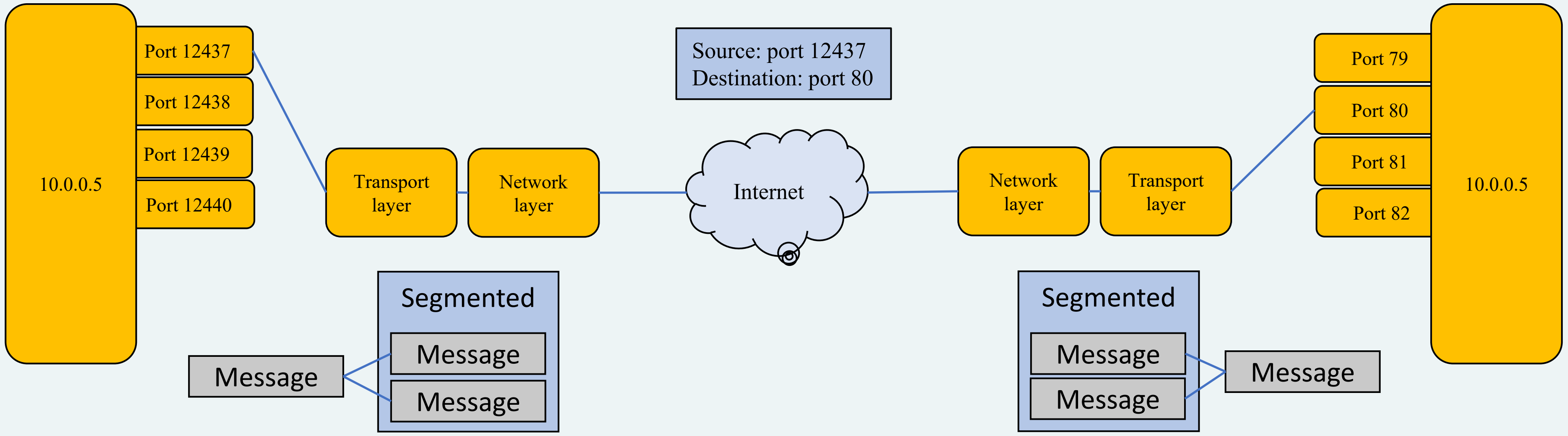


Role of the Transport layer

- The Transport Layer is responsible for establishing a temporary communication session between two applications and delivering data between them.
- TCP/IP uses two protocols to achieve this:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
- Primary Responsibilities of Transport Layer Protocols
  - Tracking the individual communication between applications on the source and destination hosts
  - Segmenting data for manageability (maximum packet size for route)
  - Reassembling segmented data into streams of application data at the destination
  - Identifying the proper application for each communication stream

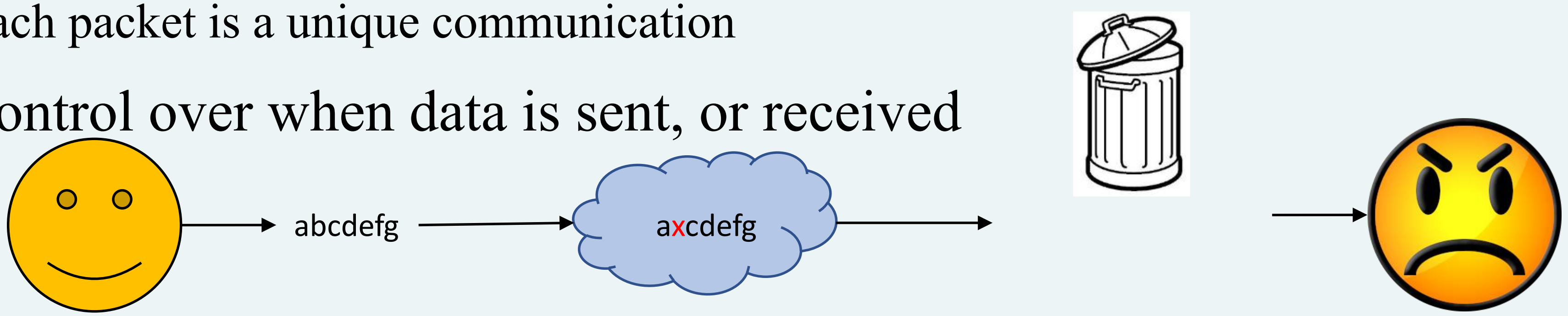
Transport layer (UDP/TCP)

- Letting multiple applications use one network connection simultaneously.
  - The TCP/UDP protocols creates 65,535 ports on your computer per network (IP address assigned to you)
  - One application can use multiple ports at the same time if it wants
  - A connection is defined by source IP : source port : destination port : destination IP



User Datagram Protocol (UDP)

