Review for Chapter 3 in the book "A Top-Down Approach to Networking"

Introduction and the Tansport-Layer Services (3.1)

What is the role of the Transport Layer

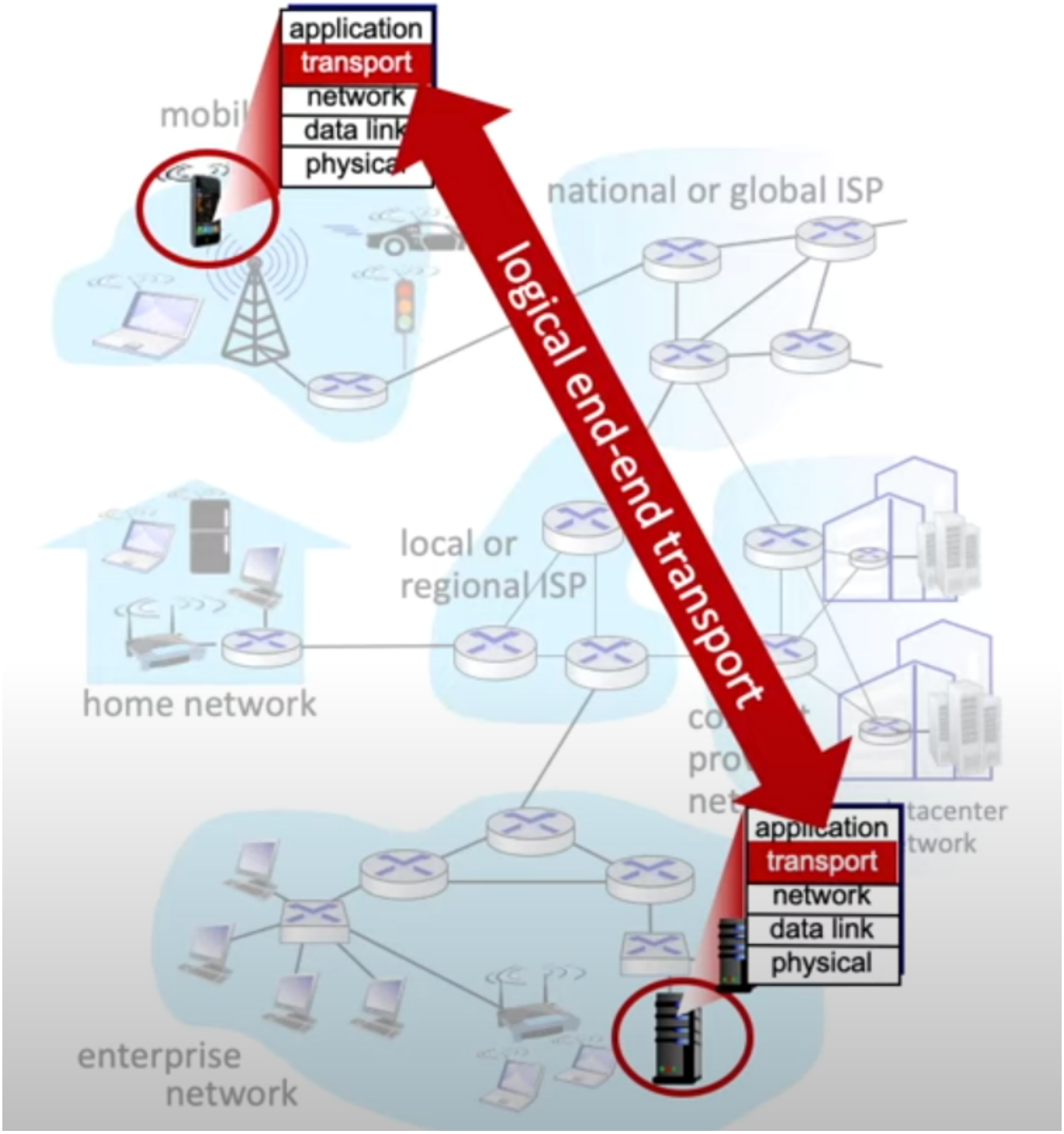
The role of the Transport layer is for end-to-end communication between applications on different hosts. It establishes a "logical communication" between applications on these hosts.

Transport Protocols actions in end systems (hosts) Sender: Breaks application messages into segments, passes to Network Layer
Receiver: Reassembles segments into messages, passes to Application Layer

Two main transport protocols available are TCP and UDP

Logical Communication

Logical Communication is a concept used to describe how applications or processes on different devices communicate as if they were directly connected (i.e they weren't on other sides of the world, and are rather right next to each other). This allows applications to "think" they are connected to each other, when in reality they're communicating across multiple routers and network segments.



Transport Layer Services

Multiplexing and Demultiplexing is also used in the transport layer alongside its protocols
Multiplexing: Allows data from multiple applications to share the same physical network link
Demultiplexing: Ensures incoming data packets are directed to the correct application based on information such as port numbers

Reliability and Flow Control: TCP provides these features to ensure that data is delivered reliably without overwhelming the network

Conncetion Setup: TCP requires a handshake between the client and server, before data can be sent, unlike UDP which is connectionless

Transport vs. Network Layer Services and Protocols

Although both layers work together to provide communication between devices, each layer has a distinct purpose and role to accomplish.

transport Layer: Provides End-to-End communcation, its role is to ensure the data is delivered to the correct application on the receiving device, regardless of the network path (logical communication)
Network Layer: Handles host-to-host communication across a network, focusing on the routing of data packets from one device to another also across various networks.

Aspect	**Transport Layer**	**Network Layer**
Addressing	Uses port numbers to identify applications on devices	Uses IP addresses to identify devices
Reliability	TCP offers reliable, ordered delivery with error-checking. USP, however, is unreliable	IP does not provide reliability, packets may be lost or reordered
Flow Control	Manages data flow to prevent overloading the receiver (e.g TCP flow control)	Not provided by IP; Transport layer handles it
Error Checking	Ensures data integrity (TCO checksums and retransmissions)	Limited error detection with IP checksums, no corrections
Multiplexing	Uses multiplexing and demultiplexing to handle multiple application streams (via port numbers)	Does not hande applications; only delivers packets to the correct host
Connection Management	TCP is connection-oriented while UDP is not	IP is connectionless

The Transport Layer mostly uses either TCP or UDP while the Network Layer most uses IP, ICMP or IGMP.

