1902EC505 - COMPUTER NETWORKS

Prof S.Praveen Kumar M.E,(Ph.D), EMCAA, MISTE, IAENG.,
Assistant Professor/CSE
E.G.S Pillay Engineering College, Nagapattinam

Course Outcomes

- At the end of this course students can able to

UNIT 4 - TRANSPORT LAYER

- Functionalities of Transport Layer
- Transport Layer Services
- Elements of Transport Layer Protocols
  - UDP
  - TCP
- Congestion Control
- Congestion Avoidance
- Transport for Real Time Applications(RTP)

FUNCTIONALITIES OF TRANSPORT LAYER

- The transport layer is responsible for process-to-process delivery of the entire message.
- A process is an application program running on a host.
- The network layer oversees source-to-destination delivery of individual packets, it does not recognize any relationship between those packets
- The transport layer, ensures that the whole message arrives intact and in order, overseeing both error control and flow control at the source-to-destination level.

FUNCTIONS OF TRANSPORT LAYER

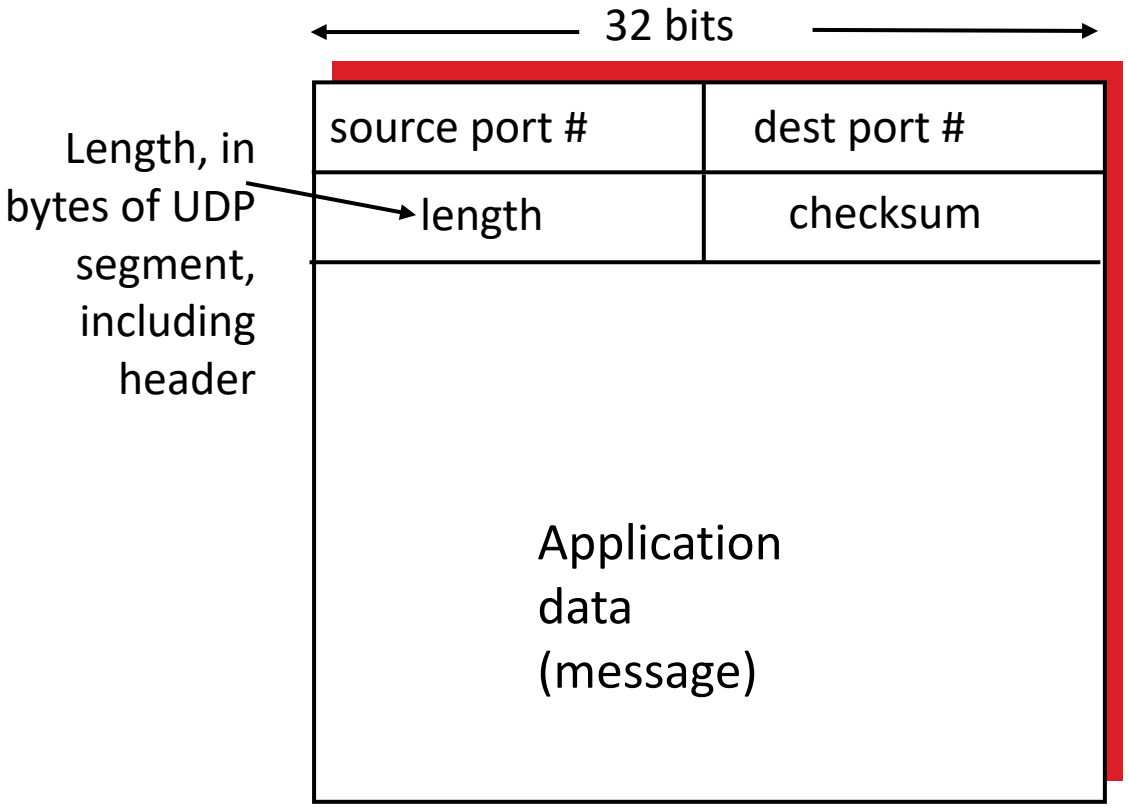
- **1.Service-point addressing:** The transport layer header includes a type of address called a *service-point address (or port address).*
- The transport layer gets the entire message to the correct process on the destination computer by the help of this address.
- **2.Segmentation and reassembly:** A message is divided into segments, containing a sequence number. These numbers enable the transport layer to reassemble the message correctly upon arriving at the destination.
- **3.Connection control:** The transport layer can be either connectionless or connection oriented. A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine.
- A connection oriented transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data are transferred, the connection is terminated.
- **4.Flow control:** Flow control at this layer is performed at end to end
- **5.Error control:** Error control at this layer is performed process-to-process rather than across a single link.

- **TRANSPORT LAYER SERVICES:**
- There are two types of services provided by transport layer. They are given below:
  - **Connectionless transport service**
    - **UDP:** User Datagram Protocol, used for connectionless transport service.
  - **Connection oriented transport service**
    - **TCP:** Transmission Control Protocol used for connection oriented transport service

UDP – USER DATAGRAM PROTOCOL

- It is a connectionless transport layer protocol.
- There is no handshaking before two processes start to communicate.
- It provides unreliable data transfer service.
- When a process sends a message into a UDP socket, UDP provides no guarantee that the message will reach the receiving process.
- Messages do arrive to the receiving process may arrive out of order.
- UDP does not include congestion control mechanism.
- So sending process can pump data into a UDP socket at any rate.

- Internet telephony is make using of UDP.
- Multicasting is supported by UDP.



