CS3.301 Operating Systems and Networks

Transport Layer and how it works!

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

SERC
Software Engineering Research Centre

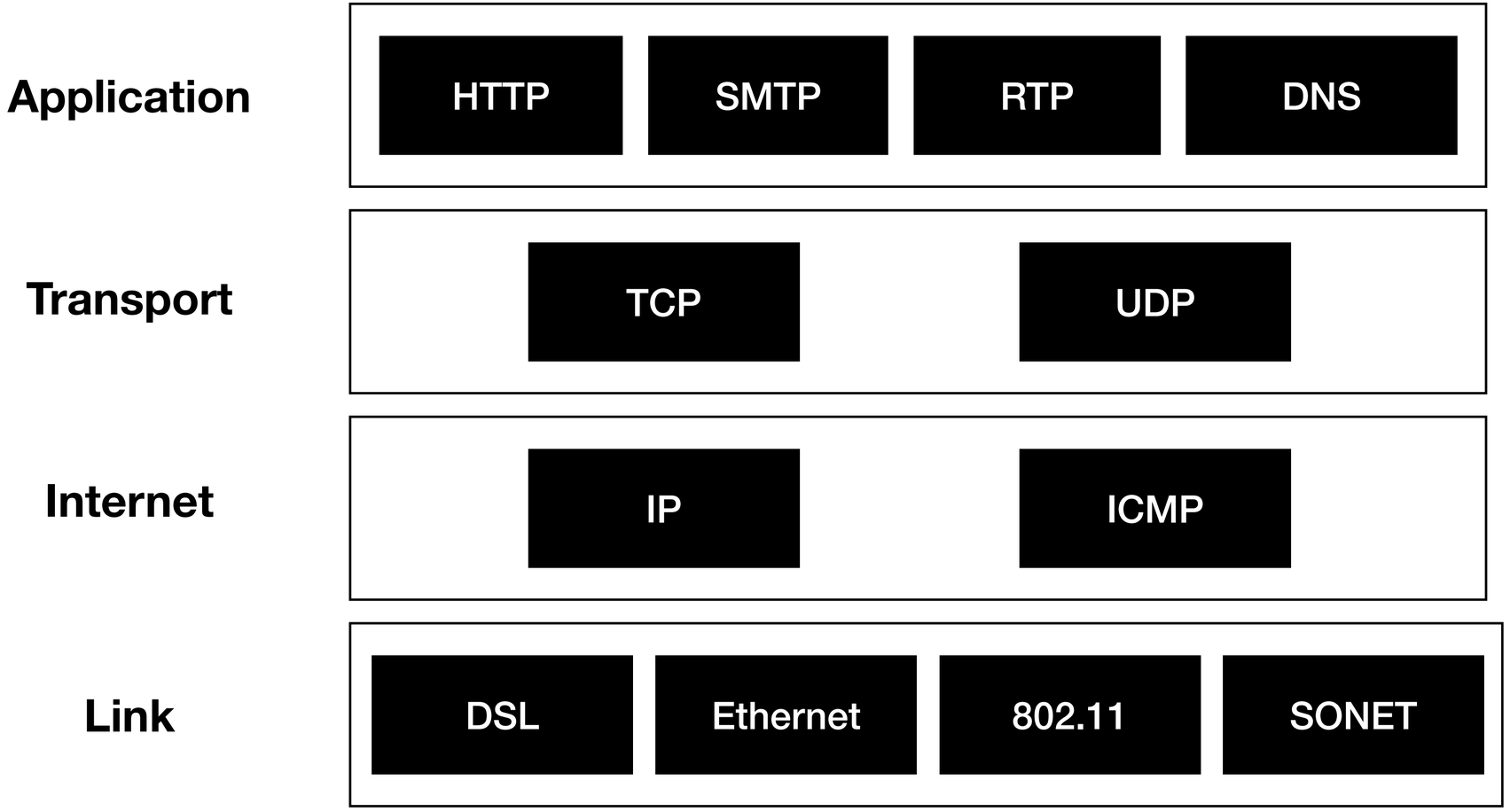
Acknowledgement

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

Network Protocol Stack



Onto Transport Layer

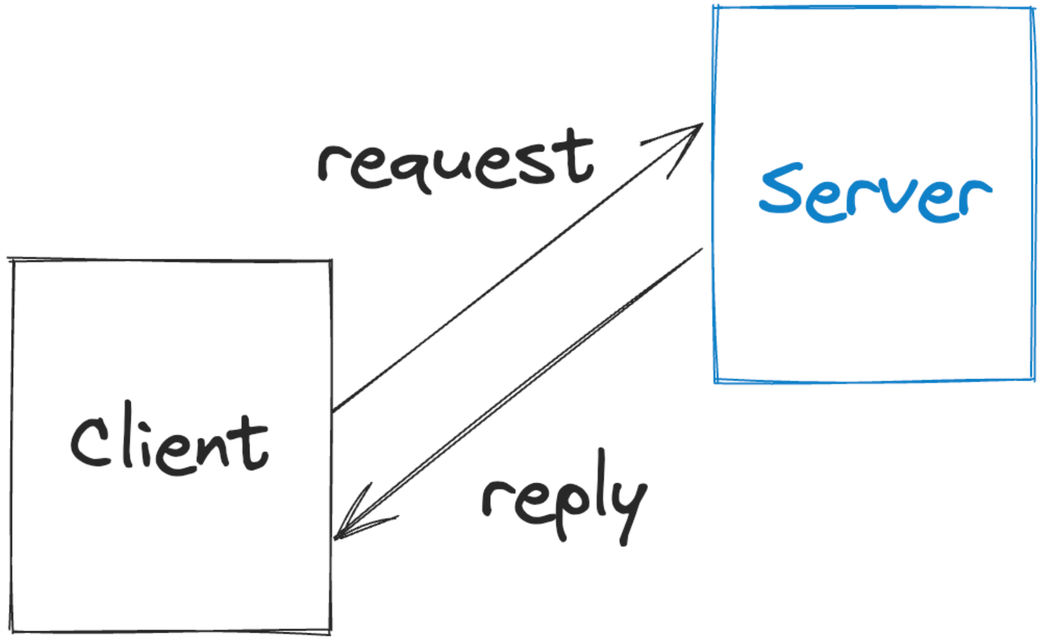
Service to Service Delivery

- **Multiplexing** and **Demultiplexing**
- Addressing scheme: **Ports**
