LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 634: Networks Programming

Lecture 03: Review of Basic Networking Concepts (TCP/UDP)

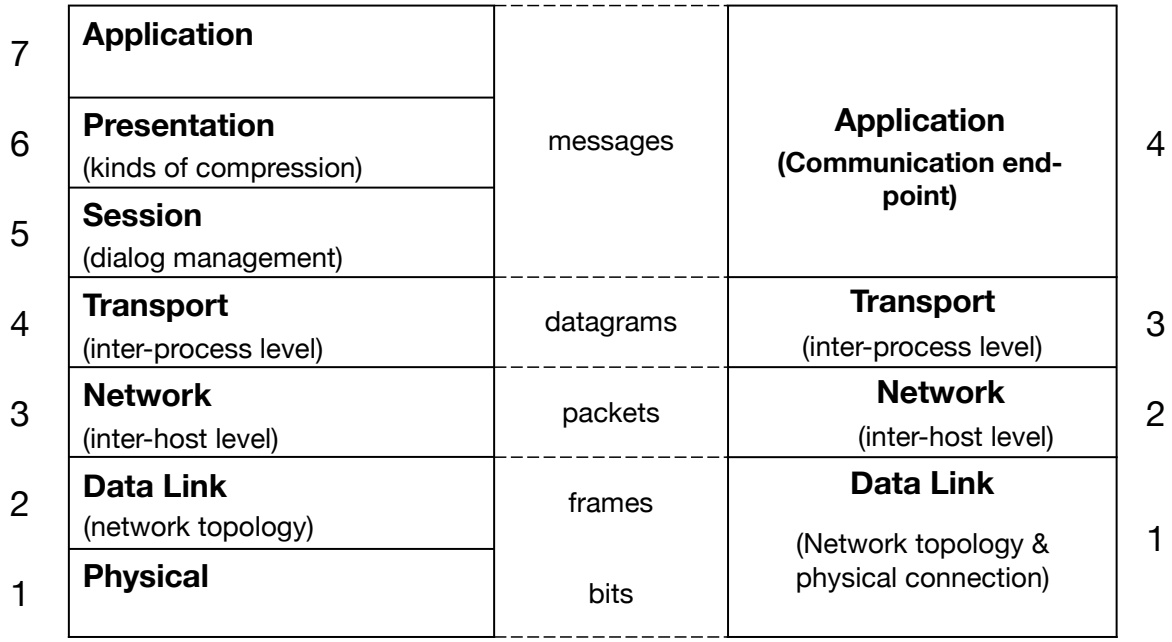
Instructor: Haidar M. Harmanani

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Recap

7-Layer OSI Model

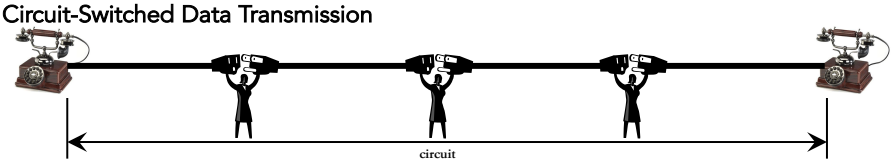
7	**Application**	messages
6	**Presentation** (kinds of compression)	messages
5	**Session** (dialog management)	messages
4	**Transport** (inter-process level)	datagrams
3	**Network** (inter-host level)	packets
2	**Data Link** (network topology)	frames
1	**Physical**	bits



4-Layer Simplified Model of TCP/IP

Communication Networks can be divided into two basic types by method of data transmission: circuit-switched and packet-switched.

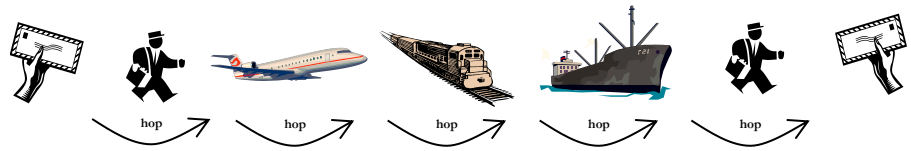
Circuit-Switched Data Transmission



- Non-shared dedicated communication line is established
- Information transmitted without division
- Connection established once, then all data transmitted through this connection.

The TCP/IP Internet uses packet-switched data transmission, provided by IP (Network) layer, responsible for forwarding of IP packets.

Packet-Switched Data Transmission

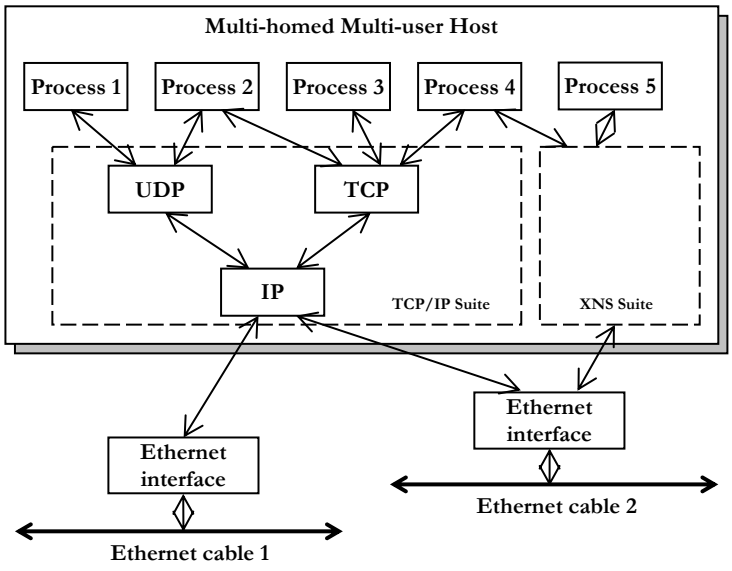


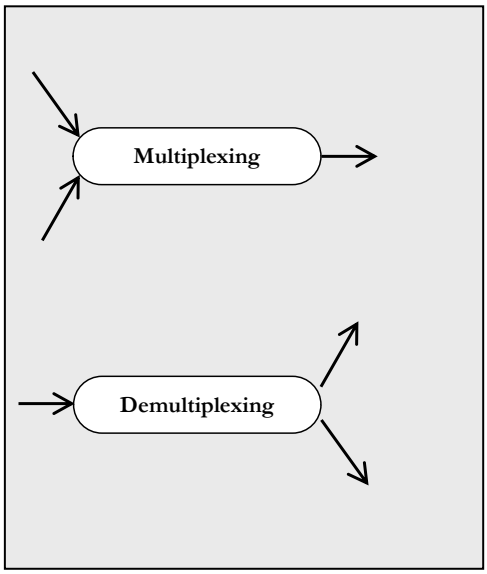
- Shared communication links are used instead of dedicated line
- Information is divided into pieces – packets.
- Each packet contains the address of destination and separately routed over shared data links.

Data Transmission Method

Multiplexing and Demultiplexing in TCP/IP Example.

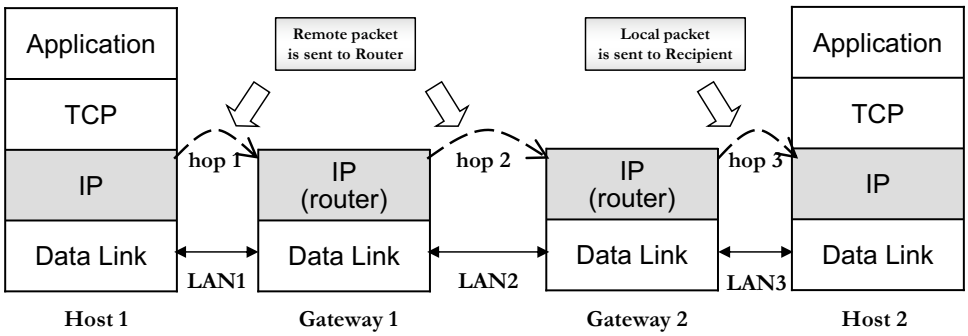
Multiplexing means "to combine many into one".
For network this means combining of data accepted from different functionalities of neighbor layer.
Demultiplexing is reverse of multiplexing.





Routing

Router is "intelligent" gateway, making a decision, what **route** (path) the packet should take.



In TCP/IP routing is made on IP Layer. Each packet could have its own route.

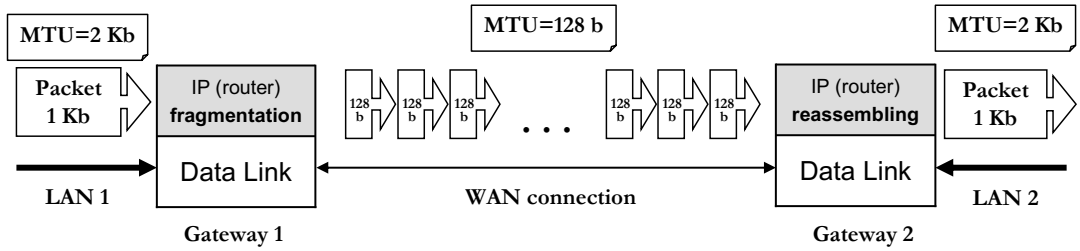
The TCP/IP Internet uses Distributed Dynamic Routing.

