

Lecture 20

Transport Layer, plays an important part in the ISO/OSI architecture, especially when the underlying network layer is unreliable, then all the complexity of flow control and error control comes on the shoulder of transport layer or becomes responsibility of transport layer.

The local transport layer communicates with the remote transport layer entity using services of lower layers. The general service provided by a transport protocol is the end-to-end transport of data in a way that shields the user from the details of the underlying communication system.

Like data link layer, network layer, transport layer also comes in two flavors. It is **connection oriented** and **connectionless** transport layer.

- A connection oriented transport service provides establishment, maintenance, and termination of transport layer connection between the end stations i.e. source and receiver.
- Since a connection needs to be established, the service also becomes reliable one.
- Example of one such connection oriented service is TCP.

In Connectionless Service

- A connection less transport service does not require connection establishment and since there is no establishment, there should be no termination also.
- This happens to be a unreliable service
- Example of one such connection less service is UDP.

We will study in detail about the TCP protocol, i.e. connection oriented service by the transport layer.

This book and your course

- First discusses the transport layer services with an underlying connection-oriented network layer.
- Then again discusses the transport layer services with an underlying connectionless network layer.
- TCP protocol, TCP services, TCP mechanisms
- UDP protocol.

So these are the four units which we will consider in this and forthcoming lectures.

Transport Layer with an underlying connection-oriented network layer

Since the underlying network layer is connection oriented, so issues of ordered delivery, reliability and loss of segments or packets is the concern of network layer. So there is no need of error-control at the transport layer. The issues of major concern for the transport layer are

1. Service Point Addressing
2. Multiplexing
3. Flow Control
4. Connection Establishment and Termination

We will examine each issue in detail starting with the first issue i.e. Service point Addressing

Service point Addressing

On each machine, we can have different TS (Transport Service) users, one corresponding to each open port. TS user on one machine wants to establish a connection with the TS user on another machine. For that connection to be established, each user has to specify their global address and the port number, which is known as socket (IP address, port number). There may be lot of open ports in a system so that much possible number of TS users are possible. A connection is established on a socket at one end and another socket of the same port on the other end. The socket's abstract name is service point.

Multiplexing

Multiplexing here means multiplexing multiple transport connections on a single network layer connection, which is generally required if the available bandwidth is more than or equal to the integration of individual requirement of each connection, thus making an effective utilization of the available bandwidth. This kind of multiplexing is known as **Upward multiplexing**.

There is another kind of multiplexing known as **Upward Multiplexing**, where a single transport user uses multiple network layer connections, which is generally required to increase the throughput.

Flow Control

We will not be discussing much about flow control mechanism as we have already discussed it in the data link layer, we discussed two algorithms regarding the flow control at the data link layer, i.e.

1. Stop and Wait Flow Control
2. Sliding window protocol

And there is another one, which we skipped during the discussion of data link layer

3. Credit Allocation Flow Control

Flow control happens to be a simpler mechanism at the data link layer and Transport layer, but it is a complex mechanism at the transport layer, this is worth notifying, since transport layer is responsible for communication between source and sender, and data link layer is responsible for communication between two nodes, so there is lot of transmission delay or in other words propagation delay, which can also be stated as

1. The transmission delay or propagation delay (one-side) is generally long compared to actual transmission time. This means that there is considerable delay in the communication of flow control information.
2. Source cannot effectively use the timeout mechanism, since transmission delay or propagation delay (one-side) is over the internet, which the delay can be variable.

In response to the flow control, transport entity at the receiving side does

1. Do nothing
2. Refuse to accept further segments from network device
3. Use a fixed sliding window protocol
4. Use a credit allocation flow control.

First alternative means that the segments that overflow the buffer are discarded. In other words, if receiving side cannot handle more incoming fragments, it will simply ignore them which is actually not a possible alternative.

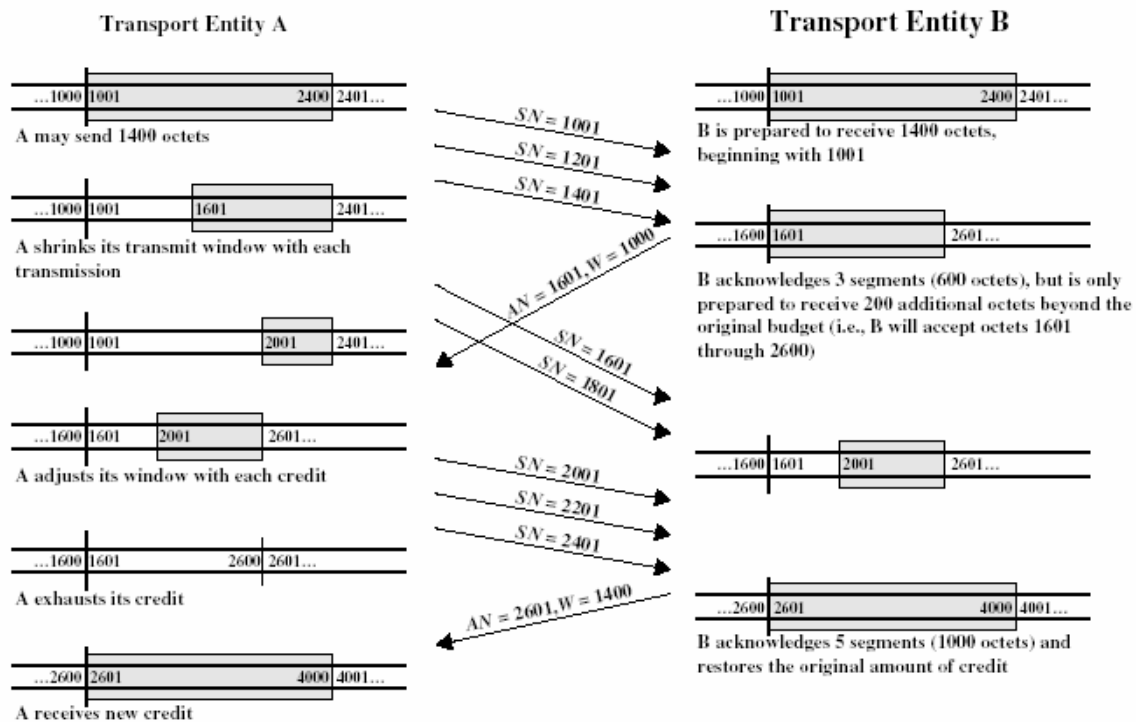
It is also called a *backpressure mechanism* that relies on the network service to do the work. This will trigger flow control procedures within the network service that throttle the network at the sending end. What network layer at receiving end does is sending a message back to the sender (using IP protocol) to slow down whenever receiver is overwhelmed by too much of incoming fragments. This approach is detection of flow control and then taking action to cure from excessive flow.

