The Transport Layer:

TCP, UDP

UDP

- UDP is a simple, unreliable datagram protocol.
- User Datagram Protocol.
- UDP is a connectionless protocol, and UDP sockets are an example of datagram sockets.
- Lack of reliability.
- Each UDP datagram has a length.
- Connectionless service.

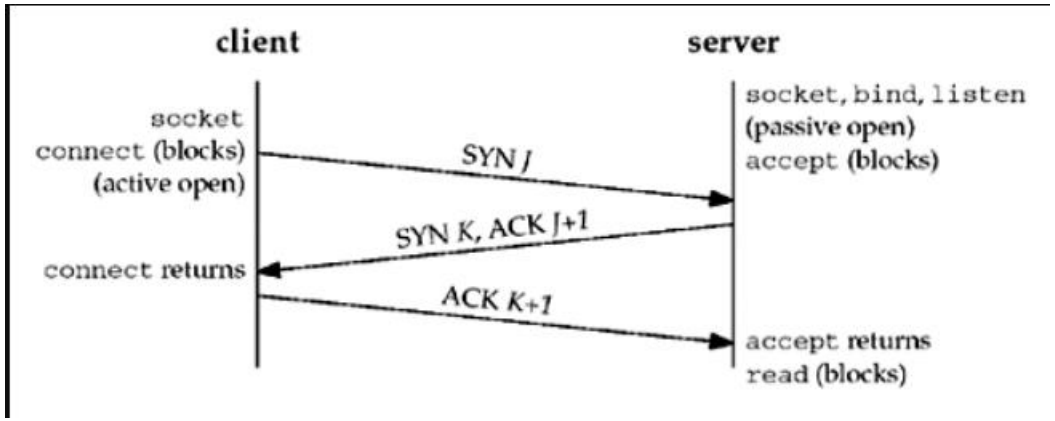
TCP

- TCP is a sophisticated, reliable byte-stream protocol.
- Transmission Control Protocol.
- TCP is a connection-oriented protocol that provides a reliable, full-duplex byte stream to its users.
- **Connection**: TCP provides connections between clients and servers. A TCP client establishes a connection with a server, exchanges data across the connection, and then terminates the connection.
- **Reliability**: TCP requires acknowledgment when sending data. If an acknowledgment is not received, TCP automatically retransmits the data and waits a longer amount of time.
- **Round-trip time** (RTT): TCP estimates RTT between a client and server dynamically so that it knows how long to wait for an acknowledgment.
- **Sequencing**: TCP associates a sequence number with every byte (**segment**, unit of data that TCP passes to IP.) it sends. TCP reorders out-of-order segments and discards duplicate segments.
- **Flow control**
- **Full-duplex**: an application can send and receive data in both directions on a given connection at any time.

TCP	UDP
Connection oriented	Connection less
Reliable	Less reliable
Error control is mandatory	Error control is optional
Slow transmission	Fast transmission
More overhead	Less overhead
Flow control, Congestion control	No Flow control, Congestion control

TCP Connection Establishment and Termination

Three-Way Handshake



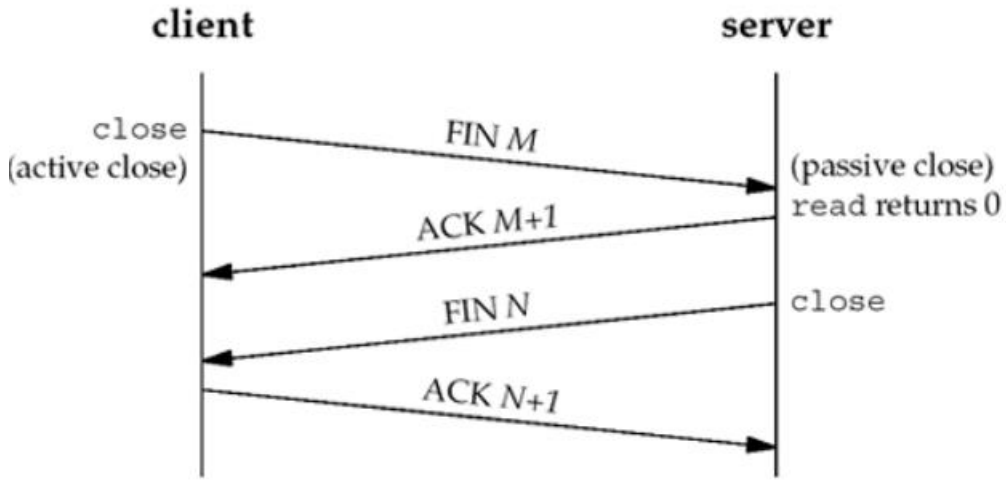
1. Server: passive open, by calling socket, bind, and listen
2. Client: active open, by calling connect. The client TCP to send a "synchronize" (SYN) segment with no data but it contains client's initial sequence number for the data to be sent on the connection.
3. Server: acknowledges (ACK) client's SYN. The server sends its SYN and the ACK of the client's SYN in a single segment which also contains its own SYN containing the initial sequence number for the data to be sent on the connection.
4. Client: acknowledges the server's SYN.