- Two strategies/protocols that allows this
  - Transmission Control Protocol (TCP) - favours reliability
  - User Datagram Protocol (UDP) - favours efficiency

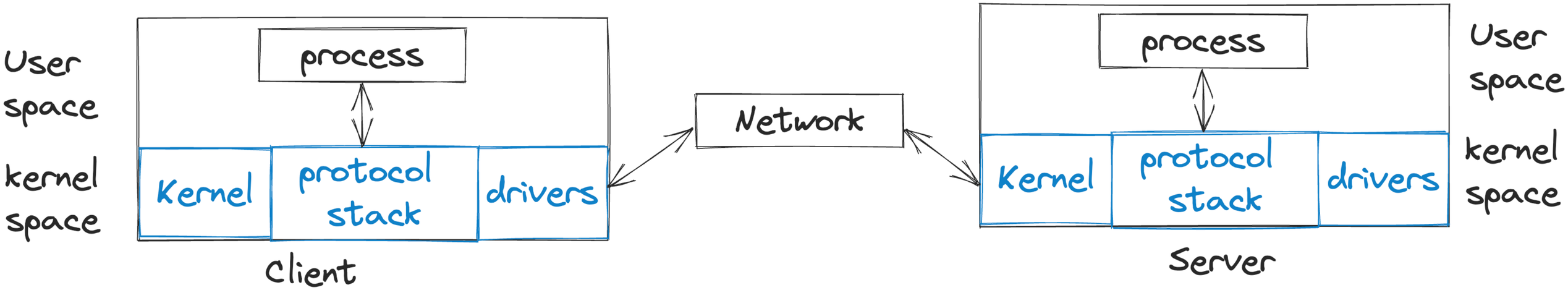
Lets go back

Two process wants to communicate

- Client sends a request to the server
- The server provides a reply based on the request made by client
- Eg: File transfer to get files
- Eg: web browsing, sent a url and get page
- How to transport data from process to process?