UDP segment format

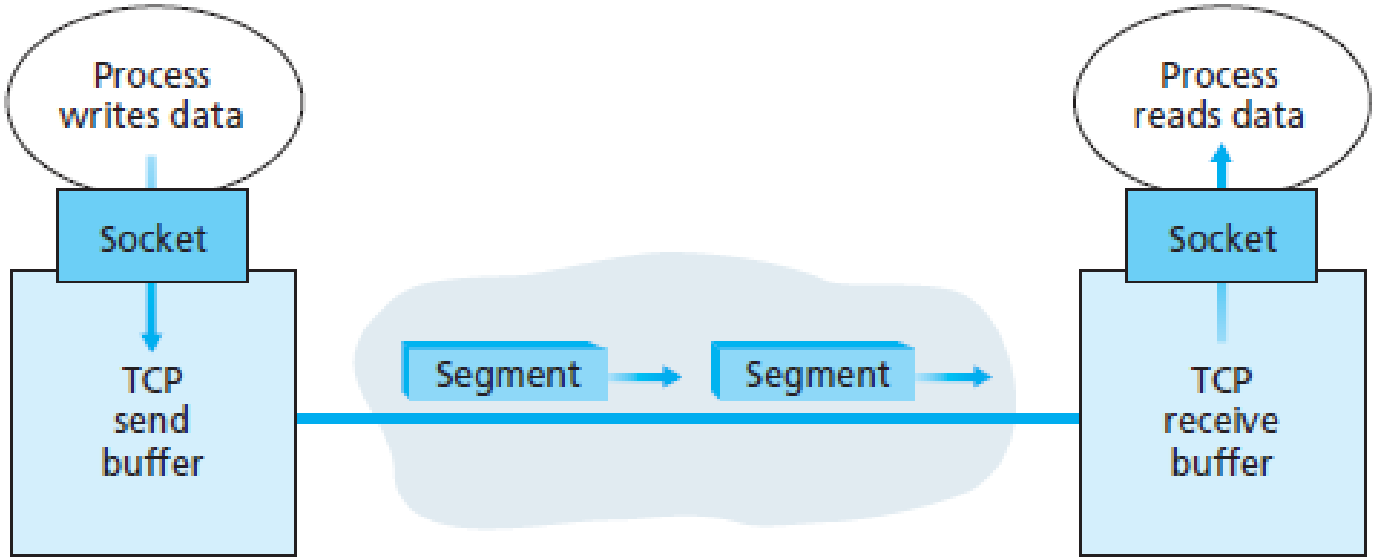
TCP-TRANSMISSION CONTROL PROTOCOL

- TCP is a connection oriented reliable transport layer protocol.
- E-Mail, Web and File transfer all use TCP. Because it provides reliable data transfer service and guarantees that all the data will go its destination.
- TCP is said to be connection oriented because, before one application process can begin to send data to another, two processes must first **"Handshake"** with each other(Connection Establishment).
- Before sending data after connection establishment both client and server establishes TCP variables.
- TCP Variables: (i) Seq #s

  (ii)Buffers,Flowcontrol info(ex: RcvWindow)

- Once a TCP connection is established, the two application processes can send data to each other.
- **Full Duplex Service:** A TCP connection provides full duplex service.
- If there is a connection between Process A on one host and Process B on another host, then the data can flow from both sides of Process A and Process B at the same time.
- **Point to Point Connection:** A TCP connection is always point to point.
- That is between a single sender and single receiver.
- Multicasting is not possible with TCP.

- **TCP Send & Receive Buffers:**
- **Client Process:** Process that initiates connection
- **Server Process:** Process that establishes connection
- **Data Transfer Steps:**

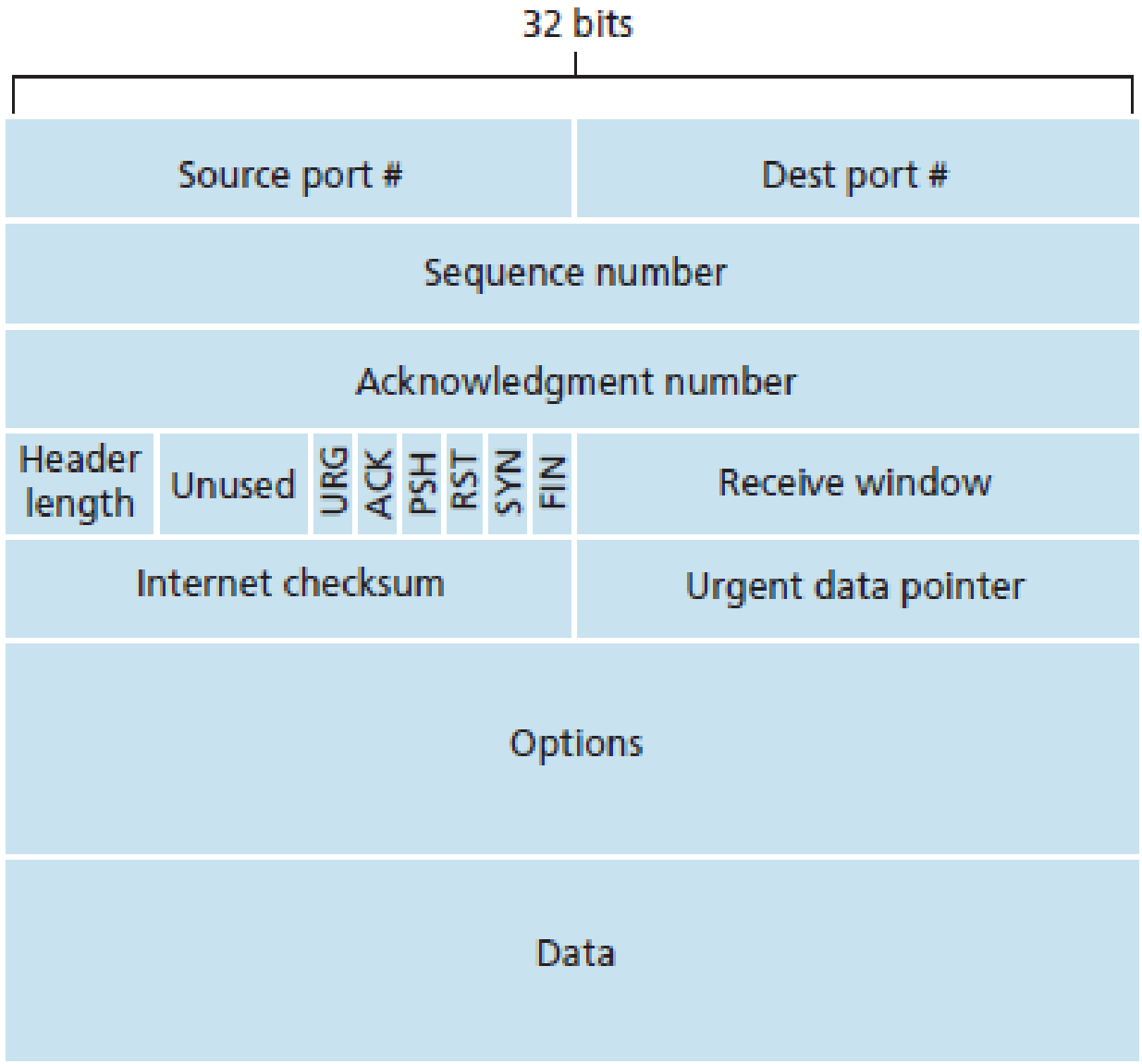


TCP Send and Receive Buffers

- Each side of connection has its own send and receive buffers.