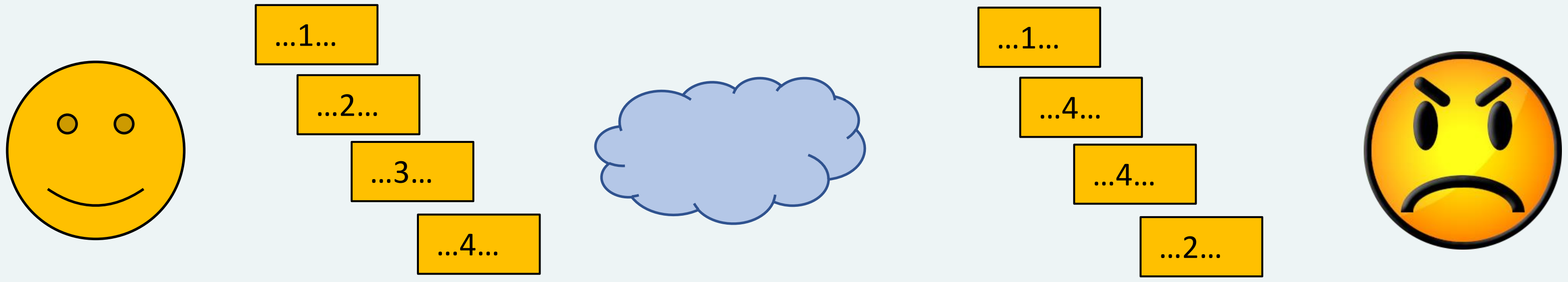
- UDP is a lightweight, connectionless choice
  - UDP has small data packet sizes, about 60% even less than TCP
    - UDP header: 8 bytes
    - TCP header: 20 bytes
  - No connection to create and maintain
    - Each packet is a unique communication
  - No control over when data is sent, or received



Sensitivity: Internal

User Datagram Protocol (UDP) (Cont’d)

- UDP does not guarantee in order package delivery
  - The package will not necessarily arrive at the receiving application in the order that they were sent
  - No congestion control in UDP
  - No guarantee that a packet only arrives once
  - Lightweight, but not that reliable this is where TCP comes
  - No guarantee that a packet only arrives once



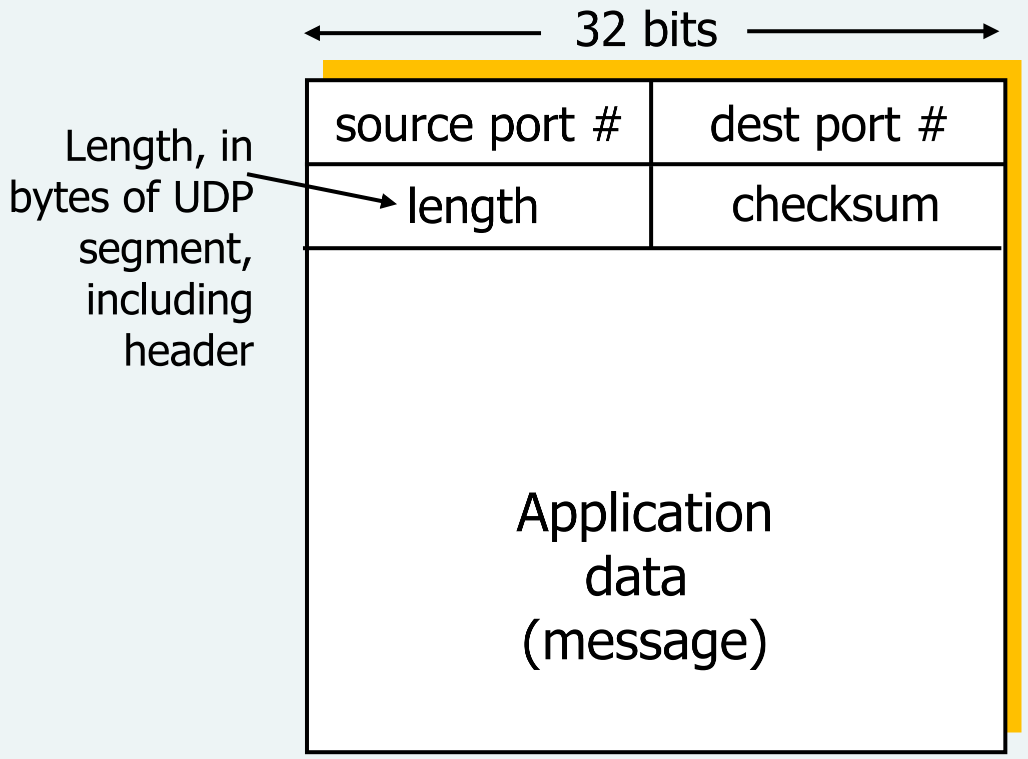
User Datagram Protocol (UDP)

- No frills, bare bones Internet transport protocol
- Best effort service, UDP segments may be:
  - Lost
  - Delivered out of order to application
- Connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- So why do we like UDP?
  - No connection establishment (which can add delay)
  - Simple: no connection state at sender, receiver
  - Small segment header(8bytes)
  - No congestion control: UDP can blast away as fast as desired

Uses for UDP

- UDP is used for:
  - Streaming multimedia apps
    - Loss tolerant
    - Rate sensitive
  - DNS
  - SNMP
- Reliable transfer over UDP
  - add reliability at application layer
  - application-specific error recover!