The role of Operating System



- Software component in the OS that supports network calls - **Protocol stack**
- Provides Service primitives which are nothing but system calls - **Some API?**

Any idea on what should be some functionalities that should be made available?

Hint: Think of process API

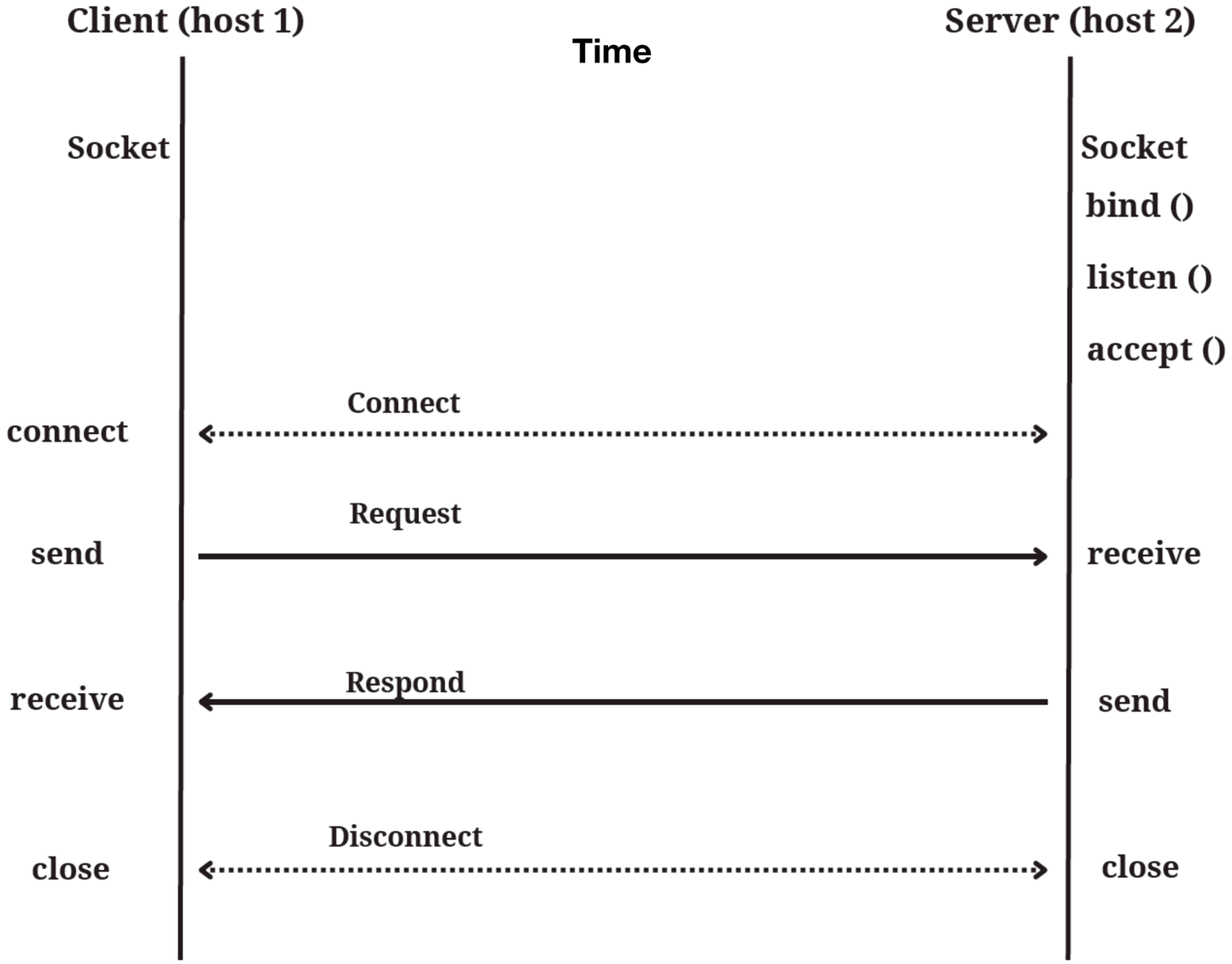
The Socket API

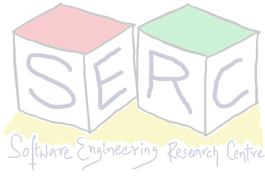
- Simple abstraction to use the network
- The network service API used to write all network applications
- Part of all OS and all language [Berkley Unix, 1983]
  - Allows user space applications to interact with networking subsystem
- Two services:
  - **Streams:** Reliable (Connection-oriented)
  - **Datagram:** Unreliable (Connection-less)
- Allows applications to attach to the network at different ports

