**Department of Computer Science and Engineering**

Subject Name: **COMPUTER NETWORKS** Subject Code: **CS T52**

**UNIT-IV**

Transport layer - Services - Berkeley Sockets -Example – Elements of Transport protocols

– Addressing - Connection Establishment - Connection Release - Flow Control and Buffering

– Multiplexing – Congestion Control - Bandwidth Allocation - Regulating the Sending Rate – UDP- RPC – TCP - TCP Segment Header - Connection Establishment - Connection Release - Transmission Policy - TCP Timer Management - TCP Congestion Control

OMPUTER NETWORKS Page 1

**2 MARKS**

**1.What is transport layer?**

The Transport layer is the fourth layer of the OSI reference model. In computer networking, a **transport layer** provides end-to-end or host-to-host communication services for applications within a layered architecture of network components and protocols. The **transport layer** provides services such as connection-oriented data stream support,reliability, flow control, and multiplexing.

**2.Write the relationship between transport and network layer?(April/May 2013) Transport layer:**

* Logical communication between processes.
* Responsible for checking that data available in session layer are error free.
* Protocols used at this layer are :

o TCP(Transmission Control Protocol)

o UDP(User Datagram Protocol )

o SCTP(Stream Control Transmission Protocol)

**Network layer:**

* Logical communication between hosts.
* Responsible for logical addressing and translating logical addresses (ex. amazon.com) into physical addresses (ex. 180.215.206.136)
* Protocols used at this layer are :
  + IP(Internet Protocol)
  + ICMP(Internet Control Message Protocol)
  + IGMP(Internet Group Message Protocol)
  + RARP(Reverse Address Resolution Protocol)
  + ARP(Address Resolution Protocol)

**3.Define Berkeley sockets?**

**Berkeley sockets** (or BSD **sockets**) is a computing library with an applicationprogramming interface (API) for internet **sockets** and Unix domain **sockets**, used for inter-process communication (IPC).The API has evolved with little modification from a [de facto](http://en.wikipedia.org/wiki/De_facto_standard) [standard](http://en.wikipedia.org/wiki/De_facto_standard) into part of the [POSIX](http://en.wikipedia.org/wiki/POSIX) specification. **POSIX sockets** are basically Berkeley sockets.

**4.What are the Elements Of Transport protocols?**

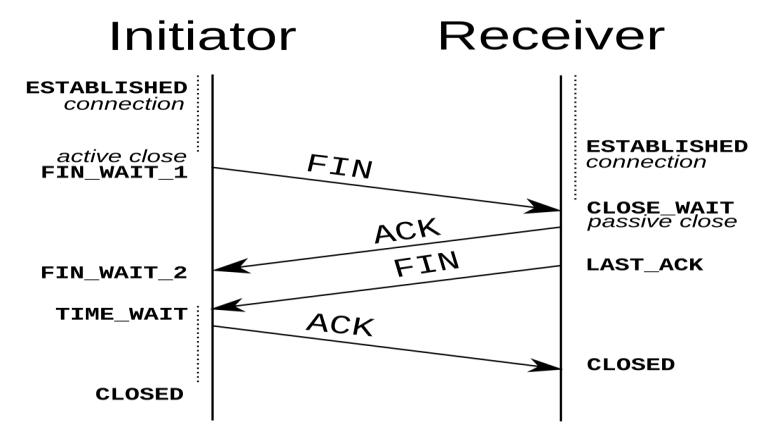
* Addressing
* Connection Establishment
* Connection Release

OMPUTER NETWORKS Page 2

* Flow Control and Buffering
* Multiplexing

**5.What is Connection Establishment?**

To **establish** a **connection**, the three-way (or 3-step) handshake occurs: SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A. SYN-ACK: In response, the server replies with a SYN-ACK.



**6.What is Connection Release?**

Dropping connections isn't trivial either, as it turns out. An asymmetric release means that either host can destroy the connection (like hanging up the telephone). But data can be lost since there is no coordination between parties, so data that is "in the pipe" is lost once the connection is destroyed. Symmetric release requires both parties to agree to a release. If both parties know they are done sending data, and agree, then no data is lost.

**7.What is Two army problem?**

Blue army #1 sends message: attack at time X. Blue army #2 receives this message and sends an acknowledgment to it. Does the attack happen at time X? No, since blue army #2 can't know that it's ack was received. Adding an ack to the ack (three-way handshake) doesn't help, since now blue army #1 doesn't know if his ack to the ack got through, and if it didn't blue army #2 won't attack, so blue army #1 shouldn't attack either.

**8.What is Transmission delay?(Nov/Dec 2012)**

In a [network](http://en.wikipedia.org/wiki/Computer_network) based on [packet switching,](http://en.wikipedia.org/wiki/Packet_switching) **transmission** **delay** (or **store-and-forward** **delay**, also known as **packetization delay**) is the amount of time required to push all ofthe packet's bits into the wire. Transmission delay is a function of the packet's length and has nothing to do with the distance between the two nodes. This delay is proportional to the packet's length in bits. It is given by the following formula:

seconds

Software quality assurance and testing Page 3

where

is the transmission delay in seconds N is the number of bits, and

R is the rate of transmission (say in bits per second)

**9.What is round-trip time?(Nov/Dec 2012)**

In telecommunications, the **round**-**trip** delay **time** (RTD) or **round**-**trip time** (**RTT**) is the length of **time** it takes for a signal to be sent plus the length of **time** it takes for an acknowledgment of that signal to be received.

**10.Define Flow control?**

* **Flow control** is the management of data **flow** between computers or devices orbetween nodes in a network so that the data can be handled at an efficient pace.
* Too much data arriving before a device can handle it causes data overflow, meaning the data is either lost or must be retransmitted.
* The ARQ protocol also provides **flow control**, which may be combined with congestion avoidance.

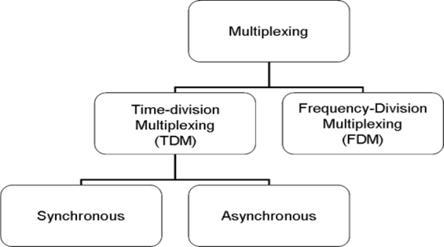
**11.What is buffering?**

* The sender's transport layer must worry about overwhelming both the network and the receiver. The network may exceed the carrying capacity, and the receiver may run out of buffers.
* Buffers are statically allocated kernel memory so that storing received TPDUs can be done quickly.
* Buffering isn't the only thing that limits the flow control in the transport layer.

**12.What is Multiplexing?**

In telecommunications and **computer networks**, **multiplexing** (sometimes contracted to muxing) is a method by which multiple analog message signals or digital data streams are combined into one signal over a shared medium. The aim is to share an expensive resource.

**13.What are the different types of Multiplexing?(Nov/Dec 2014)**

****

Software quality assurance and testing Page 4

There are two basic forms of multiplexing used:

* Time division multiplexing (TDM)
* Frequency division multiplexing (FDM)

**14.What is meant by demultiplexing?(April/May 2014)**

To separate two or more channels previously multiplexed. Demultiplexing is the reverse of [multiplexing.](http://www.webopedia.com/TERM/M/multiplexing.html) Demultiplex (DEMUX) is the reverse of the multiplex (MUX) process – combining multiple unrelated analog or digital signal streams into one signal over a single shared medium, such as a single conductor of copper wire or fiber optic cable.

**15.Define congestion Control?**

**Congestion Control**. When one part of the subnet (e.g. one or more routers in an area)becomes overloaded, **congestion** results. Because routers are receiving packets faster than they can forward them, one of two things must happen.

**16.What is Bandwidth allocation?**

**Bandwidth allocation** is the process of assigning radio frequencies to differentapplications. The radio spectrum is a finite resource creating the need for an effective **allocation** process.

**17.What is Dynamic bandwidth allocation?**

**Dynamic bandwidth allocation** is a technique by which traffic bandwidth in a sharedtelecommunications medium can be allocated on demand and fairly between different users of that bandwidth. Where the sharing of a link adapts in some way to the instantaneous traffic demands of the nodes connected to the link.

**18.Define UDP?(Nov/Dec 2011)(Nov/Dec 2013)**

**UDP** (**User Datagram Protocol**) is a communications protocol that offers a limited amountof service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). **UDP** is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as **UDP/IP.**

**19.What is the drawback of UDP?(April/May 2012)**

* There are no guarantees with udp. a packet may not be delivered, or delivered twice, or delivered out of order; you get no indication of this unless the listening program at the other end decides to say something. UDP has no flow control. implementation is the duty of user programs.

Software quality assurance and testing Page 5

 Routers are quite careless with udp. they never retransmit it if it collides, and it seems to be the first thing dropped when a router is short on memory. udp suffers from worse packet loss than tcp.

**20.What is Datagram? (Nov/Dec 2011)**

A **datagram** is a basic transfer unit associated with a packet-switched network. The delivery, arrival time, and order of arrival need not be guaranteed by the network.

**21.Define Remote Procedure call?**

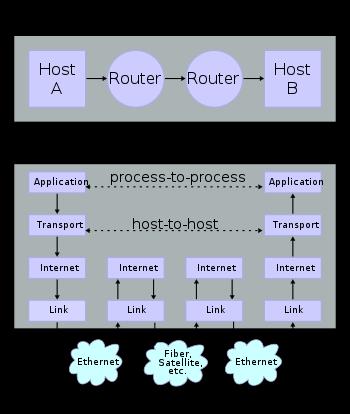
In computer science, a **remote procedure call** (**RPC**) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared **network**) without the programmer explicitly coding the details for this remote interaction.

**22.Define TCP?**

* The Transmission Control Protocol (TCP) is a core [protocol](http://en.wikipedia.org/wiki/Communications_protocol) of the [Internet Protocol](http://en.wikipedia.org/wiki/Internet_Protocol_Suite) [Suite.](http://en.wikipedia.org/wiki/Internet_Protocol_Suite)
* TCP provides [reliable,](http://en.wikipedia.org/wiki/Reliability_%28computer_networking%29) ordered, and [error-checked](http://en.wikipedia.org/wiki/Error_detection_and_correction) delivery of a stream of [octets](http://en.wikipedia.org/wiki/Octet_%28computing%29) between applications running on hosts communicating over an IP network.
* TCP is the protocol that major Internet applications such as the [World Wide Web,](http://en.wikipedia.org/wiki/World_Wide_Web) [email,](http://en.wikipedia.org/wiki/Email) [remote administration](http://en.wikipedia.org/wiki/Remote_administration) and [file transfer](http://en.wikipedia.org/wiki/File_transfer) rely on.

**23.Define TCP/IP? (Nov/Dec 2011)**

It is commonly known as **TCP**/**IP**, because its most important protocols, the Transmission Control Protocol (**TCP**) and the Internet Protocol (**IP**), were the first networking protocols defined in this standard.



**24.List out the services provided by TCP?(Nov/Dec 2012)**

Software quality assurance and testing Page 6

TCP/IP services are divided into two groups: services provided to other protocols and services provided to end users directly.

**Services Provided to Other Protocols**

These services are designed to actually accomplish the internetworking functions of the protocol suite. For example, at the network layer, [IP provides functions](http://www.tcpipguide.com/free/t_IPFunctions.htm) such as addressing, delivery, and datagram packaging, fragmentation and reassembly.