Distributed Dynamic Routing

uses a mixture of global and local information to make routing decision.

Fragmentation and Reassembling

1. **Fragmentation**: Breaking up of a packet into smaller pieces (transmission units).
2. **Maximal Transmission Unit (MTU)**: Maximal packet size held by network layer, depending on Data Link characteristics
3. **Reassembling: R**everse of fragmentation.



In TCP/IP fragmentation and reassembling is done at IP layer.

The IP layer performs these activities depending on requirement of specific Data Link layers,

hiding the technological differences between the networks.

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Part II: Transport Layer

Transport Layer: Port Numbers

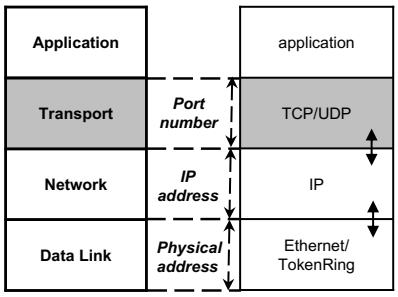
Port Number is unique 16-bit identifier of Application Communication Endpoint, used by Transport Protocols.

TCP ports and UDP ports are independent.

Servers are normally known by their *well-known* port number.
For example:

- FTP server always uses TCP port 21.
- Telnet server is on TCP port 23.
- TFTP server is on UDP port 69.

Clients usually need any unique free port. Client port numbers are called *ephemeral* ports.



Port Number Reservation:

- 1– 1023 *Well-Known Ports:* Used by well-known servers across all Internet.
- 1024–49151 *Registered Ports:* Used by servers known across specific networks.
- 49152–65535 *Dynamic Ports:* Used as ephemeral port numbers.

The ranges of Registered and Dynamic Ports are configurable.

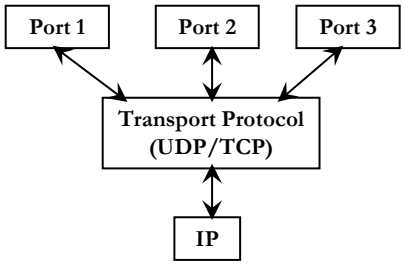
File /etc/services

on most Unix systems the well-known port numbers are contained in this file.

```
#grep telnet /etc/services
telnet 23/tcp

#grep domain /etc/services
domain 53/udp
domain 53/tcp
```

Transport Protocols provide multiplexing / demultiplexing based on Port Number



Association and Transport Address

Communication link between two processes is completely specified by **5-tuple Association**:

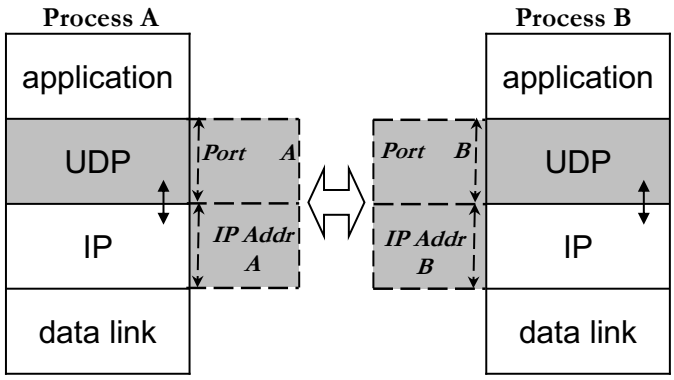
- Transport Protocol (UDP / TCP)
- Local IP Address
- Local Port
- Foreign IP Address
- Foreign Port

Transport Address (or Socket) is half-Association:

- Transport Protocol (UDP / TCP)
- Local IP Address
- Local Port

Socket Pair defines the Association

UDP Association Example

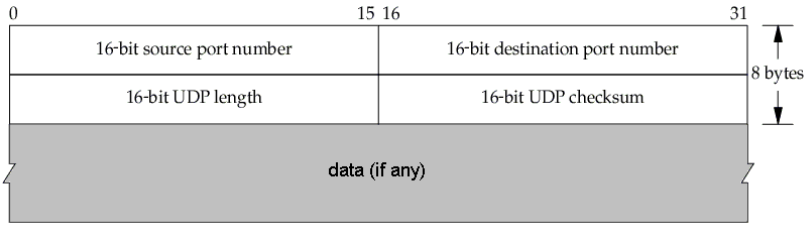


Socket A = { UDP, Port A, IP Address A }

Socket B = { UDP, Port B, IP Address B }

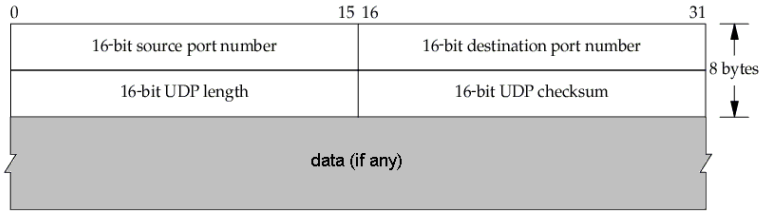
Association = { UDP, Port A, IP Address A, Port B , IP Address B }

Transport Layer: UDP Protocol



- UDP – User Datagram Protocol
  - Provides unreliable connectionless datagram delivery service.
- UDP is a simple protocol
  - Each output operation by a process produces exactly one UDP datagram, which causes one IP datagram to be sent.

Transport Layer: UDP Protocol



- Source and Destination Port Numbers
  - Identify the sending process and the receiving process
- UDP length
  - The length of the UDP header and the UDP data in bytes.
  - Can be calculated from IP header
- UDP checksum
  - Covers the UDP Pseudo-Header (see below) and the UDP data.
- In order to double-check that the data has arrived at the correct destination, the UDP checksum is calculated on UDP Pseudo-Header, containing:
  - UDP header itself
  - IP Header fields: Source IP Address, Destination IP Address, Protocol type.
- DATA field may be empty

Reliable Stream Transport: TCP

- Computer communication networks provide unreliable packet delivery
  - Packets maybe lost, destroyed, duplicated, delivered out of order or after a delay
- At high level, application programs send a large volume of data from one computer to the next
  - Hard to use unreliable connectionless delivery
  - Programmers have to build error detection and recovery into each application program

Reliable Stream Transport: TCP

- TCP is remarkable since it solves a difficult problem well
  - No other protocol has proved to work better
- TCP has various properties
  - Connection Oriented
    - An application must first request a connection to a destination
  - Point to Point Communication
    - A TCP connection has two endpoints
  - Complete Reliability
    - Data sent across a connection will be delivered exactly as sent
  - Full Duplex Communication
    - A TCP connection allows data to flow in either direction

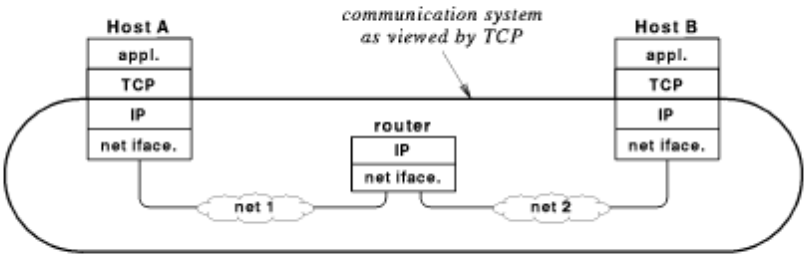
Reliable Stream Transport: TCP

- TCP has various properties (Cont.)
  - Stream Interface
    - An application sends a continous sequence of octets across a connection
  - Reliable Connection Startup
    - Two applications that create a connection must agree to the new connection
  - Graceful Connection Shutdown
    - An application can open a connection, send data, an then request to shutdown the connection

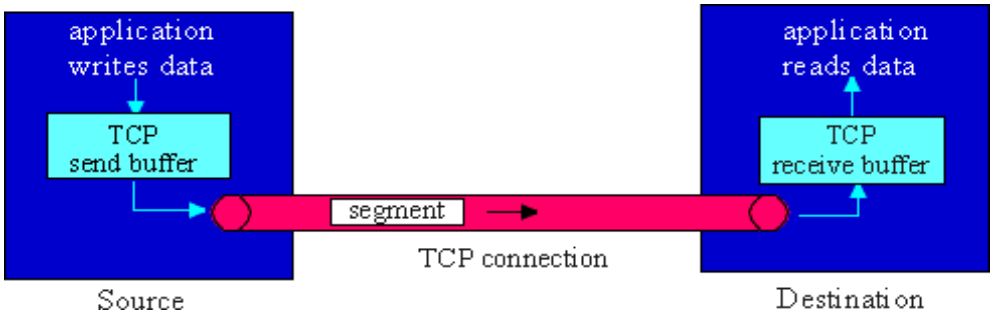
End-to-End Service and Datagrams

- End-to-End protocol
  - Provides a connection directly from an application on one computer to an application on another computer
- A TCP connection is a virtual connection
  - Achieved through software since the underlying Internet does provide hardware or software support for connections
- TCP treats IP as a packet communication system that connects hosts at two endpoints of a connection while IP treats a TCP message as data to be transferred
  - A TCP message is encapsulated in an IP datagram
  - When a datagram arrives on the destination host, IP passes the contents to TCP
  - IP does not read or interpret the messages