32 bits
source port #
dest port #
Length, in bytes of UDP segment, including header
length
checksum
Application data (message)

UDP segment format

Transmission Control Protocol (TCP)

- Transmission control protocol (TCP) is a network communication protocol designed to send data packets over the Internet, ensuring the delivery of messages over supporting networks and Internet.
  - Reliable, connection-based choice.
  - Negotiation is required before sending the data.
  - In order delivery, as the segments are numbered

Wanna connect?
Alright then!
I am data
Sure
Got it!

TCP (Cont’d)

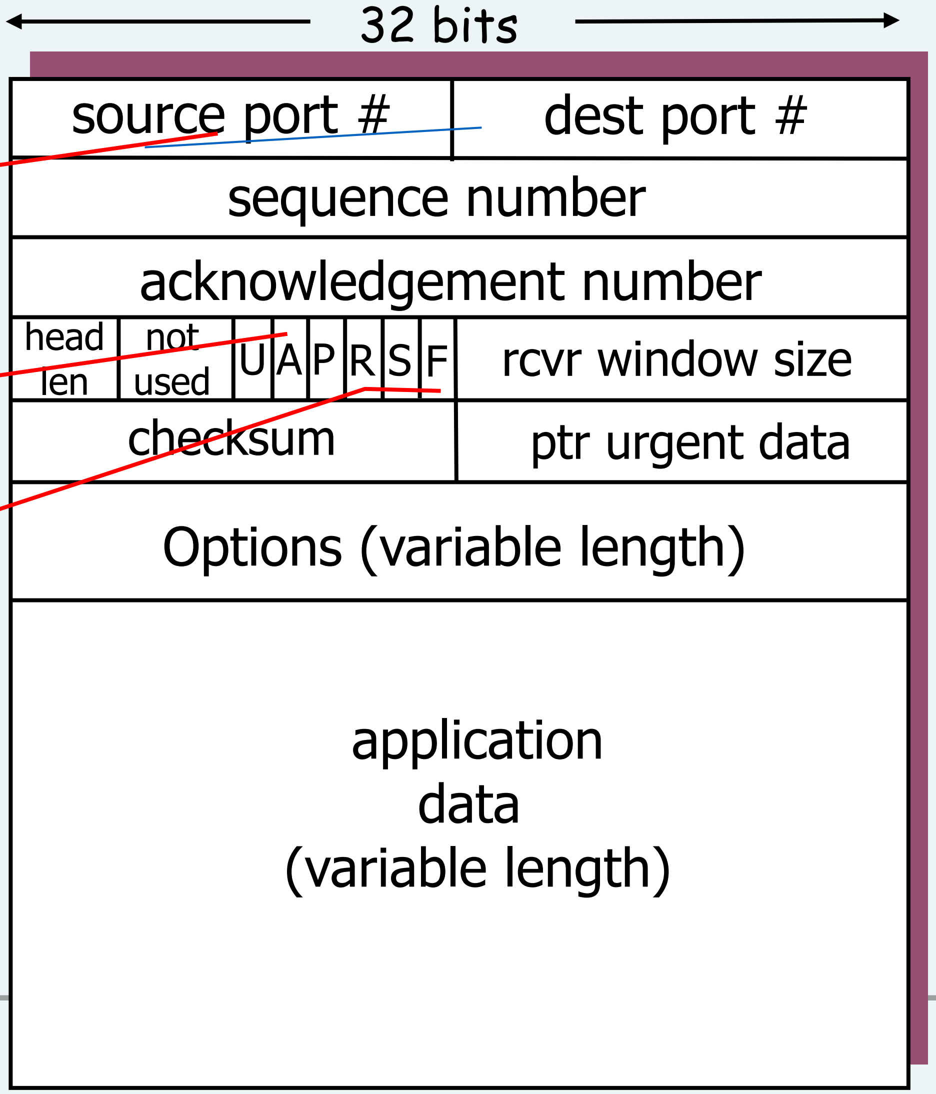
- Congestion control:
    - delays transmission when the network is congested
- Error detection:
    - no improvement, but checksum now mandatory
- Bigger header than UDP (60% larger than UDP)
- Data does not always get sent out immediately
    - Data can be stored before transmission
    - Data can be packet together
    - Delay can be bad e.g. Skype voice call
- Bigger overhead:
    - Retransmission of packets
    - Acknowledgement of packets (e.g. HD video)