**End-User Services**

* WWW services are provided through the [Hypertext Transfer Protocol (HTTP),](http://www.tcpipguide.com/free/t_TCPIPHypertextTransferProtocolHTTP.htm) a TCP/IP application layer protocol. HTTP in turn uses services provided by lower-level protocols. All of these details are of course hidden from the end users, which is entirely on purpose.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **25.Discuss the TCP connections needed in FTP?(Nov/Dec 2014)** | | | |  |  |
| TCP |  | connections | needed | in | FTP. |
| FTP | establishes | two connections | between the hosts. One | connection | is used |

for data transfer, the other for control information. The control connection uses very simple rules of communication. The data connection needs more complex rules due to the variety of data types transferred.

**26.What is the use of option field in TCP?(April/May 2012)**

* **Options :** Provides a way to add extra facilities not covered by the regular header.eg,

1. Maximum TCP payload that sender is willing to handle. The maximum size of segment is called MSS (Maximum Segment Size).
2. Window scale option can be used to increase the window size. It can be specified by telling the receiver that the window size should be interpreted by shifting it left by specified number of bits. This header option allows window size up to 230.

**27.What do you mean by receive window?(April/May 2014)**

* In [computer networking,](http://en.wikipedia.org/wiki/Computer_networking) **RWIN** [(TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) Receive Window) is the amount of [data](http://en.wikipedia.org/wiki/Data_%28computing%29) that a [computer](http://en.wikipedia.org/wiki/Computer) can accept without acknowledging the sender. If the sender has not received acknowledgement for the first [packet](http://en.wikipedia.org/wiki/Packet_%28information_technology%29) it sent, it will stop and wait and if this wait exceeds a certain limit, it may even [retransmit.](http://en.wikipedia.org/wiki/Retransmission_%28data_networks%29)
* The limitation caused by window size can be calculated as follows:



Software quality assurance and testing Page 7

* where RWIN is the TCP Receive Window and RTT is the [round-trip time](http://en.wikipedia.org/wiki/Round-trip_delay_time) for the path.

**28.Why an Application developer would ever choose to build an application over UDP rather than over TCP?(Nov/Dec 2012)**

* UDP is a transport protocol that does not check for errors. One would use it when speed of the transport is desired, not quality or reliability.
* If you were streaming video or using voice, then getting the packets to the destination quickly is more important than making sure each sound gets to the receiver.

**11 MARKS**

The transport layer is not just another layer. It is the heart of the whole protocol hierarchy. Its task is to provide reliable, cost-effective data transport from the source machine to the destination machine, independently of the physical network or networks currently in use. Without the transport layer, the whole concept of layered protocols would make little sense. We will study the transport layer in detail, including its services, design, protocols, and performance.

**1..Briefly explain transport layer services? (Nov/Dec 2011) (April/May 2012) (Nov/Dec 2013)**

* **Services Provided to the Upper Layers**
* **Transport Service Primitives**
* **Berkeley Sockets**
* **An Example of Socket Programming:**

– **An Internet File Server**

**Services Provided to the Upper Layers**

* Efficient, reliable and cost-effective service to users (application layer) o despite limitations of network layer
* Features (a lot like the Network layer?)

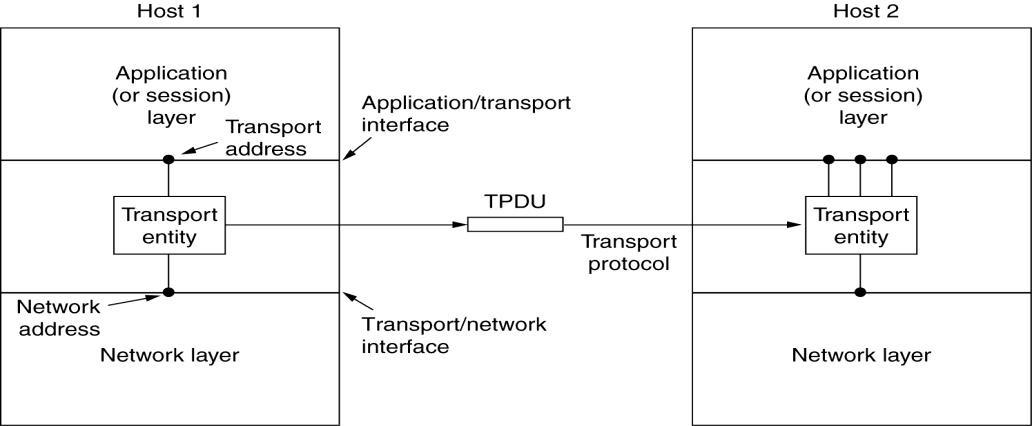
o Connection oriented vs. Connectionless

o Addressing and Flow Control

Software quality assurance and testing Page 8

* But Transport layer can make lower subnet reliable (QoS), and gives standard interface
* Major boundary between user and network!

o Few users write code for network layer o Many write code for transport layer

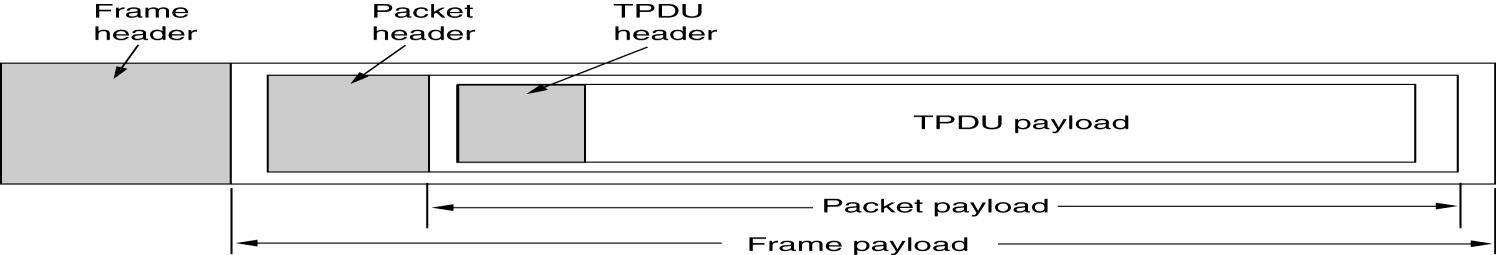


The network, transport, and application layers.

* Logical location of transport entity
* Physical: OS, separate process, network card

**TPDU**

* **TPDU (Transport Protocol Data Unit)** is a term used for messages sent fromtransport entity to transport entity.
* 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.



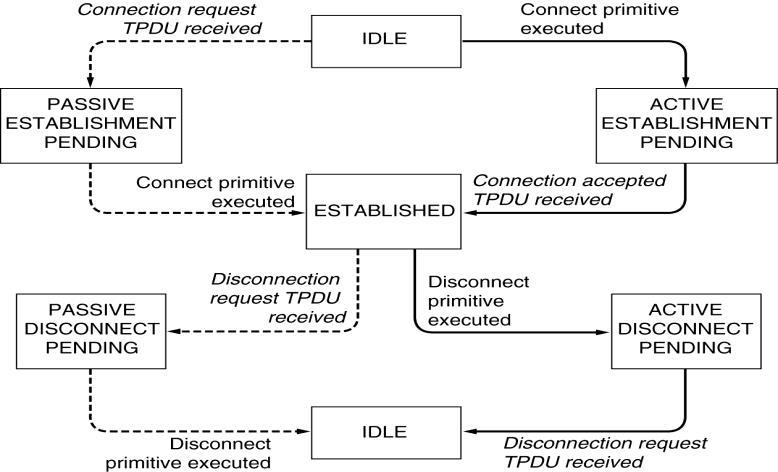
Software quality assurance and testing Page 9

The nesting of TPDUs, packets, and frames.

Disconnection has two variants: asymmetric and symmetric.

* In the asymmetric variant, either transport user can issue a DISCONNECT primitive, which results in a DISCONNECT segment being sent to the remote transport entity. Upon its arrival, the connection is released.
* In the symmetric variant, each direction is closed separately, independently of the other one. When one side does a DISCONNECT, that means it has no more data to send but it is still willing to accept data from its partner. In this model, a connection is released when both sides have done a DISCONNECT.

State Diagrams



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

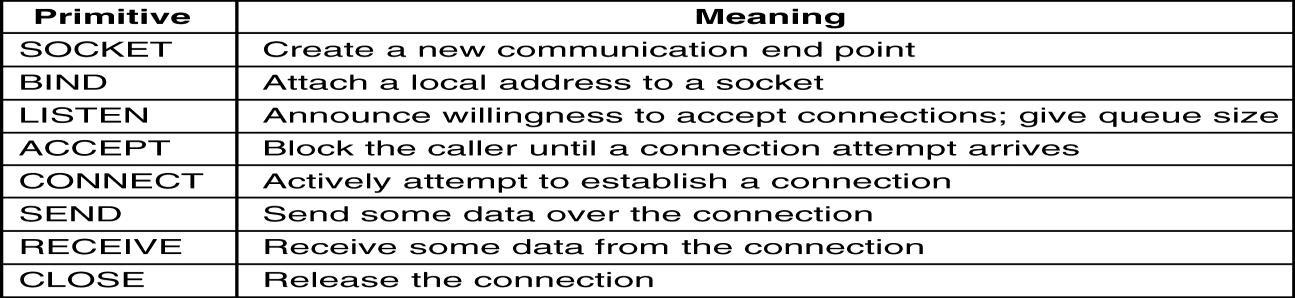
**2.Explain briefly about Berkeley Sockets.**

This methods provided by the Berkeley sockets API library:

* socket() creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.
* bind() is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.
* listen() is used on the server side, and causes a bound TCP socket to enter listening state.
* connect() is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.

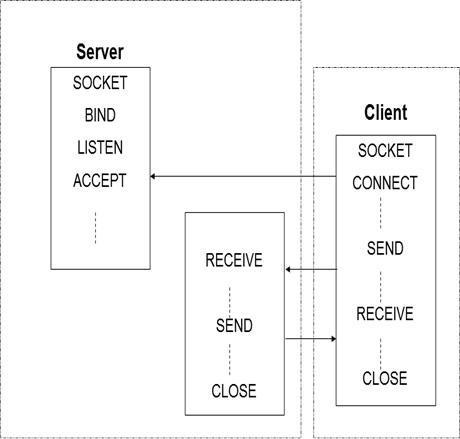
Software quality assurance and testing Page 10

* accept() is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
* send() and recv(), or write() and read(), or sendto() and recvfrom(), are used for sending and receiving data to/from a remote socket.
* close() causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.



The socket primitives for TCP.

The socket API is often used with the TCP protocol to provide a connection-oriented service called a **reliable byte stream**. Two examples are **SCTP** (**Stream Control** **Transmission Protocol**) defined in RFC 4960 and **SST** (**Structured Stream Transport**)(Ford, 2007).



The working principal of Berkeley socket.