End-to-End Service and Datagrams



Flow of TCP segments



Achieving Reliability

- Major problems
  - Unreliable delivery by the underlying communication system
  - Computer system reboot
- The first problem is obvious since packets may be duplicated, lost, delayed, or delivered out of order
- Computer reboot is another difficult problem
  - If a connection is established and one computer is rebooted is a problem since the protocol software on the computer that did not reboot considers the connection to be valid
  - Must be able to reject packets from a previous reboot

Achieving Reliability

- TCP converts the unreliable, best effort service of IP into a reliable service
  - It ensures that each segment is delivered correctly, only once and in order
- Converting an unreliable connection into a reliable connection is basically the same problem at the data link layer, and essentially the same solution is used:
  - TCP numbers each segment and uses an ARQ protocol to recover lost segments.
  - Some versions of TCP implement Go Back N and other versions implement Selective Repeat.

Transport Layer: TCP – Transmission Control Protocol

- Provides reliable connection-oriented full-duplex byte stream service over unreliable connectionless packet-switched IP Network
- TCP provides reliability by doing the following:
  - The application data is broken into TCP Segments – the best sized chunks passed by TCP to IP.
  - Each TCP Segment has its Sequence Number.
  - When TCP receives data from the other end of the connection, it sends an Acknowledgment.
  - When TCP sends a segment it maintains a Timer. If an acknowledgment isn't received in time, the segment is retransmitted.
  - TCP maintains a Checksum on its header and data. If a segment arrives with an invalid checksum, TCP discards and doesn't acknowledge it, expecting the retransmission from sender after expiration of its timeout.
  - TCP Re-Sequences the data if necessary, passing the received data in the correct order to the application.
  - TCP provides discarding of duplicate data.
  - TCP provides Flow Control. Each end of a TCP connection has a finite amount of buffer space. A receiving TCP allows the sender to send as much data as the receiver has buffers for.

TCP

- However, there are a few important differences between the transport layer and the data link layer.
  - At the data link layer, we view a protocol as being operated between two nodes connected by a point-to-point link.
  - At the transport layer, this protocol is implemented between two hosts connected over network.
    - Packets can arrive out-of-order, and packets may also be stored in buffers within the network and then arrive at much later times.
    - The round-trip time will change with different connections and connection establishment is more complicated.

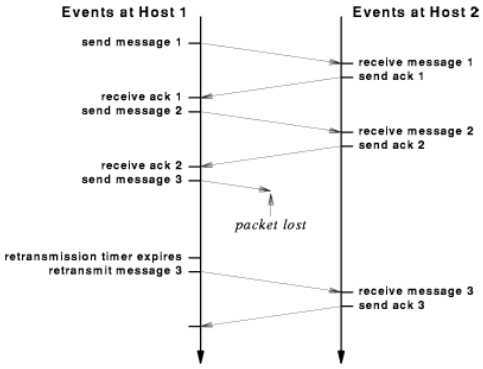
Achieving Reliability in TCP

- Use a positive acknowledgment with retransmission
  - Recipient communicate with source
    - Send back an acknowledgment message as data is received
  - Sender keeps a record of packets it sends and waits for an acknowledgment before sending the next packets
    - Start a timer when a packet is sent
    - Retransmit if timer expires before an acknowledgment is received
    - Duplicate packets are detected by assigning each packet a sequence number and requiring the receiver to indicate which sequence numbers it has received

Achieving Reliability in TCP

- How long should a TCP wait before retransmitting?
  - Waiting too long for an acknowledgment on a LAN leaves the network idle and does not maximize throughput
    - Retransmit after a few milliseconds
  - The same strategy does not work well on a long-distance satellite connection
- Another challenge for TCP is that bursts of datagrams can cause congestion
  - Can cause retransmission delays along a given path to change rapidly

Achieving Reliability in TCP

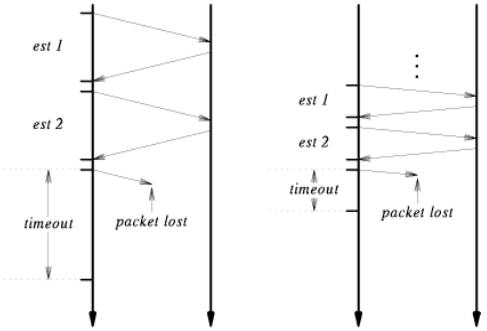


Adaptive Retransmission

- TCP uses an adaptive retransmission mechanism
  - Monitor current delay on each connection and change the retransmission timer to accommodate changing conditions
- How can TCP estimate the retransmission timeout?
  - Estimate the round trip delay for a connection
    - Whenever a message is sent to which a response is expected, TCP record the time at which the message was sent
    - When a response is received, TCP subtracts the time the message was sent from the current time
  - Generate a sequence of round trip estimates
    - TCP uses a statistical function to produce a weighted average
    - TCP estimates of the variance and uses a linear combination of the estimated mean and variance as a value for retransmission

Retransmission Times

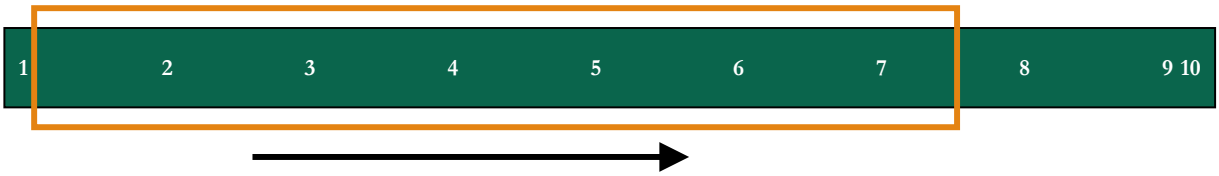
- TCP sets the retransmission timeout to be slightly longer than the mean round trip delay
  - Wait long to determine that a packet is lost but not longer than necessary



Sliding Windows

- A simple positive acknowledgement protocol wastes substantial amount of network bandwidth
  - Must delay sending a new packet until an acknowledgment is received for the previous packet
- Use a window mechanism to control the flow of data
  - Transmit multiple packets before waiting for acknowledgment
- The protocol places a fixed size window on the sequence of packets and transmits all packets that lie inside the window.
  - Window slides as long as acknowledgments are received
- Performance depends on window size and speed at which network accepts packets

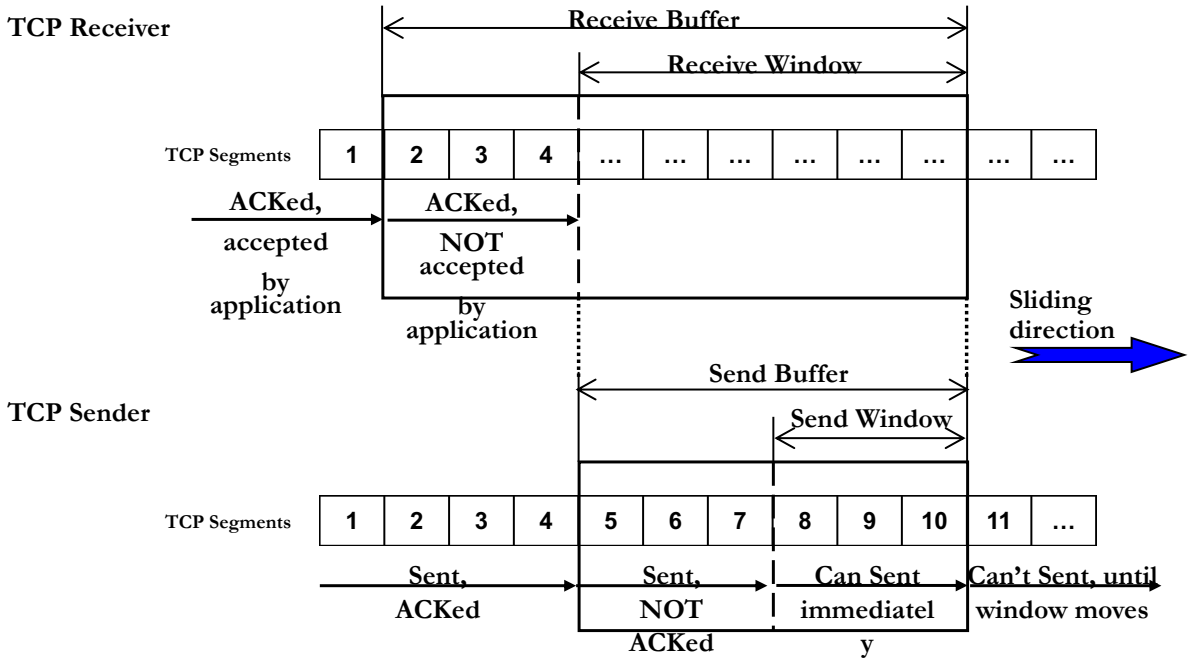
Sliding Windows



- Sender is permitted to transmit eight packets before receiving an acknowledgment
  - Allocate a buffer to hold incoming data and send the size of the buffer (8) to the other end
  - As data arrive, the receiver sends acknowledgment as well as the remaining buffer size
  - Buffer size notification is called a *window advertisement*

TCP Sliding Window

Sliding Window is algorithm of positive acknowledgement with re-transmission.