Application Scenarios

TCP is well-suited for applications that need reliable data transfer, like file transfer (FTP), web browsing (HTTP), and email (SMTP).
UDP is ideal for applications that prioritize speed over reliability, like live video streaming, online gaming, and voice-over IP (VoIP), where occasional packet loss is acceptable.

Multiplexing and Demultiplexing (3.2)

Multiplexing

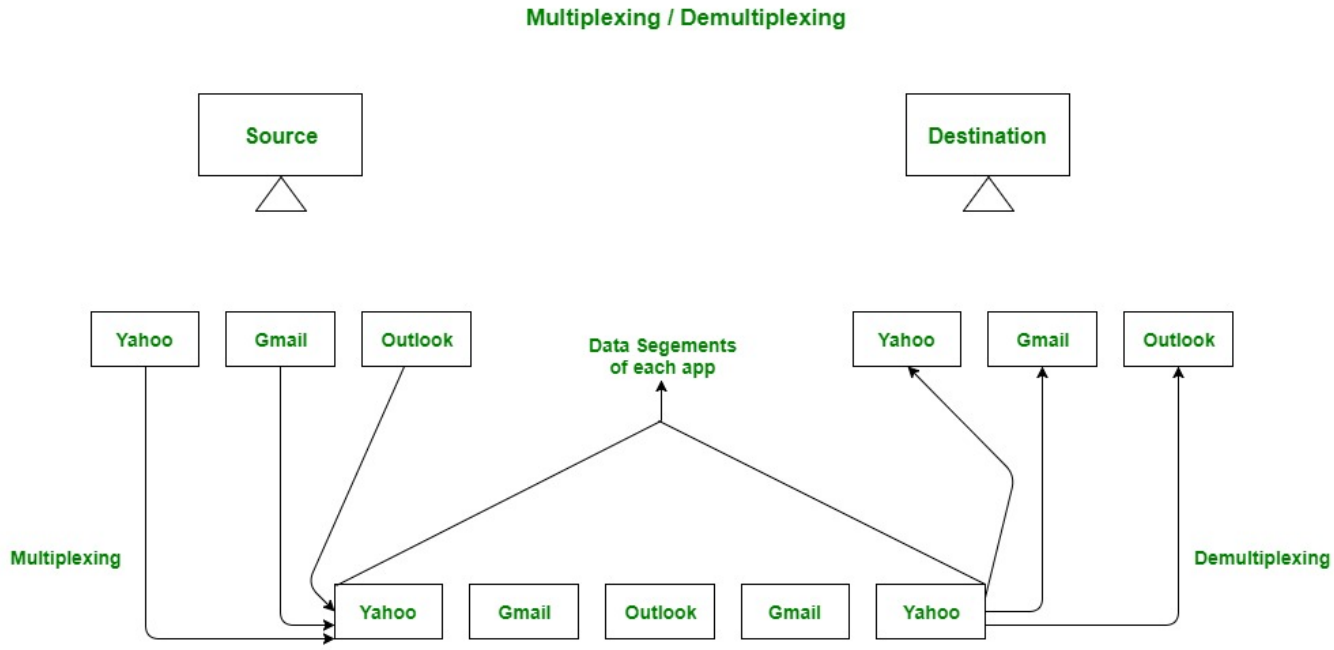
Imagine you have three friends (A, B, and C) who want to send messages to their friend D, but there’s only one phone line. Instead of taking turns and using the line one-by-one (which would be slow), a system is set up to combine all their messages together. This combined stream of messages is then sent over the line, and this process is called multiplexing

The process of combining data from multiple applications at the sender's transport layer and preparing it for transmission over the network. The transport layer attaches special information such as port numbers to each segment, identifying which application the data belongs to and making sure it reaches the correct person.

Demultiplexing

Now, when the messages reach D, he wants to separate them and figure out which message came from A, B, and C. The process of separating the combined messages into their original, individual parts is called demultiplexing

Unlike multiplexing, demultiplexing if the reverse process, occuring at the receiver's end. The transport layer inspects the headers of incoming segments and uses port numbers to direct each segment to the correct application. When the segment arrives, the transport layer checks the destination port number in the header to determine which socket should receive the data.



Role of Port Numbers

Port numbers are used in both multiplexing and demultiplexing to ensure each segment is directed to the correct application on both the sender's and receiver's sides.
Common port numbers used can be port 80 (for HTTP) or 443 (for HTTPS). There can also be unique port numbers for custom applications that are above 1024, which are typically chosen dynamically.

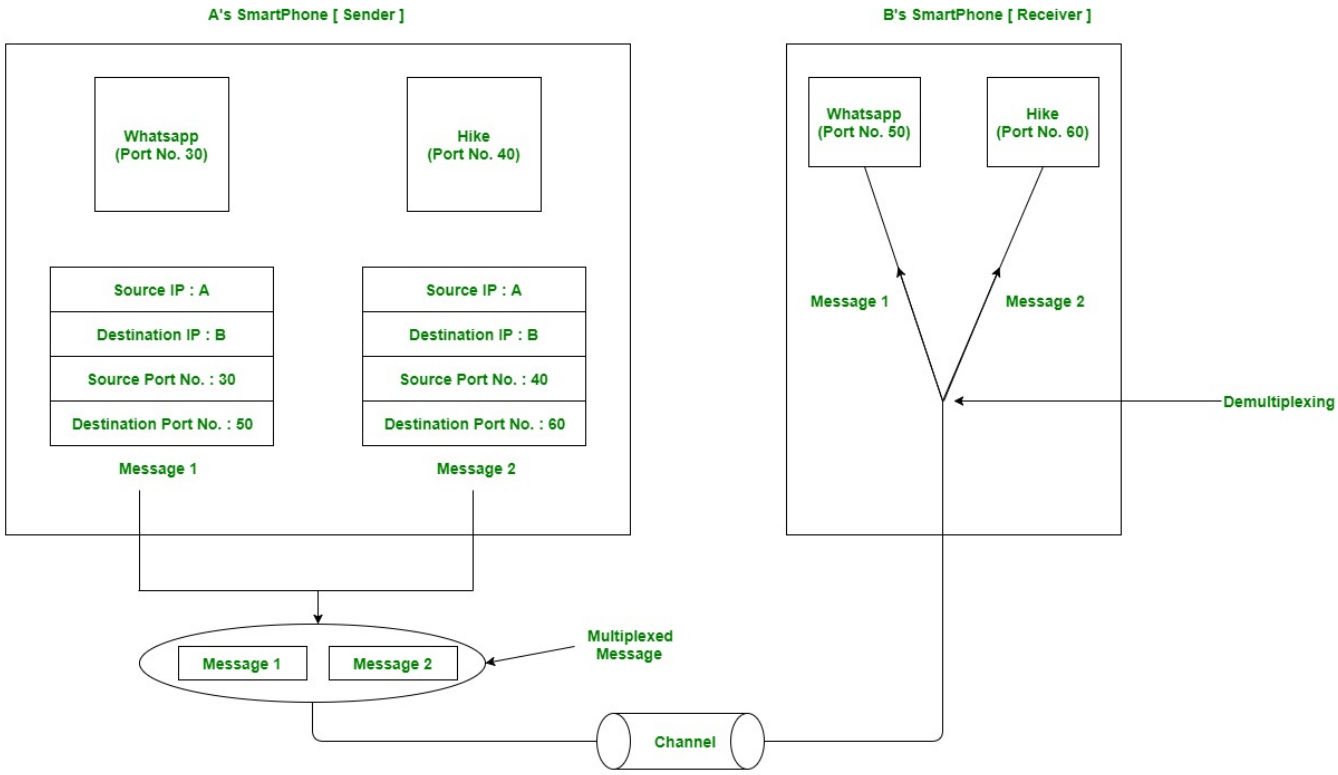
Multiplexing and Demultiplexing with TCP and UDP