**3.Discuss about socket programming with TCP ? (Nov/Dec 2011) SocketProgrammingExample:**

**Internet File Server**

**Socket Programming with TCP**

Software quality assurance and testing Page 11

* Connection oriented

– Handshaking procedure

* Reliable byte-stream

TCP-client in Java import java.io\*; import java.net.\*; Class TCPClient {

public static void main (String argv[]) throws Exception { String sentence;

String modifiedSentence;

BufferedReader inFromUser = new BufferedReader( new InputStreamReader(system.in));

Socket clientSocket = new Socket(”hostname”, 6789);

DataOutpuStream outToServer = new DataOutputStream( clientSocket.getOutputStream());

BufferedReader inFromServer =

new BufferedReader(new InputStreamReader( clientSocket.getInputStream()));

sentence = inFromUser.readLine(); outToServer.writeBytes(sentence + ’\n’); modifiedSentence = inFromServer.readLine();

System.out.println(”FROM SERVER: ” + modifiedSentence); clientSocket.close(); } }

import java.io\*; import java.net.\*;

* Imports needed packages Class TCPClient {

public static void main (String argv[]) throws Exception {

* Standard Java initiation

String sentence;

String modifiedSentence;

* Declares two string objects

BufferedReader inFromUser = new BufferedReader( new InputStreamReader(system.in))

* Creates a stream that handels input from the user

Socket clientSocket = new Socket(”hostname”, 6789) ;

* Initiate a TCP-connection with the ”hostname” through port 6789
* Client performes a DNS lookup to obtain host IP. DataOutpuStream outToServer = new DataOutputStream(

clientSocket.getOutputStream())

Software quality assurance and testing Page 12

• Creates a stream that handels output to server BufferedReader inFromServer =

new BufferedReader(new InputStreamReader( clientSocket.getInputStream()));

* + Creates a stream that handels input from server sentence = inFromUser.readLine()
  + Puts the input from user into string object outToServer.writeBytes(sentence + ’\n’);
  + Transform sentence to bytes & sends to server modifiedSentence = inFromServer.readLine();
  + Puts input from server into modified sentence

System.out.println(”FROM SERVER: ” + modifiedSentence); clientSocket.close(); } }

* + Prints modifiedSentence and closes the connection

1. **Explain in detail about Elements of Transport Protocols. Elements of Transport Protocols**

* The transport service is implemented by a **transport protocol** used between the two transport entities
* Transport protocols resemble the data link protocols
* Both have to deal with error control, sequencing, and flow control, among other issues.
* Differences are due to major dissimilarities between the environments in which the two protocols operate
  + **Addressing**
  + **Connection Establishment**
  + **Connection Release**
  + **Flow Control and Buffering**
  + **Multiplexing**
  + **Crash Recovery**

**Addressing**

Software quality assurance and testing Page 13

* When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to.
* The method normally used is to define transport addresses to which processes can listen for connection requests. In the Internet, these end points are called **ports**.
* We will use the generic term **TSAP**, (Transport Service Access Point).
* The analogous end points in the network layer (i.e., network layer addresses) are then called **NSAPs**.
* **IP** addresses are examples of **NSAPs**.

**Establishing a connection**

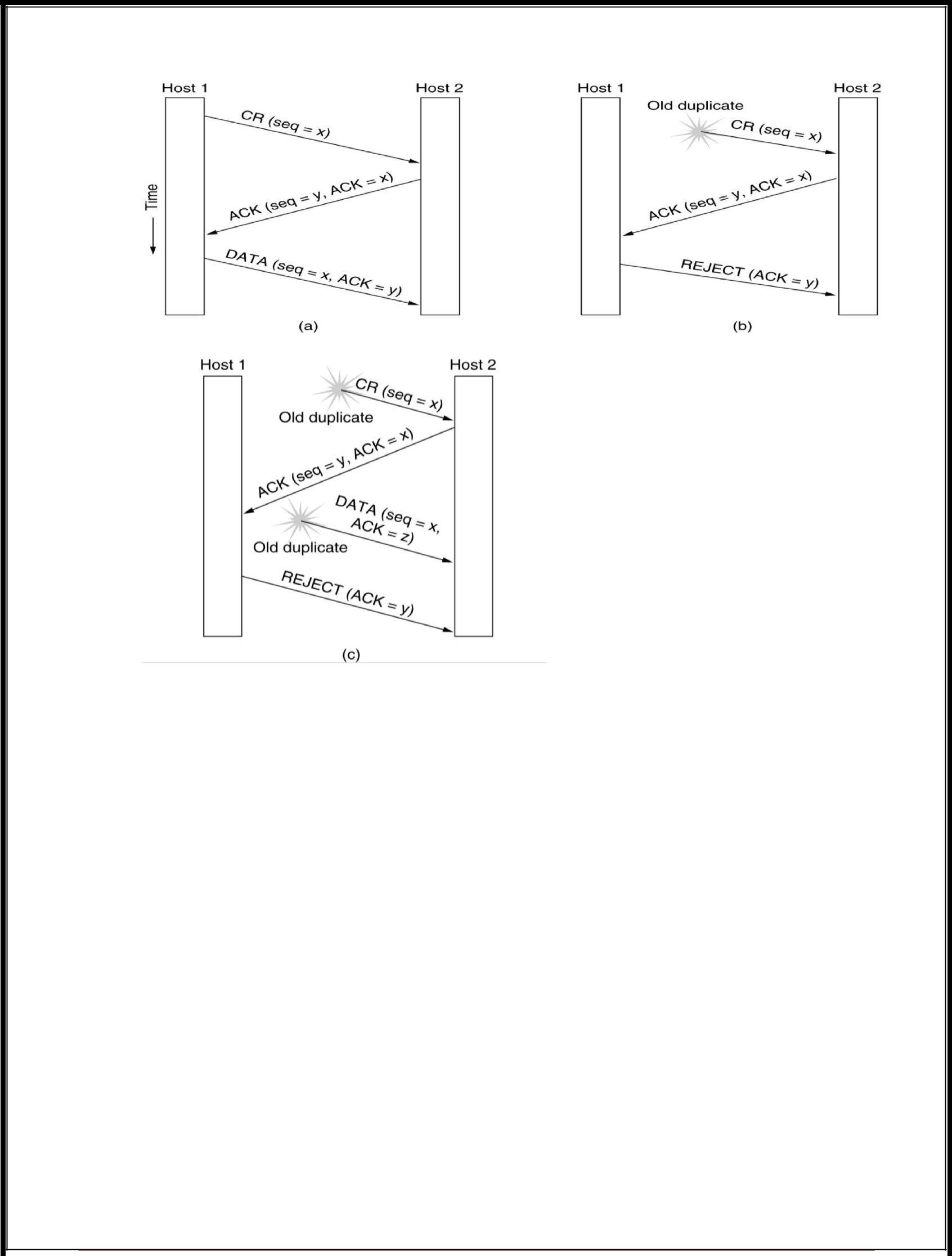
* It would seem sufficient for one transport entity to just send a CONNECTION REQUEST TPDU to the destination and wait for a CONNECTION ACCEPTED reply.
* The problem occurs when the network can lose, store, and duplicate packets. This causes serious complications.
* The crux of the problem is the existence of delayed duplicates.
* It can be attacked in various ways
* Using throw-away transport addresses. In this approach, each time a transport address is needed, a new one is generated. When a connection is released, the address is discarded and never used again.
* Give each connection a connection identifier. After each connection is released, each transport entity could update a table listing obsolete connections. Whenever a connection request comes in, it could be checked against the table, to see if it belonged to a previously-released connection.
* This scheme has a basic flaw: it requires each transport entity to maintain a certain amount of history information indefinitely. If a machine crashes and loses its memory, it will no longer know which connection identifiers have already been used.
* Rather than allowing packets to live forever within the subnet, devise a mechanism to kill off aged packets that are still hobbling about. If no packet lives longer than some known time, the problem becomes somewhat more manageable.

Software quality assurance and testing Page 14

* Packet lifetime can be restricted to a known maximum using one of the following techniques:
  + Restricted subnet design.
  + Putting a hop counter in each packet.
  + Timestamping each packet.
* Once both transport entities have agreed on the initial sequence number, any sliding window protocol can be used for data flow control.
* Three-way handshake protocol is used
* This protocol does not require both sides to begin sending with the same sequence number.
* Host 1 chooses a sequence number, x, and sends a CONNECTION REQUEST TPDU containing it to host 2.
* Host 2 replies with an ACK TPDU acknowledging x and announcing its own initial sequence number, y.
* Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data TPDU that it sends.

**Connection Establishment**

Software quality assurance and testing Page 15



Three protocol scenarios for establishing a connection using a three-way

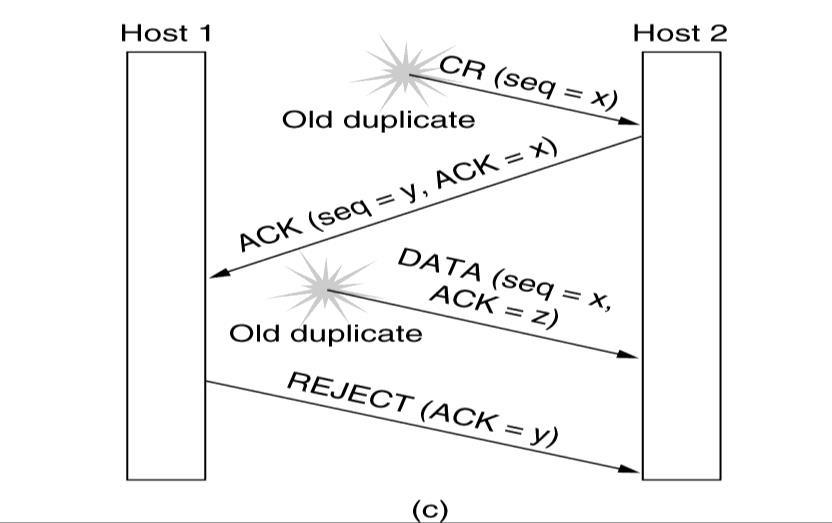
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| handshake. | | CR | denotes | CONNECTION | REQUEST. |
| (a) |  |  | Normal |  | operation, |
| (b) | Old | CONNECTION | REQUEST | appearing out of | nowhere. |

(c) Duplicate CONNECTION REQUEST and duplicate ACK.

* Fig b: shows how the three-way handshake works in the presence of delayed duplicate control TPDUs .
* The first TPDU is a delayed duplicate CONNECTION REQUEST from an old connection.
* This TPDU arrives at host 2 without host 1's knowledge. Host 2 reacts to this TPDU by sending host 1 an ACK TPDU, in effect asking for verification that host 1 was indeed trying to set up a new connection.
* When Host 1 Rejects Host 2 ack, Then Host 2 realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage

Software quality assurance and testing Page 16

* The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet. This case is shown in fig c
* Host 2 gets a delayed CONNECTION REQUEST and replies to it.
* Host 2 has proposed using y as the initial sequence number for host 2 to host 1 traffic, knowing full well that no TPDUs containing sequence number y or acknowledgements to y are still in existence.
* When the second delayed TPDU arrives at host 2, the fact that z has been acknowledged rather than y tells host 2 that this, too, is an old duplicate.

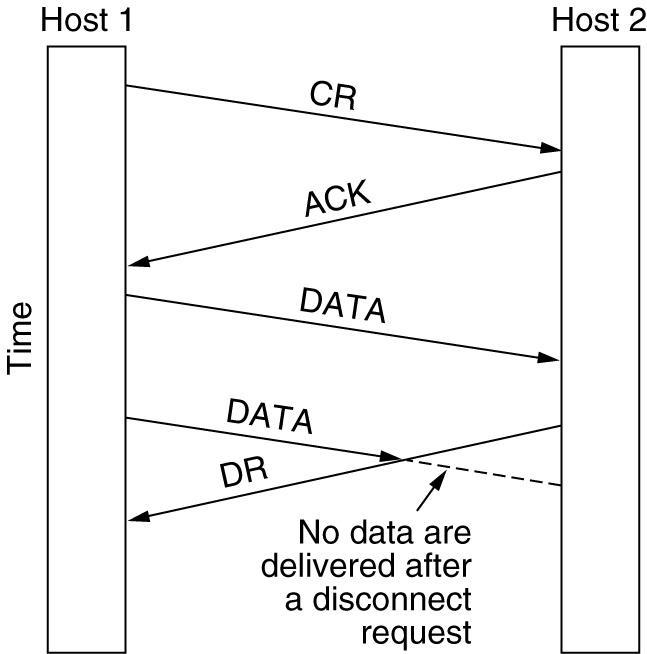


**Releasing a connection**

* 1. Are of 2 types :
* **Asymmetric release**
  1. Like telephone s/m: when one party hangs up, the connection is broken.
* **Symmetric release**
  1. It treats the connection as 2 separate unidirectional connections and requires each one to be released separately.