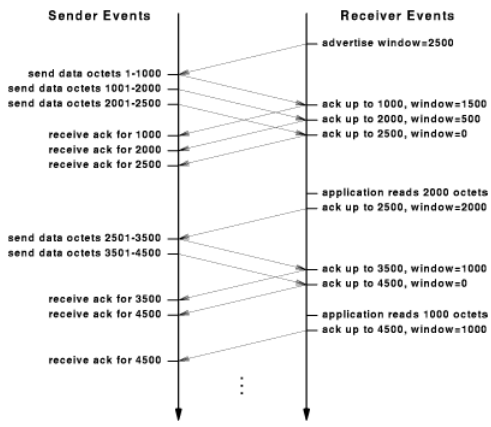
TCP Sliding Window

- Receive Window
  - Consists of any space in Receive Buffer that is not occupied by data.
  - Data remains in Receive buffer until target application will accept it.
  - The size of Receive Window is advertised by TCP Receiver to TCP Sender.
- Send Buffer
  - Begins from first un-acknowledged segment.
  - Has the size of advertised Receive Window Size.
- Send Window
  - Covers unused part of the Send Buffer.

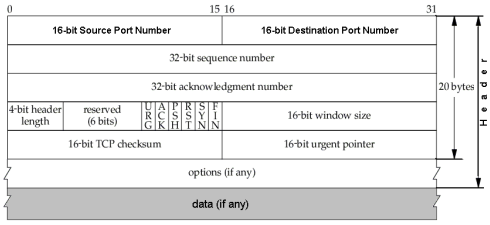
Sliding Windows

- If the receiving application can read data as quickly as it arrives
  - A receiver will send a positive window advertisement along with each acknowledgement
- If the sending side operates faster than the receiving side
  - Incoming data will eventually fill the receiver's buffer
  - Receiver advertises a *zero window*
  - The sender will not send any further data until the receiver advertises a positive window again

Sliding Windows

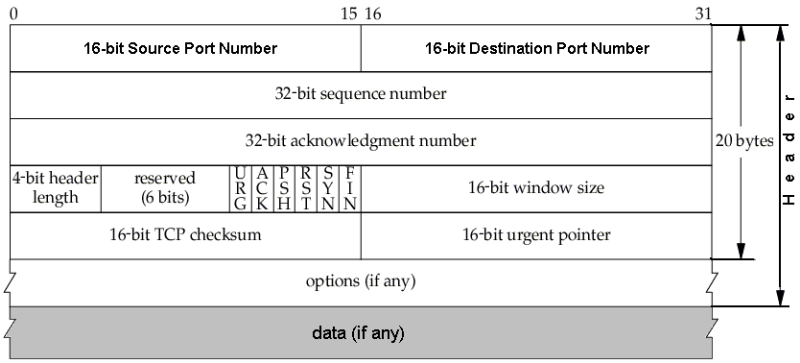


TCP Segment Structure



- Source and Destination port numbers
  - Identify the sending process and the receiving process.
- Sequence Number
  - Unique identifier of TCP Segment, equal to the sequence number of segment 1st data byte in stream between TCP sender and TCP receiver. Byte numeration begins from ISN (initial sequence number) chosen by TCP sender. Wraps back around to 0 after reaching 2^32 - 1.
- Acknowledgement Number
  - Used with ACK flag. The number of next byte, which TCP receiver is ready to receive.
- Header Length
  - The length of the header in 32-bit words.
- Flags: URG – urgent data (used with Urgent Pointer), ACK – acknowledge (used with Acknowledge Number)
  PSH - flush receiver data from its cache to process, RST – reset the connection ("port unreachable" reply)
  SYN – connect (used with Sequence Number = ISN), FIN – finalize the connection
- Window Size
  - Number of bytes, which receiver is ready to accept (flow control)
- TCP Checksum
  - Covers TCP Pseudo-Header (TCP Header + IP Addresses and Protocol fields) + TCP Data
- Urgent Pointer
  - Used with URG flag. Offset from Sequence Number to last byte of urgent data.
- Options (<=40 bytes)
  - Maximal Segment Size (MSS) announcement, Timestamp, Window scale, etc.

TCP Segment Structure



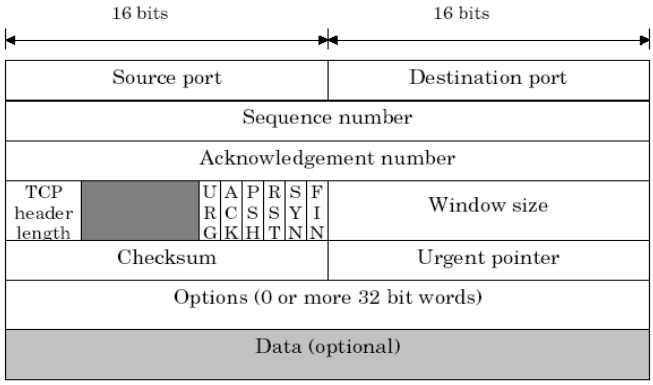
TCP Segment Format

- Source Port
  - Identifies the application program that sent the data
- Destination Port
  - identifies which application program on the receiving computer should receive the data
- Sequence Number
  - Indicates the sequence number for the data being carried in the segment
  - Used by the receiver to reorder segments that arrive out of order and to compute an acknowledgement number

TCP Segment Format

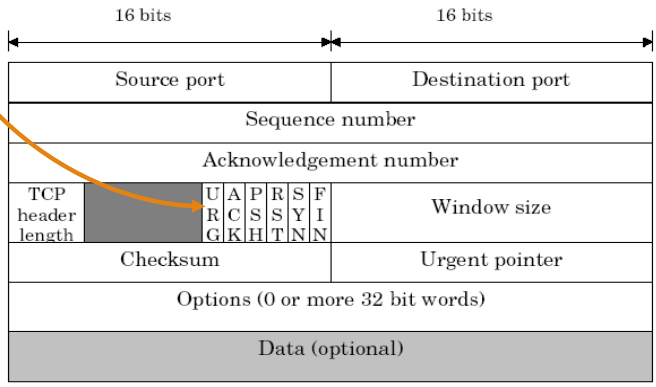
- Acknowledgement Number
  - Specifies the sequence number of the data that has been received
- Window
  - Specifies how much additional buffer space is available for more data
- Checksum
  - A checksum that covers the TCP segment header and the data

TCP Header



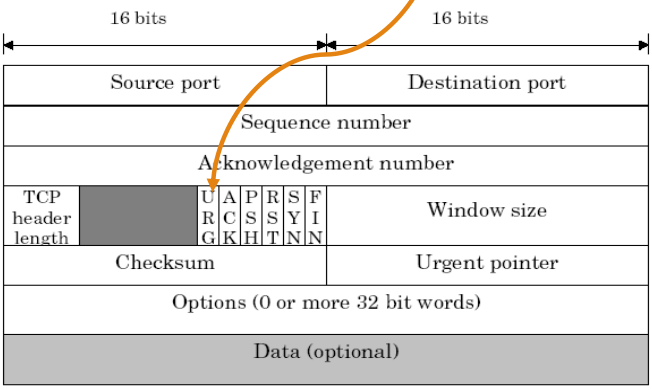
TCP Header

Connection establishment



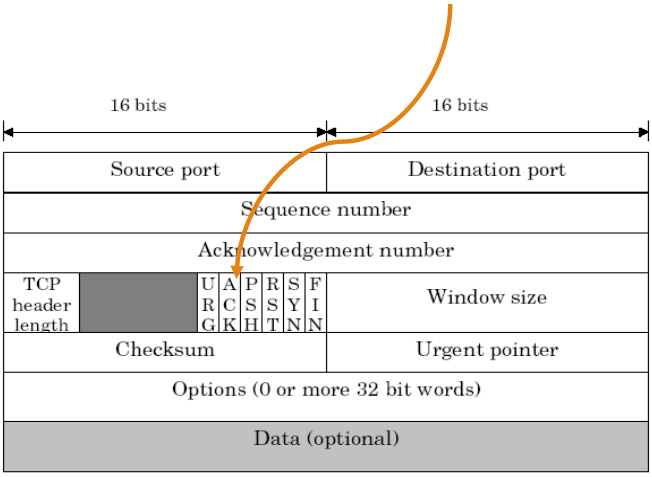
TCP Header

URG → Urgent Pointer: to allow the sender to signal the receiver without getting TCP itself involved in the reason for the interrupt.