Third and fourth mechanism is a preventive approach, objective is to prevent the over flow to happen at the first place, for that we have flow control protocols working at sender and receiver ends. There are lots of protocols, one is sliding window protocol and other one is credit allocation flow control, which is a modification of sliding window protocol with credit allocation added to it.

We will not be discussing sliding window flow control here, as we already had a discussion on it during data link layer protocol lecture.

But we will discuss credit allocation flow control protocol in detail, which is better than sliding window protocol (infact improved version of sliding window protocol), because here size of the window can vary dynamically which was fixed with sliding window protocol.

In Credit Allocation flow control, each transmitted segment includes three things in its header related to flow control: Sequence Number (SN), Acknowledgment Number (AN) and Window as shown in figure below.



When transport entity sends a segment, it includes the Sequence Number of the first octet of the segment

When a transport entity sends an acknowledgment, it includes the Acknowledgment Number and Window, which looks like (AN = i, W= j).

- All octets through sequence number $SN = i - 1$ have been received successfully and next octet expected is with sequence number $SN = i$.
- Permission is granted to send an additional window of $W = j$ octets of data; that is grants an additional j octets beginning with octet i .

The above explanation will change in case of piggybacking. With piggybacking, each segment will carry both data and acknowledgment. We will only discuss for the first case i.e. without piggybacking.

In the diagram above, we have made an assumption, that transport entity can send a maximum of 200 octets of data in each segment and initial credit granted to A is 1400 octets beginning with 1001, i.e. A can only send maximum of 1400 octets of data before receiving an acknowledgment, that makes only 7 segment.

So A starts with sending the first segment, it's Sequence Number $SN = 1001$, as shown in figure below.

A sends the second and third segment successively, with Sequence Number $SN = 1201$ and $SN = 1401$ respectively as shown in figure above. Now A can further send only

Now B thinks of acknowledging back, so B will acknowledge back with $AN = 1601$ and $W = 1000$, which can be interpreted as, the receiver B has received data till Sequence Number $SN=1600$ with next expected Sequence Number $SN = 1601$ and A is granted an additional allocation of 1000 i.e. from 1601 to 2601.

But the acknowledgment $AN = 1601$ and $W = 1000$ reaches A after he was through with sending two more fragments i.e. when $SN = 1601$ and $SN = 1801$ has already gone, so the current size of window is 2001 to 2401, which increases effectively by 600 only making it 2601.

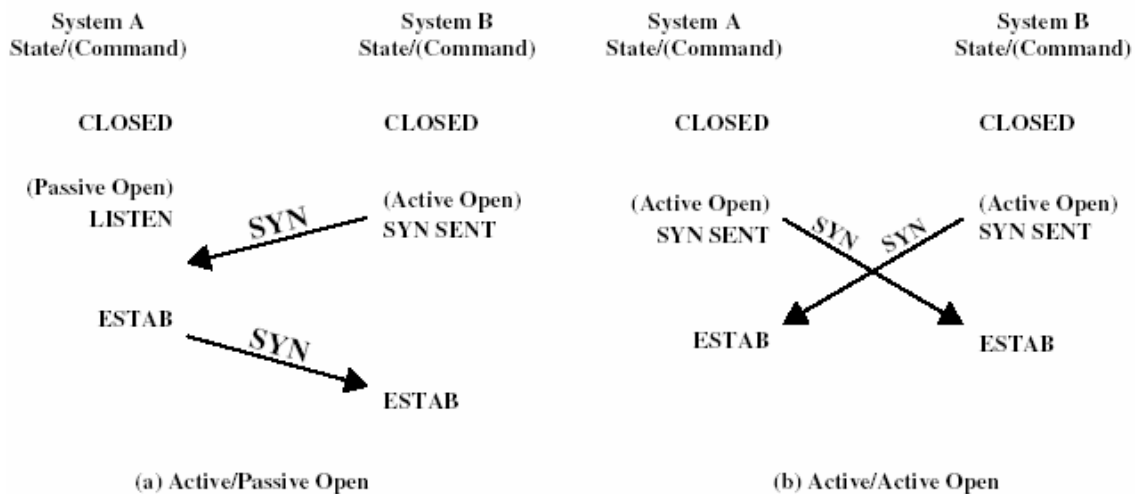
Now, A sends three successive frames, with Sequence Numbers $SN = 2001$, $SN = 2201$, $SN = 2401$, and window reduces with this.

B acknowledges back, with $AN = 2601$ and $W = 1400$, which can be interpreted as, the receiver B has received data till Sequence Number $SN = 2600$ with next expected Sequence Number $SN = 2601$ and A is granted an additional allocation of 1400 i.e. from 2601 to 4001.

Connection Establishment and Termination

Connection establishment is always preferred although Transport layer is working with reliable network layer, because it serves three purposes

1. Allows each end station to ensure that the other exists.
2. Allows negotiation of parameter, which is called as option negotiation.
3. And after negotiation is over, setting the value of parameters for which the negotiation was done.



Above diagram depicts the connection establishment procedure, two entities are there, system A and system B. Initially both are in **closed** state. Now one of the systems, who wish to establish connection, goes to **Active Open** state and other one who doesn't has a desire to establish connection moves to **Passive Open** state. In Passive Open state, a station is listening for the request from others for establishing a connection

- So let's say station B is in Active Open state and station A is in Passive Open state listening for requests from other users if any.
- Station B will issue a SYN frame, as a request to station A for establishing a connection and moves to **SYN SENT** state.
- Station A if also agrees to request, will respond back by sending a SYN for B as an acknowledgment.
- After SYN is received by B, B will move to **established** state and A will also move to **established** state.

This kind of communication for establishing a connection at the transport layer is called as two-way handshake.

Connection-termination is also handled similarly.

- Any station can send the FIN to the other station for disconnecting the connection.
- Let's say B initiates the connection termination, after sending FIN, B will move to **FIN WAIT** state.
- When a FIN is received by station A, he will move to **CLOSE WAIT** state and sends an acknowledgment as another FIN frame and moves to **CLOSED** state if no job is left.
- This acknowledgment FIN when received by B will terminate the connection (if it has no pending data or job left to process) move to **CLOSED** state.