**Asymmetric Release**

Software quality assurance and testing Page 17



Abrupt disconnection with loss of data.

* After the connection is established, host 1 sends a TPDU that arrives properly at host 2.
* Then host 1 sends another TPDU.
* Unfortunately, host 2 issues a DISCONNECT before the second TPDU arrives.
* The result is that the connection is released and data are lost.

**Symmetric Release**

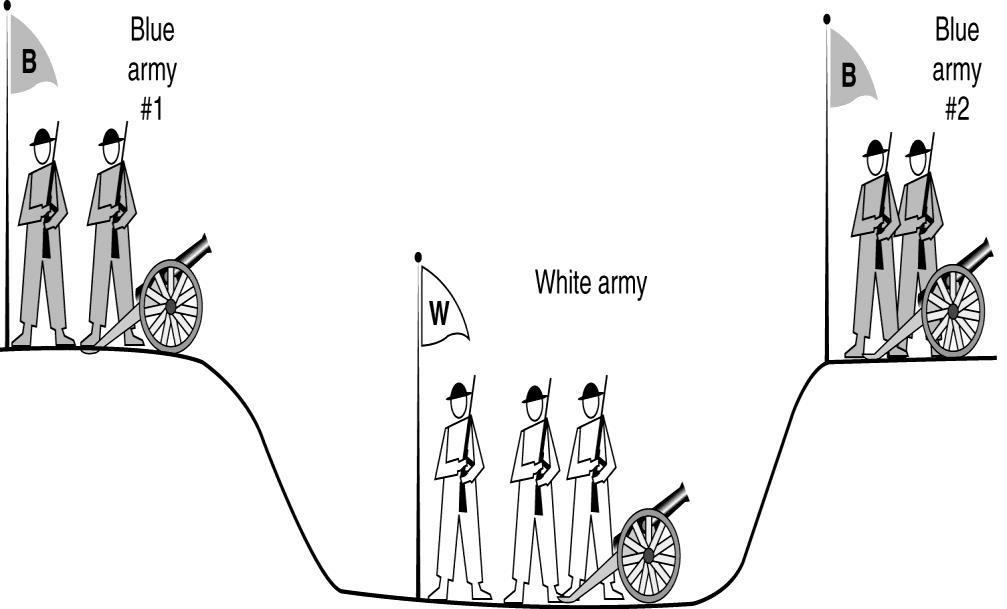
* A more sophisticated release protocol is needed to avoid data loss.
* One way is to use **symmetric release**, in which each direction is released independently of the other one.
* Here, a host can continue to receive data even after it has sent a DISCONNECT TPDU.
* Symmetric release does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
* One can envision a protocol in which host 1 says: **I am done. Are you done too**? If host 2 responds: I **am done too. Goodbye**, the connection can be safely released

**The Two-Army problem**

* Imagine that a white army is encamped in a valley, as shown in fig.
* On both of the surrounding hillsides are blue armies.
* The white army is larger than either of the blue armies alone, but together the blue armies are larger than the white army.
* If either blue army attacks by itself, it will be defeated, but if the two blue armies attack simultaneously, they will be victorious.

Software quality assurance and testing Page 18

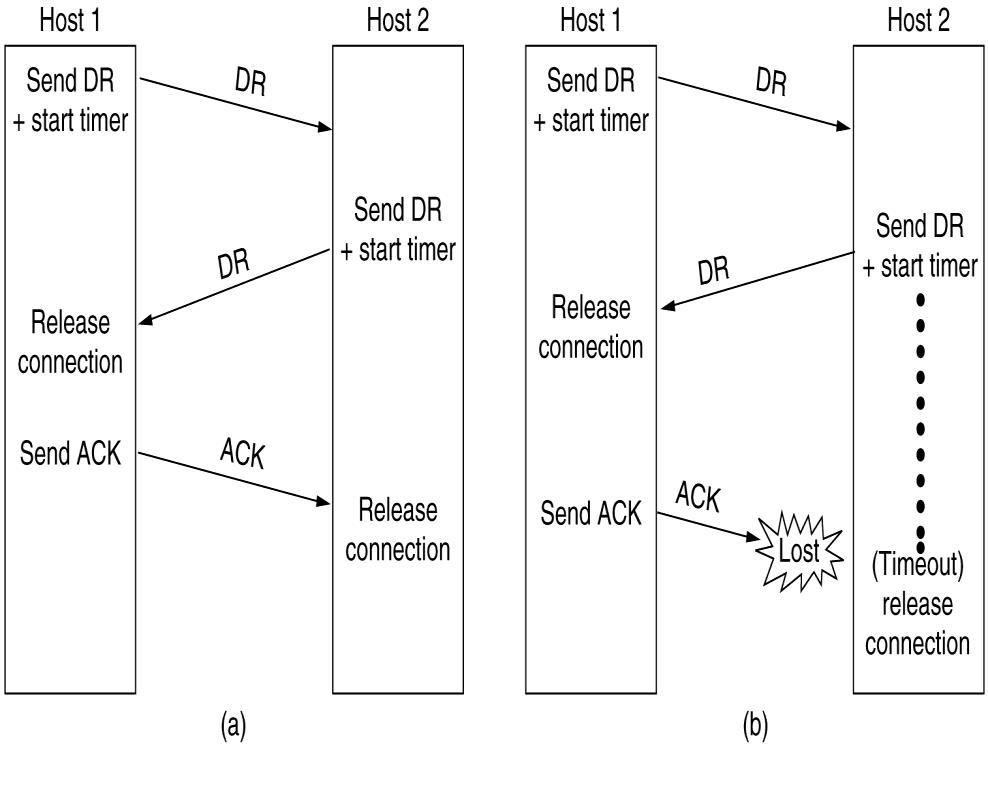
 The two-army problem.



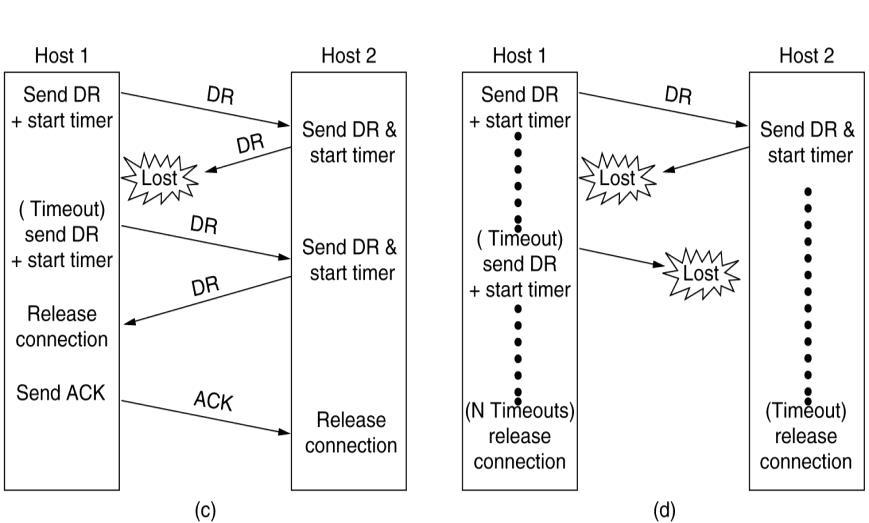
* The blue armies want to synchronize their attacks.
* However, their only communication medium is to send messengers on foot down into the valley, where they might be captured and the message lost (i.e., they have to use an unreliable communication channel).
* Does a protocol exist that allows the blue armies to win?
* Suppose that the commander of blue army #1 sends a message reading: ''I propose we attack at dawn on March 29. How about it?''
* Now suppose that the message arrives, the commander of blue army #2 agrees, and his reply gets safely back to blue army #1.
* Will the attack happen? Probably not, because commander #2 does not know if his reply got through. If it did not, blue army #1 will not attack, so it would be foolish for him to charge into battle.
* Now improve the protocol by making it a three-way handshake.
* The initiator of the original proposal must acknowledge the response.
* Assuming no messages are lost, blue army #2 will get the acknowledgement, but the commander of blue army #1 will now hesitate.
* After all, he does not know if his acknowledgement got through, and if it did not, he knows that blue army #2 will not attack.
* We could make a four-way handshake protocol, but that does not help either.
* To see the relevance of the two-army problem to releasing connections, just substitute ''disconnect'' for ''attack.'‘
* If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen.

Software quality assurance and testing Page 19

Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) final ACK lost.



(c) Response lost. (d) Response lost and subsequent DRs lost.



* While this protocol usually works, in theory it can fail if the initial DR and N retransmissions are all lost.
* The sender will give up and release the connection, while the other side knows nothing at all about the attempts to disconnect and is still fully active.
* This situation results in a **half-open connection**.
* We could avoid this problem by not allowing the sender to give up after N retries but forcing it to go on forever until it gets a response.

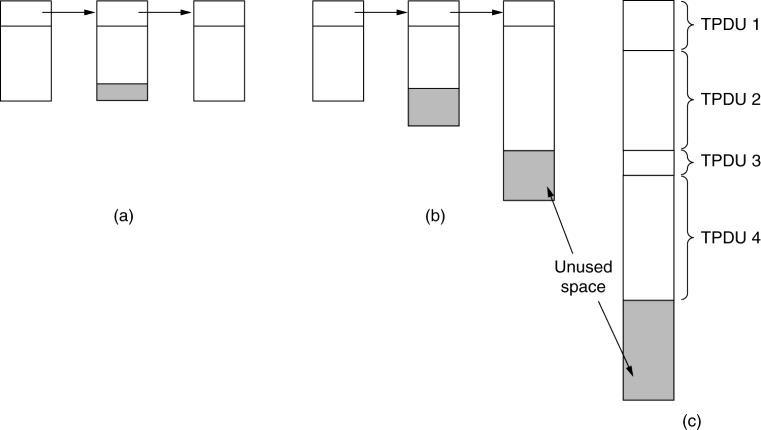
Software quality assurance and testing Page 20

* One way to kill off half-open connections is to have a rule saying that if no TPDUs have arrived for a certain number of seconds, the connection is then automatically disconnected

**Flow Control and Buffering**

* How connections are managed when in use: o **Flow control**
* In some ways the flow control problem in the transport layer is the same as in the data link layer, but in other ways it is different.
* The basic similarity is that in both layers a sliding window or other scheme is needed on each connection to keep a fast transmitter from overrunning a slow receiver.
* The main difference is that a router usually has relatively few lines, whereas a host may have numerous connections.
* This difference makes it impractical to implement the data link buffering strategy in the transport layer.
* if the network service is unreliable, the sender must buffer all TPDUs sent, just as in the data link layer.
* with reliable network service, other trade-offs become possible.
* In particular, if the sender knows that the receiver always has buffer space, it need not retain copies of the TPDUs it sends.
* if the receiver cannot guarantee that every incoming TPDU will be accepted, the sender will have to buffer anyway.
* In the latter case, the sender cannot trust the network layer's acknowledgement, because the acknowledgement means only that the TPDU arrived, not that it was accepted.
* Even if the receiver has agreed to do the buffering, there still remains the question of the buffer size.
* If most TPDUs are nearly the same size, it is natural to organize the buffers as a pool of identically-sized buffers, with one TPDU per buffer, as in Fig (a).
* if there is wide variation in TPDU size, a pool of fixed-sized buffers presents problems.
* If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a short TPDU arrives.
* If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for long TPDUs, with the attendant complexity.