The Socket API

Function	Description
socket()	Creates a new socket of a certain type (depending on TCP or UDP) and returns file descriptor
bind()	Associates the socket with a specific IP and port
listen()	For server sockets, it allows sockets to listen for incoming connections
accept()	For server sockets, it waits for client to connect and then return a new file descriptor
connect()	For client sockets, it initiates a connection to a server.
send() / receive()	Transmit data or receive data
close()	Terminate the connection

Using Sockets





More about Ports

- Application process is identified by tuple (IP address, Protocol, Port)
    - Port are 16-bit integers representing “mailboxes” that process leases
- Servers are often bind to “well-known-ports”
- Clients are assigned ephemeral ports
    - Chosen by the OS temporarily

Some well Known Ports

Port	Protocol	Use
20, 21	FTP	File Transfer
22	SSH	Remote login
25	SMTP	Email
80	HTTP	World wide web
443	HTTPS	Secured web
543	RTSP	Media Player Control

An Opportunity for a Context Switch?

- The calls of establishing socket are blocking calls
  - connect(), accept(), receive()
  - Once the call is made, OS halts the program to wait to receive some response
  - They are essentially **System calls**
  - Trap instruction is called and there is an opportunity for a context switch

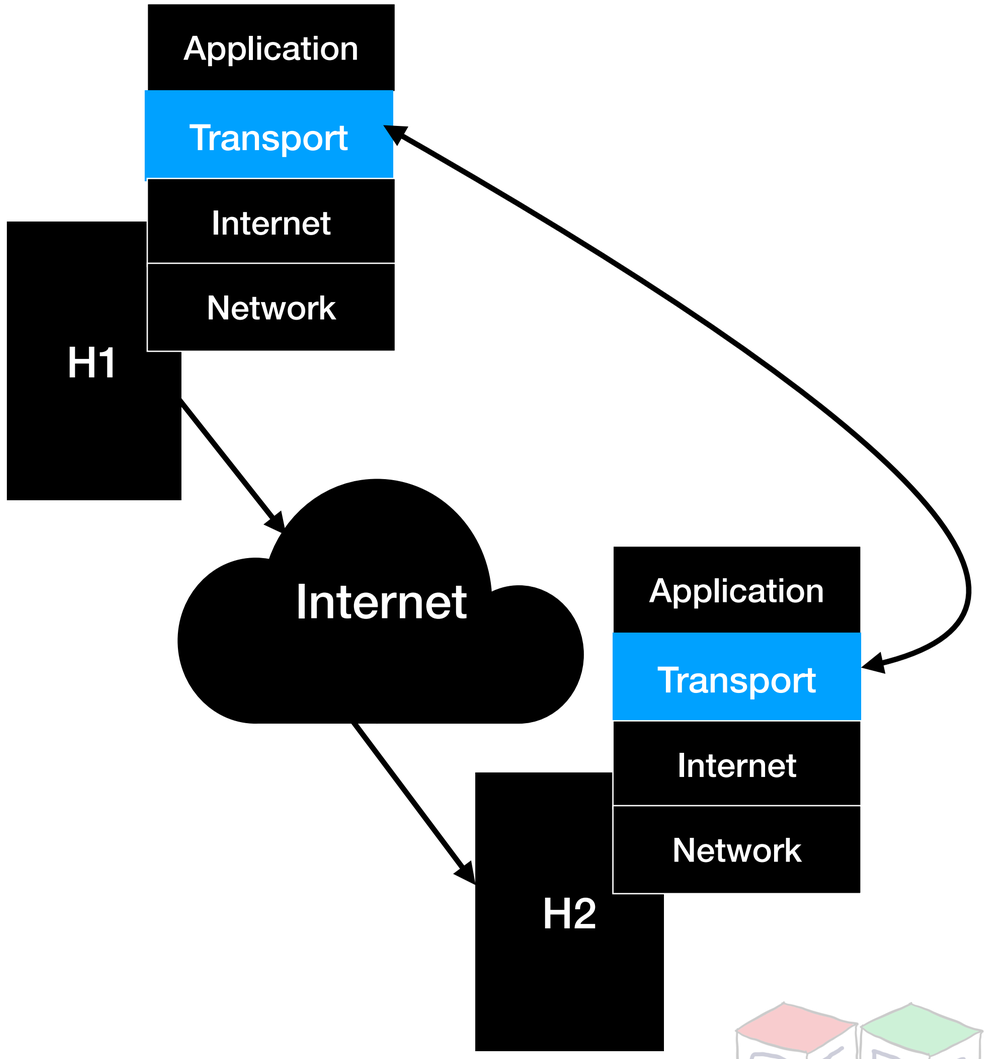
Let us take a step back

Types of Links

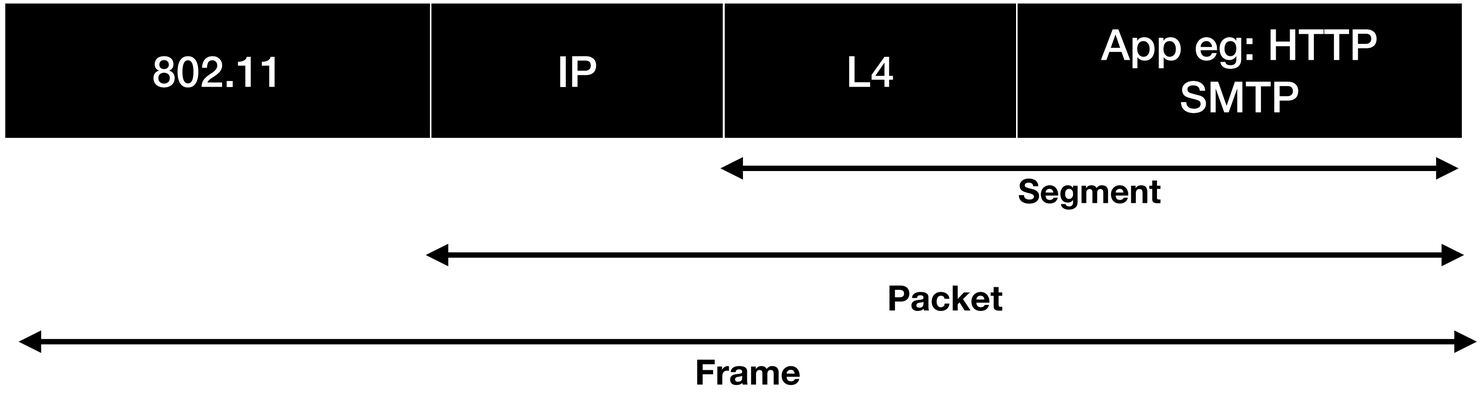
- **Full Duplex**
  - Bidirectional
  - Both sides at the same time
- **Half-duplex**
  - Bidirectional
  - Both the sides but only one direction at a time (eg: walkie talkies)
- **Simplex**
  - Unidirectional

Transport Services and Protocols

- Provides logical communication between application processes running on different hosts
- Transport protocols actions in the end systems:
  - Sender: breaks application messages into segments, passes to network layer
  - Receiver: reassembles messages into messages, passes to application layer
- Protocols: TCP, UDP