- 1. The client passes a stream of data through the socket.
- 2. Once the data passes through the client socket, the data is now in the hands of TCP running in the client.
- 3. TCP directs this data to the connections send buffer.
- From time to time TCP will grab **chunks of data** from the send buffer.
- 4. The maximum amount of data that can be grabbed and placed in the segment is limited by **MAXIMUM SEGMENT SIZE(MSS).**
- 5. The maximum segment size(MSS) is set by **MAXIMUM TRANSMISSION UNIT(MTU).**
- 6. When TCP receives a segment at the other end, the segments data is placed in the TCP connections receive buffer.
- 7. The application reads the stream of data from this buffer.

TCP Segment Structure:

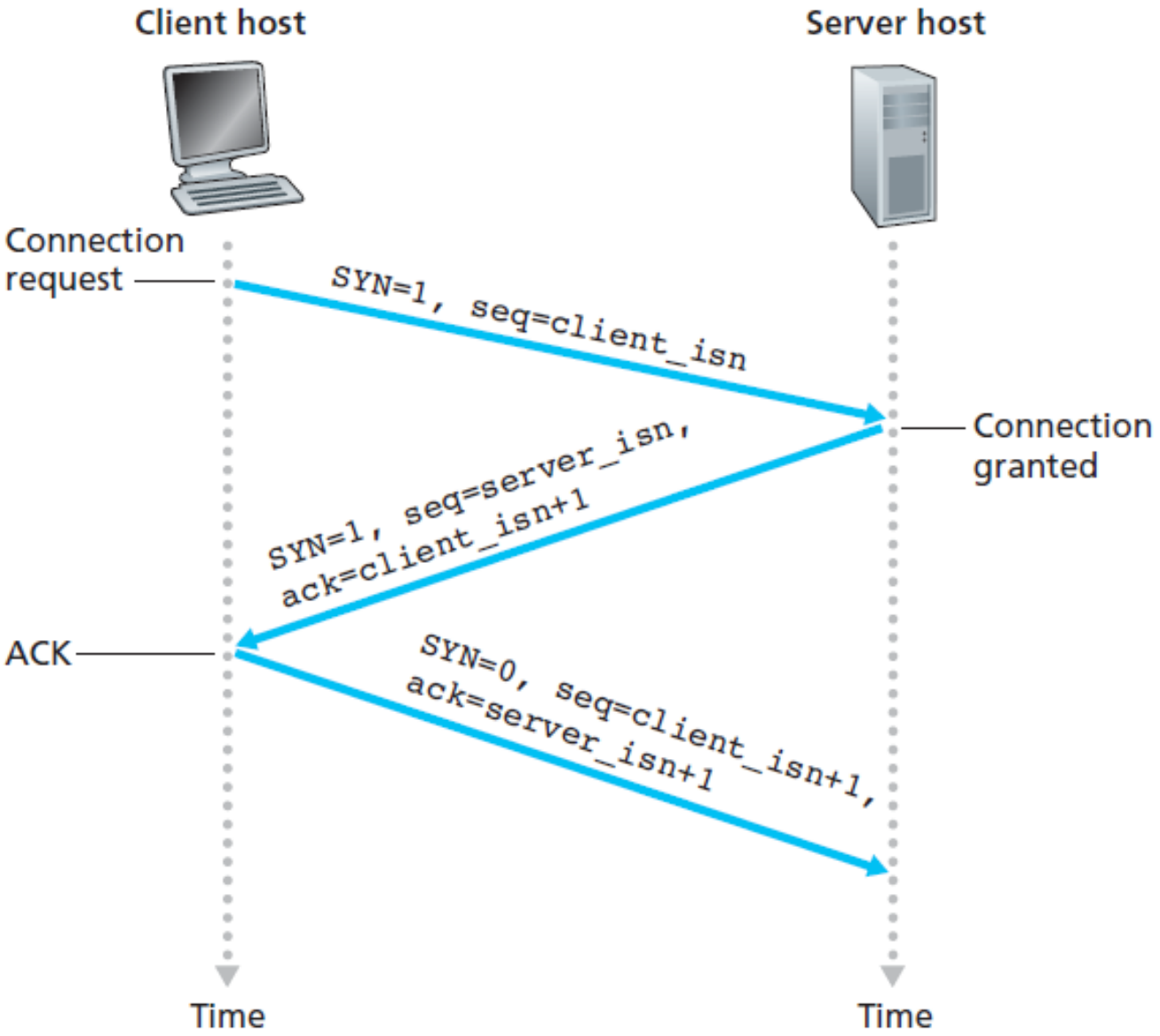


- The 32 bit sequence number and acknowledgement number field are used by the TCP for reliable data transfer service.
- The 16 bit receive window field is used for flow control. It is used to indicate the no of bytes that a receiver is willing to accept.
- The 4 bit header length field specifies the length of the TCP header.
- The flag field contains 6 bits.
- The ACK bit is used to indicate that the variable carried in the acknowledgement field is valid
- That is it contains ack for a segment that has been successfully received.
- The RST, SYN and FIN bits are used for connection setup and teardown.

- **TCP Connection Management:**
- **Establishing TCP Connection(3 way handshaking):** Consider a process running in one host(client) wants to initiate a connection with another process in another host(Server).
- The client application process first informs the client TCP that it wants to **establish a connection** to a process in the server.
- The TCP in the client then proceeds to establish a TCP connection with the TCP in the server in the following manner:

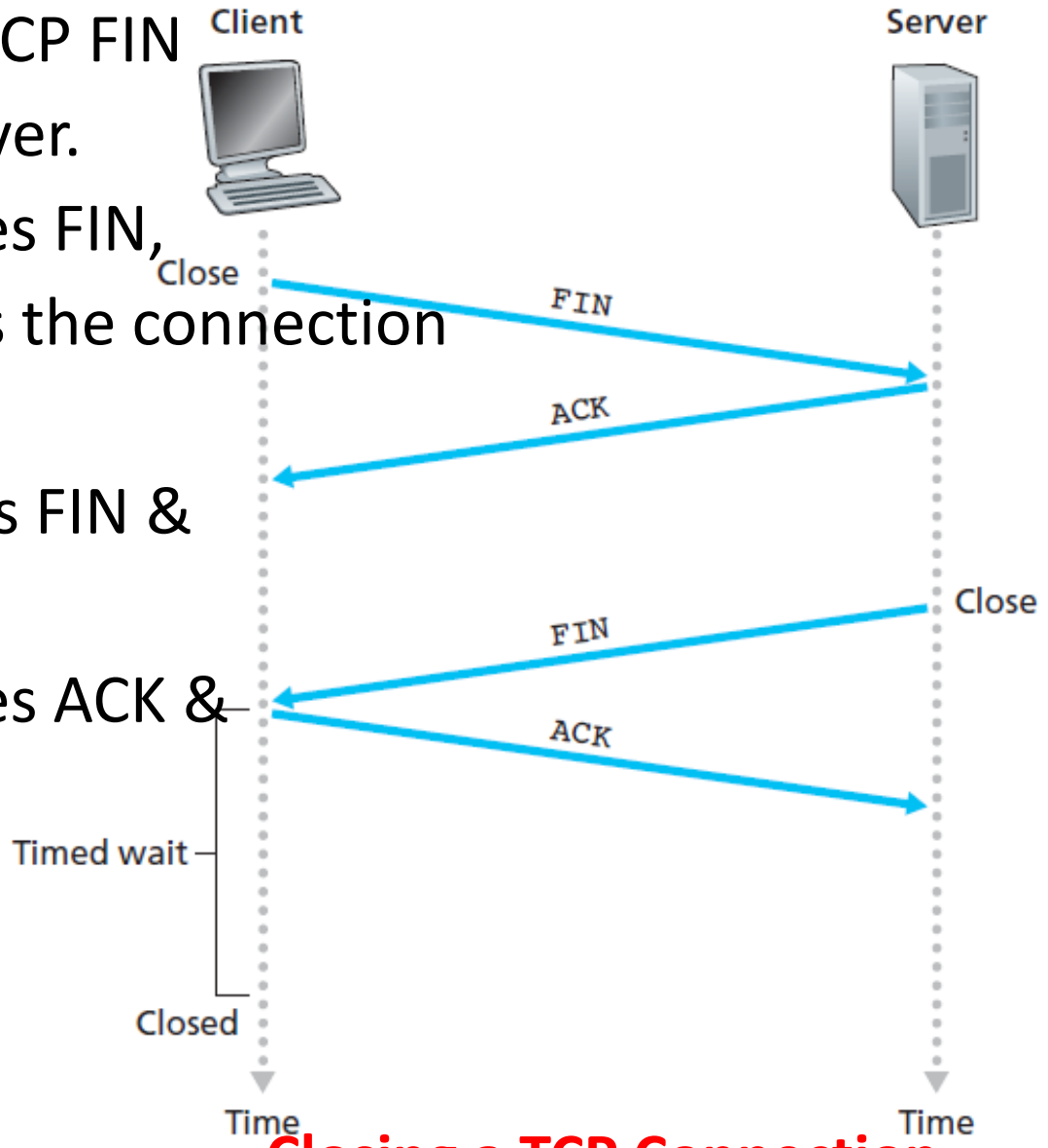
- **Step 1:** The client side TCP first sends a special TCP segment to the server side TCP, which contains no data.
- The SYN bit is set to 1 and it is referred as **SYN segment**.
- The client randomly chooses an initial sequence number client_isn and puts this number in the sequence number field of the initial TCP SYN segment.
- **Step 2:** Once the TCP SYN segment arrives a the server host, the server extracts the TCP SYN segment, allocates the TCP buffers and variables to the connection and sends a connection granted segment to the client TCP.
- This segment contains 3 important information such as
- (i)SYN bit is set to 1
- (ii)The acknowledgement field of the TCP segment is set to client_isn+1.

- The server chooses its own initial sequence number server_isn and puts this value in the sequence number field of TCP segment header.
- This connection granted segment is saying that I received your initial sequence number client_isn. I agree to establish this connection.
- My own initial sequence number is server_isn.
- The connection granted segment is sometime referred as **SYNACK segment.**
- **Step 3:** Upon receiving the SYNACK segment the client also allocates buffers and variables to the connection.
- The client host then sends the last segment that acknowledges the server's connection granted segment by putting the value server_isn+1 in the Ack field of the TCP segment header.
- The SYN bit is set to '0', because the connection is established.



TCP 3 way handshaking

- **Connection Tear Down(Closing a Connection):**
- **Step 1:** Client sends TCP FIN control segment to server.
- **Step 2:** Server receives FIN, replies with ACK, closes the connection and sends FIN.
- **Step 3:** Client receives FIN & replies with ACK.
- **Step 4:** Server receives ACK & connection is closed.