Transport Layer with an underlying connection-less un-reliable network layer

Since the underlying network layer is connection-less, all the burden falls onto the shoulders of transport layer. Now transport layer has concerns about errors also, whether data fragment is received or not, whether data fragments are received in order or not, if not in order and to know which one is missing, and how to do deal with retransmission, so all these issues are taken care of besides issues which will still be there as they were with connection-oriented reliable network layer. So the following issues are there

1. Ordered delivery

2. Retransmission Strategy
3. Duplicate Detection
4. Flow Control
5. Connection Establishment
6. Connection Termination
7. Crash Recovery

Ordered Delivery

Since the underlying network layer service is unreliable, that means the fragments can arrive out of order, they need to be reassembled at the destination either by the network layer or if not by network layer then definitely by transport layer. So transport layer service reassembles the fragments in orderly fashion and finally gets an assembled complete message provided no fragment is missing. How assembling is done at the network layer, each IP packet has an identification field, which increases sequentially with every subsequent packet.

Concept is same over here also, but the only difference is, instead of having identification field, each fragment carries the start of octet i.e. first fragment may have a sequence number $SN = 0$, and is carrying 1000 octets of data. Second fragment will then have a sequence number $SN = 1001$. Thus if the frames happen to be out of order at the destination, they can easily be reassembled based on their sequence number.

Retransmission Strategy

There are two reasons for retransmission of segment

1. Transmitted segment may be lost during transit
2. Transmitted segment is received but contains error.

Second case is easy to handle, since CRC is there with every segment, if the segment is received by the receiver, he can determine the CRC, if it is not 0, this means there is some error, and can ask for the retransmission by sending a negative acknowledgment.

But with the first case, if a sender sends a segment, which never reaches the receiver, receiver will not be able to send any negative acknowledgment. In that case a timer is required at the sender side, which when blows off, is an indication for the sender to retransmit the sender, but what should be the value of the timer is a major issue, since between sender and receiver there may be multiple hops, and how much should be the value of retransmission timer. Actually it should be variable, which must change with the changing network configuration. This is a major research topic, choosing the value of retransmission timer.

There are other issues associated with the value of retransmission timer

1. The receiving entity may not acknowledge every received fragment immediately, may acknowledge collectively. So setting the value of retransmission timer should take this thing in to consideration
2. If a segment is retransmitted, and later an acknowledgment arrives, sender cannot know whether this is the acknowledgment of initial transmission or acknowledgment of retransmission.
3. We discussed above, changing network conditions.

Retransmission also relates to one more problem which is duplicate, discussed below.

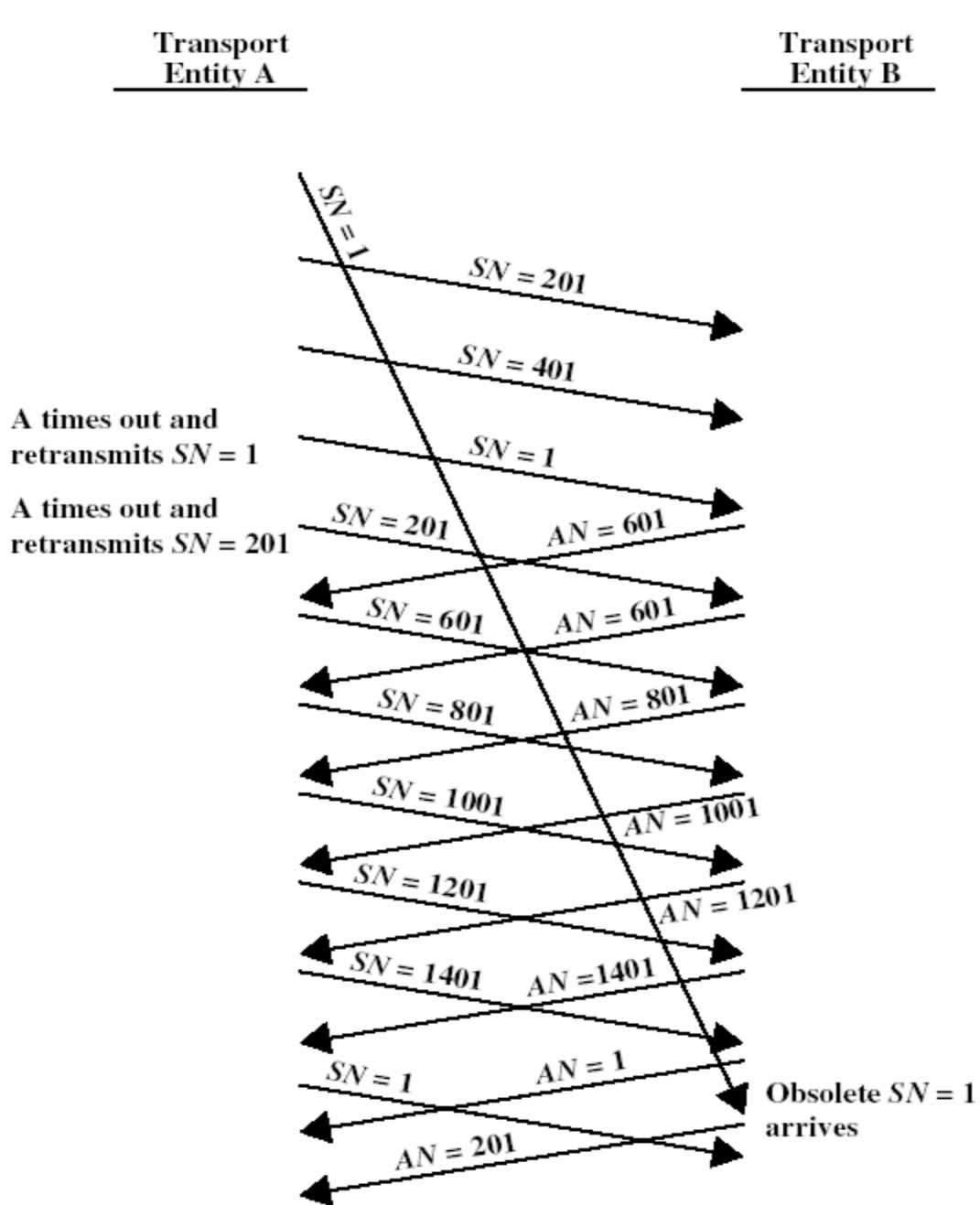
We will discuss about retransmission timer in detail later.