Software quality assurance and testing Page 21



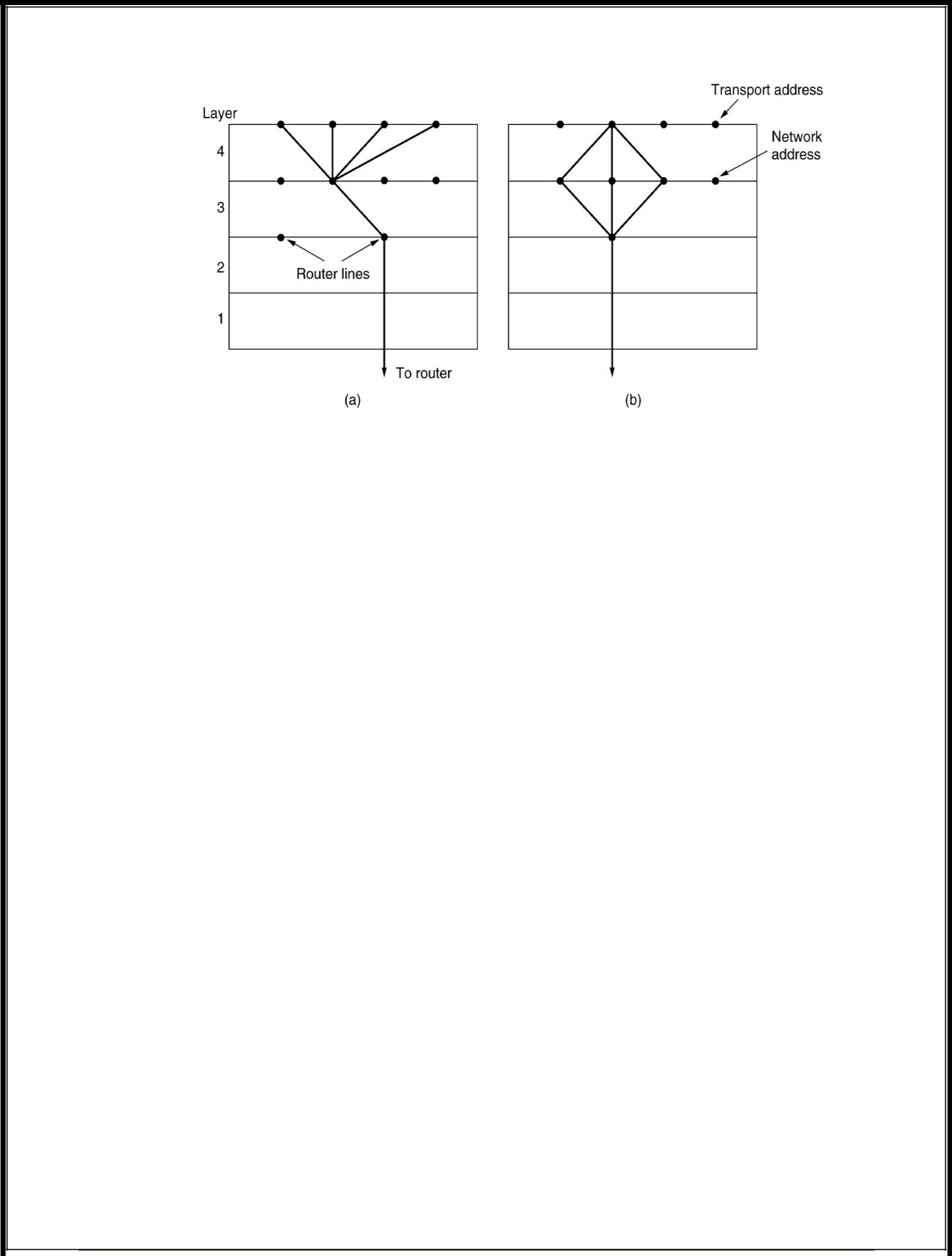
Chained fixed-size buffers. (b) Chained variable-sized buffers.

* 1. One large circular buffer per connection.
* Another approach to the buffer size problem is to use variable-sized buffers, as in Fig(b).
* The advantage here is better memory utilization, at the price of more complicated buffer management.
* A third possibility is to dedicate a single large circular buffer per connection, as in Fig. (c).
* This system also makes good use of memory, provided that all connections are heavily loaded, but is poor if some connections are lightly loaded.

**Multiplexing**

* In the transport layer the need for multiplexing can arise in a number of ways.
* For Eg, if only one network address is available on a host, all transport connections on that machine have to use it.
* When a TPDU comes in, some way is needed to tell which process to give it to.
* This situation, called **upward multiplexing**, is shown in fig a.
* In this figure, 4 distinct transport connections all use the same network connection (e.g., IP address) to the remote host.
* If a user needs more bandwidth than one virtual circuit can provide, a way out is to open multiple network connections and distribute the traffic among them on a round-robin basis, as indicated in fig b.
* This modus operandi is called **downward multiplexing**
* (a) Upward multiplexing. (b) Downward multiplexing.

Software quality assurance and testing Page 22



**7.Explain in detail about the principle of congestion control? (April/May 2014)**

**Congestion Control**

Congestion occurs at routers, so it is detected at the network layer. However, congestion is ultimately caused by traffic sent into the network by the transport layer. The only effective way to control congestion is for the transport protocols to send packets into the network more slowly.

* Desirable bandwidth allocation
* Regulating the sending rate

**Principles of Congestion Control**

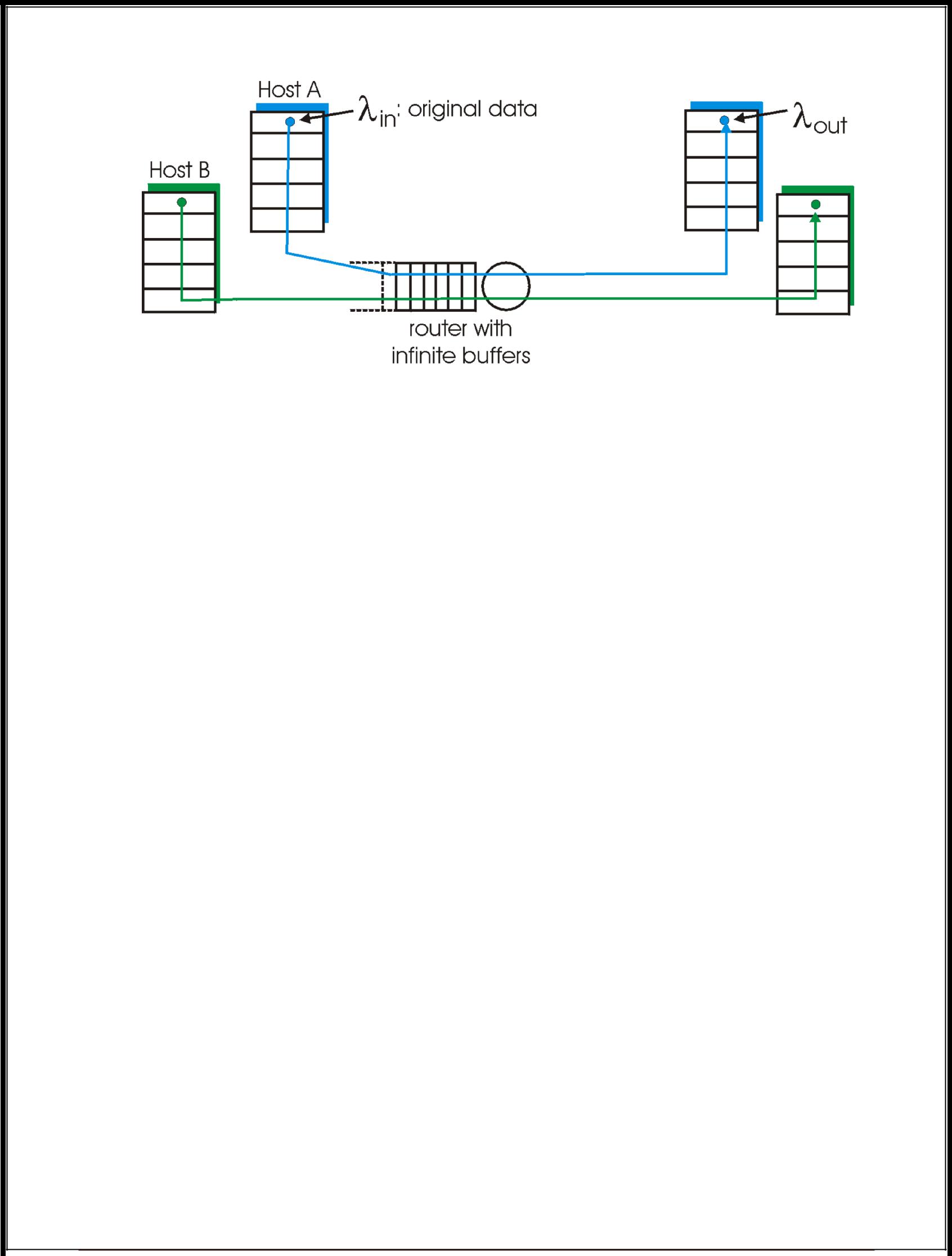
* Congestion:
* informally: “too many sources sending too much data too fast for *network* to handle”
* manifestations:

o lost packets (buffer overflow at routers)

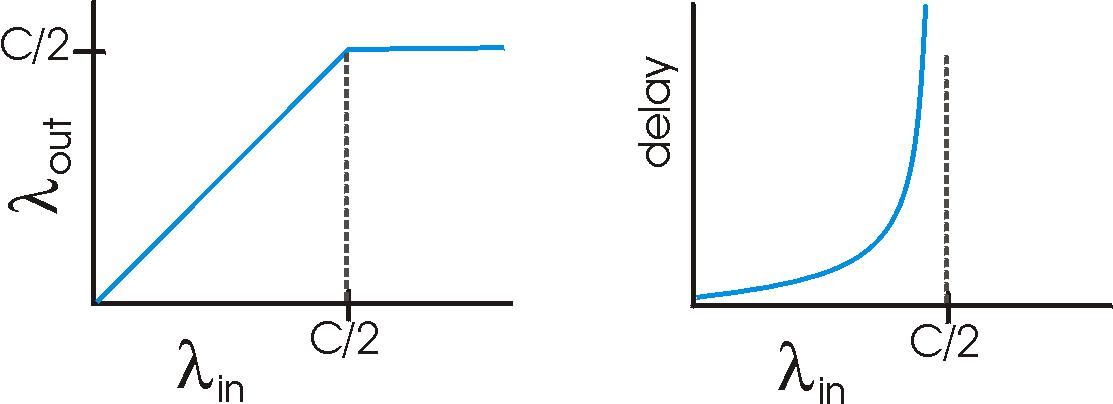
* 1. long delays (queuing in router buffers)
* a highly important problem!