The client's initial sequence number as *J* and the server's initial sequence number as *K*.
The acknowledgment number in an ACK is the next expected sequence number for the end sending the ACK.
Since a SYN occupies one byte of the sequence number space, the acknowledgment number in the ACK of each SYN is the initial sequence number plus one.

TCP Options

- MSS option. The TCP sending the SYN announces its **maximum segment size** (the maximum amount of data that it is willing to accept in each TCP segment) on this connection. The sending TCP uses the receiver's MSS value as the maximum size of a segment that it sends.
- Window scale option. The maximum window that the TCP can advertise to the other is 65,535, because the corresponding field in the TCP header occupies 16 bits. However, high-speed connections, common in today's Internet, or long delay paths require a larger window to obtain the maximum throughput possible.
- To provide interoperability with older implementations that do not support this option, TCP can send the option with its SYN as part of an active open, but it can scale its windows only if the other end also sends the option with its SYN. Similarly, the server's TCP can send this option only if it receives the option with the client's SYN. Implementations ignore options that they do not understand.
- Timestamp option. This option is needed for high-speed connections to prevent possible data corruption caused by old, delayed, or duplicated segments. Since it is a newer option, it is negotiated similarly to the window scale option.

TCP Connection Termination



It takes four segments to terminate a connection:

1.One end calls close first by sending a FIN segment to mean it is finished sending data. This is called **active close**.

2.The other end that receives the FIN performs the **passive close**. The received FIN is acknowledged by TCP (sending an ACK segment).

The receipt of the FIN is also passed to the application as an end-of-file.

3.Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.

4.The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN

A FIN occupies one byte of sequence number space just like a SYN. Therefore, the ACK of each FIN is the sequence number of the FIN plus one.

Services Provided to the Upper Layers

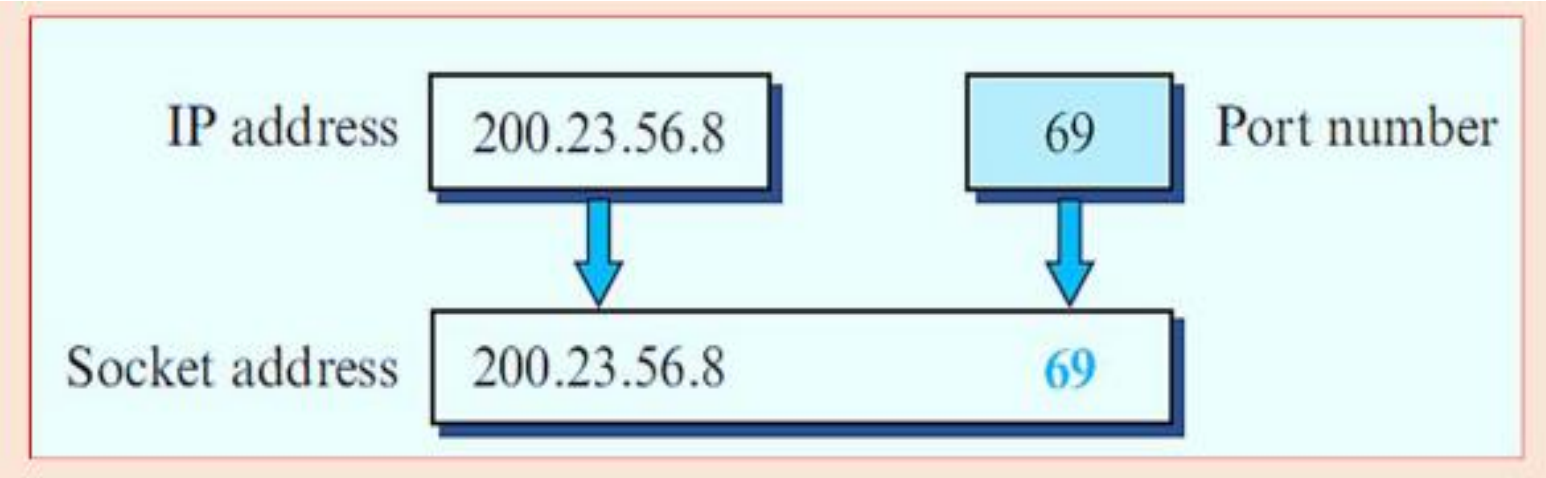
- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer.
- To achieve this goal, the transport layer makes use of the services provided by the network layer.
- The hardware and/or software within the transport layer that does the work is called the transport entity.
- The transport entity can be located in the operating system kernel, in a separate user process, in a library package bound into network applications, or conceivably on the network interface card