Duplicate Detection

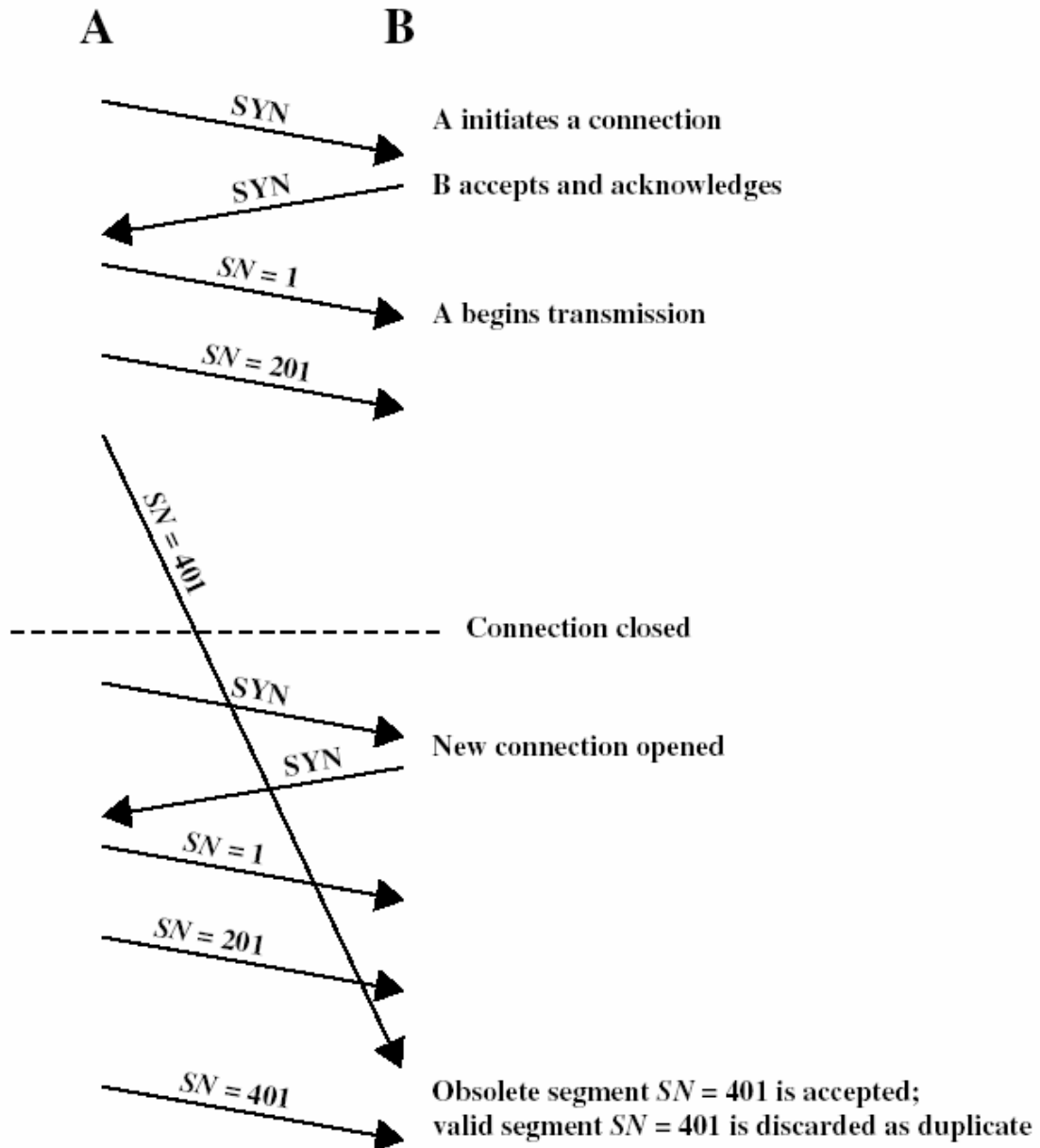
There are three scenarios which lead to duplicate, obviously they are related to retransmission in one way or the other. First discuss the simplest case which leads to no duplicate, If a data segment is lost, timer at the sender side goes off, and sender retransmit the same data segment with the same sequence number again, receiver

acknowledges back, this is perfect case, for which timer and retransmission were defined, but definition of timer and retransmission leads to further problem.

1. It may happen that the data segment is delayed, timer at the sender side goes off, and he re-transmits the data segment. That re-transmitted data segment is received. After some time that delayed data segment appears which confuses the receiver.



2. It may also happen that the delayed data segment may appear after the close of current connection i.e. in another session but between the same two transport users.



- It may happen that a data segment is transmitted by the sender, which is received by the receiver, the receiver acknowledges it back, but the acknowledgment is lost, so after some time the sender's timer will go off, and he retransmits the segment with the same sequence number. The receiver can easily deal with it by assuming that the acknowledgment that he sent may be lost during transit, which made the sender retransmit the segment again. But the receiver should again acknowledge back.

So the first two cases are difficult to handle but the third case can be easily handled by the receiver.

The solution to this problem is dealt with changing the connection establishment strategy, which we will discuss later. Earlier we were using two-way handshake and now three-way handshake will be used which will resolve the above-mentioned two issues.

Flow Control

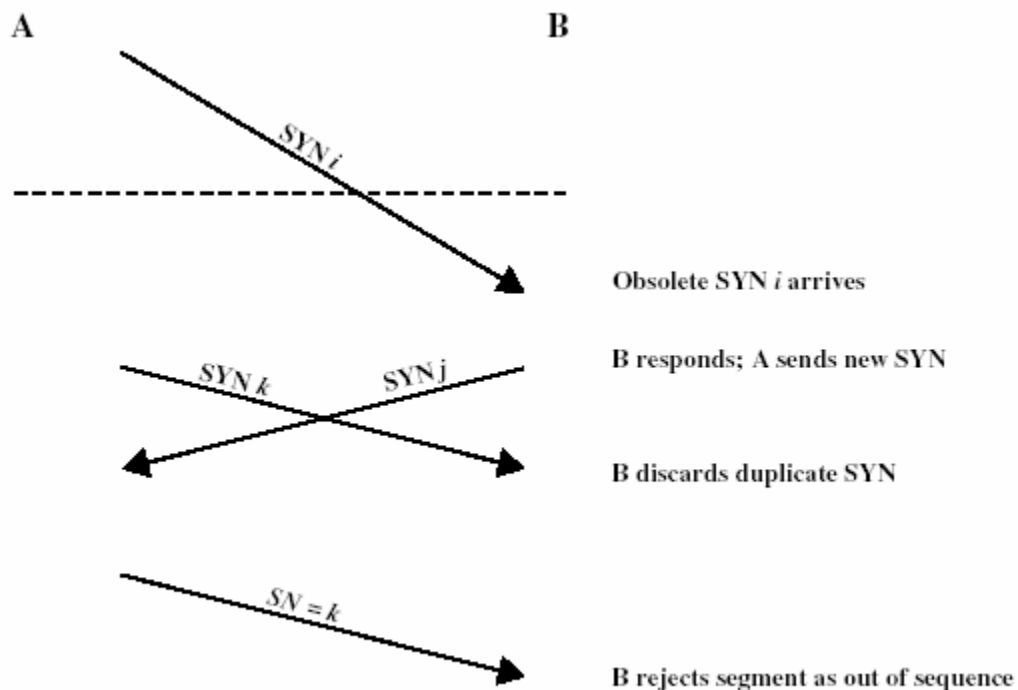
The credit allocation flow control mechanism is already quite robust and requires little enhancement. We use credit allocation flow control but that some times lead to deadlock as described below.