TCP Packet Structure

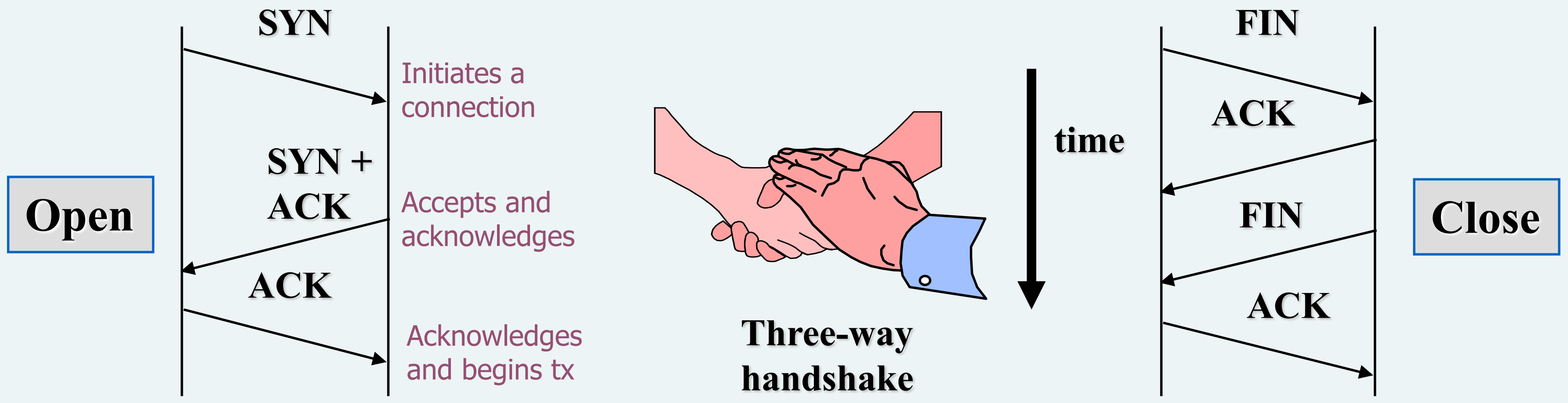
Ports individually identify processes /applications. (e.g. well know ports – FTP 21, HTTP 80, IRC 194)

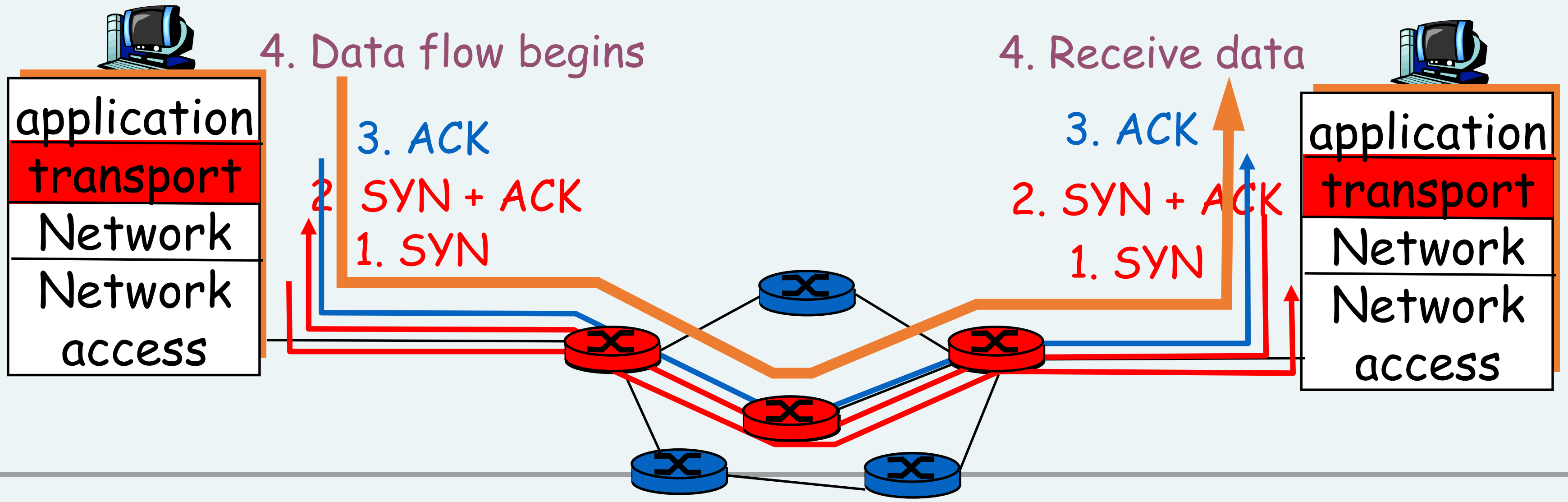
ACK: ACK # valid

RST, SYN, FIN: connection estab (setup, teardown commands)