TPDU (Transport Protocol Data Unit)

- **TPDU** (**Transport Protocol Data Unit**) for messages sent from transport entity to transport entity.
- Thus, TPDUs (exchanged by the transport layer) are contained in packets (exchanged by the network layer).
- In turn, packets are contained in frames (exchanged by the data link layer).
- When a frame arrives, the data link layer processes the frame header and passes the contents of the frame payload field up to the network entity.
- The network entity processes the packet header and passes the contents of the packet payload up to the transport entity.

Socket Addresses

- A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection.
- The combination of an IP address and a port number is called a socket address.
- The client socket address defines the client process uniquely and server socket address defines the server process uniquely.



UDP (User Datagram Protocol)

- The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- UDP is a very simple protocol using a minimum of overhead.
- If a process wants to send a small message and does not care much about reliability, it can use UDP.
- Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.
- UDP packets, called user datagrams
- UDP Doesn’t ensure orderly arrival of data
- It has limited error checking

UDP advantages(Applications)

- Protocol with less overhead
- Sending a message using UDP requires less interaction between sender and receiver
- Convenient protocol for multimedia and multicasting applications
- Used in some route updating protocols like RIP(routing information protocol)
- Used in DNS(domain name system)

UDP header

- UDP transmits **segments** consisting of an 8-byte header followed by the payload

UDP header

- Source port, destination port:
  - Identifies the end points within the source and destination machines.
- UDP length:
  - Includes 8-byte header and the data
- UDP checksum:
  - It is an optional field used for error detection

Applications of UDP

- RPC(Remote procedural call)
- RTTP(Real time transport protocol)

Remote procedure call

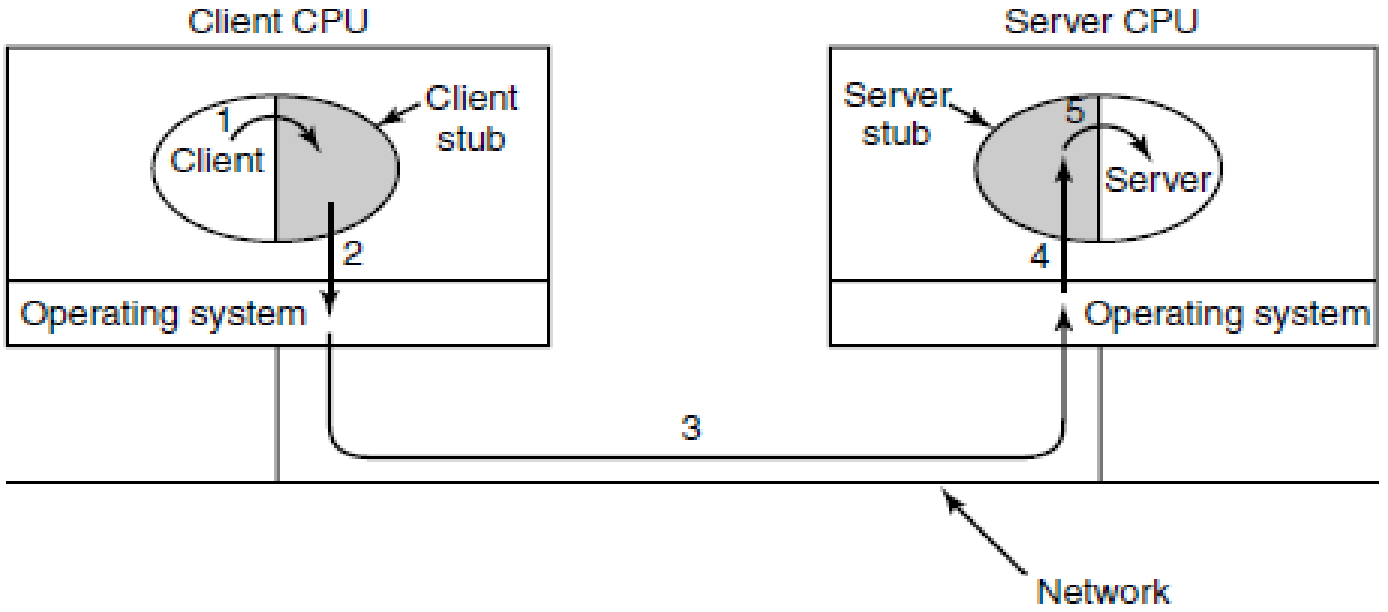
When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2.

- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing is visible to the application programmer.
- This technique is known as **RPC** (**Remote Procedure Call)**
- Traditionally, the calling procedure is known as the client and the called procedure is known as the server.

Remote procedure call

- In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub, that** represents the server procedure in the client's address space.(stub knows which machine to contact for that procedure)
- Similarly, the server is bound with a procedure called the **server stub.**
- **These procedures hide the fact** that the procedure call from the client to the server is not local

Remote procedure call



. Steps in making a remote procedure call. The stubs are shaded.

Remote procedure call

- Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.
- Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called **marshaling.**
- **Step 3 is the operating system sending the** message from the client machine to the server machine.
- Step 4 is the operating system passing the incoming packet to the server stub.
- step 5 is the server stub calling the server procedure with the unmarshaled parameters.
- The reply traces the same path in the other direction.

Features of TCP protocol

- **Transport Layer Protocol**
  - TCP is a transport layer protocol as it is used in transmitting the data from the sender to the receiver.
- **Reliable**
  - TCP is a reliable protocol as it follows the flow and error control mechanism.
  - It also supports the acknowledgment mechanism, which checks the state and sound arrival of the data.
- **Order of the data is maintained**
  - This protocol ensures that the data reaches the intended receiver in the same order in which it is sent.
- **Connection-oriented**
  - It is a connection-oriented service that means the data exchange occurs only after the connection establishment.
  - When the data transfer is completed, then the connection will get terminated.

Features of TCP protocol

- **Full duplex**
  - It is a full-duplex means that the data can transfer in both directions at the same time.
- **Stream-oriented**
  - TCP is a stream-oriented protocol as it allows the sender to send the data in the form of a stream of bytes and also allows the receiver to accept the data in the form of a stream of bytes.

The TCP Segment Header

- Every segment begins with a fixed-format, 20-byte header.
- The fixed header may be followed by header options
- Segments without any data are legal and are commonly used for Acknowledgements and control messages.

The TCP Segment Header

- Source Port, Destination Port : Identify local end points of the connections
- Sequence number: Specifies the sequence number of the segment
- Acknowledgement Number: Specifies the next byte expected.
- TCP header length: Tells how many 32-bit words are contained in TCP header
- Next comes a 6-bit field that is not used.
- Now come six 1-bit flags

The TCP Segment Header

- URG: It is set to 1 if URGENT pointer is in use, which indicates start of urgent data.
- ACK: It is set to 1 to indicate that the acknowledgement number is valid.
- PSH: Indicates pushed data
- RST: It is used to reset a connection that has become confused due to reject an invalid segment or refuse an attempt to open a connection.
- FIN: Used to release a connection.
- SYN: Used to establish connections.

The TCP Segment Header

- Window size - This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits
- Checksum - This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP.
- Urgent pointer - This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data.
- Options field -provides a way to add extra facilities not covered by the regular header.