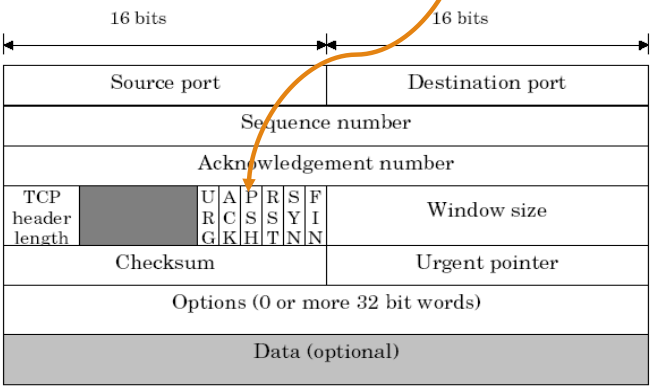
TCP Header

ACK → 1 (acknowledgement number is valid);
0 (ignore acknowledgement field)



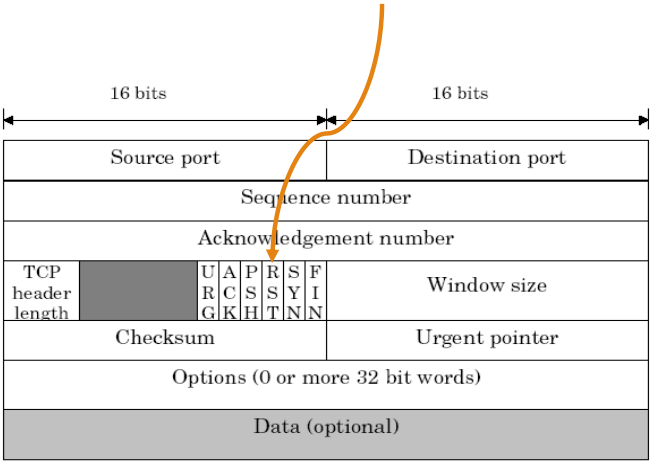
TCP Header

PSH → PUSHed data: receiver is requested to deliver the data to the application upon arrival and not buffer it.



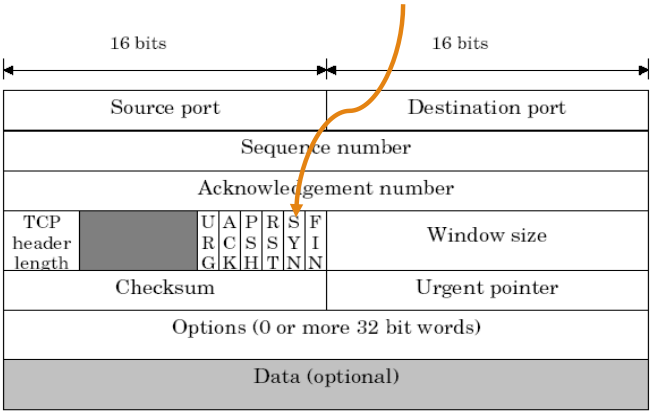
TCP Header

RST → reset a connection confused due to host crash or some other reason



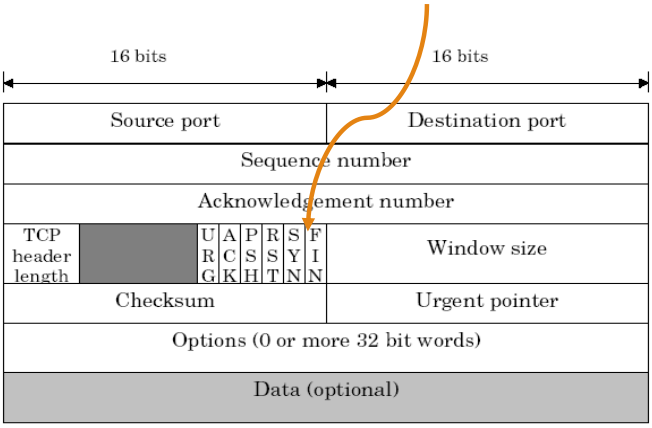
TCP Header

SYN → used to establish a connection; denote CONNECTION REQUEST and CONNECTION ACCEPTED, with the ACK bit used to distinguish between those two possibilities



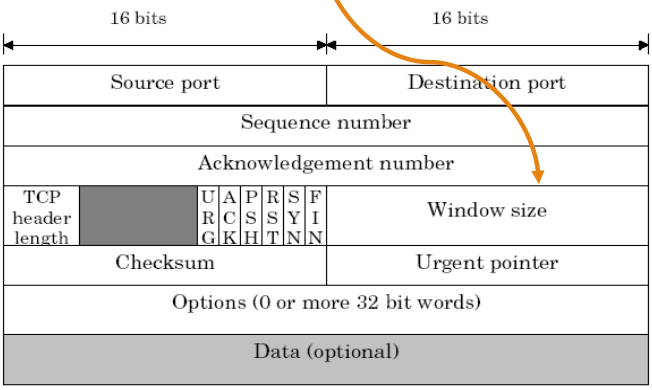
TCP Header

FIN → used to release a connection



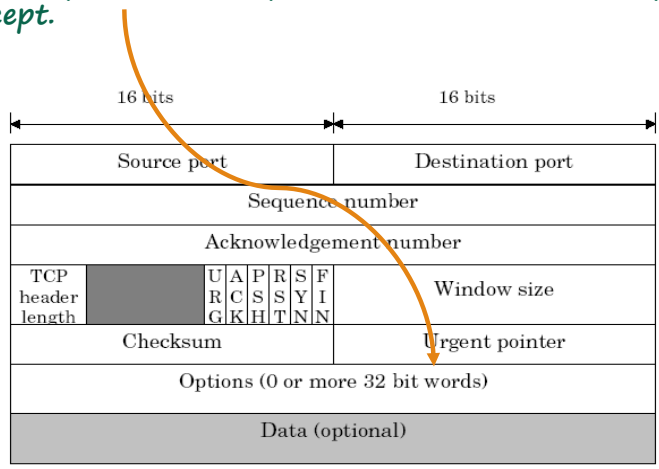
TCP Header

Flow control → sliding window



TCP Header

Provides a way to add extra facilities not covered by the regular header; most important option → host specifies the maximum TCP payload it is willing to accept.



Connection Establishment

- In TCP, a connection establishment phase is required, to ensure that the receiving process is available and to synchronize sequence numbers, etc.
- Connection establishment is a basic problem that arises in many other places as well. We look at the generic problem of connection establishment first.

Connection Establishment

- Consider the following simple connection establishment protocol
  - The client transmits a special packet requesting a connection.
  - The server responds to a connection request with an accept packet.
  - After receiving the accept packet the client and server begin sending data using some ARQ protocol with initial sequence numbers of zero.
  - If the client does not receive a reply to a connection request after a certain time, it times-out and retransmits the request.

This simple scenario is complicated by the fact that the network can lose, store and re-order packets.

Example: Suppose the client wants to send one data segment. First it requests a connection, then sends the data and closes the connection. Suppose the client times out after doing each of these steps and then retransmits. This results in the following sequence.

Client Transmits:

CONNECT CONNECT DATA(0) DATA(0) CLOSE CLOSE

Since data can be reordered in the network, the server could receive:

CONNECT DATA(0) CLOSE CONNECT DATA(0) CLOSE

In this case the server may think that the client opened two sessions and sent two different data packets.

Similar problems can arise if a client crashes and then re-opens a connection, while a packet from a previous connection is still in the network.

We will consider some possible solutions to these problems next.

First Solution

- The server could keep track of the last sequence number used for a given host and assign the next sequence number to start the next connection (this could be sent to the server in the "connection accept" packet).
- In the previous case, when the server received the second connect, it would tell the client to begin using sequence number 1. In this case when the server receives the second data packet 0 it would know that this was from a previous connection.

Problems

- There are still several problems with the above solution:
  - This requires the server to store in memory a sequence number for each connection even after the connection has been released. For large servers this is undesirable.
  - Even if the server keeps track of sequence numbers, they will eventually "wrap around" and be reused. If a packet is delayed in the subnet long enough an old packet can still be confused with a new packet that has the same sequence number.
  - Either the client or server could crash and loose track of the sequence numbers.

A connection establishment algorithm that avoids the above problems can be designed if the following two assumptions hold

(1) There is a maximum time that packets can be delayed in the subnet. (This type of guarantee must be provided by the network layer, e.g., with the use of the time-to-live field in IP packets.)