TCP Connection Management: 3 Way Handshake





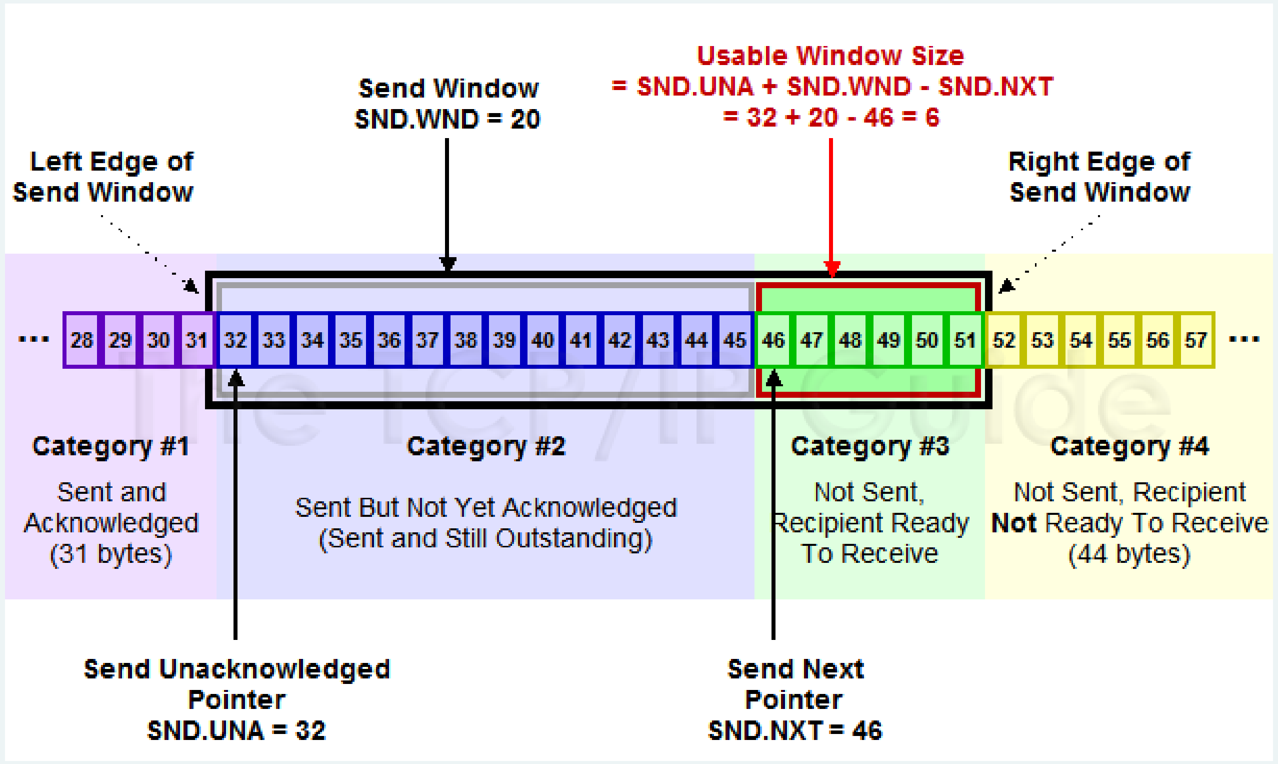
derby.ac.uk

Sensitivity: Internal

TCP Characteristics

- Positive acknowledgement
  - A message is transmitted and then a positive acknowledgement is waited upon
  - If the positive acknowledgement does not arrive in a certain period of time, the message is retransmitted
- Sequenced
  - Messages are numbered in sequence so that none are lost or duplicated
  - Messages are delivered at the destination in the same order they were sent by the source (from the receiving programs perspective)

TCP Windowing



Messages vs. streams

- UDP is message-oriented
    - Application sends data in distinct chunks (e.g. a letter)
- TCP is stream-oriented
    - Used as a continuous flow of data
    - Split up in chunks by TCP (e.g. phone conversation)

Which one is better?

	TCP	UDP
• Text communication		
• Streaming communication		
• Video downloading		
• Torrenting		

Other uses for TCP

- When the data can not be lost or the order of delivery is important.
    - File transfers
    - Remote access
- When delivery acknowledgements are needed
    - Implement acknowledgements in your application for important packets

Multimedia streaming

- UDP or TCP can be both applied on multimedia streaming
- Traditionally people have chosen UDP for such applications:
  - It has less overhead
  - Less latency
  - Data loss can be masked
- Recently, TCP has been used for streaming as:
  - Its overhead can be mitigated to not deteriorate performance
  - Avoidance of firewalls which block UDP for security reasons

Other uses for UDP

- Small question-and-answer transactions
  - DNS lookups
  - No need to acknowledge creation and closure of the connection e.g. fire and forget
- Bandwidth-intensive applications that tolerate packet loss (e.g. YouTube..)