- If B sends ($AN=i, W=0$), temporarily closing the window. Later, B sends ($AN=i, W=j$), but this segment is lost. Thus A is waiting for the opportunity to send and B thinks that it has granted the opportunity.
- A *window timer* can be used. This timer is reset with every segment containing AN and W fields.

Connection Establishment and Termination

As we discussed above, the need for three-way handshake, since two-way handshake leads to problem of handling duplicates, which will be completely eradicated by using three-way handshake.

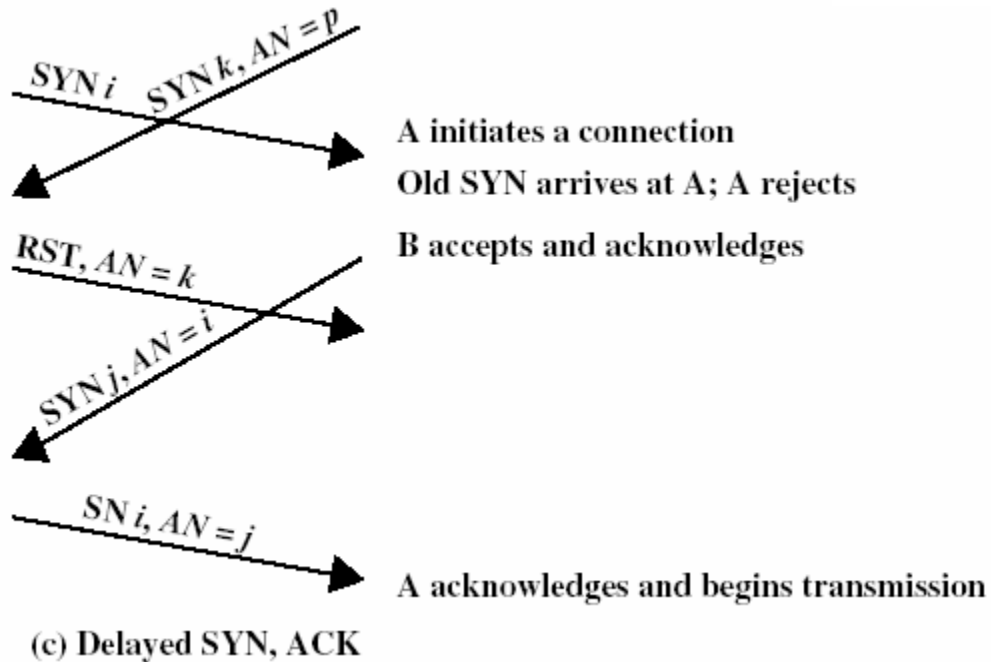
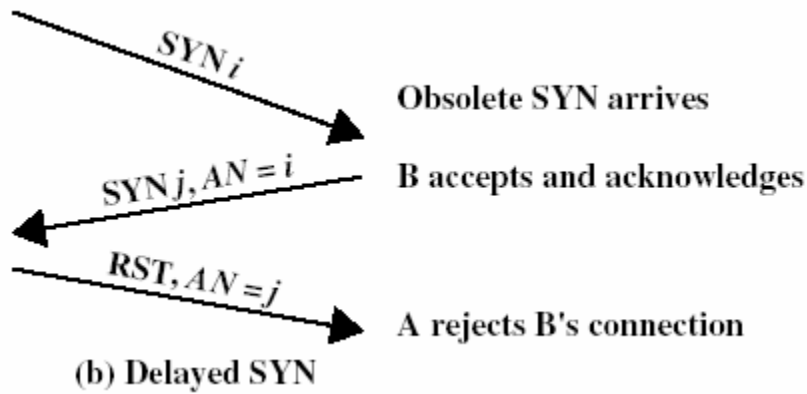
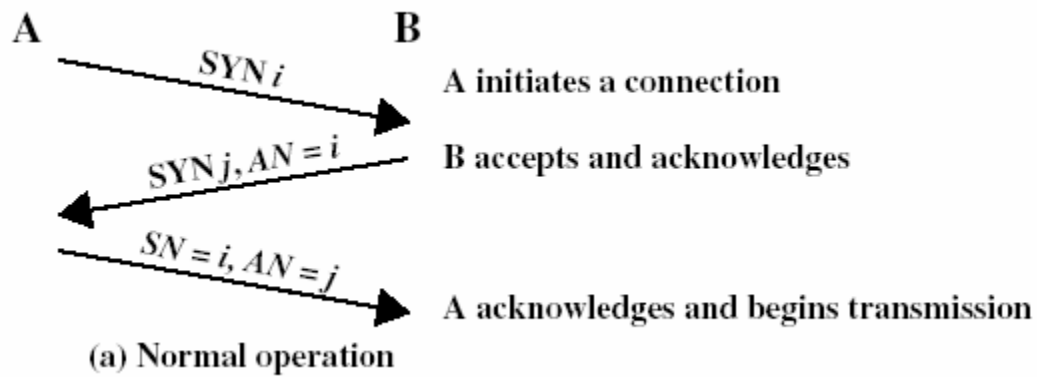
Possible solution to above mentioned problem is to start each new connection with a different sequence number, as shown in figure below.



Three way handshake requires sender say A to first send $SYN\ i$ to B, where i is the sequence number for the first frame fragment of A.

The other side, say B, responds by sending another SYN that acknowledges the sequence number i of A and includes its own sequence number j which will be used as Sequence number in B's first segment.