**Causes/costs of congestion: scenario 1**

Software quality assurance and testing Page 23



* two senders, two receivers
* one router,
* infinite buffers
* no retransmission



* Throughput increases with load
* Maximum total load C (Each session C/2)
* Large delays when congested

o The load is stochastic

**Goals of congestion control**

* Throughput:

o Maximize goodput

Software quality assurance and testing Page 24

* 1. the total number of bits end-end
* Fairness:
  1. Give different sessions “equal” share.

o Max-min fairness

* Maximize the minimum rate session.

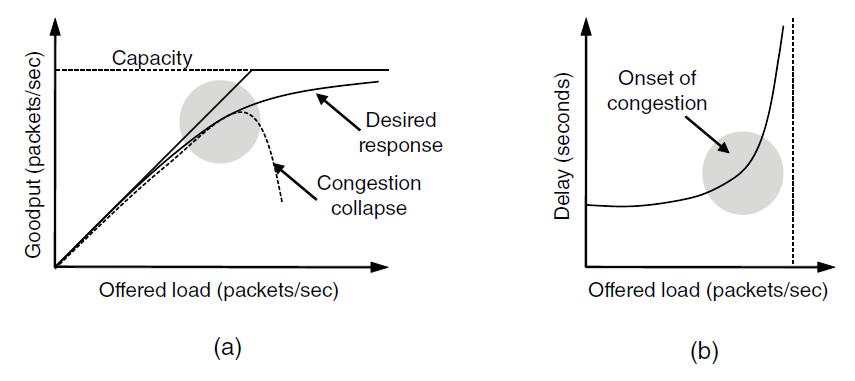
1. Single link:
   * Capacity R
   * sessions m
   * Each sessions: R/m

**Desirable Bandwidth Allocation (1)**

* It is to find a good allocation of bandwidth to the transport entities that are using the network.
* A good allocation will deliver good performance because it uses all the available bandwidth but avoids congestion, it will be fair across competing transport entities, and it will quickly track changes in traffic demands.

**Efficiency and power**

* As the load increases goodput initially increases at the same rate, but as the load approaches the capacity, goodput rises more gradually.
* Initially the delay is fixed, representing the propagation delay across the network.
* As the load approaches the capacity, the delay rises, slowly at first and then much more rapidly. This is again because of bursts of traffic that tend to mound up at high load.



(a) Goodput and (b) delay as a function of offered load

**Max-Min Fairness**

Software quality assurance and testing Page 25

* Model: Graph G(V,e) and sessions s1 … sm
* For each session si a rate ri is selected.
* The rates are a Max-Min fair allocation:

1. The allocation is maximal
   * No ri can be simply increased
   1. Increasing allocation ri requires reducing
      * Some session j
      * rj ≤ ri

* Maximize minimum rate session.
* Model: Graph G(V,e) and sessions s1 … sm
* Algorithmic view:
  1. For each link compute its fair share f(e).
     + Capacity / # session

o select minimal fair share link.

o Each session passing on it, allocate f(e).

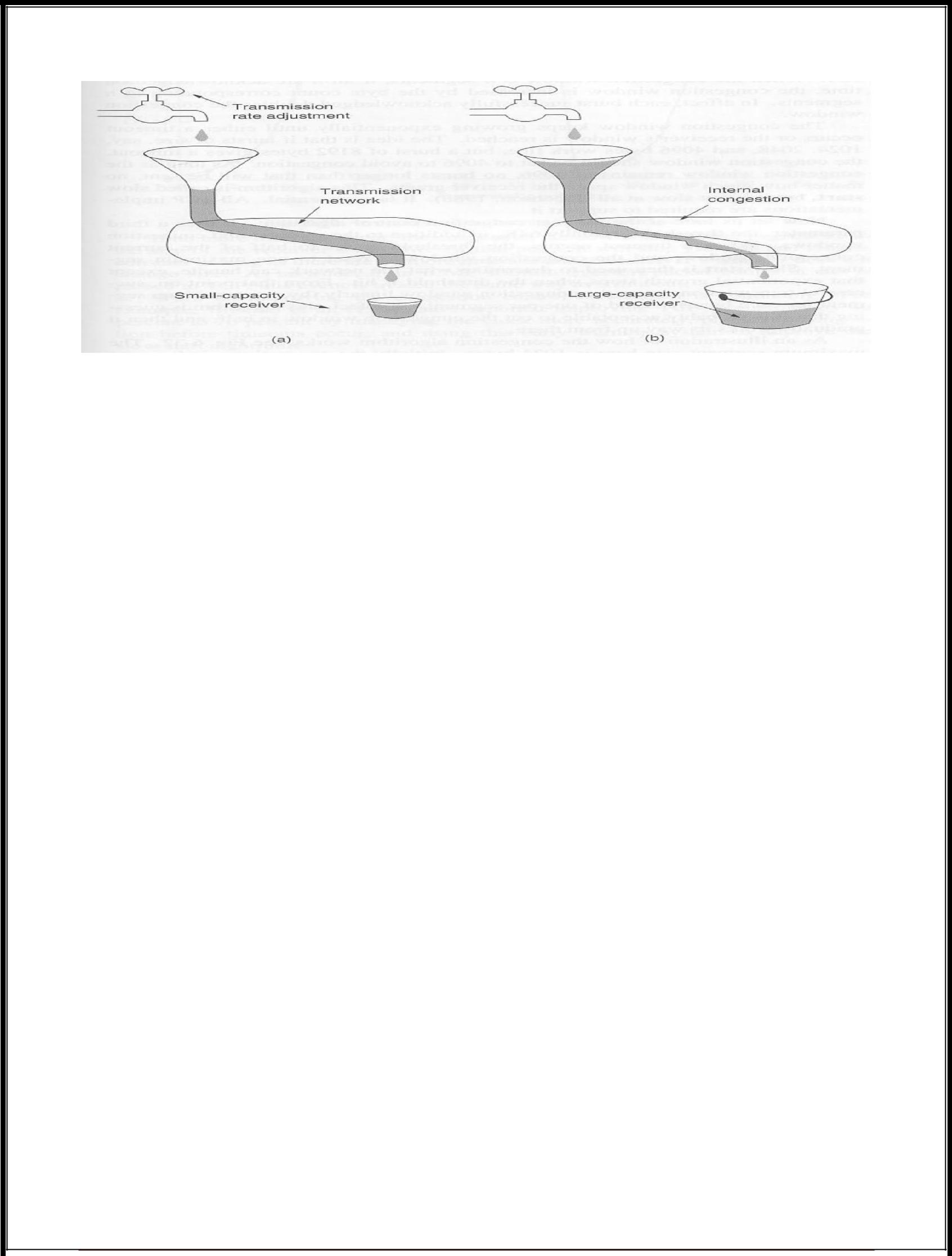
o Subtract the capacities and delete sessions

* 1. continue recessively.
* Fluid view.

**Regulating the Sending Rate**

* When the load offered to any networks is more than it can handle, congestion builds up. The Internet is no exception.
* Algorithms have been developed over the past decade to deal with congestion.
* Although the network layer also tries to manage congestion, most of the heavy lifting is done by TCP because the real solution to congestion is to slow down the data rate.

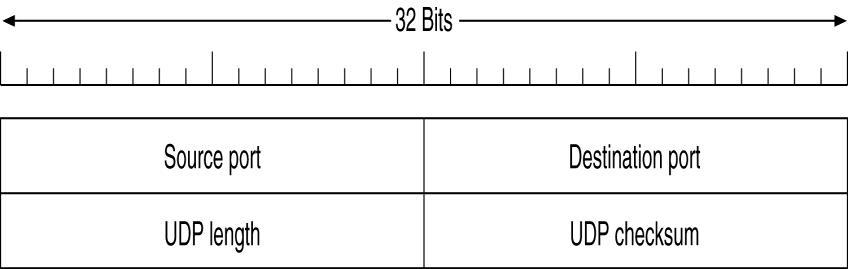
Software quality assurance and testing Page 26



* 1. A fast network feeding a low capacity receiver
  2. A slow network feeding a high capacity receiver
* In theory congestion can be dealt with by employing a principle borrowed from physics: the law of conservation of packets. The idea is not to inject a new packet into the network until an old one leaves (i.e. is delivered). TCP attempts to achieve this goal by dynamically manipulating the Window size.

**8.Explain briefly about the UDP segment structure? (April/May 2014)**

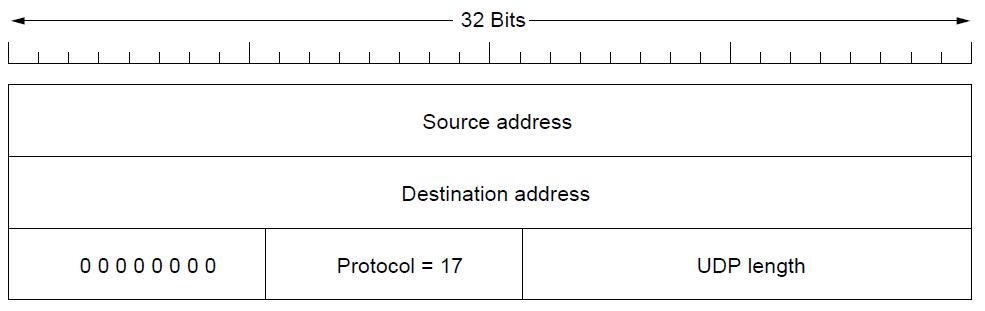
* The Internet protocol suite also supports a connectionless transport protocol, UDP (User Data Protocol)
* UDP provides a way for applications to send encapsulated raw IP datagrams and send them without having to establish a connection.
* Many client-server applications that have 1 request and 1 response use UDP rather than go to the trouble of establishing and later releasing a connection.
* A UDP segment consists of an 8-byte header followed by the data.



The UDP header.

Software quality assurance and testing Page 27

* The two ports serve the same function as they do in TCP: to identify the end points within the source and destination machines.
* The UDP length field includes the 8-byte header and the data.
* The UDP checksum is used to verify the size of header and data.



The IPv4 pseudoheader included in the UDP checksum.