(2) Each host has a clock available, which it uses to decide on initial sequence numbers. This clock can be thought of as a counter with N bits, that increments every r seconds (N is the number of bits used for sequence numbers). We assume this clock keeps time even if the host crashes (to avoid the third problem above). These clocks do not need to be synchronized between two different hosts.

Given the above assumptions the following connection establishment procedure is used:

(1) The client picks an initial sequence number based on its clock and includes it in the connection request packet.

(2) The server responds to the connection request acknowledging the clients sequence number and including its own initial sequence number (based on its own clock).

(3) The client then sends its first data packet, using its initial sequence number and acknowledges the servers initial sequence number.

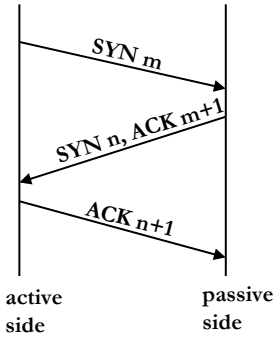
Both sides agree that the connection is established only after all three of these steps have taken place. This type of exchange is referred to as a *3 –way handshake*.

After a connection is established, the hosts increment their sequence numbers according to the rules of the ARQ protocol.

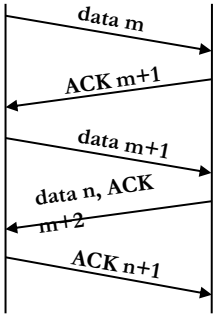
Establishing and Closing Connections

- TCP uses a three way handshake to guarantee that connections are established or terminated reliably
  - Necessary and sufficient to ensure unambiguous agreement despite packet loss, duplication, and delay
- Terms
  - Synchronization segment (SYN segment)
    - Messages in a 3-way handshake used to create a connection
  - Finish or FIN
    - Messages in a 3 way handshake used to close a connection

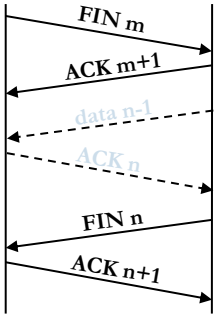
Transport Layer: TCP Protocol



TCP Connection Establishing. Three – Way Handshake.



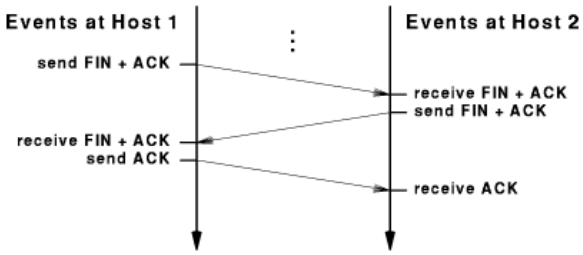
Simple data exchange example.



TCP Connection Termination. Modified Three – Way Handshake.

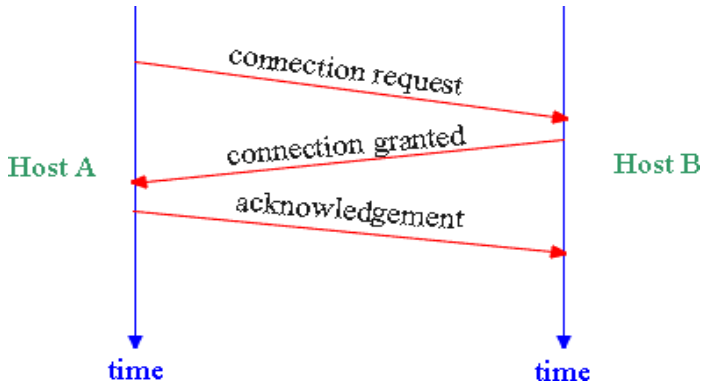
Three Way Handshake

- Part of the 3-way handshake used to create a connection requires each end to generate a random 32-bit sequence number
  - If an application attempts to establish a new TCP connection after a computer reboots, TCP chooses a new random number



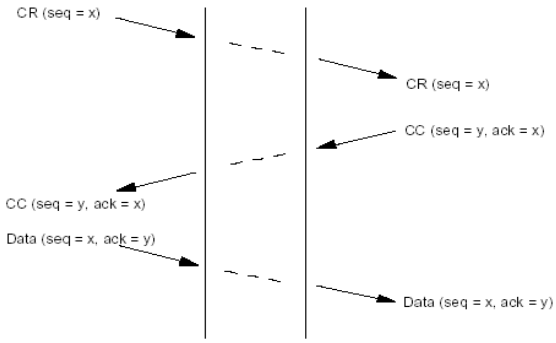
Three Way Handshake

1. A: I would like to talk to you B; A sends a SYN packet to B
2. B: Ok, let's talk; B sends a SYN-ACK packet to A
3. A: Thanks; ACK from A to B



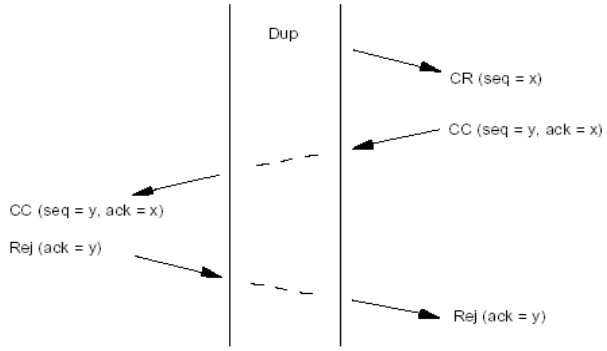
Three Way Handshake

Schematically, it looks like this:



CR= connection request, CC = connection accepted.

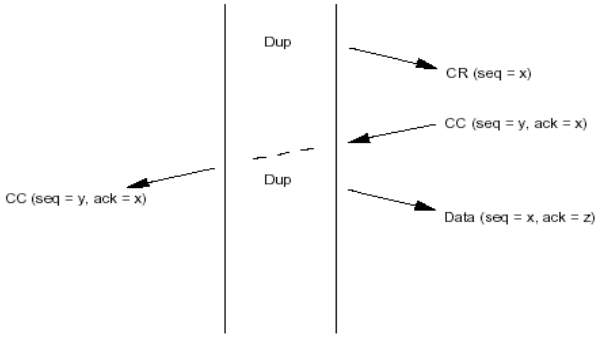
Avoiding Duplicate Connections



What happens if a duplicate CR shows up? The originator is informed as follows:

- The client knows that it has already given up on starting a connection with seq=x. It may tell the server, so that the server does not hold on to false connection (or the server may simply time-out and close the connection.)

Delayed Request and Delayed Third Leg



- As in the previous example, suppose that both the connection request packet and the first data packet sent from the client were repeated and show up after the connection is closed.
- In this case, how does the server know to not accept the data?

For the above protocol to work correctly, it needs to be guaranteed that two different packets in the network cannot have the same sequence numbers.

To prevent this, two things are needed:

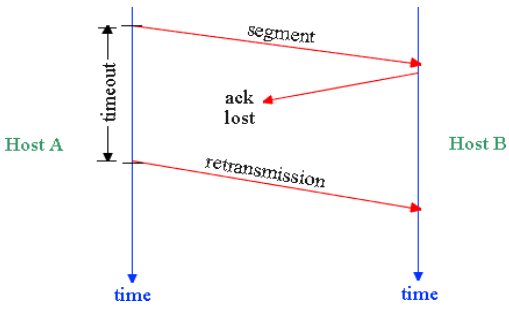
(1) The clock has to advance fast enough, so that two consecutive packets from the same host do not have the same sequence numbers.

(2) The time it takes for the sequence numbers to wrap around is longer than the time a packet can be delayed in the network. (This time depends on the number of bits in a sequence number and the rate of the clock.)

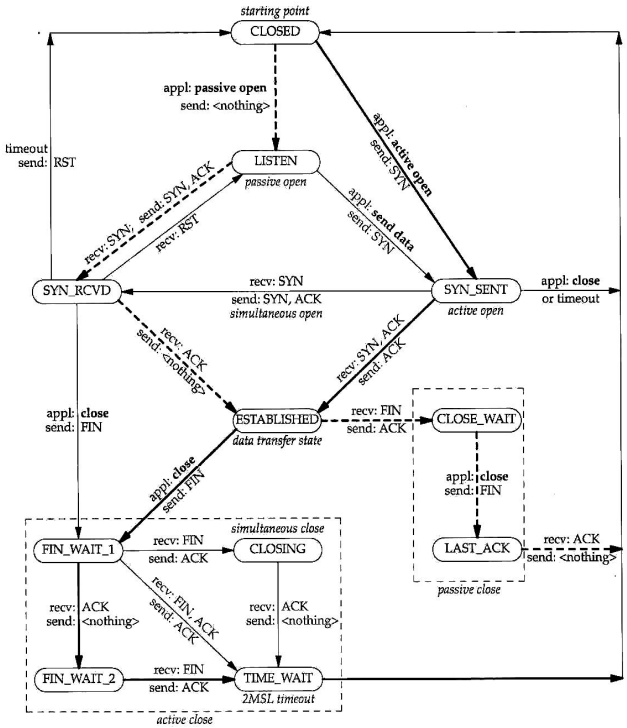
Even if the above two conditions are met, there is still the possibility of duplicate sequence numbers, when a host has a very slow transmission rate and connections last a very long time.