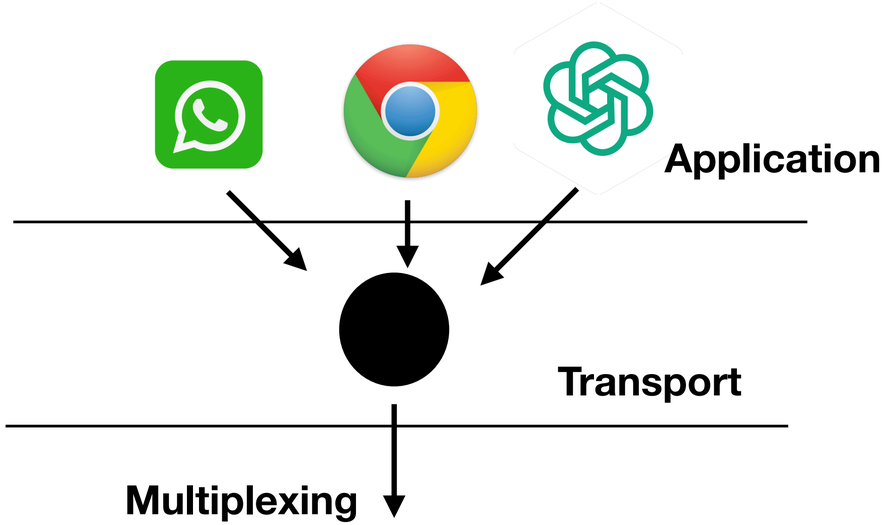
Quick Recap



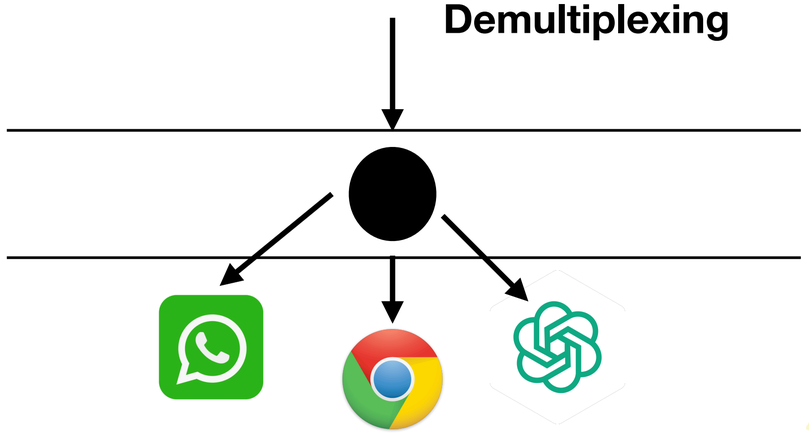
- Segments carry data across the network
- **Segments** are carried within the packets, within frames
- Each layer adds a **header** (Above L4 will be replaced by its header)

Multiplexing and Demultiplexing

- **Multiplexing as sender**: Handle data from multiple sockets, add transport header

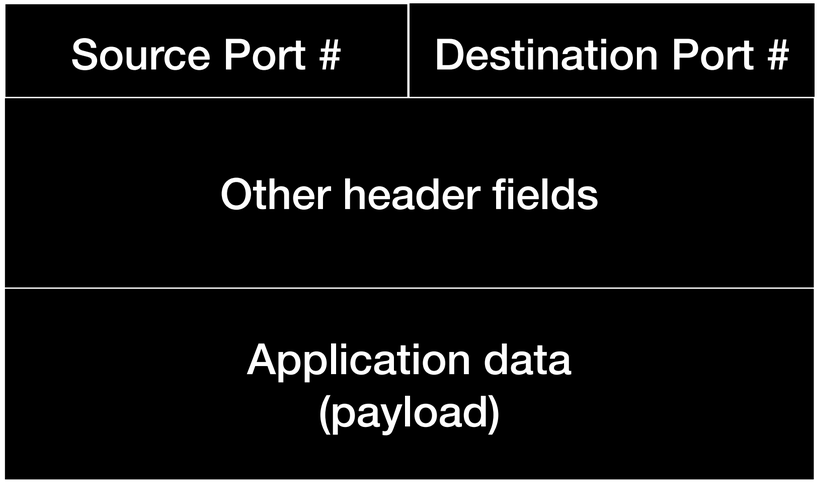


- **Demultiplexing as receiver**: Use header info to deliver received segments to correct socket



Working of Demultiplexing

- Host receives IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport layer segment
  - Each segment has source and destination port number
- IP addresses and ports are used to direct segment to appropriate socket



TCP/UDP Segment format

Connection Oriented vs Connectionless

Demultiplexing Scenarios

- **Connection oriented (TCP)**
  - TCP socket identified by 4 tuple
    - Source IP, destination IP, source port and destination port
  - Receiver uses all 4 to direct segment to appropriate socket
  - Server may support many TCP sockets
    - **Each socket has it own client**