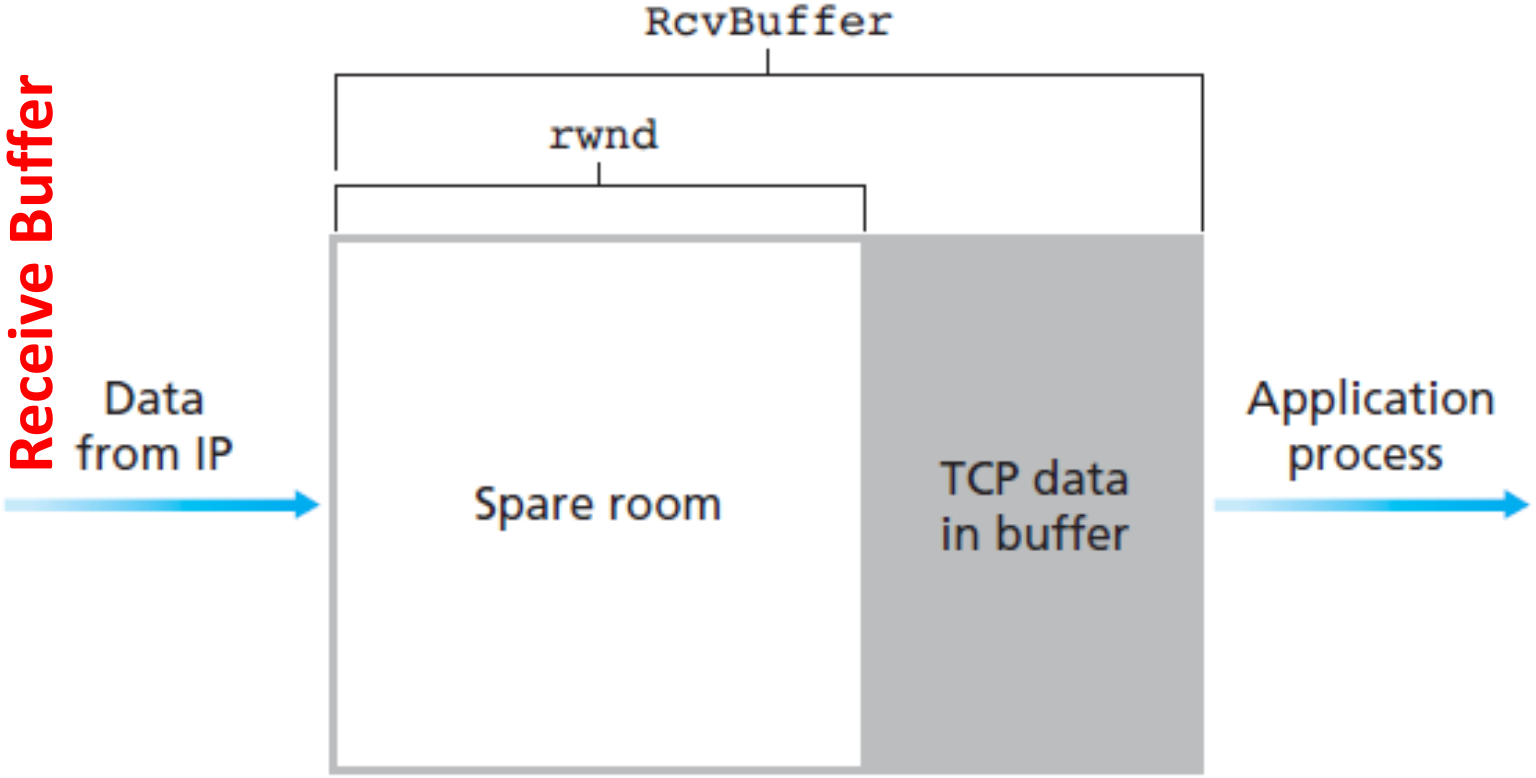


Closing a TCP Connection

TCP FLOW CONTROL

- The hosts on each side of the TCP Connection sets send and receive buffer.
- When the TCP connection receives bytes that are correct and in sequence, it places the data in receive buffer.
- The associated application process will read data from this buffer.
- Mostly, the receiving application may be busy with some other task.
- If the application is relatively slow at reading the data, the sender can very easily overflow the receiver's receive buffer by sending too much data too quickly.

- **Speed Matching Service:** TCP provides a flow control service to eliminate the possibility of the sender overflowing the receiver's buffer by speed matching service.
- **Matching the rate at which the sender is sending against the rate at which the receiving application is reading.**
- **Receive Window(rwnd):** It is used to give the sender an idea about how much free buffer space is available at the receiver.



- **Example with a context of file transfer:**
- Consider that host A is sending a large file to host B.
- Host B allocates receive buffer to this connection and denotes its size by **RcvBuffer.**
- From time to time the application process in Host B reads from the buffer and define the following variables:
- **LastByteRead:** The sequence number of the last byte in the data stream read from the buffer by the application process in Host B.
- **LastByteRcvd:** The sequence number of last byte in the data stream that has arrived from the network and has been placed in the receive buffer at Host B.
- TCP is not permitted to overflow the allocated buffer.
- LastByteRcvd – LastByteRead ≤ RcvBuffer

- The receive window denoted as **RcvWindow** is set to the amount of spare room in the receive buffer.
- RcvWindow = RcvBuffer – [LastByteRcvd – LastByteRead]
- TCP in Host B tells TCP in host A about how much spare room it has in the connection buffer by placing its current value of RcvWindow in the receive window field.
- Host A in turn keep track of two variables namely **LastByteSent and LastByteAcked.**
- The difference between the above two variables is the **amount of unacknowledged data**, Host A has to be sent to Host B.
- By keeping the amount of unacknowledged data less than the value of RcvWindow, Host A assures that it is **not overflowing the receive buffer at host B.**
- **LastByteSent – LastByteAcked ≤ RcvWindow**