Acknowledgements and Retransmissions

- When host A sends a segment, it starts a timer and waits for host B to acknowledge the reception of the segment.
- If the acknowledgment is not received before a timeout, then host A retransmits the segment.
- When a host B receives a segment containing data, it sends an acknowledgment.



TCP State Transition Diagram



The Legend:

- normal transition for Client
- normal transition for Server
- appl: state transition when application issues operation
- recv: state transition taken when segment received
- send: what is sent for this transaction

TIME_WAIT State

The endpoint that initiates termination of connection, goes through TIME_WAIT state and remains there for the period:

2 * MSL (maximum segment lifetime)

MSL is configurable life time of IP packet in the network. The actual duration of 2MSL varies from 30 sec up to 4 min.

The TIME_WAIT state has two reasons:

1. Reliable termination of TCP connection (ability to reply ACK for resent FIN)
2. To allow old duplicate segments to expire in the network
   (1MSL for original duplicate segments
   + 1 MSL for replies to be lost from network)

For the period of 2MSL the TCP prevents the *incarnation* (the restart from the same port) of the connection.

Flow Control and Congestion Control

In a network, it is often desirable to limit the rate at which a source can send traffic into the subnet.

If this is not done and sources send at too high of a rate, then buffers within the network will fill-up resulting in long delays and eventually packets being dropped.

Moreover as packets gets dropped, retransmission may occur, leading to even more traffic.

When sources are not regulated this can lead to congestion collapse of the network, where very little traffic is delivered to the destination.

Flow Control and Congestion Control

Two different factors can limit the rate at which a source sends data.

→ the inability of the destination to accept new data.
Techniques that address this are referred to as
flow control.

→ the number of packets within the subnet.
Techniques that address this are referred to as
congestion control.

Flow Control and Congestion Control

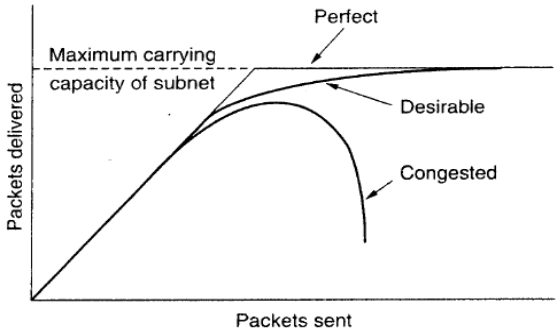
Flow control and congestion control can be addressed at the transport layer, but may also be addressed at other layers.

For example, some DLL protocols perform flow control on each link.
And some congestion control approaches
are done at the network layer.

Both flow control and congestion control are part of TCP.

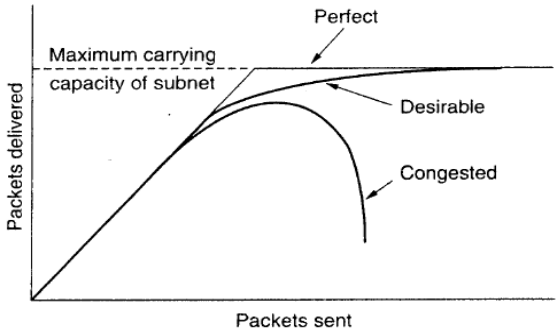
Congestion

Congestion arises when the total load on the network becomes too large. This leads to queues building up and to long delays (recall the M/G/1 model). If sources retransmit messages, then this can lead to even more congestion and eventually to *congestion collapse*. This is illustrated in the figure below. Notice, as the offered load increase, the number of packets delivered at first increases, but at high enough loads, this rapidly decreases.



Congestion

Notice, as the offered load increase, the number of packets delivered at first increases, but at high enough loads, this rapidly decreases.



For example, originally the Internet (TCP/IP) did not implement congestion control.

This led to a series of congestion collapses starting in 1986.

[V. Jacobson, "Congestion Avoidance and Control", *Proc. of SIGCOMM '88.*]

To avoid this type of behavior, congestion control was incorporated into TCP.

The first order goal of congestion control is to avoid this type of congestion collapse.

It is still desirable to get the best performance possible. Congestion control can also be used to ensure that users get a desired Quality of Service (QoS) (i.e. throughput, delay).

Also ensuring *fairness* between different sessions; in other words two sessions with the same requirements should receive equitable treatment from the network. (Exactly what is a fair treatment can be interpreted in several ways.)

Approaches to Congestion Control

Congestion control may be addressed at both the network level and the transport layer.

At the *network layer* possible approaches include

Packet dropping → when a buffer becomes full a router can drop waiting packets – if not coupled with some other technique, this can lead to greater congestion through retransmissions.

Packet scheduling → certain scheduling policies may help in avoiding congestion – in particular scheduling can help to isolate users that are transmitting at a high rate.

Approaches to Congestion Control

Dynamic routing → when a link becomes congested, change the routing to avoid this link – this only helps up to a point (eventually all links become congested) and can lead to instabilities

Admission control/Traffic policing – Only allow connections in if the network can handle them and make sure that admitted sessions do not send at too high of a rate – only useful for connection-oriented networks.

Approaches to Congestion Control

An approach that can be used at either the network or transport layers is

Rate control → this refers to techniques where the source rate is explicitly controlled based on feedback from either the network and/or the receiver.

For example, routers in the network may send a source a "choke packet" upon becoming congested. When receiving such a packet, the source should
lower it rate.

Approaches to Congestion Control

These approaches can be classified as either "congestion avoidance" approaches, if they try to prevent congestion from ever occurring, or as "congestion recovery" approaches, if they wait until congestion occurs and then react to it. In general, "better to prevent than to recover."

Different networks have used various combinations of all these approaches.

Traditionally, rate control at the transport layer has been used in the Internet, but new approaches are beginning to be used that incorporate some of the network layer techniques discussed above.

Congestion Control in TCP

TCP implements end-to-end congestion control.

IP does not provide any explicit congestion control information; TCP detects congestion via the ACK's from the sliding-window ARQ algorithm used for providing reliable service.

Specifically, assume that the RTT for a session is known, so that the proper time-out time can be used.

When the source times out before receiving an ACK, the most likely reason is because a link became congested. TCP uses this as an indication of congestion. In this case, TCP will slow down the transmission rate.

Congestion Control

- Packet loss is most likely to be caused by congestion
  - Cannot inject additional copies of a message using retransmission
    - May reach a state of congestion collapse
- TCP uses packet loss as a measure of congestion
  - If a message is lost, TCP begins congestion control
  - Rather than sending enough data to fill the receiver's buffer
    - Send a single message
    - If ack. is received without loss, TCP doubles the amount of data
    - If ack. is received without loss, TCP sends four more messages
    - The exponential increase continues until TCP sends half of the advertised window at which time TCP slows down the rate of increase

Congestion Control in TCP

TCP controls the transmission rate of a source by varying the window size used in the sliding window protocol (this is the same technique used for flow control).

Recall, that for a window of size of W packets, the effective rate under a sliding window protocol (ignoring errors) is given by

$$R_{eff} = \frac{Wd}{d + IR} \text{ (assuming that } W < 1 + (IR)/d)$$

Here I is the round-trip time, and d is the number of bits per packet. Thus a smaller value of W will result in a smaller effective rate.

Timeout time

Should be set approximately equal to the round trip time, i.e. the time from when a segment is transmitted until an ACK for the segment is received.

Choosing to small of a time-out time will result in unneeded retransmissions, and choosing too large a value will result in long delays before a packet is retransmitted.

However, at the transport layer, the round trip time will vary with each pair of host (or more precisely with each route between any pair of hosts).

The round-trip time will also depend on the queuing delays within the network; these will vary with each segment sent.

TCP Congestion control

TCP tries to adaptively find the "right" congestion window size for the network. If window is too large, network will become congested; if too small then inefficient. More over, the "right" window size may change over time, depending on network load.

When segments are successfully acknowledged, without the sender timing out, TCP increases the window size. Whenever TCP times-out while waiting for a segment it decreases the window size. In this manner TCP continually adapts the window size.

In addition to the congestion window, the sender also keeps track of a parameter called the *threshold*;

TCP Congestion control

We see that starting at WC = 1, the congestion window will increase by one for every ACK that is received until it becomes larger than the threshold, this phase is called *slow start*.

After the congestion window becomes larger than the threshold, the congestion window is increased by one for every congestion window worth of ACKs that are received; this phase is called the *congestion avoidance* phase.