Small Client-Server network

- This coursework requires you to produce a network application which demonstrates:
  - client-server networking and/or peer-to-peer (P2P) networking principles
  - communication security principles e.g. authentication, and encryption

- If you cannot identify a project that you want to implement then you should consider something like the Simple File Transfer Protocol (SFTP – RFC 913).

Part 2

1. Overview of Network Application Programming
2. Python’s socket Module
3. Building a Simple Network Application (Client-Server)
4. How to handle Multiple Clients

Networked Application Programming

- **What is Network Application?**
  - A network application is a software program or service that relies on network resources to perform specific functions, enabling communication, data sharing, and collaboration among devices connected to a network.
  - This includes the Internet, local networks (LANs), and other types of network connections.
  - Messaging Apps:
    - Examples: WhatsApp, Slack, Microsoft Teams
    - Messaging applications allow real-time text, voice, and video communication between individuals and groups, enhancing collaboration and connectivity.

Q. What pivotal roles do Network Applications play in today’s interconnected world?

Networking Concepts

- Network Protocols: These are rules and conventions for communication between network devices. Common protocols include TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).
  - TCP is reliable and makes sure that data is delivered in order, making it suitable for applications like web browsing and email.
  - UDP, on the other hand, is faster but does not guarantee delivery or order, making it suitable for applications like live video streaming and online gaming.
- IP Addresses: Every device on a network has a unique IP (Internet Protocol) address, which acts as its identifier. There are two versions of IP addresses: IPv4 and IPv6.
  - IPv4 addresses are 32-bit numbers, typically written as four decimal numbers separated by dots (e.g., 192.168.1.1). IPv6 addresses are 128-bit numbers, written as eight groups of four hexadecimal digits separated by colons.

What We would focus on

- Networking Protocols:
    - TCP/IP: The foundational protocol of the internet, providing reliable, ordered, and error-checked delivery of a stream of bytes.
    - UDP: A simpler, connectionless Internet protocol that offers a direct way to send and receive datagrams without guaranteeing delivery or order.
- Python for Network Programming
    - Python's simplicity and extensive standard library make it ideal for network programming. It allows for rapid development of networked applications with readable and maintainable code.

Networking Concepts

- Ports: Ports are numerical identifiers for specific processes or services on a device. For example, web servers typically use port 80 for HTTP traffic and port 443 for HTTPS traffic.
    - When data is sent to an IP address, the port number indicates which application should receive the data.
- Sockets: A socket is an endpoint for communication between two devices. It combines an IP address and a port number to uniquely identify a connection.
    - Sockets provide a way for software to read and write data across the network.

Built-in Python libraries (Networking)

- *socket*: This library provides low-level access to network interfaces, allowing you to create and manage network connections using both TCP and UDP protocols.
- *http.client* and *http.server*: These libraries offer higher-level functions for creating HTTP clients and servers, making it easier to build web-based applications.
- *urllib* and *requests*: These libraries simplify working with URLs and handling HTTP requests, enabling you to interact with web APIs and download content from the web.
- *asyncio*: This library provides support for asynchronous programming, allowing you to handle multiple network connections concurrently without blocking the main execution thread.