Both TCP and UDP use port numbers, but differ in how they handle multiplexing and demultiplexing.

TCP: Is connection-oriented, meaing it requires both a unique sorce and destination port, as well as source and destination IP to identify a unique socket pair.
UDP: Is connectionless, meaning it uses port numbers only to deliver data to the correct application without needing to establish a connection

Unique Identifiers

for TCP, the combination of source IP, source port, destination IP, and destination port (4-tuple) uniquely identifies a connection
for UDP, each segment is sent independently, so only the destination port and IP are used to identify the intended application.



Connectionless Transport: UDP (3.3)

What is UDP?

UDP (User Datagram Protocol) is a transport layer protocol tht provides a connectionless, lightweight communication method. Unlike TCP, UDP does not need to establish a handshake / connection to send and receive data, it also does not break large files down, keeping them in their same format.
This allows UDP to be simple and fast, and without needing to follow many steps to send data, senders and receivers and share data quickly and effiecently.

Why is there a UDP?

Although it may seem that UDP is useless compared to TCP, there are still some advantages for using it.
UDP does not require a connection, meaning that it's very simple to send and receive messages, UDP has a small header size and has no congestion control meaning it can work as fast as it wants even in the face of congestion.

Characteristics

No Connection Setup: UDP is connection less, meaning that a handshake or connection is not needed to send / receive data.
Unreliable: Although being fast, UDP lacks organization, retransmission for missed packets, duplicated packets, or no packets at all.
No Flow Control: Unlike TCP, UDP does not have flow control, allowing a sender to send large amounts of data even if it means overwhelming the receiver.
No Congestion Control: UDP does not monitor or adjust sending rates based on network congestions, which could lead to a network overload

Structure of UDP segments

UDP segments consist of a simple header with four fields

```
Source Port: Identifies the application or process sending the data
Destination Port: Specifies the application or process that should receive the
data on the destination host
Length: Specifies the length of the UDP segment (Header + Data)
Checksum: Used for optional error-checking of the segment's header and data. If an
error is detected, UDP discards the segment without retrying
```

these small headers contribute to UDP's low overhead, making it ideal for fast speed

Benefits of UDP

Low Latency: Due to being connectionless, having no retransmissions nor acknowledgements, this allows UDP to achieve very low-latency communication, ideal for real-time applications.
Efficiency: Due to UDP's simple structure and a lack of a connection setup, this makes UDP a resource-efficient protocol, minimizing both network and computational overhead.