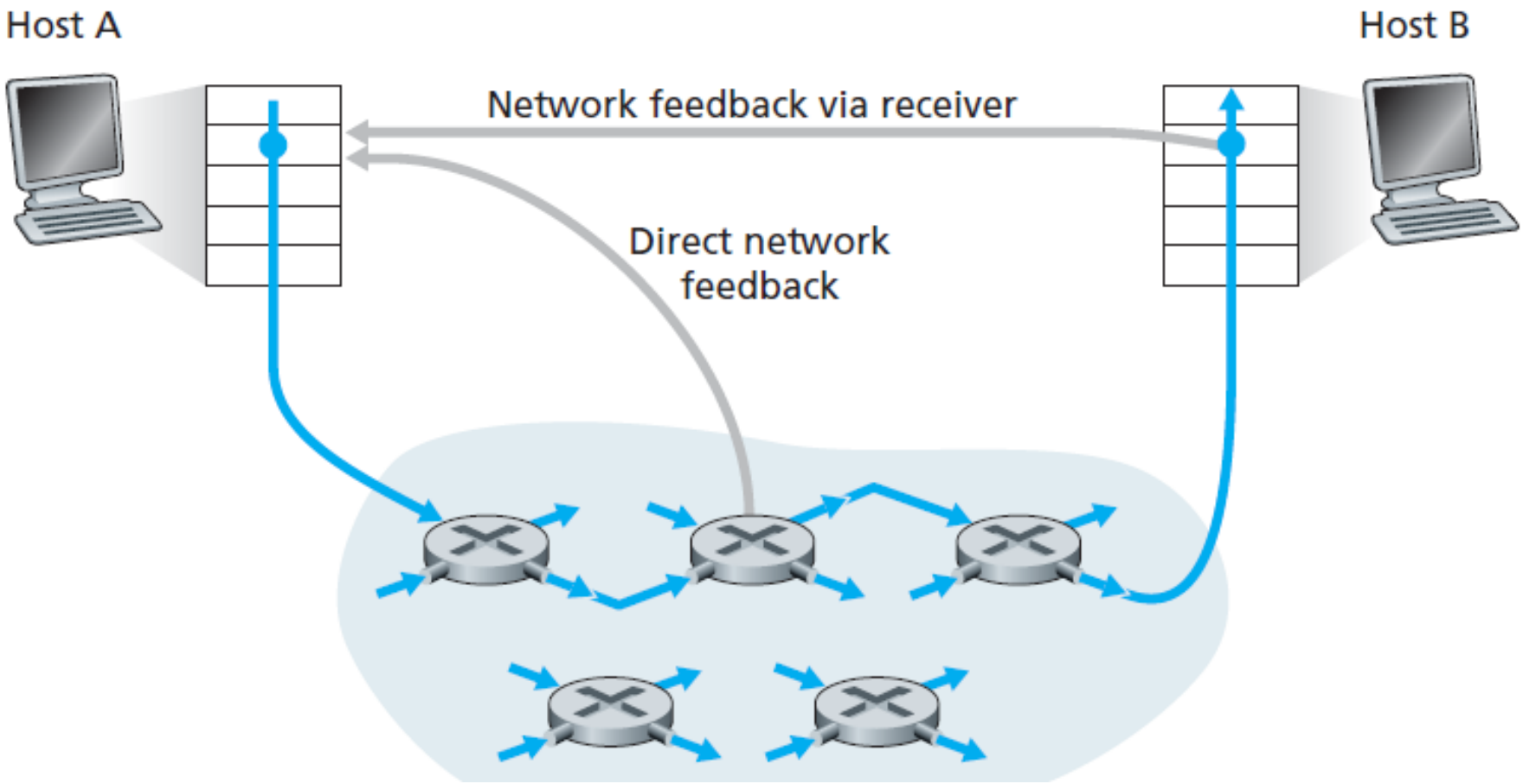
Amount of Unacknowledged Data

CONGESTION CONTROL

- **Congestion:** It refers to a network state where a node or link carries so much data that it can't handle and resulting in queuing delay or packet loss and the blocking of new connections.
- Congestion occurs when bandwidth is insufficient and network data traffic exceeds capacity.
- **Approaches to congestion control:**
- (i)End to End Congestion Control
- (ii)Network Assisted Congestion Control

- **(i)End to End Congestion Control:** In this approach, the network layer provides no support to transport layer for congestion control.
- The presence of congestion in the network must be inferred by the end systems, based only on observed network behavior like packet loss and delay.
- TCP must necessarily take this end to end approach towards congestion control, since network layer provides no feedback to the end systems regarding network congestion.
- TCP segment loss is taken as indication of network congestion.

- **(ii)Network Assisted Congestion Control:** In this approach the network layer components such as routers provide explicit feedback from the network to sender in two ways as follows:
- **WAY 1:** Direct feedback is sent from a network router to the sender.
- This form of notification typically takes the form of a choke packet saying that “I am Congested”.
- **WAY 2:** This form of notification occurs when a router marks a field in a packet flowing from sender to receiver to indicate congestion.
- Upon receipt of a marked packet, the receiver then notifies the sender about congestion



Two feedback pathways for network indicated congestion information

TCP CONGESTION CONTROL

- TCP provides a reliable transport service between two processes running on different hosts.
- TCP uses end to end congestion control approach.
- **Approaches:**
- (i)Limiting each sender, the rate at which it sends traffic into its connection as a function of perceived network congestion.
- (ii)If TCP sender perceives, that there is a little congestion on the path, between itself and its destination, then the sender increases its sending rate.
- (iii)If the sender perceives that there is congestion along the path, then the sender reduces its sending rate.

- **The above approaches raises 3 questions:**
- (i)How does a TCP sender limit the rate at which it sends traffic into its connection?
- (ii)How does a TCP sender perceive that there is a congestion on the path between itself and its destination?
- (iii)What algorithm should sender use to change its sending rate as a function of perceived end to end congestion control?

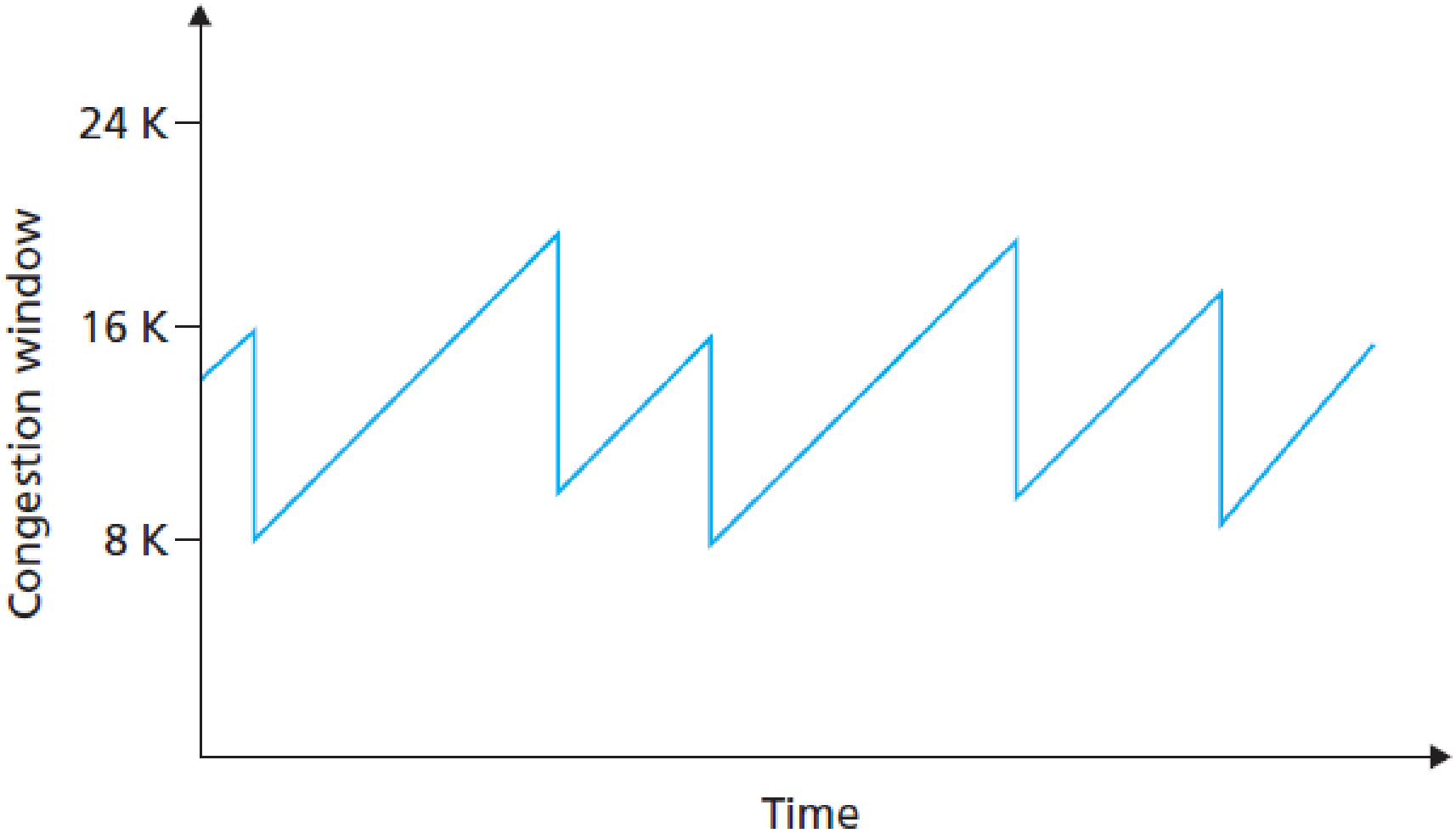
- **(i)How does a TCP sender limit the rate at which it sends traffic into its connection?**
- Each side of a TCP connection consist of send buffer, receive buffer & variables like LastByteRead, RcvWindow..
- The TCP Congestion control mechanism has each side of connection, keep track of a variable called congestion window congwin.
- It imposes a condition on the rate at which a TCP sender can send traffic into network.
- The amount of unacknowledged data at a sender may not exceed the minimum of congwin.
- The above constraints limits the amount of unacknowledged data at the sender and indirectly limits the senders sending rate.
- **By adjusting the value of congwin, the sender can adjust the rate at which it sends data into its connection.**