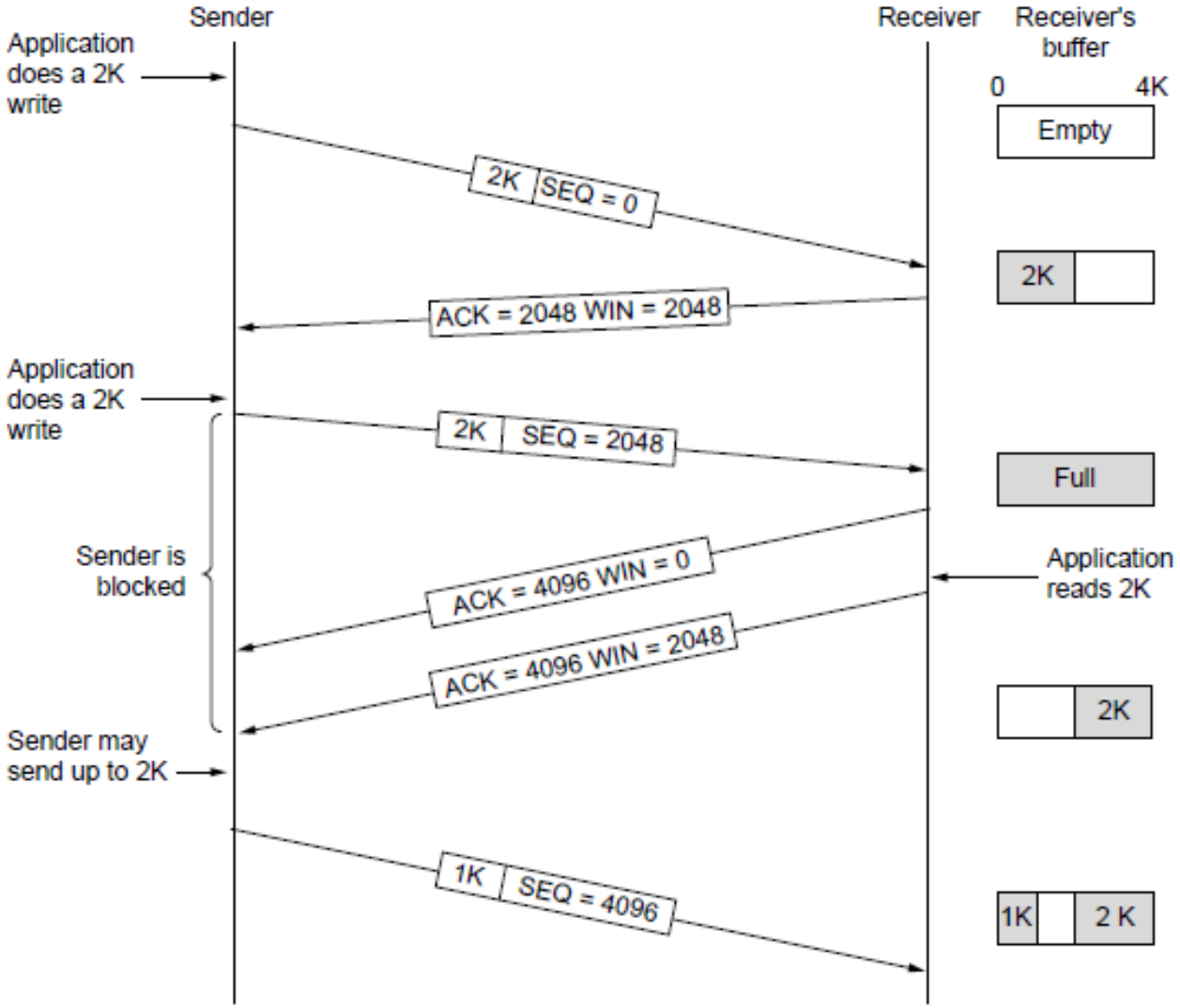
most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept

TCP Transmission Policy

- **TCP Transmission Policy** is not directly tied to acknowledgements as it is in most data link protocols

- Acknowledgement doesn’t allow the sender to transmit more
- TCP uses the concept of window size for managing transmission of data.
- Window field explicitly tells the sender how much it can transmit .
- Basically, the window size indicates the size of the receive buffer

TCP Transmission Policy



TCP Transmission Policy

- 1. In the above example, the receiver has 4096-byte buffer.
- 2. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
- 3. Now the receiver will advertise a window of 2048 as it has only 2048 of buffer space, now.
- 4. Now the sender transmits another 2048 bytes which are acknowledged, but the advertised window is’0’.
- 5. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a layer window.