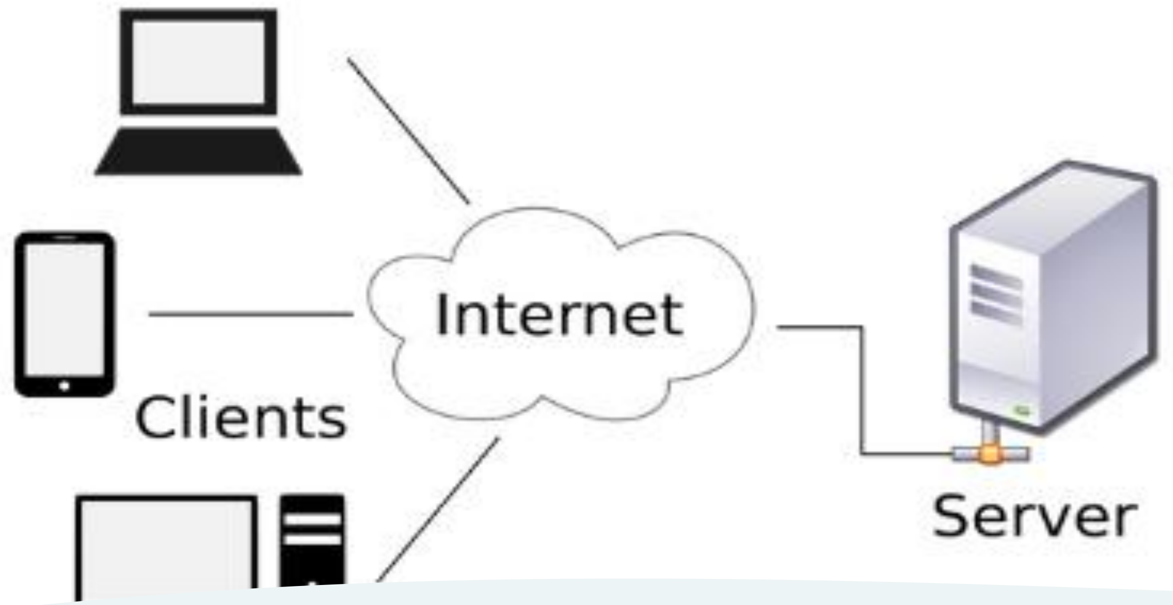
Understanding Sockets

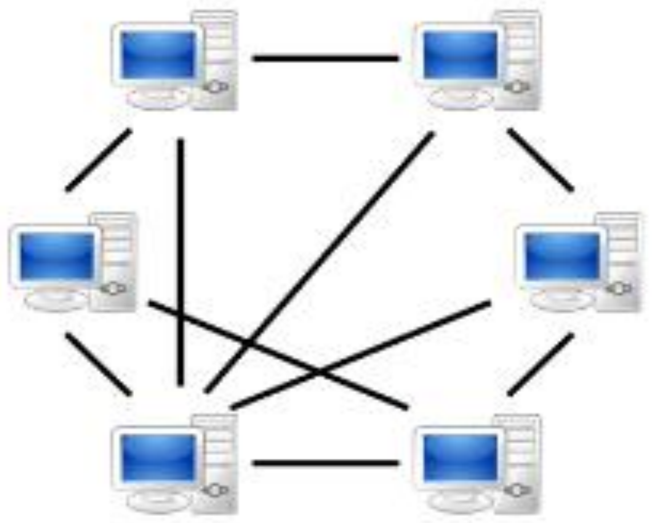
- *A socket is essentially a combination of an IP address and a port number, creating a unique endpoint for network communication. There are two main types of sockets:*
    - *Stream Sockets (TCP): These sockets use the Transmission Control Protocol (TCP) to provide reliable, connection-oriented communication.*
        - *They make sure that data is delivered in the correct order and without errors.*
    - *Datagram Sockets (UDP): These sockets use the User Datagram Protocol (UDP) to provide connectionless communication.*
        - *They are faster but do not guarantee delivery or order, making them suitable for applications where speed is more critical than reliability.*

Handling Errors: Sockets

- *Network communication can be unpredictable, so it’s important to handle errors gracefully.*
- *Python’s socket module raises exceptions for various errors, such as* ***socket.error, socket.timeout, and socket.gaierror.*** *You can use try-except blocks to handle these exceptions and make sure your program can recover from errors:*

```
import socket
try:
# Example code that might raise an exception
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 65432))
server_socket.listen(1)
except socket.error as e:
print('Socket error: {}'.format(e))
except Exception as e:
print('Other error: {}'.format(e))
```





Communication Models

- **Client-Server Model: The most common architecture, where one program (the client) requests services, and another program (the server) provides those services.**
- **Peer-to-Peer (P2P): In P2P networks, all nodes can act as both clients and servers, sharing resources directly without centralized servers.**
- **Hybrid models?**

Implementing TCP client-server

- *Client-server communication is a fundamental concept in network programming.*
- *It involves two main components: the client, which initiates the communication, and the server, which responds to the client's requests.*
- *This code section will guide you through the process of implementing client-server communication in Python using TCP.*
- *TCP (Transmission Control Protocol) is a connection-oriented protocol that makes sure reliable and ordered delivery of data. It is commonly used in applications where data integrity is crucial, such as web servers and email clients.*
- *First, we create a simple TCP server that listens for incoming connections and echoes back any data it receives. Second, the server will run indefinitely, handling one connection at a time.*

Implementing TCP server - Listener

```
import socket

def start_tcp_server():
# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the address and port
server_address = ('localhost', 65432)
print('Starting up on {} port {}'.format(*server_address))
server_socket.bind(server_address)

# Listen for incoming connections (clients)
server_socket.listen(1)
```

TCP server – Listen indefinitely (Loop 1)

```
while True:
# Wait for a connection
print('Waiting for a connection')
connection, client_address = server_socket.accept()
try:
print('Connection from', client_address)
# Receive the data in small chunks and retransmit it
```

TCP server – Run indefinitely (Loop 2)

```
while True:
data = connection.recv(16)
print('Received {!r}'.format(data))
if data:
print('Sending data back to the client')
connection.sendall(data)
else:
print('No more data from', client_address)
break
finally:
# Clean up the connection
connection.close()

# Start the TCP server
start_tcp_server()
```

TCP client – connect to server

```
import socket

def start_tcp_client():
# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's port
server_address = ('localhost', 65432)
print('Connecting to {} port {}'.format(*server_address))
client_socket.connect(server_address)

try:
# Send data
message = b'This is the message. It will be repeated.'
print('Sending {!r}'.format(message))
client_socket.sendall(message)
```