- **Connectionless (UDP)**
  - UDP socket identified by 2 tuple
    - Destination IP and port
  - Receiver uses the port to redirect to the corresponding socket
  - UDP segments with same destination port but different IP or source port
    - **Redirected to same socket**

TCP vs UDP

TCP	UDP
Connection Oriented	Not Connection Oriented
Reliability (order is maintained and retransmission)	Unreliable
Higher overhead - reliability, error checking, etc	Low overhead
Flow control (based on network)	No implicit flow control
Error detection - retransmit erroneous packets	Has some error checking - Erroneous packets are discarded without notification
Congestion Control	No Congestion Control
Use cases: HTTP/HTTPS, File transfer, Mail	Use cases: Streaming data, VoIP, DNS queries, ..

Connection Oriented and Reliability

- **Connection Oriented**
  - In TCP, the connection is first established before the data is transmitted
  - In UDP there is no notion of connection starting and ending (use timeout)
- **Reliability**
  - Confirmation of data delivery (Acknowledgement is there) in TCP
    - Order is preserved or maintained
    - Error can be handled (Awareness). TCP can handle it.
  - In UDP there is no confirmation, the client trusts that there is someone to receive the data (Fire and Forget)
    - No error awareness (at L4). Protocol does not handle it

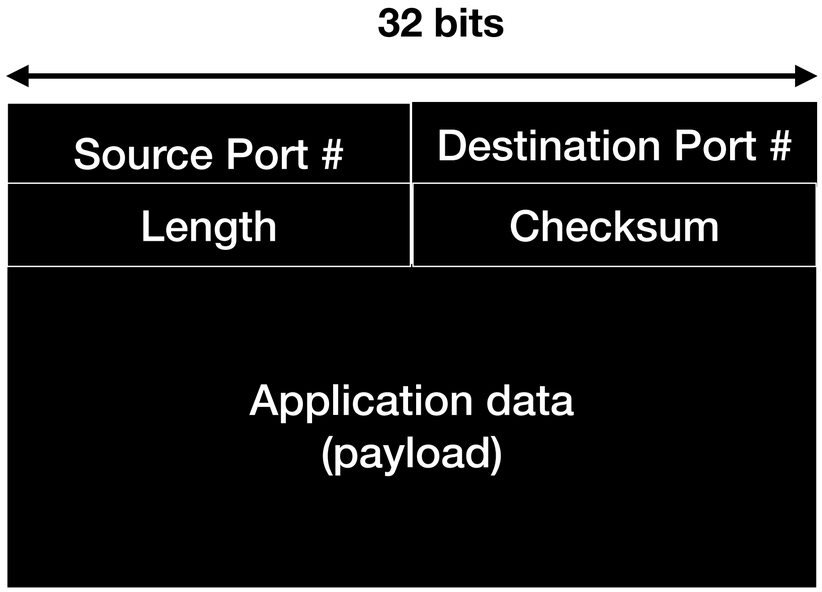
Flow Control and Overhead

- Flow Control
  - TCP can adjust the transmission rate to use maximum available bandwidth
  - Check how much the receiver can receive and adjust accordingly

- Overhead
  - TCP Adds a larger header to the data ~ 20 bytes or even more
  - TCP has more features that does not exist in UDP
  - In UDP the header length is ~ 8 bytes

UDP Segment Header

- Length: In bytes of the UDP segment including the header
- Checksum: For error detection (16 bit value which represents the sum of UDP header, payload and Pseudo header from IP layer)
  - Supports Error detection
  - Makes use of 1’s compliment arithmetic to find the sum



UDP Segment Format

Checksum Process

- **Sender**
  - All contents of the header including IP addresses are treated as sequence of 16 bit integers
  - Checksum: addition (one's complement) of segment content
- **Receiver**
  - Compute checksum of received content
  - Check if received and header checksum are equal - No error
  - Else, Error detected

Example

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Add it back	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

TCP is the most used protocol on the internet. How does TCP work?

What all you need to provide some features that TCP provides?

Thank you

Course site: karthikv1392.github.io/cs3301_osn
Email: karthik.vaidhyanathan@iiit.ac.in
Twitter: @karthi_ishere