Silly Window Syndrome

- Silly Window Syndrome is a problem that arises due to the poor implementation of TCP.
- It degrades the TCP performance and makes the data transmission extremely inefficient.
- The problem is called so because-
  - It causes the sender window size to shrink to a silly value.
  - The window size shrinks to such an extent where the data being transmitted is smaller than TCP Header

Causes of Silly Window Syndrome

- The syndrome may arise because of the following problems –

    1. Sender windows transmit one byte of data repeatedly.

    1. Receiver windows accept one byte of data repeatedly.

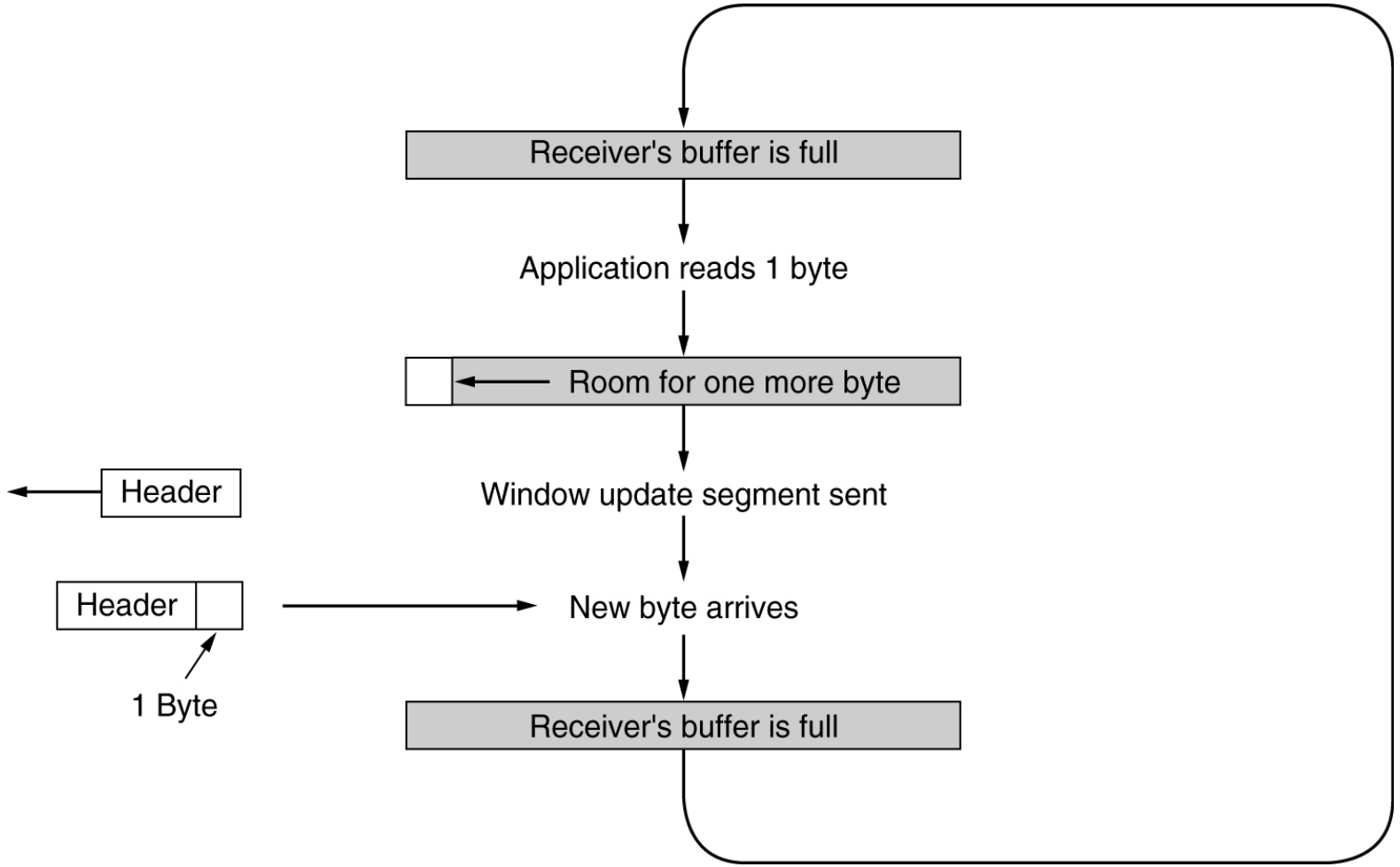
Sender windows transmit one byte of data repeatedly

- If an application which will generate only one byte of data. The TCP will transmit this small segment of data.
- All time the application generates a single byte of data and the window transmits it.
- This is because the transmission process becomes slow and inefficient. Here the sender window transmits one byte of data repeatedly.