- **(ii)How does a TCP sender perceive that there is a congestion on the path between itself and its destination?**
- Let us define a loss event at a TCP sender as the occurrence of either time out/receipt of duplicate acks(NAKs) from the receiver.
- When there is a excessive congestion, the one or more router's buffer along the path overflows, causing a datagram containing TCP segment to be dropped.
- The dropped datagram, in turn results in a loss event at the sender, either a timeout or three duplicate acks, which is taken by the sender as an indication of congestion on path.
- TCP uses acknowledgments to trigger its increase in congestion window size & it is said to be **self clocking**.

- **(iii)What algorithm should sender use to change its sending rate as a function of perceived end to end congestion control?**
- The TCP sender uses **Celebrated TCP Congestion Control Algorithm** to change its sending rate as a function of end to end congestion control.
- **Celebrated TCP Congestion Control Algorithm:** This algorithm has 3 major components.
- (a)Additive increase and multiplicative decrease(AIMD)
- (b)Slow start
- (c)Reaction to time out events

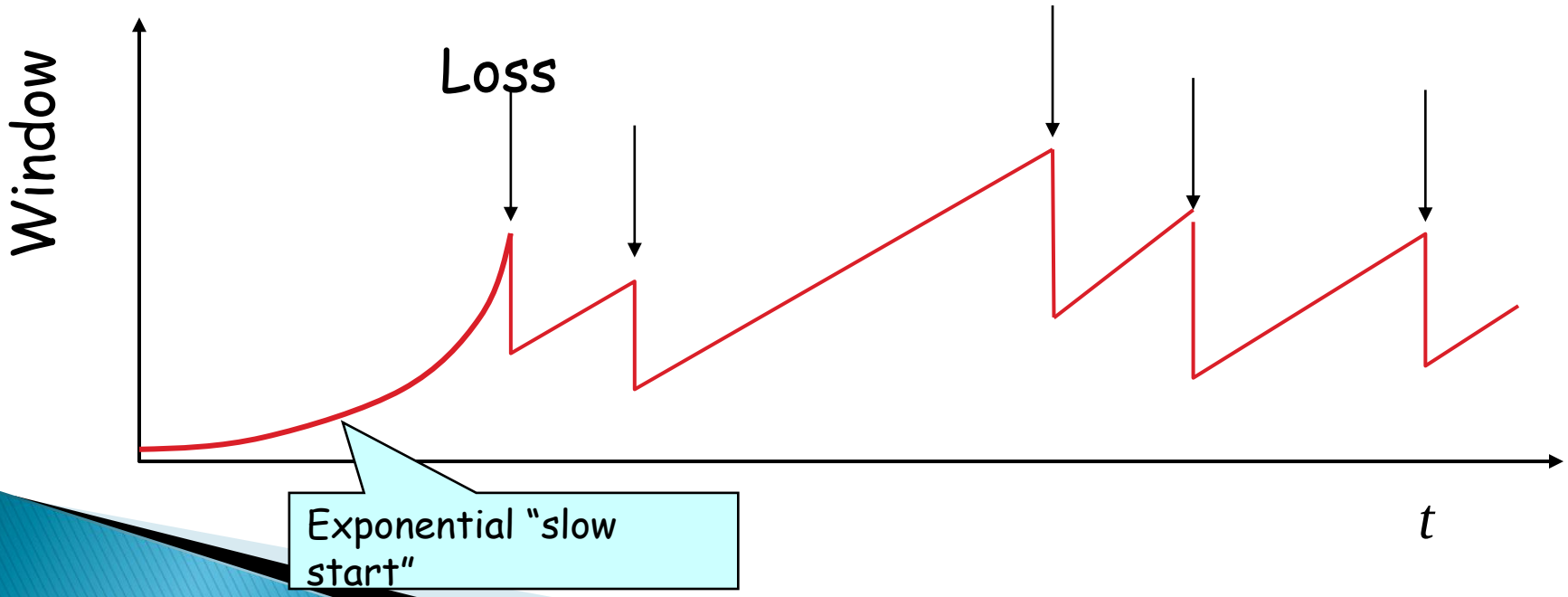
- **(a)Additive increase and multiplicative decrease(AIMD):**
- The basic idea behind the TCP Congestion control is for the sender to reduce its sending rate by decreasing its congestion window size.
- A TCP sender additively increases its sending rate, when it perceives that the end to end path is congestion free.
- A TCP sender multiplicatively decreases its sending rate when it detects that the path is congested by loss event.
- The TCP sender halves the current value of congestion window after a loss event.
- If the value of TCP sender's congwin is currently 20kb and if a loss is detected congwin is cut into half to 10kb.
- If another loss event occurs, congwin is further reduced to 5kb.

- The value of congwin may continue to drop, but it is not allowed to drop below 1mss.
- This linear increase phase of TCP congestion control mechanism is known as **congestion avoidance.**
- The value of congwin repeatedly goes through cycles during which it increases linearly & then suddenly drop to half of its current value, giving rise to a saw-toothed pattern in long lived TCP connections.



AIMD Congestion Control

- **(b)Slow Start:** When a TCP connection begins, the value of congwin is typically initialized to 1MSS/RTT.
- Instead of increasing its rate linearly during this initial phase, a TCP sender increases it rate exponentially, until there is a loss event.
- If there is a loss event the congwin is cut into half & then **grows linearly as in AIMD.**



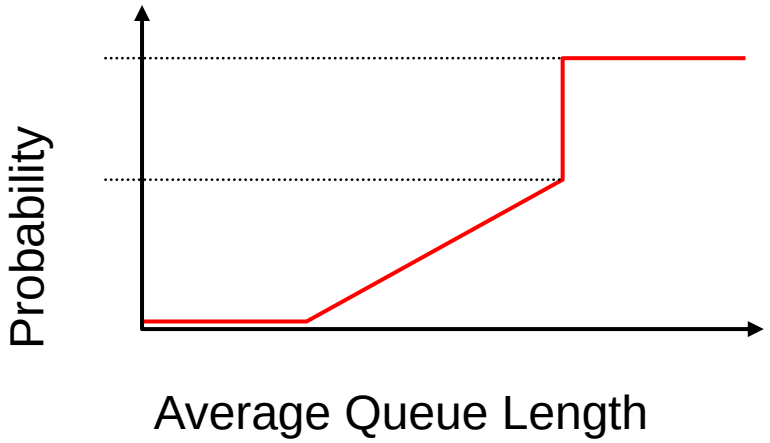
- **(c)Reaction to time out events:**
- When a TCP connection begins, it enters into slow start phase.
- If a loss event occurs AIMD saw-toothed pattern begins.