TCP Congestion control

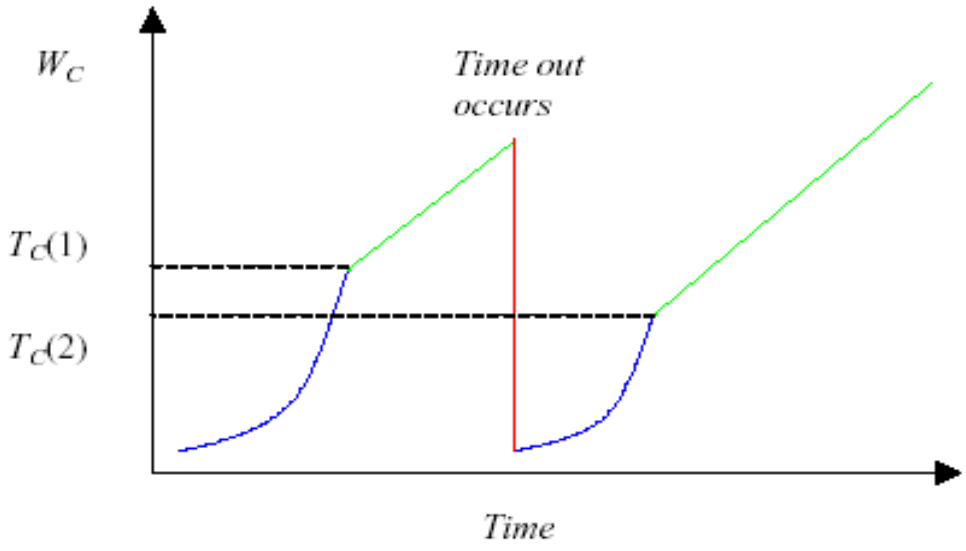
Assume that in approximately one RTT, a congestion window worth of data can be sent and acknowledged (this is reasonable if the time to send a windows worth of data is much less than the RTT).

With this assumption:

→ During the slow start phase, the congestion window will double during each RTT, i.e. it will grow exponentially fast.

→ During the congestion avoidance phase, the window will increase by 1 during each RTT; in this case, it is growing at a linear rate.

An example of this algorithm is shown below. The slow start phase is shown in blue and the congestion avoidance phase is shown in green. In this figure one time out occurred as indicated. Here TC(1) indicates the initial threshold and TC(2) indicates the threshold after the time out occurs.



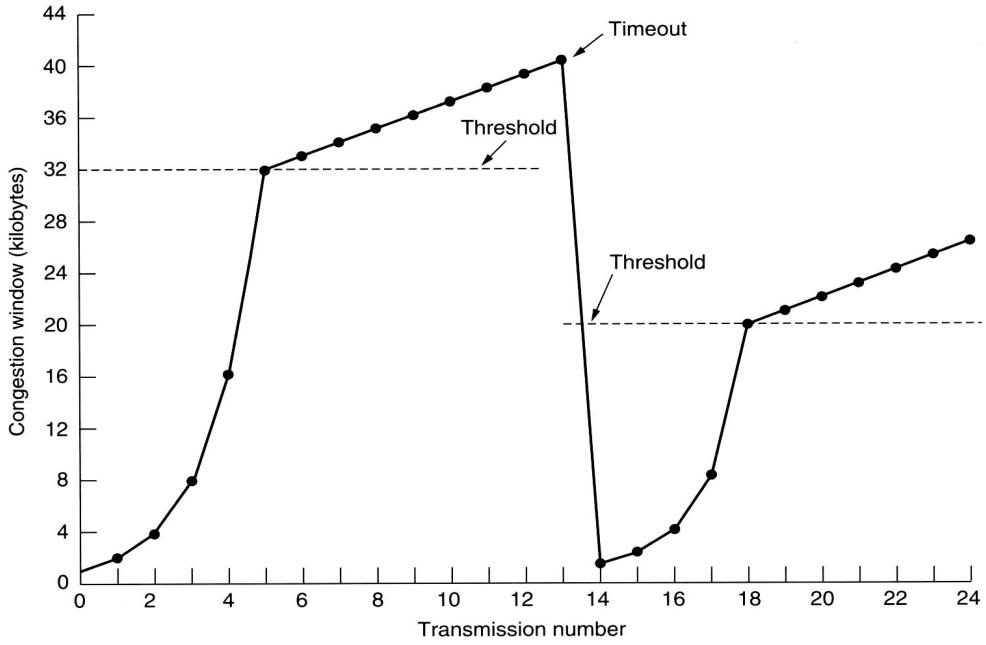


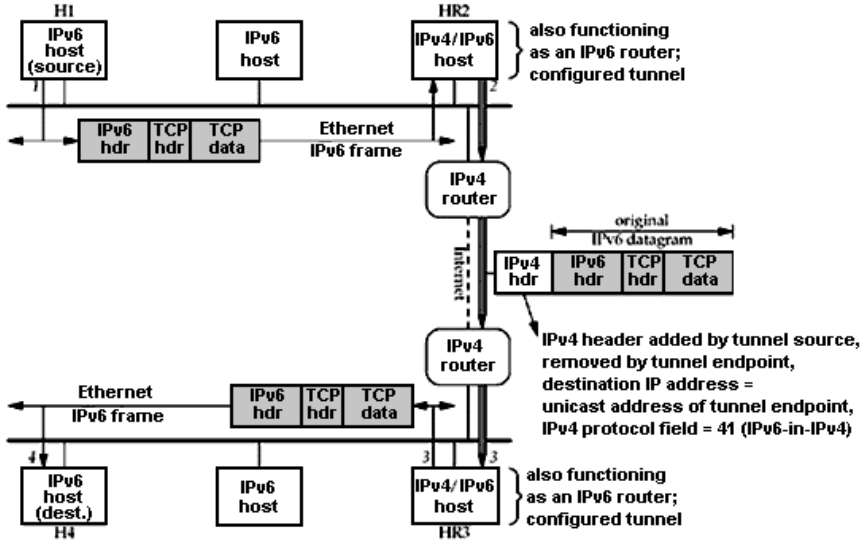
Fig. 6-32. An example of the Internet congestion algorithm.

Modern protocols in TCP/IP Protocol Suite

- IPv6 - Internet Protocol version 6.
  - Network Layer protocol, designed in the mid-1990s as a replacement for IPv4. The major change is a larger address comprising 128 bits to deal with the explosive growth of the Internet in the 1990s.
  - IPv6 addresses are 128 bits long and are usually written as eight 16-bit hexadecimal numbers.
  - IPv6 provides packet delivery service for TCP, UDP, SCTP, and ICMPv6.
- ICMPv6 - Internet Control Message Protocol version 6.
  - Is integral part of IPv6 implementation.
  - ICMPv6 combines the functionality of ICMPv4, IGMP, and ARP.
- SCTP - Stream Control Transmission Protocol.
  - Transport Layer protocol, providing *Reliable Connection-Oriented Full-Duplex Message Service*.
  - Designed in 2000-2002.
  - SCTP connection called *Association*, because SCTP is *Multihomed* and involves a set of IP addresses and a single port for each side of an Association.
  - SCTP provides a *Message Service*, which maintains record boundaries. As with TCP and UDP, SCTP can use either IPv4 or IPv6, but it can also use both IPv4 and IPv6 simultaneously on the same association.
  - SCTP can provide multiple streams between connection endpoints, each with its own reliable sequenced delivery of messages. A lost message in one of these streams does not block delivery of messages in any of the other streams.

Virtual Network and Tunneling: The 6bone.

The 6bone is a *virtual network* that was created in 1996 for users with islands of IPv6-capable hosts wanted to connect them together using a virtual network without waiting for all the intermediate routers to become IPv6-capable. The 6bone is established on top of the existing IPv4 Internet using *tunnels*.



TCP/IP Summary

- IPv4
  - Internet Protocol version 4.
- IPv6
  - Internet Protocol version 6.
- TCP
  - Transmission Control Protocol.
- UDP
  - User Datagram Protocol.
- SCTP
  - Stream Control Transmission Protocol.
- ICMP
  - Internet Control Message Protocol
  - (version 4).
- IGMP
  - Internet Group Management Protocol.
- ARP
  - Address Resolution Protocol.
- RARP
  - Reverse Address Resolution Protocol.
- ICMPv6
  - Internet Control Message Protocol version 6.
- BPF
  - BSD Packet Filter. Provides access to the datalink layer in Berkeley-derived kernels.
- DLPI
  - Datalink Provider Interface. Provides access to the datalink layer in System V R4.

Internet

- Set of interconnected independent computer networks.
- Uses common suite of protocols called TCP/IP.
- Managed by groups of representatives:
  - **IAB** -Internet Activity Board
  - **NIC** -National Information Center
  - **FNC** -Federal Network Center

Internet Services

Transport Level:

- Unreliable packet delivery
- Reliable stream transport

Application Level:

- File Transfer (FTP, TFTP)
- Electronic Mail (SMTP)
- Remote Login (TELNET)
- Network File System (NFS)
- Remote Program Execution
- Shared peripheral devices