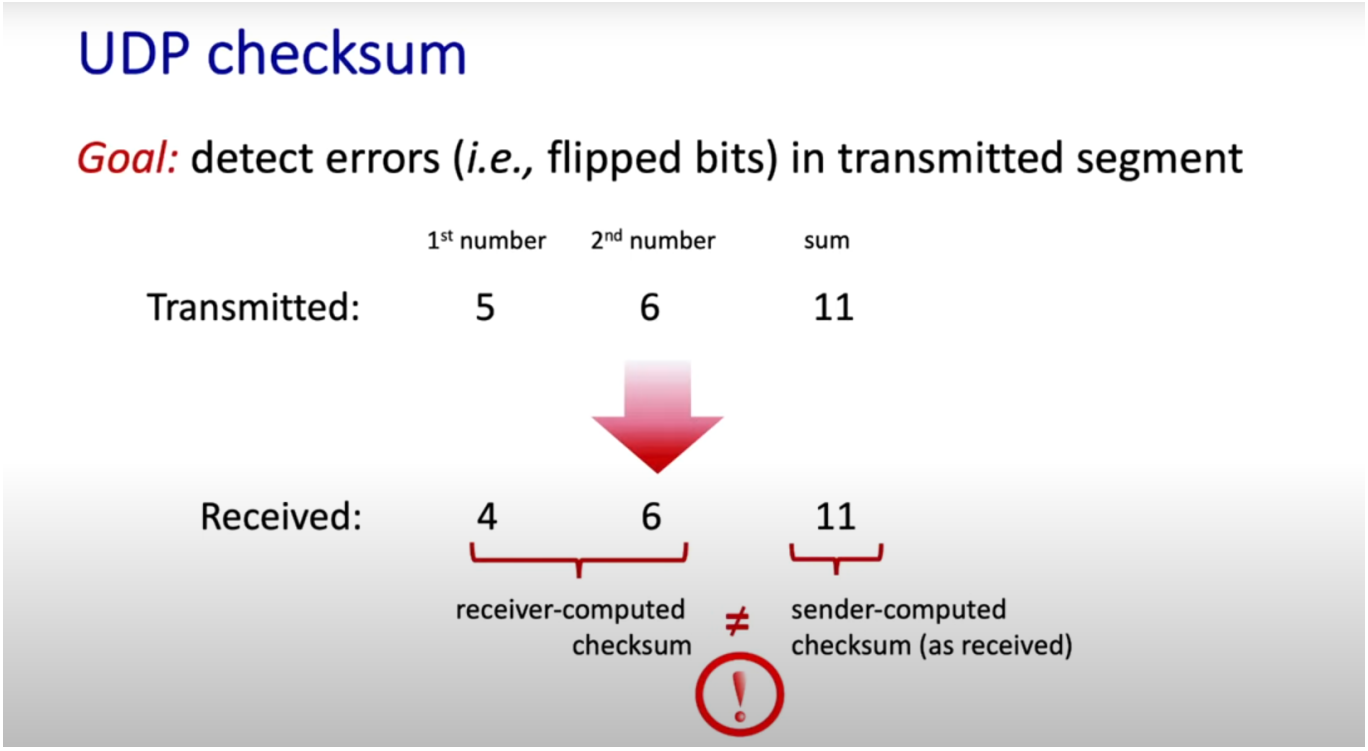
UDP Usage

Streaming Media: Applications like video streaming and online gaming use UDP as they can tolerate data loss and it allows the minimization of delays, ensuring perfect data accuracy
VoIP: For real-time audio communication
DNS Queries: DNS commonly uses UDP due to its small requests, and lost data can simply be retransmitted by the application without significant delay

UDP Checksum

The checksum is calculated by summing all 16-bit words in the UDP header and data payload. It detects any errors in the transmitted segment, if caught, it will completely discard the packet.

Sender	**Receiver**
Treat contents of UDP segments (including header fields and IP addresses) as a sequence of 16-bit integers	Compute checksum of received segment
Checksum: Addtion of segment content (one's completion sum)	Check if computed checksum equals checksum field value
checksum value put into UDP checksum field	if not equal, then there is an error, if equal, then no error (perhaps...)



Internet Checksum

An Internet Checksum is a basic error-checking algorithim used in internet protocols to verify data intergrity during transmission.

```
HOW IT WORKS

divide the data into 16-bit words, each 2 bytes long

then, all the 16-bit words are summed using one's complement arithmetic, if the
sum exceeds 16 bits, it will be "wrapped around" and added back to the lower 16
bits

After summing all the words, the resut is inverted (one's complement) to form the
checksum (each bit is flipped meaning 1 becomes 0 and vice-versa)

the checksum is then placed in the header and sent along with the data

VISUAL EXAMPLE

Suppose we have three 16-bit words

0xABCD
0x1234
0xFFFF

SUM OF WORDS

0xABCD + 0x1234 = 0xBE01
0xBE01 + 0xFFFF = 0x1DE00

Since 0x1DE00 exceeds 16 bits, we wrap the overflow and add it to the lower bits
```

```
0xDE00 + 0x0001 = 0xDE01

The one's complement of `0xDE01` is `0x21FE`, which would be the checksum.

### Summary of Calculations:

1. `0xABCD + 0x1234 = 0xBE01`
2. `0xBE01 + 0xFFFF = 0x1DE00`
3. Wrap around the carry: `0xDE00 + 0x0001 = 0xDE01`
4. One’s complement of `0xDE01 = 0x21FE`

Thus, `0x21FE` is the final checksum.
```

Connection-Oriented Transport: TCP (3.4)

What is TCP

TCP is a connection based protocol, meaning it requires a handshake / connection in order to send and receive data. This ensures that both the server and client are prepared to communicate and send packets to each other.

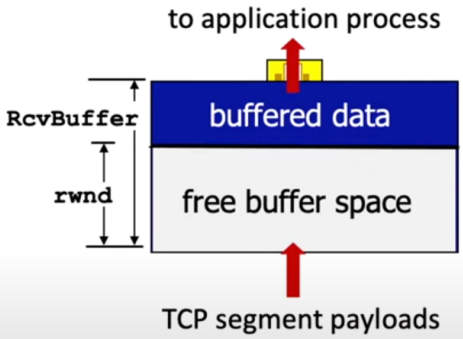
Reliable Data Transfer

TCP offers a great sense of reliability, ensuring that the data sent from and end-point to another is delivered accurately and in the correct order.
TCP achives this through its mechanisms such as;
Acknowledgements (ACKs): when the receiver successfully receives a segment, it sends an acknowledgement back to the sender.
Retransmission of Lost Data: if a segement is lost, and there is no ACK received, then TCP retransmits it again.
Sequence Numbers: Each segment has a sequence number, allowing the receiver to reasseble the data in the correct order.

Flow Control

TCP implements a system known as Flow Control which prevents the sender from overwhelming the receiver's buffer capacity. This done by using a mechanism called the Receiver Window
The receiver indicates the maximum amount of data it is ready to accept, helping to control the pace of data flow and prevent buffer overflows.

- TCP receiver “advertises” free buffer space in **rwnd** field in TCP header
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unACKed (“in-flight”) data to received **rwnd**
- guarantees receive buffer will not overflow



TCP receiver-side buffering

Congestion Control

TCP also implements congestion control mechanisms to manage network traffic and avoid overwhelming the network with excessive data.
Congestion Windows (cwnd): TCP uses cwnd to control the volume of data that can be sent on the network before receiving an acknowledgment.
Slow Start: TCP begins transmission at a slow rate to assess the network capacity, increasing the rate according to its received ACKs, the rate doubles each RTT until a threshold is reached.
Congestion Avoidance: When TCP detects potential congestion (e.g packet loss) it reduces the data transmission rate to east the network traffic and avoid further congestion.
Fast Retransmit and Fast Recovery: When a packet loss is detected, TCP reduces the sending rate and attempts to quickly retransmit the missing segment without waitng for the retransmission timer to expire.

Receiver Window vs Congestion Window

Receiver Window (rwnd)

Purpose: The purpose of the receiver window is to indicate how much data the receiver can get without being overwhelmed, based on the buffer space available in the receiver's memory
Mechanism: The receiver advertises its receive buffer in every TCP header it sends (called rwnd), so the sender know the max data it can send

Congestion Windows (cwnd)

Purpose: The congestion window is the exact opposite, as it is managed by the sending side and determines how much data it can send before waiting for an ACK. It is the estimate of the capactiy of the entwork path between the sender and receiver.
Mechanism: The congestion window's size is adjusted based on network conditions using TCP Reno or TCP Cubic (which increase or decrease cwnd) depending on the presence of absence of congestion signals
Congestion Control: The Congestion Control mechanism prevents the sender from sending too much data too quickly and causing congestion in the network

Feature	Receiver Windows (rwnd)	Congestion Windows (cwnd)