CONGESTION AVOIDANCE

- **Router-based Congestion Avoidance:**
  - **DECbit:**
    - Routers explicitly notify sources about congestion.
- Each packet has a **"Congestion Notification"** bit called the DECbit in its header.
- If any router on the path is congested, it sets the DECbit.
- To notify the source, the destination copies DECbit into ACK packets.
- **Source adjusts rate to avoid congestion.**
  - Counts fraction of DECbits set in each window.
  - If <50% set, increase rate additively.
  - If >=50% set, decrease rate multiplicatively.

- **Random Early Detection (RED):**
- Routers implicitly notify sources by dropping packets.
- RED drops packets at random, and as a function of the level of congestion.
- RED is based on DECbit, and was designed to work well with TCP.
- RED implicitly notifies sender by dropping packets.
- Drop probability is increased as the *average* queue length increases.

- **Basic idea of RED:**
  - Router notices that the queue is getting backlogged.
  - Randomly drops packets to signal congestion
- **Packet drop probability:**
  - Drop probability increases as queue length increases
  - If buffer is below some level, don't drop anything
  - Otherwise, set drop probability as function of queue



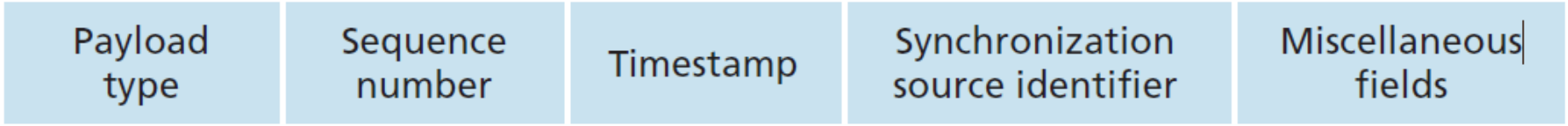
- **Properties of RED:**
- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Drops packet in proportion to each flow's rate
  - High-rate flows have more packets
  - Hence, a higher chance of being selected
- Drops are spaced out in time
  - Which should help desynchronize the TCP senders
- Tolerant of burstiness in the traffic
  - By biasing the decisions on *average* queue length

- **Transport for Real Time Applications:**
- **RTP – Real Time Transport Protocol:**
- RTP is a standard used for transporting common formats such as PCM, ACC, and MP3 for sound and MPEG and H.263 for video.
- **RTP Basics:**
- The sending side encapsulates a media chunk within an RTP packet, then encapsulates the packet in a UDP segment, and then hands the segment to IP.
- The receiving side extracts the RTP packet from the UDP segment, then extracts the media chunk from the RTP packet, and then passes the chunk to the media player for decoding and rendering.

- **Example:** Consider the use of RTP to transport voice.
- Suppose the voice source is PCM-encoded (that is, sampled, quantized, and digitized) at 64 kbps.
- Further suppose that the application collects the encoded data of 160 bytes in a chunk.
- The sending side precedes each chunk of the audio data with an **RTP header that includes the type of audio encoding, a sequence number, and a timestamp.**
- The RTP header is normally 12 bytes. The audio chunk along with the RTP header form the **RTP packet.**
- **The RTP packet is then sent into the UDP socket interface.** At the receiver side, the application receives the RTP packet from its socket interface.
- The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk.

- RTP does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service (QoS) guarantees; it does not even guarantee delivery of packets or prevent out-of-order delivery of packets.
- RTP encapsulation is seen only at the end systems. Routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't.
- RTP allows each source (for example, a camera or a microphone) to be assigned its own independent RTP stream of packets. For example, for a video conference between two participants, four RTP streams could be opened—two streams for transmitting the audio (one in each direction) and two streams for transmitting the video (again, one in each direction).
- However, many popular encoding techniques— including MPEG 1 and MPEG 2—bundle the audio and video into a single stream during the encoding process.
- When the audio and video are bundled by the encoder, then only one RTP stream is generated in each direction.

- **RTP Packet Header Fields**
- The four main RTP packet header fields are the payload type, sequence number, timestamp, and source identifier fields.



- **The payload type** field in the RTP packet is 7 bits long.
- For an audio stream, the payload type field is used to indicate the type of audio encoding (for example, PCM, adaptive delta modulation, linear predictive encoding) that is being used.

- If a sender decides to change the encoding in the middle of a session, the sender can inform the receiver of the change through this payload type field.
- The sender may want to change the encoding in order to increase the audio quality or to decrease the RTP stream bit rate.
- ***Sequence number field.*** *The sequence number field is 16 bits long.*
- *The sequence* number increments by one for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
- For example, if the receiver side of the application receives a stream of RTP packets with a gap between sequence numbers 86 and 89, then the receiver knows that packets 87 and 88 are missing.
- The receiver can then attempt to conceal the lost data.

- ***Timestamp field.*** *The timestamp field is 32 bits long. It reflects the sampling* instant of the first byte in the RTP data packet.
- The receiver can use timestamps to remove packet jitter introduced in the network and to provide synchronous playout at the receiver.
- The timestamp is derived from a sampling clock at the sender.
- ***Synchronization source identifier (SSRC).*** *The SSRC field is 32 bits long. It identifies* the source of the RTP stream.

Payload-Type Number	Audio Format	Sampling Rate	Rate
0	PCM μ-law	8 kHz	64 kbps
1	1016	8 kHz	4.8 kbps
3	GSM	8 kHz	13 kbps
7	LPC	8 kHz	2.4 kbps
9	G.722	16 kHz	48–64 kbps
14	MPEG Audio	90 kHz	—
15	G.728	8 kHz	16 kbps

Table 7.2 ♦ Audio payload types supported by RTP

Payload-Type Number	Video Format
26	Motion JPEG
31	H.261
32	MPEG 1 video
33	MPEG 2 video

Table 7.3 ♦ Some video payload types supported by RTP