TCP client – process server response

```
# Look for the response
amount_received = 0
amount_expected = len(message)

while amount_received < amount_expected:
data = client_socket.recv(16)
amount_received += len(data)
print('Received {!r}'.format(data))

finally:
print('Closing socket')
client_socket.close()

# Start the TCP client
start_tcp_client()
```

TCP client – process server response

```
# Look for the response
amount_received = 0
amount_expected = len(message)

while amount_received < amount_expected:
data = client_socket.recv(16)
amount_received += len(data)
print('Received {!r}'.format(data))

finally:
print('Closing socket')
client_socket.close()

# Start the TCP client
start_tcp_client()
```

Implementing UDP Client-Server Communication

- UDP (User Datagram Protocol) is a connectionless protocol that does not guarantee the delivery or order of data.
- It is suitable for applications where speed is more important than reliability, such as live video streaming and online gaming.
- In this example, the client sends a message to the UDP server, which then echoes the message back to the UDP client.
- Let's create a simple UDP server that listens for incoming messages and echoes them back to the sender.

UDP Server: Create socket, bind to address/port

```
import socket

def start_udp_server():
# Create a UDP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the address and port
server_address = ('localhost', 65432)
print('Starting up on {} port {}'.format(*server_address))
server_socket.bind(server_address)
```

UDP Server: Wait for message

```
while True:
# Wait for a message
print('Waiting for a message')
data, address = server_socket.recvfrom(4096)

print('Received {} bytes from {}'.format(len(data), address))
print(data)

if data:
sent = server_socket.sendto(data, address)
print('Sent {} bytes back to {}'.format(sent, address))

# Start the UDP server
start_udp_server()
```

UDP Client: Create socket and Server address/port

```
import socket

def start_udp_client():
# Create a UDP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Server address
server_address = ('localhost', 65432)
message = b'This is the message. It will be repeated.'
```

UDP Client: Send and Receive response from server

```
try:
        # Send data
        print('Sending {!r}'.format(message))
        sent = client_socket.sendto(message, server_address)

        # Receive response
        print('Waiting for a response')
        data, server = client_socket.recvfrom(4096)
        print('Received {!r}'.format(data))

    finally:
        print('Closing socket')
        client_socket.close()

# Start the UDP client
start_udp_client()
```

Handling Multiple Clients?

- To handle multiple clients simultaneously, you can use threading.
- Each client connection is handled in a separate thread, allowing the server to manage multiple connections concurrently.
- We would discuss Python threading next week.

Multi-threaded TCP server

```
import socket
import threading

def handle_client(connection, client_address):
    try:
        print('Connection from', client_address)
        while True:
            data = connection.recv(16)
            if data:
                print('Received {!r}'.format(data))
                connection.sendall(data)
            else:
                print('No more data from', client_address)
                break
    finally:
        connection.close()
```

Create socket, bind and listen for connections

```
def start_threaded_tcp_server():
# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the address and port
server_address = ('localhost', 65432)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(5)
print('Waiting for a connection')
```

TCP Server: Create thread for each connection

```
while True:
connection, client_address = server_socket.accept()
client_thread = threading.Thread(target=handle_client,
args=(connection, client_address))
client_thread.start()

# Start the threaded TCP server
start_threaded_tcp_server()
```

Choosing Between TCP and UDP for Client-Server Communication

- TCP: Use TCP when you need reliable, ordered delivery of data.
- This is suitable for applications like web servers, email clients, and file transfers, where data integrity is critical.
- UDP: Use UDP when you need fast, efficient communication and can tolerate some data loss.
- This is suitable for applications like live video streaming, online gaming, and voice-over-IP (VoIP), where speed is more important than reliability.

Summary

- Whether you are creating simple client-server applications, developing network tools, or building complex network systems, Python provides the tools and flexibility needed to achieve your goals.
- By understanding the basics of sockets and client-server communication, you can leverage Python to build strong and efficient network applications.
- Keep experimenting, explore additional features, and continue to deepen your knowledge of Python’s network programming capabilities.

Reading list

- Websites:
  - https://diveintopython3.net/
  - https://www.slitherintopython.com/
  - https://www.w3schools.com/python/
  - https://realpython.com/
  - https://automatetheboringstuff.com/
  - https://www.reddit.com/r/Python/
- Home install:
  - Ensure you have a Jetbrains account (www.jetbrains.com/student)
  - Install Python 3.8.5 or later and PyCharm (https://www.jetbrains.com/pycharm/download)
- Week 2 objectives:
  - Become familiar with Python basics and socket programming (selection, iteration, functions, classes, libraries)

UNIVERSITY OF DERBY

University of Derby, Kedleston Road, Derby, DE22 1GB
T +44 (0)1332 591044 **E** opendays@derby.ac.uk

THANK YOU

derby.ac.uk

Sensitivity: Internal