**Remote Procedure Call**

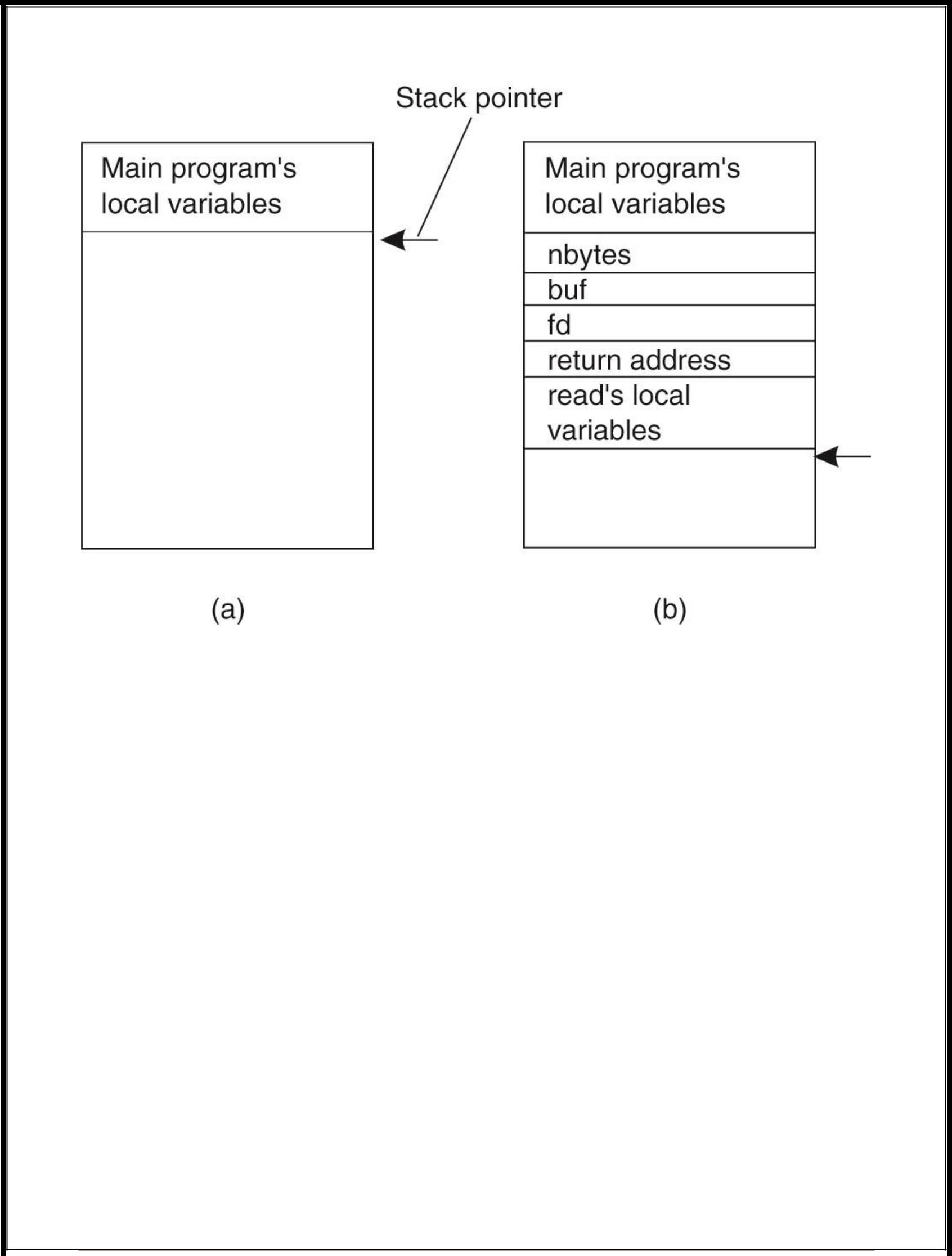
**Remote Procedure Calls (RPC)**

* RPC is a powerful technique for constructing distributed, client-server based applications.
* It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure.
* The two processes may be on the same system, or they may be on different systems with a network connecting them.
* By using RPC, programmers of distributed applications avoid the details of the interface with the network
* Avoid explicit message exchange between processes
* Basic idea is to allow a process on a machine to call procedures on a remote machine

– Make a remote procedure possibly look like a local one

**Conventional Procedure Call**

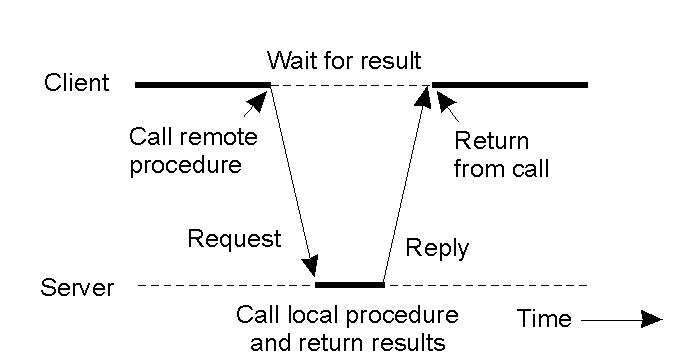
Software quality assurance and testing Page 28



(a) Parameter passing in a local procedure call: the stack before the call to read. (b) The stack while the called procedure is active.

* How are parameter passed in a remote procedure call, while making it look like a local procedure call?

**Client and Server Stubs**

****

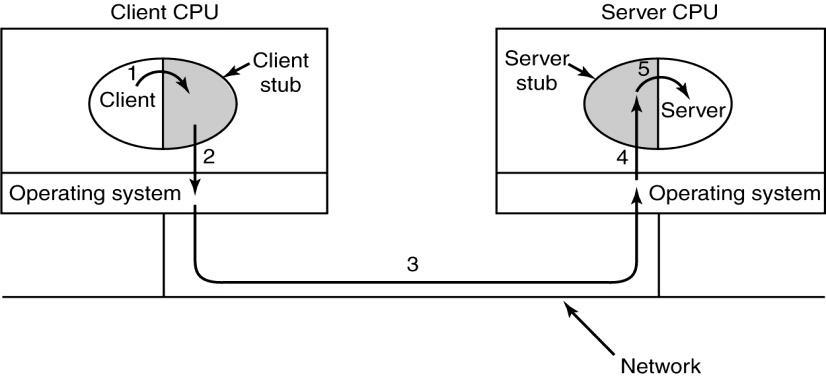
Software quality assurance and testing Page 29

**Principle of RPC between a client and server program.**

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

**RPC: The basic mechanism**

****

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

* 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.

Software quality assurance and testing Page 30

* Step 4 is the operating system passing the incoming packet to the server stub.
* Finally, step 5 is the server stub calling the server procedure with the unmarshaled parameters. The reply traces the same path in the other direction.

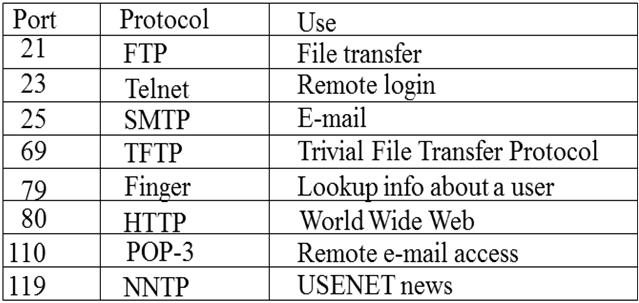
**The Internet Transport Protocols: TCP**

* TCP (Transmission Control Protocol) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork.
* An internetwork differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters.

**The TCP Service Model**

TCP service is obtained by both the sender and receiver creating end points, called sockets. Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a port. A port is the TCP name for a TSAP. For TCP service to be obtained, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine.

* full duplex and point-to-point
* byte stream
* immediate data
* urgent data



Some assigned ports.

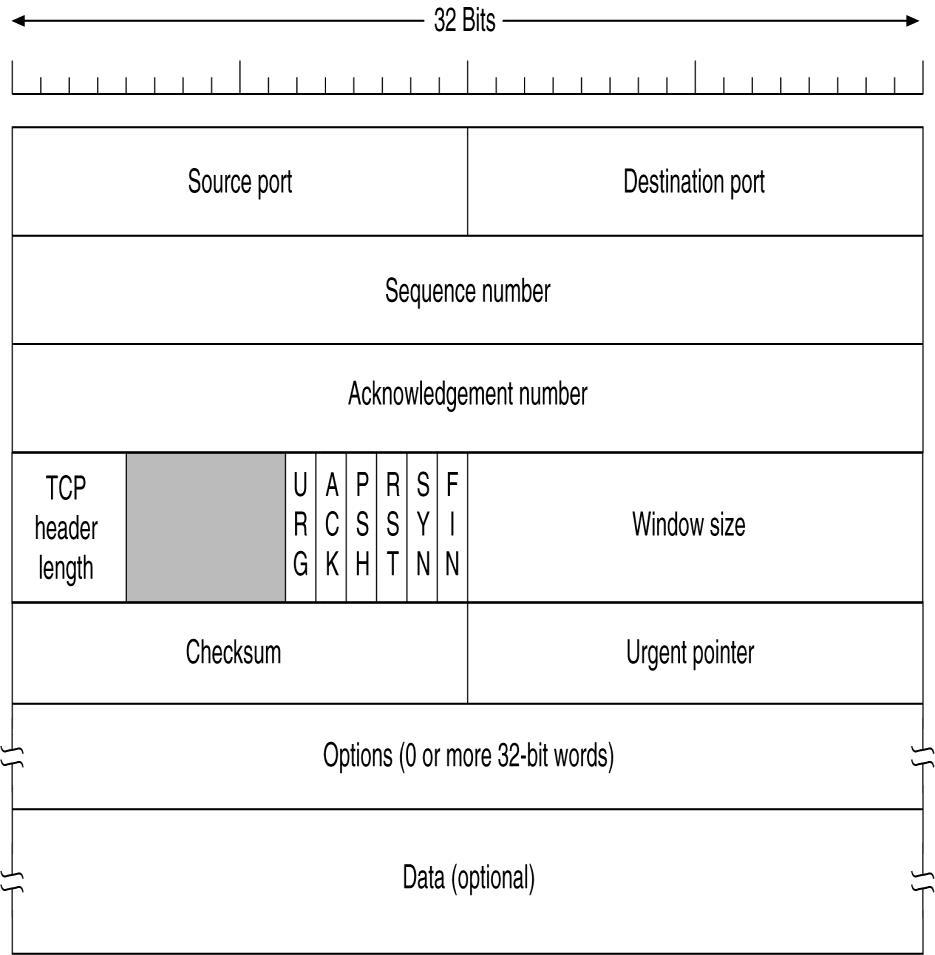
**10.Explain briefly about the TCP segment structure? (April/May 2013) The TCP Segment Header**

* Every segment begins with a fixed-format 20-byte header.
* The fixed header may be followed by header options.
* After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the

Software quality assurance and testing Page 31

* 1. first 20 refer to the IP header and
  2. second to the TCP header.
* Segments without any data are legal and are commonly used for
  1. acknowledgements and

o control messages.



TCP Header

* The **Source port** and **Destination port** fields identify the local end points of the connection.
* A port plus its host's IP address forms a 48-bit unique end point (TSAP).
* The **Sequence number** and **Acknowledgement number** fields perform their usual functions.
* **Acknowledgement number** specifies the next byte expected, not the last bytecorrectly received.
* Both are 32 bits long
* The **TCP header length** tells how many 32-bit words are contained in the TCP header.
* This information is needed because the Options field is of variable length, so the header is, too.
* Next comes a 6-bit field that is not used .

Software quality assurance and testing Page 32

* Now come six 1-bit flags.
* **URG** is set to 1 if the Urgent pointer is in use.
  1. The **Urgent pointer** is used to indicate a byte offset from the current sequence number at which urgent data are to be found.
* The **ACK** bit
  1. set to 1 to indicate that the Acknowledgement number is valid.
  2. If ACK is 0, the segment does not contain an acknowledgement so the Acknowledgement number field is ignored.
* The **PSH** bit indicates PUSHed data.
  1. Applications can use the PUSH flag, which tells TCP not to delay the transmission.
* The **RST** bit
  1. used to reset a connection that has become confused due to a host crash or

some other reason.

1. It is also used to reject an invalid segment or refuse an attempt to open a connection.
   1. if you get a segment with the RST bit on, you have a problem on your hands.