Now A acknowledges the $SYN\ j/ACK\ i$ received from B in its first data segment, as shown below.



Crash Recovery

When the system upon which a transport entity is running fails and subsequent restarts, the state information of all the active connections is lost. The affected connection the other side becomes half open.

It may also happen that the system that crashed never turns up. Since the other system has crashed permanently, sender will retransmit multiple times the same data fragment, and he will never receive response. So for how long he should continue sending data fragment. There is need to define the persistent timer, it is the time duration for which sending transport entity retransmits the data fragment even without receiving any response, and after this timer expires connection is closed.

Second scenario is the system goes down for a moment and starts immediately, the other side, i.e. sending transport entity will then have a half-open connection. Since the system has come up again, but he has lost all state information and is not aware of any active connection. So the response to every data segment that crashed-recovered system or failed system receive is sending a RST i segment. When RST i is received by sender, he abnormally terminates the connection.

TCP (Transmission Control Protocol)

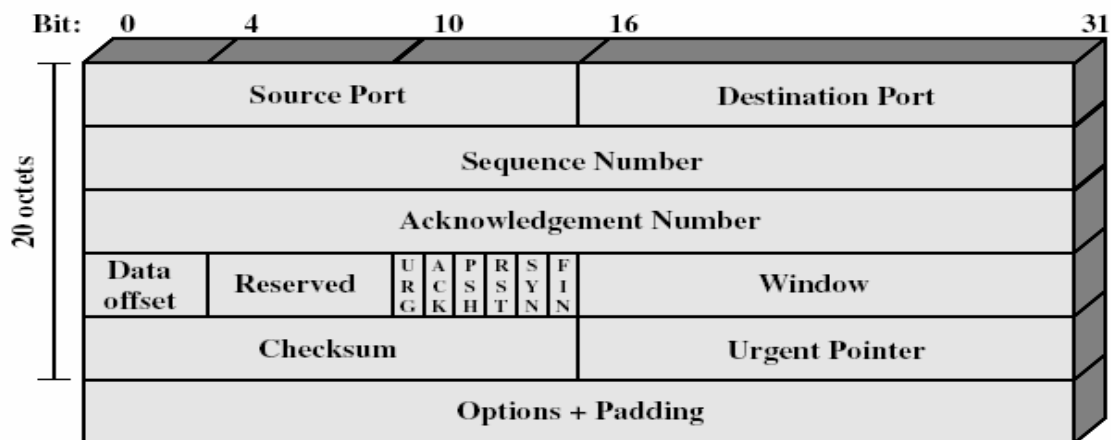
There are two transport layer protocols, one is connection oriented and the other one is connection less. Connection oriented protocol is TCP and connection less protocol is UDP. We will first discuss TCP protocol and UDP protocol.

Within TCP protocol, we will discuss TCP services, TCP header format, and TCP mechanisms.

TCP Services

TCP is designed to provide reliable service between TS users which are connected through a variety of reliable and unreliable networks and internets. TCP is a **stream-oriented** protocol, which means that it accept stream of octets from the user, group them into segments and number each octet in the stream. The unit of transfer between two TCP entities is **segment**.

TCP Segment format



Description of field is as follows

1. **Source Port** (16 bit): Specifies the port number at which the connection is to be established at the source end.
2. **Destination Port** (16 bit): Specifies the port number at which the connection is to be established at the destination end.

3. **Sequence Number** (32 bit): This 32-bit field defines the number assigned to the first byte of data contained in this segment, as we said above regarding TCP is a stream-oriented protocol. The sequence number tells the destination which byte in this sequence comprises the first byte in the segment. As told above in three way hand shake, during connection establishment, each user creates an initial sequence number. For example if A chooses initial sequence number to be say, 420, and the first segment is carrying 1000 bytes, the sequence number is 422 (because 420, 421 are used for connection establishment). And the second segment will have the sequence number 1422 and so on.
4. **Acknowledgment Number** (32 bit): This 32 bit field defines the byte number that the TCP entity expects to receive next. If the receiver has successfully received segment till byte N, the acknowledgment number will be N+1.
5. **Data offset** (4-bit): Also known as Header length in some of the text. This 4-bit field indicates the number of 4-byte words in header length. The minimum length of the header is 20 (5*4) or 60 bits.
6. **Reserved** (6 bit): Reserved for future use.
7. **Flags** (6 bit):
 - URG: Urgent Flag indicates that the value of urgent pointer be considered and is valid.
 - ACK: Acknowledgment field indicates that the acknowledgment field value be considered and is valid, it is used in context with piggybacking.
 - PSH: Generally, when the TCP user has sufficient data such that a segment can be formed, it does so and sends the data segment to transport layer for transmission. But for music kind of data, one cannot afford to wait for filling up the whole data segment, so wherever stream of data is expected continuously, data stream push is specified.
 - RST: Reset the connection
 - SYN: Synchronizes the connection or for establishing the connection.
 - FIN: no more data to send, for termination of the connection.
8. **Window** (16 bit): This is for flow control credit allocation, in octets. It contains the number of data octets that the other party must maintain.
9. **Check-Sum** (16 bit): This field is meant for error detection. The checksum field applies to the entire segment plus a **pseudoheader** prefixed to the header. Pseudoheader include the following fields from the IP header : Source IP Address, Destination IP Address, protocol, and segment length field. By including the pseudoheader, **TCP protects itself from misdelivery by IP**. That is, if IP delivers a segment to the wrong host, even if the segment contains no bit error, the receiving TCP entity will detect the delivery error.
10. **Urgent Pointer** (16 bit): It points to the last octet in a sequence of urgent data. This allows the receiver to know how much urgent data are coming. This field is only valid if the URG flag is set. It defines the number that must be added to the sequence number to obtain the number of the urgent byte in the data segment.

Detailed Discussion: Since TCP is a stream oriented protocol, this means data is presented from the user to the TCP service as it appears. TCP service forms the segments out of it and transmits the segments one of the other. Since TCP is a connection oriented protocol, a virtual connection also needs to be established before the transmission of segment and all the segments must follow the same path (virtual circuit) to reach the destination and thus arrive in order. Now suppose that at the user end (i.e. application layer), some wrong data is passed to the TCP service, which will be transmitted sooner, so general tendency is to press Ctrl + C. This will also become part of the same segment. Sending station will like that at the receiver end Ctrl + C is processed first. To make is processed first before the rest of data, and Ctrl + C being present in the later part of the data, URG Flag is set and Urgent Pointer contains the value of the byte at which Ctrl + C happens to be in the segment.