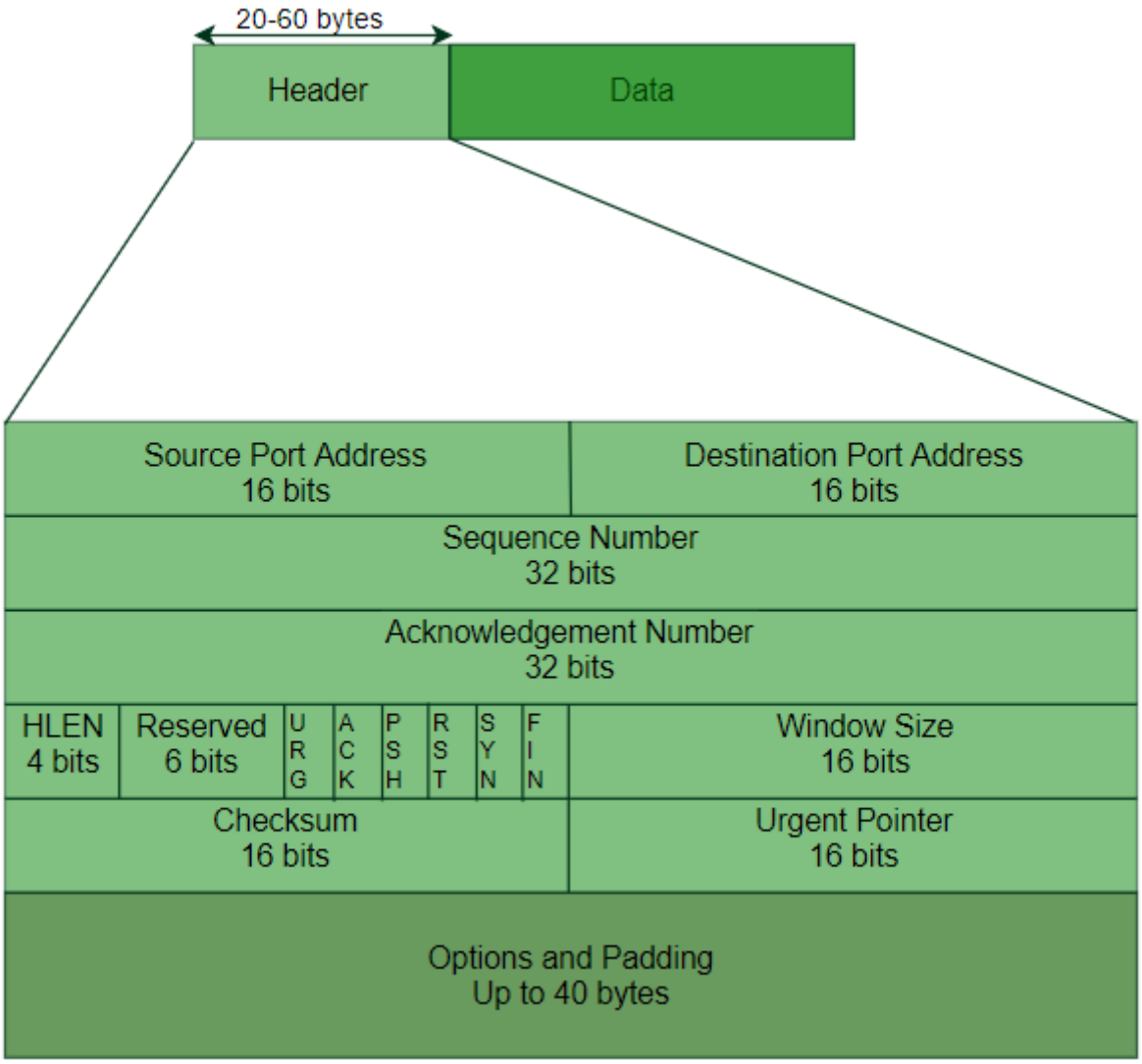
Feature	Receiver Windows (rwnd)	Congestion Windows (cwnd)
Controlled by	Receiver	Sender
Function	Flow Control	Congestion Control
Managed Based on	Receiver's Buffer Space	Network Conditions (capacity, congestion)
Advertised	Yes, included in TCP header	No, internal to the sender
Adjustments	Based on receiver's buffer space availability	Based on congestion signals

```
HOW THEY WORK TOGETHER

The actual amount of data a sender can send at any given moment is the minimum of
the rwnd and cwnd

Allowed Sending Rate = min(rwnd,cwnd)

This means the sender respects both the receiver's capacity (rwnd) and the
network's capacity (cwnd) which helps maintain a stable and reliable data transfer
without overwhelming either the receiver nor the network.
```

TCP Segment Structure

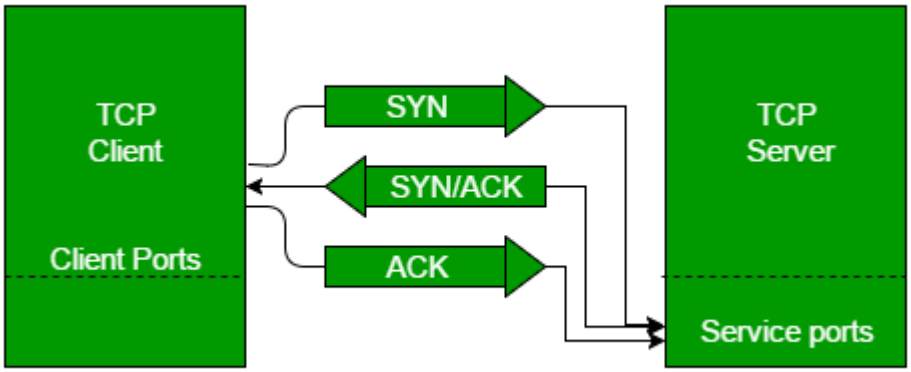
A TCP Segment consists of several fields in its header

```
Source and Destination Ports: Identifying the communicating applications
Sequence Number: Specifies the position of the data within the stream, used to
ensure ordered delivery
Acknowledgement Number: Indicates the next sequence number the sender expects,
used for acknowledgment
Flags: Control information (SYN, ACK, FIN)
Reciever Windows: Used for flow control, indicating the available buffer size on
the receiver's end
Checksum: For error checking the header and data
```



TCP Three-Way Handshake

To establish a connection, TCP requires a three-way handshake in order to send and receive data
SYN: TCP used this mechanism to indicate to the server, that the client wants to establish a connection
SYN-ACK: The server then responds with a SYN-ACK segment, acknowledging the client's request and signaling it's ready to communicate
ACK: Cliend sends an ACK to confirm the connection, completing the handshake



Connection Teardown

Once communication is complete, TCP uses a four-step process to gracefully close the connection, this process involves a series of FIN (finish) and ACK segments to ensure all data has been sent and received before closing the connection.