Receiver window accepting one byte of data repeatedly

- Suppose the receiver is not able to process all the incoming data.
- In such a case, the receiver advertises a small window size.
- The process repeats and the window size becomes too small.
- Thus, the receiver repeatedly advertises window size of one byte.
- Finally, the receiving process becomes slow and inefficient.

Silly Window Syndrome- **Receiver window accepting one byte of data repeatedly**



Silly Window Syndrome- **Receiver window accepting one byte of data repeatedly**

1. Initially the TCP buffer on the receiving side is full and the sender knows this(win=0)
2. Then the interactive application reads 1 character from tcp stream.
3. Now, the receiving TCP sends a window update to the sender saying that it is all right to send 1 byte.
4. The sender obligates and sends 1 byte.
5. The buffer is now full, and so the receiver acknowledges the 1 byte segment but sets window to zero.
6. This behavior can go on forever.

Silly Window Syndrome Solutions

- Nagle's algorithm and Clark's solution
- Nagle’s Algorithm tries to solve the problem caused by the sender delivering 1 data byte at a time.

- Clark’s Solution tries to solve the problem caused by the receiver sucking up one data byte at a time.

Nagle's algorithm

Nagle’s algorithm suggests-

- Sender should send only the first byte on receiving one byte data from the application.
- Sender should buffer all the rest bytes until the outstanding byte gets acknowledged.
- In other words, sender should wait for 1 RTT.
- After receiving the acknowledgement, sender should send the buffered data in one TCP segment.
- Then, sender should buffer the data again until the previously sent data gets acknowledged.

Clark’s solution

- Clark’s solution suggests-
- Receiver should not send a window update for 1 byte.
- Receiver should wait until it has a decent amount of space available.
- Receiver should then advertise that window size to the sender.

TCP congestion control.

- Congestion control refers to techniques and mechanisms that can-
  - Either prevent congestion before it happens
  - Or remove congestion after it has happened

TCP congestion control.

- TCP uses a congestion window(cwnd) in the sender side to do congestion avoidance
- The congestion window indicates the maximum amount of data that can be sent out on a connection with out being acknowledged.
- TCP detects congestion when it fails to receive an acknowledgement with in the estimated time out.
- In such situation it decreases the congestion window to one maximum segment size.

Congestion Control in TCP

- Congestion in TCP is handled by using these **three phases:**

1. Slow Start
2. Congestion Avoidance(Additive Increase)
3. Congestion Detection(Multiplicative Decrease)

Phase 2 & 3 combinely known as AIMD technique Additive Increase Multiplicative Decrease

Congestion Detection Phase

- In this phase, the sender identifies the segment loss
- When sender detects the loss of segments, it reacts in different ways depending on how the loss is detected
- **Case-01: Detection On Time Out**
  - In this, the timer time-out expires even before receiving acknowledgment for a segment.
  - In this case sender sets the slow start threshold to half of the current congestion window size.
  - Each time a timeout occurs, the source sets CongestionWindow to half of its previous value.
  - This halving of the CongestionWindow for each timeout corresponds to the “multiplicative decrease” part of AIMD(Additive Increase /Multiplicative Decrease)
  - Slow start phase is resumed

Congestion Detection Phase

- **Case-02: Detection On Receiving 3 Duplicate Acknowledgements-**
- If three ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received.
- This is called fast transmission and fast recovery.
  - Sender receives 3 duplicate acknowledgements for a segment.
  - This case suggests the weaker possibility of congestion in the network.
  - There are chances that a segment has been dropped but few segments sent later may have reached.
- In this case, sender reacts by-
  - Setting the slow start threshold to half of the current congestion window size.
  - Decreasing the congestion window size to slow start threshold.
  - Resuming the congestion avoidance phase.