When the receiving TCP receives a segment with the URG flag set, it looks for the value of Urgent Pointer, extracts the urgent data using the value of urgent pointer and delivers it out of order to the receiving application.

11. Options (Variable): For future use.

We will now discuss Connection establishment, Data Transfer, and Connection Termination in context to TCP.

Connection Establishment and termination: Earlier we discussed two approaches to connection establishment, one with the connection oriented network layer and the other one with the connection less network layer. The approaches were Two-way hand shake and Three-way handshake. TCP uses Three-way handshake for connection establishment and connection termination.

Data Transfer: TCP uses sliding window flow control, which a window of fixed size is used at both the sender and receive end. When ever sender has sufficient data to form a segment, a segment is formed and is transmitted. Maximum number of segments that can be transmitted without receiving the acknowledgment depends upon the size of the window.

Various policies implemented by TCP are as follows

1. Send Policy
2. Deliver Policy
3. Accept Policy
4. Retransmit Policy
5. Acknowledge Policy

Policy means how TCP protocol behaves or works or handles sending of data, delivery of data and so on.

Send Policy: As discussed above, TCP is a stream oriented protocol, Transport layer receives a stream of data, form segment whenever sufficient data is available for forming the segment or other-wise if sufficient data is not available and data is to be pushed, then without waiting for sufficient data to form the segment. Whatever data is available segment is formed and is transmitted.

Accept Policy: Since TCP is a connection oriented protocol, a virtual connection is to be established for delivery of data, but type of connection established actually depends upon

the network layer, because it is network layer who is responsible for routing the data packets. If the underlying network layer is connection oriented, then data packets will arrive in order. But if the underlying network layer is connection less, then data packets may arrive in order or may arrive out of order. If the data packets arrive out of order, then transport layer has to reassemble the data segments to form the original data. There are two approaches for reassembling data that is received out of order

1. **In-order:** Accepts only segments that arrive in order, any segment that arrive out of order are discarded, although this is not a practical approach.
2. **In-window:** Accepts all segments that are within the receive window.

Retransmit Policy: TCP maintains with itself a queue of segments that have been sent but not yet acknowledged. TCP also maintains retransmission timer, which starts after a segment is transmitted, and which goes off after certain duration of time, and if no acknowledgment is received before the timer goes off, that segment is retransmitted. There are three policies for retransmission

1. **First Only:** It maintains one retransmission timer for the entire queue. If an acknowledgment is received remove the appropriate segment from the queue and reset the timer. If no acknowledgment then, retransmit only one segment which is at the beginning of the queue.
2. **Batch:** It maintains one retransmission timer for the entire queue. If an acknowledgment is received remove the appropriate segments from the queue and reset the timer. If no acknowledgment then, retransmit all segments in the queue.
3. **Individual:** It maintains one retransmission timer for each individual segment. If Acknowledgment is received remove the appropriate segment or segments from the queue, and destroy the corresponding timer or timers. If no acknowledgment, retransmit the corresponding segment and reset the timer.

Acknowledge policy

1. **Immediate:** When segment is accepted at the receiver side, immediately an acknowledgment is sent containing the appropriate acknowledgment number.
2. **Cumulative:** When segment is accepted, look for the need of acknowledgment, if there is enough data to send back (piggybacking), then piggyback the acknowledgment, otherwise wait for enough data to accumulate. But receiver cannot wait for long duration of time, so another timer is started, called as window timer. If enough data is accumulated before the window timer goes off, then acknowledgment is piggybacked and window timer is reset. If enough data is not accumulated before the window timer goes off, then transmit an empty segment only containing the appropriate acknowledgment number.

Retransmission Timer Management

As discussed above, a timer is maintained by the source which goes off after a certain period of time. If acknowledgment is received before the timer goes off, it is fine, but if no acknowledgment is received and the timer goes off, then, retransmission of the corresponding segment is done, we are taking into consideration Individual Retransmit Policy. Now the question arises

How much should be the value of retransmission timer?

Let's see why this question is important, and what is so difficult about it. A TCP user on one machine may establish a connection with a TCP user who is sitting next to his room (means that the two users are not far apart).

There may be a case when a TCP user on one machine may establish a connection with a TCP user who is geographically far apart.

If my machine maintains a constant value of retransmission timer, then it will fit to either of the above mention two cases. That constant value may be either too less for the second case or may be too large for the first case.

This means that the TCP cannot use the same retransmission time for all connections.

Selecting a fixed retransmission time for all connections can result in serious consequences. If the retransmission time does not allow enough time for the segment to reach the destination and an acknowledgment to reach the source, it can result in retransmission of segments that are still on the way. Conversely if the retransmission time is longer, it may result in inappropriate delay.

We want the value of retransmission timer to change with the changing connection, taking the underlying network conditions into consideration. Also traffic is bursty, means sometimes there is lot of traffic and sometimes there is very few traffic, thus retransmission timer need to adjust itself with the changing network conditions.

Retransmission time can be made dynamic by basing it on the round trip time (**RTT**). The most common way to calculate the value of retransmission time is

$$\text{Retransmission time} = 2 * \text{RTT}$$

How to calculate Round-Trip Time (RTT) now, since retransmission time is not dependent upon Round-trip Time (RTT). A lot of research has been done on calculating the value of Round-Trip Time.

Simple Average

One way of calculating the RTT is, TCP sends a segment, starts a timer, when the acknowledgment arrives back, that value is taken as the value of the RTT. And for the subsequent calculation of RTT, an average is taken.

$$\text{ARTT}(N) = \frac{1}{N} \sum_{i=1}^{i=N} \text{RTT}(i)$$

The above equation can be rewritten as

$$\text{ARTT}(N) = \frac{N-1}{N} \text{ARTT}(N-1) + \frac{1}{N} \text{RTT}(N)$$

Exponential Average

Note that each term in the summation is given equal weight; i.e. each term is multiplied by the same constant $\frac{1}{N}$, infact best approach is to give higher weight to recent instances

because they are most likely to reflect future behavior and give lower weight to past instances. This is achieved by using a constant α

$$\text{RTT} = \alpha * \text{previousRTT} + (1 - \alpha) * \text{currentRTT}$$

RTT Variance Estimation

The above formulas are used to calculate the value of Round-Trip Time, however they did not cope well with a situation in which the round-trip time exhibits a relatively high variance. For example

1. If the data rate on a connection is very low, then the transmission delay will be relatively large compared to propagation time. Thus actually propagation delay is very less, but due to high transmission delay, RTT will also increase, which should not increase.
2. The peer TCP entity may not acknowledge each segment immediately because of its own processing delays and because it exercises its privilege to use cumulative acknowledgments.