TCP Sequence Number

The TCP Sequence Number plays a crucial part in helping the sender and receiver ensure data is transmitted accurately and in the correct order.

Intial Sequence Number (ISN): Each TCP connection begins with a unique sequence number known as the ISN, once a connection is established, both the client and server select their own ISN, starting their own sequence numbers for data transfer.

Byte-Level Tracking: TCP is a byte-oriented protocol, meaning each byte in the data stream has a sequence number, not just each packet. The sequence number in the TCP header represents the first byte of data in that specific TCP packet.

Incrementing the Sequence Number: As data is transmitted, the sequence number increases by the number of bytes sent, for example, if the initial sequence number is 1000, and 500 bytes of data are sent in the first segment, the sequence number in the next segment will be 1500.

Reliability: Sequence numbers allow the receiver to reorder out-of-sequence packets and detect missing data. When the receiver acknowledges received data, it sends back an ACK number indicating the next expected sequence number. This allows the sender to knows up to which byte the data has been successfully received.

Duplicates and Retransmissions: If packets are lost, delayed, or duplicated, sequence numbers help identify them.

Principles of Reliable Data Transfer (RDT) (3.5)

RDT Goals

The main goal of a Reliable Data Transfer protocol is to ensure data is delivered from the sender to the receiver without any errors and in correct order.

Finite State Machines (FSM) for RDT

FSMs are introduced as a way to visualize the actions of the sender and receiver under different network conditions. FSMs help illustrate how each side should respond to different events, such as receiving data correctly, encountering an error or receiving duplicate media.

Building Reliable Data Transfer Protocols

There have been multiple released versions of RDT, each increasing its services to handle challenging network conditions

RDT 1.0 - Perfectly Reliable Channel

Assumes a completely reliable network, where packets are never lost, corrupted or reordered. No special mechanisms are needed for reliability, so the sender simply sends data and the receiver receives it.

```
SENDER:

State: "Wait for data from above"
The sender remains in this state, waiting for application data to send.
When data arrives, the sender packages it and sends it to the receiver.

RECEIVER:

State: "Wait for packet from below"
The receiver stays in this state, waiting for a packet to arrive.
When a packet arrives, it’s always correct and in order, so the receiver simply
delivers the data to the application layer.
```

RDT 2.0 - Channel with Bit Errors

Accounts for the possibility of data corruption during transmission. Error detection (such as checksums) are used, as well as ACKs and Negative Acknowledgements (NAK) are also introduced. The sender retransmits the data if it receives a NAK. This version is improved in RDT 2.1 and 2.2 to handle duplicate packets and to remove the NAK by using only ACKs with sequence numbers.

```
SENDER:

State 1: "Wait for data from above"

When data arrives from the application layer, the sender packages it with a
checksum to detect corruption.
The sender transmits the packet and transitions to State 2.

State 2: "Wait for ACK or NAK"

If the sender receives an ACK, indicating the packet arrived correctly, it returns
to State 1 to send the next packet.
If it receives a NAK, indicating that the packet was corrupted, it retransmits the
packet and remains in State 2 until an ACK is received.

RECEIVER:

State 1: "Wait for packet from below"
When a packet arrives, the receiver checks for corruption using the checksum.
If the packet is correct, it sends an ACK and delivers the data to the application
layer.
If the packet is corrupted, it sends a NAK back to the sender, prompting
retransmission.
```

RDT 3.0 - Channel with Packet Loss

Adds mechanisms to deal with packet loss, which is not addressed in previous versions. Timeouts are also introduced, where the sender waits for an ACK for a certain period of time, if the sender does not receive an ACK within the timeout perios, it assumes the packet was lost and retransmits it. This approach is known as "stop and wait" as only 1 packet can be in transit at any time

```
SENDER:

State 1: "Wait for data from above"

When data is ready, the sender packages it, sends it, and starts a timer.
The sender then moves to State 2.
State 2: "Wait for ACK with timer"

If an ACK arrives before the timer expires, the sender knows the packet arrived
successfully, stops the timer, and returns to State 1.
If the timer expires before receiving an ACK, the sender retransmits the packet
and restarts the timer, remaining in State 2 until an ACK arrives.

RECEIVER:

State 1: "Wait for packet 0"

If a packet with sequence number 0 arrives and is not corrupted, the receiver
sends an ACK and moves to State 2.
If corrupted, the receiver discards the packet and waits.
State 2: "Wait for packet 1"

The receiver repeats the same process for packet 1.
```

Pipelined Reliable Data Transfer Protocols

As stop and wait protocols are inefficient for high-bandwidth or long-delay networks, the idea of Pipelining is introduced, this allos multiple packets to be send before receiving ACKs. Pipelines consist of two primary protocols;

Go-Back-N (GBN):

Allows the sender to send multiple packets up to a window size (rwnd, cwnd) represented by the letter `N` which is sent without waiting for an ACK after each one. If a packet is lost, all subsequent packets in the window are retransmitted.

Selective Repeat (SR)

Similar to GBN, but instead of retransmitting all the packets after a loss, it only retransmits the specific missing packets. This is much more effective than GBN, but requires more complex processing including understanding what packets have been acknowledged.

Trade-offs in Reliable Data Transfer Protocols

Protocol	Pros	Cons
Stop-and-Wait	Simple to use, easy to track due to ACK mechanism and is reliable	very low efficiency due to waiting for each ACK before sending a new packet, bandwidth is often underutilized
Go-Back-N	Allows multiple packets to go through before waiting for an ACK, improves throughput, if errors occur, it simply retransmits all the packets.	If a single packet is lost, all packets must be resent which can wast bandwidth
Selective Repeat	Compared to GBN, it only retransmits specific packets which are lost or corrupted, allowing greater efficiency, maintains a higher effective data transfer rate.	SR requires complex implementation and more resources and memory to account for all the specific packets which are lost or corrupted.

Unreliable Channels

An unreliable channel refers to a communication link that is prone to errors, such as corruption, packets loss, etc.