A variable measure that is easy to estimate is

$$\text{MDEV}(X) = E[|X - E[X]|]$$

Where $E[X]$ is the expected value of X , we will apply the same formulae for calculating the variance of RTT.

$$\text{AERR}(N) = \text{RTT}(N) - \text{ARTT}(N-1)$$

Where ARTT is the simple average defined above

And AERR is the mean deviation

$$\text{ADEV}(N) = \frac{1}{N} \sum_{i=1}^N | \text{AERR}(i) |$$

$$\text{ADEV}(N) = \frac{N-1}{N} \text{ADEV}(N-1) + \frac{1}{N} | \text{AERR}(N) |$$

Where ADEV is the average deviation

These results were given by Jacobson.

Jacobson results can significantly improve the TCP performance but there are two other factors

1. Which roundtrip samples should be used as input to Jacobson's algorithm?

Karn's algorithm provides solution to above stated problem.

Suppose that a segment is not acknowledged and the retransmission timer goes off. Now the sender TCP will retransmit the segment. After retransmitting, sending TCP receives the acknowledgment. Now he does not know whether this acknowledgment is for the original segment or for the retransmitted segment.

Karn's solution is very simple, Do not consider the RTT of a retransmitted segment in the calculation of the new RTT. Do not update the value of RTT until you send a segment and receive an acknowledgment without the need for retransmission.

TCP congestion Control

We have been discussing a lot about retransmission, But what is the reason for retransmission. The reason is either the segment didn't reach the destination or the acknowledgment didn't reach the source or there is an error in the segment. If there is an error, that can be resolved by retransmitting. But if it is due to delay, then question arises where the delay is?

Delay can be at two places. Consider the scenario below

A packet started at sender may pass through several routers before reaching its final destination. Every router in the path stores the incoming packet in its buffer, processes it and further transmits the packet on one of the outgoing line. If a router receives a packet faster than it can process them, he will start buffering them, but there is a limit to buffering, a time may come when the buffer of router will be full. At this time, he may have to discard the packet. So a packet is not able to reach the destination in this case and therefore no acknowledgment from the receiver. The sender has no choice but to retransmit the lost packet. This may further alleviate the congestion, since already our router who was a part of the path is overloaded, which means more retransmission and more congestion.

Let's try to understand the difference between congestion control and flow control. In the case of flow control receiver is overwhelmed with the data, and he cannot accept any more data than he can process. In Congestion control, the problem lies in the network, the router which is part of the network route is congested or overwhelmed with so many packets which exceeds his buffer and his processing capacity.

So solution to above problem is again determining the size of the window. In flow control, size of the window was determined by the receiver or he can reduce the size of window thus limiting the number of segments that source can send. But with congestion, where the problem lies in the network, a new concept comes into picture, and that is congestion window.

Now the size of the window is a dependent upon the receiver window and congestion window, formulae for that is

Actual Window Size = minimum (Receiver window size, congestion window size)

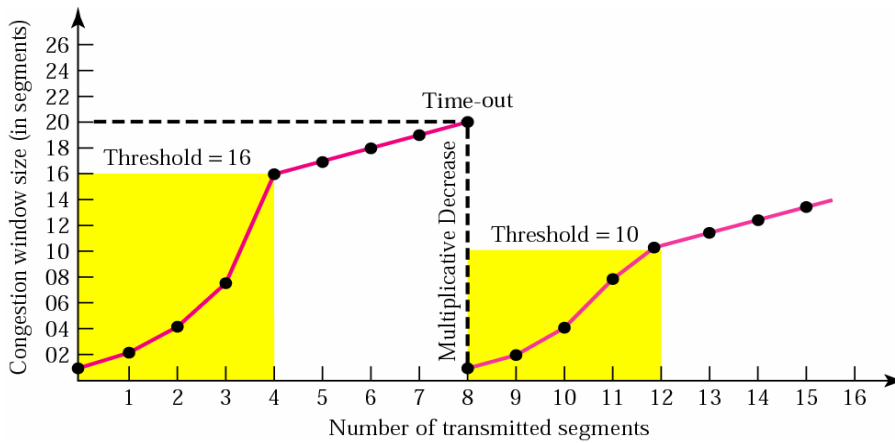
Now window can be managed by two techniques

1. **Slow Start:** At the beginning of the connection, TCP sets the congestion window size to the maximum segment size. For each segment that is acknowledged, TCP increases the size of the congestion window by one maximum segment size until it reaches a threshold of half the allowable window size. This is called slow start. Infact the size of the window increases exponentially. The sender sends one segment, receives one acknowledgment, increases the size to two segments, sends two segments, receives acknowledgments for two segments, increases size to four segments and then to 8, and so on.
2. **Additive Increase:** To avoid congestion before it happens, one must slow down this exponential growth. After the size reaches threshold, the size is increased one segment for each acknowledgment even if the acknowledgment is for several segments, i.e. there will be a linear growth. The additive increase strategy continues as long as the acknowledgment arrives before the timeout.
3. **Multiplicative Decrease:** If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that the congestion has occurred is through a lost segment. If a sender does not receive an acknowledgment for a segment before its retransmission timer has matured, it assumes that there is a congestion. The strategy says, once the timeout occurs, the threshold must be set

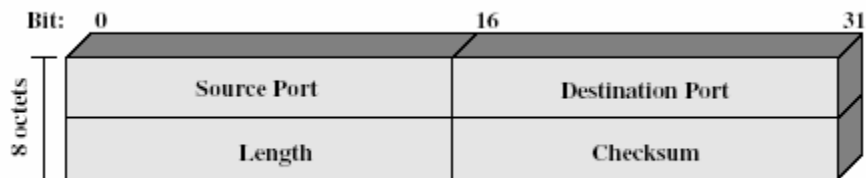
to half of the last congestion window size, and congestion window size should start from one again. For every received acknowledgment just double the size as we discussed above in slow start.

The above process can be summarized as follows

1. Set the threshold equal to half the current congestion window. Here CWND=16, so threshold is equal to 8
2. Set cwnd=1 and perform the slow start process until cwnd is equal to threshold, cwnd will be increased by 1 for each received acknowledgment.
3. After cwnd is equal to threshold, increase cwnd by one for each acknowledgment.



UDP (User Datagram Protocol)



This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.