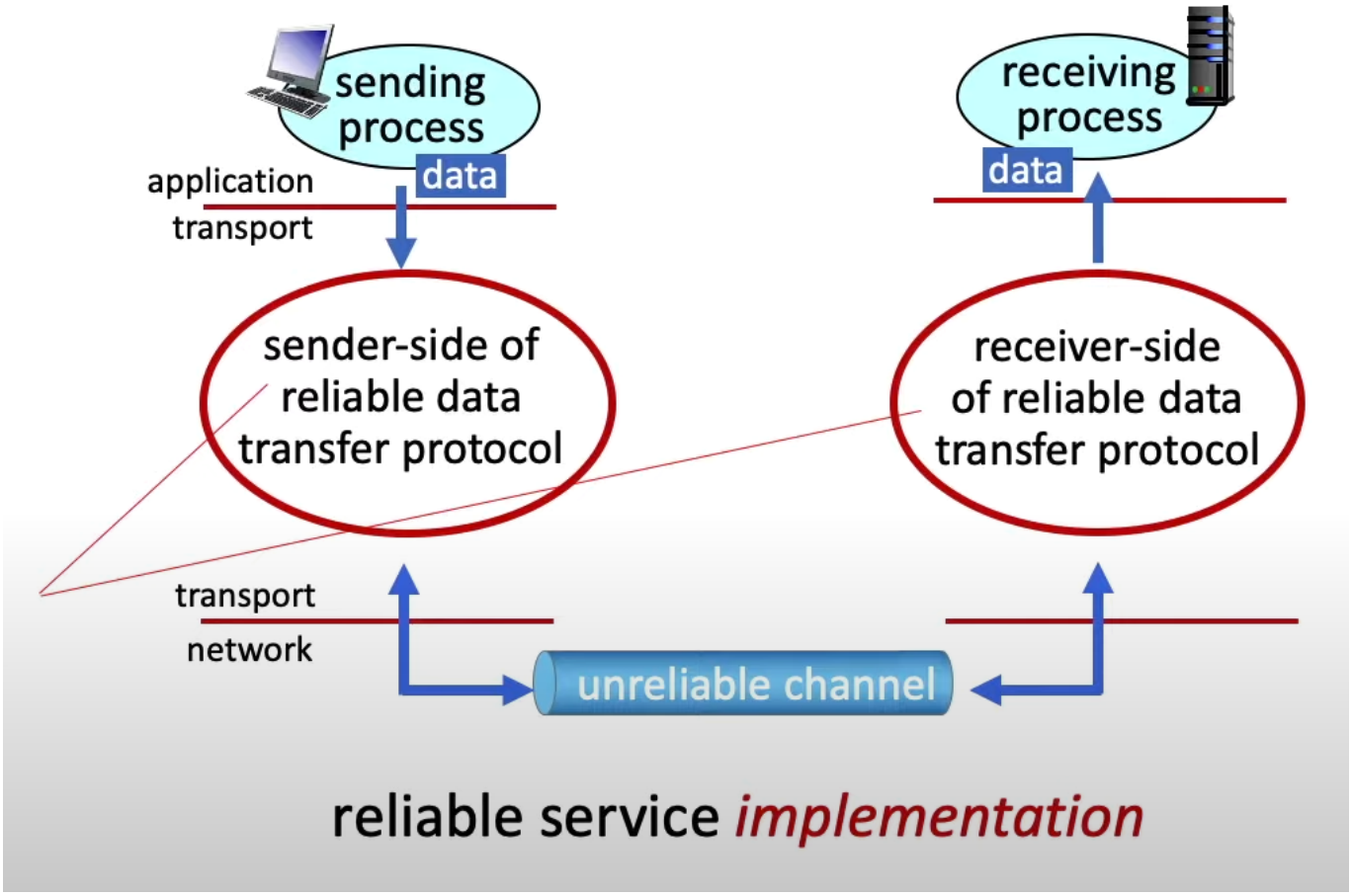
Key characteristics of an unreliable channel

```
Bit Errors: Some of the bits in the packet might get corrupted during transmission
due to noise or inteference in the communication channel.
Packet Loss: Entire packets may be lost during transmission, meaning they never
reach the receiver.
Out-of-Order Delivery: Packets might not arrive in the order they were sent.
Duplicate Packets: Some packets may be received more than once.
```

These issues make it hard to guarantee an reliable communication, thus, RDT protocols (2.0-3.0) are intorduced to provide mechanisms that detect and help solve these errors.

How protocols handle these porblems

```
Using checksums for error detection (verifying a packet has been corrupted).
Implementing retransmissions when packets are lost and/or corrupted.
Using sequence numbers to differentiate between new and retransmitted packets and
handle packets that are out of order.
```



Congestion Control Mechanisms (and with TCP) (3.6-3.7)

Understanding Network Congestion

Network congestion occurs when too much data flows through the network, exceeding its capacity. This can lead to packet delays, buffer overflows, and packet losses. Making an efficient congestion control mechanism is crucial, and with the help of TCP, it further allows us to detect for signs of congestion and can dynamically adjust the data transmission rate to avoid overloading the network.

TCP Congestion Control Mechanisms

1. Slow Start

A Slow Start is the initial phase of TCP's congestion control where the sender starts with a low transmission rate, gradually increasing it to examine the network's capactiy.

Congestion Window (cwnd): TCP uses the `cwnd` mechanism to ensure that the sender does not send a large amount of data all at once to the receiver. Rather it uses its Slow Start ability and analyzes the receiver's network and capacity conditions, thus beginning with a small amount of packets and gradually increasing by doubling the `RTT` each time, examining the conditions and its limit.

Threshold (ssthresh): The Slow Start threshold (`ssthresh`) is the point at which TCP transitions from exponential growth, to linear growth to avoid overwhelming the network.

Example: If the initial `cwnd` is set to one segment, after one RTT, it doubles to two RTT, then four, then eight, and so on, until reaching `ssthresh` or encountering packet loss, signaling potential congestion.

2. Congestion Avoidance

Once `cwnd` reaches `ssthresh`, TCP transitions into Congestion Avoidance Mode, where the rate of packet flow slows from exponential to linear growth. Instead of doubling after each ACK, TCP adds one segment to `cwnd` for each `RTT`. This phase carefully monitors for congestion to maintain a stable flow rate.

Additive Increase: In congestion avoidance, `cwnd` increases linearly, which is called 'additive increase' TCP aims to fill up available capactiy gradually without overloading the network.

Multiplicative Decrease: Upon detecting congestion (through signs of packet loss), TCP reduces `cwnd` by half, which is called 'multiplicative decrease' This reduction helps quickly ease the load on the network.

Example: If `cwnd` is at 10 segments and packet loss occurs, TCP will cut `cwnd` to 5 segments (MD), gradually resuming linear growth stabilizing.

3. Fast Retransmit

Fast retransmit is a mechanism that helps TCP quickly recover from lost packets without waiting for a timeout. If the sender receives three duplicate ACKs (indicating out-of-order packets due to lost data), it retransmits the missing segment immdediately.

Example: If TCP sends packets 1, 2, and 3, but packet 2 is lost, the receiver will repeatedly acknowledge packet 1 until packet 2 arrives. When the sender sees three duplicate ACKs for packet 1, it retransmits packet 2 right away, reducing wait time.

4. Fast Reovery

Fast recovery complements fast retransmit by allowing TCP to avoid entering slow start after retransmitting lost segments. Instead, `cwnd` is set to `ssthresh` (half of the previous `cwnd`), and the sender resumes linear growth from there.

Duplicate ACKs: Each duplicate ACK indicates successful receipt of data after the missing segment, enabling TCP to adjust `cwnd` based on network conditions even during recovery.

Costs of Congestion and Inefficiencies

Although TCP introduces many mechanisms that can counteract congestion, without the proper setups and precautions, congestion can produce negative impacts, such as waster resources and reduced throughput. When congestion leads to packet loss, TCP retransmits those packets, causing a 'congestion collapse' if unchecked. This is why TCP congestion control mechanisms are essential for managing network resources.

Packet Loss: When routers and network devices run out of buffer space, packets are dropped, requiring the sender to retransmit them. This increases network load as the same data is sent multiple times, wasting bandwidth.

Retransmission Costs: Retransmitting packets due to loss conumes extra bandwidth and resources that could be used for new data. TCP's retransmission can also create a 'congestion collapse' if too many retransmission

occur simultaneously, further congesting the network.

Congestion Collapse: Is a severe network condition in which the increased network traffic, results into a dramatic decrease in network throughput, to the point where almost no useful data is transmitted. This happens when excessive packet retransmissions conume the available bandwidth without actually delivering new or useful infromation, creating a feedback loop of congestion and retransmission that worsens the situation.

Example: During a high-traffic event (e.g., streaming a popular live sports event), many users attempt to access the same resources, potentially overloading the network. TCP's congestion control mechanisms help adjust traffic flow to avoid further delays and packet loss during such high-demand periods.

Real-World Implications of Congestion Control

TCP congestion control is fundamental for maintaining efficient data transfer on the internet. By balancing the need for speed and stability, these mechanisms ensure that network resources are used effecctively without causing significant delay or data loss, crucial for applications like video streaming, online gaming, and cloud services.

TCP Congestion Control Mechanisms and Algorithms

TCP also uses several additional algorithms to handle congestion based on different network conditions and scenarios

1. TCP Reno and TCP Tahoe

TCP Tahoe: Implements a basic version of a slow start and congestion avoidance but always reduces `cwnd` to 1 segment after packet loss, making it conservative in handling congestion.

TCP Reno: Introduces fast retransmit and fast recovery, making it more efficient by not reducing `cwnd` to its minimum. Reno's fast recovery mechanism allows for quicker recover from minor packet losses.

Example: In a lightly congested network, TCP Reno can recover from single packet losses without significant throughput loss, making it effective in many scenarios.

2. TCP NewReno

TCP NewReno improves on TCP Reno by addressing multiple lost segments in a single window. It extends fast recovery by handling partial acknowledgements, allowing TCP to recover from multiple losses without dropping `cwnd` drastically.

3. TCP Cubic and TCP BBR

TCP Cubic: A modern variant often used in high-speed networks (e.g Linux Systems), where `cwnd` growth follows a cubic function. This shape allows for rapid growth at lower congestion levels, maximixing throughput, then slow growth near congestion thresholds.

TCP BBR (Bottleneck Bandwidth and RTT): Focuses on achieving high throughput and low latency by measuring actual available bandwidth and RTT rather than reacting to packet loss. It's popular in cloud computing and streaming applications

Security in TCP and UDP

Neither TCP nor UDR includes a built-in encryption or cyersecurity features to secure packet data. Both protocols were designed to provide reliable or quick transport of data, not to offer data confidentiality or integrity by themselves. Security for TCP and UDP packets is typically implemented through external protocols and systems layered on top of them.

TCP Security

TCP provides features such as sequencing and ackownledgements for reliable data delivery, but these features do not protect the data itself from interception or tampering.

TCP does not include encryption within its protocol. Any encryption applied to data sent over TCP must be done by higher-layer protocols (e.g the application layer) or by security protocols like TLS (Transport Layer Security), which is commonly layered over TCP in protocols such as HTTPS

TLS (Transport Layer Security): TLS encrypts data transmitted over TCP, providing confidentiality, integrity, and authenticity, for example, HTTPS uses TLS over TCP to secure web traffic, while SMTP and STARTTLS secures email.

UDP Security

UDP is a simpler protocol than TCP and is often used where speed is more important than reliability, such as in video streaming and gaming.

UDP also lacks encryption or security features; it does not have mechanisms for retransmission, ordering, or even error-checking beyond a basic checksum

DTLS (Datagram Transport Layer Security): DTLS is a variant of TLS that secures UDP traffic, enabling encrypted communication for protocols that use UDP, such as VoIP and online gaming.

Externam Systems for Packet Security

TCP and UDP security is often managed by additional network security systems that secure data as it moves through the network.

VPNS (Virtual Private Networks): A VPN encrypts the entire connection (including TCP and UDP packets) between a device and a remote network, providing an extra layer of security.

IPsec (Internet Protocol Security): IPsec can sure TCP and UDP packets at the IP layer through encryption and integrity checks, commonly used in VPNs for encrypting all IP-based communication between devices.

Firewalls and Intrusion Detection Systems (IDS): Firewalls and IDS solutions monitor TCP and UDP traffic for suspicious patters, limiting the exposure to attacks like packet sniffing or spoofing.

Cybersecurity Limitations

Without added security protocols like TLS or DTLS, both TCP and UDP packets can be **vulnerable to interception** (sniffing) or **modification** (man-in-the-middle attacks).

Port Scanning: Attackers can scan TCP and UDP ports to identify open services, potentially leading to attacks on exposed applications.