* The **SYN** bit is used to establish connections.
  1. The connection request has SYN = 1 and ACK = 0 to indicate that the

piggyback acknowledgement field is not in use.

o The connection reply bears an acknowledgement it has SYN = 1 and ACK = 1.

o In essence the SYN bit is used to denote CONNECTION REQUEST and

CONNECTION ACCEPTED, with the ACK bit used to distinguish between

those two possibilities.

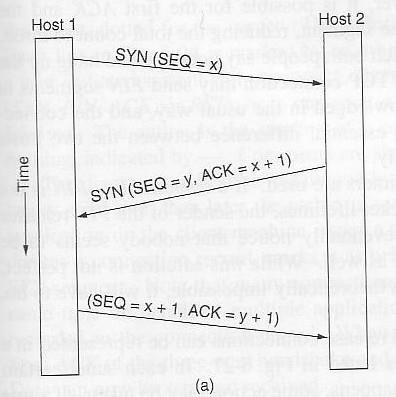
* 1. The **FIN** bit is used to release a connection. It specifies that the sender has no more data to transmit.
* Both **SYN** and **FIN** segments have sequence numbers and are thus guaranteed to be processed in the correct order.
* Flow control in TCP is handled using a variable-sized sliding window.
* The **Window size** field tells how many bytes may be sent starting at the byte acknowledged.
* A **Checksum** is also provided for extra reliability.
* The **Options** field provides a way to add extra facilities not covered by the regular header.
* The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept.

**TCP Connection Establishment**

Software quality assurance and testing Page 33

Connections are established in TCP using a three-way handshake:

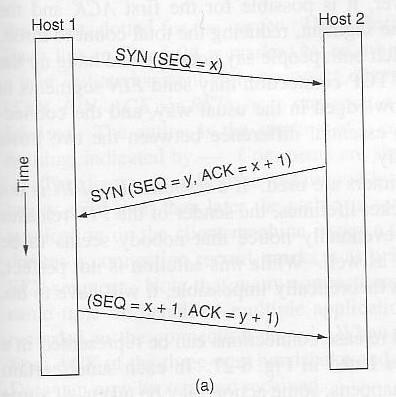
* Host 1 chooses a sequence number, *x*, and sends a CONNECTION REQUEST containing it to host 2.
* Host 2 replies with CONNECTION ACCEPTED acknowledgment *x*, and announcing its own initial sequence number, *y*.
* Finally Host 1 acknowledges host 2’s choice of an initial sequence number in the first data that it sends.



To establish a connection, one side, say a server, passively waits for an incoming connection by executing LISTEN and ACCEPT primitives

The other side, say a client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, and the max TCP segment size it is willing to accept

The CONNECT primitive sends a TCP segment with the SYN bit = 1 and ACK = 0 and waits for a response



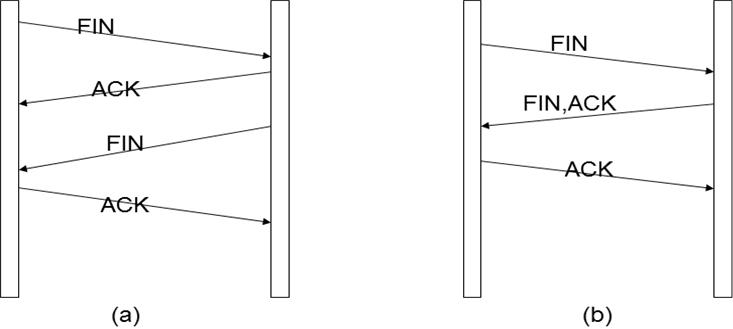
Software quality assurance and testing Page 34

When this segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a LISTEN on the port given in the Destination port field. If not, it sends a reply with the RST bit on to reject the connection.

If some process is listening on the port, that process is given the incoming TCP segment. It can either accept or reject the connection. If it accepts, an acknowledgment segment is sent back.

**TCP Connection Release**

* Client application wishes to release the TCP connection
* Client sends a TCP segment with the FIN bit set in the TCP header
* Client changes state to FIN Wait 1 state
* Server receives the FIN
* Server responds back with ACK to acknowledge the FIN
* Server changes state to Close Wait. In this state the server waits for the server application to close the connection
* Client receives the ACK
* Client changes state to FIN Wait 2. In this state, the TCP connection from the client to server is closed. Client now waits close of TCP connection from the server end



1. Normally, four TCP segments are needed to release a connection .
2. It is possible for the first ACK and the second FIN to be contained in the same segment .

Software quality assurance and testing Page 35

**TCP Transmission Policy**

* Window management is not directly tied to acks as in data link protocols
* Exclusive buffer messages manage the transmission
* If no buffer at receiver, then no transmission by sender except

– Urgent data may be sent

– One byte segment can be sent to ask receiver to renounce its buffer status (to prevent deadlock)

* Sender and receiver are not forced to transmit or receive as soon as they receive data from the application. This improves performance as follows:

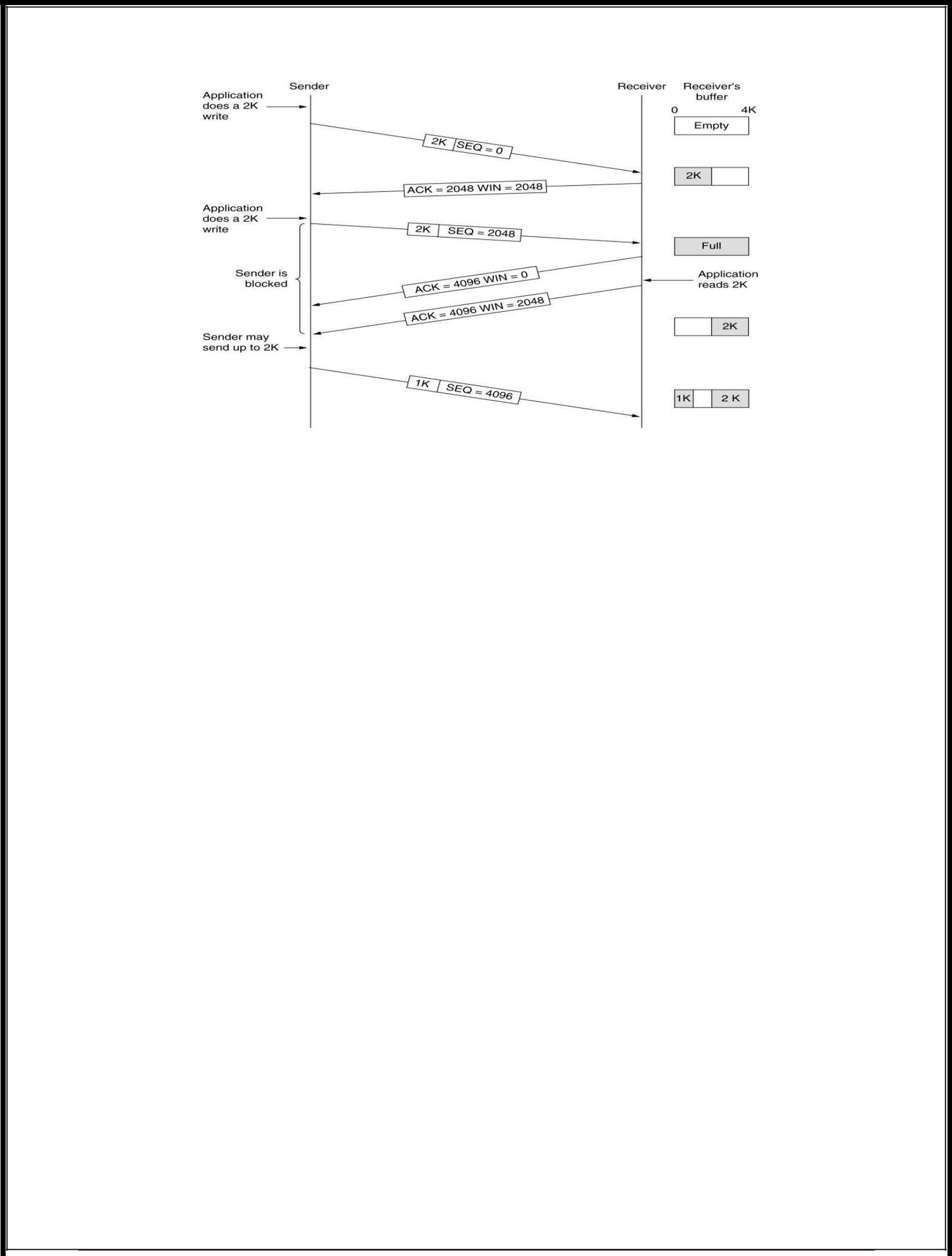
– If one byte messages are sent (like in TELNET) then use NAGLE’s algorithm

* + - Send first byte and keep the rest until ack comes back
    - As ack comes in send the rest and keep the further incoming bytes until ack is received

**TCP Sliding Window**

* Window management in TCP decouples the issues of acknowledgement of the correct receipt of segments and receiver buffer allocation.
* For example, suppose the receiver has a 4096-byte buffer, as shown in Fig. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
* However, since it now has only 2048 bytes of buffer space (until the application removes some data from the buffer), it will advertise a window of 2048 starting at the next byte expected.
* Now the sender transmits another 2048 bytes, which are acknowledged, but the advertised window is of size 0.
* 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 larger window and more data can be sent.

Software quality assurance and testing Page 36



Window management in TCP.

**Nagle's Algorithm and Silly Window Syndrome**

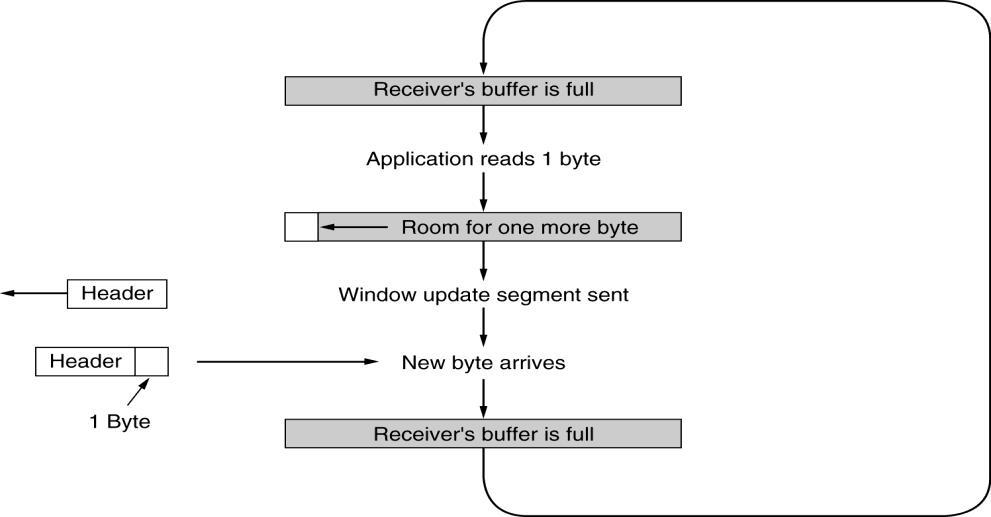
Silly Window Syndrome: Receive bytes one by one and send window messaging accordingly

* Clark’s Solution to Silly Window Syndrome

– Prevent receiver from sending one byte updates and make it wait until decent amount of space available before it sends buffer messages

– Sender may also pospone sending messages

* Nagle’s Algorithm and Clark’s solution are complementary and they can be used at the same time



Software quality assurance and testing Page 37

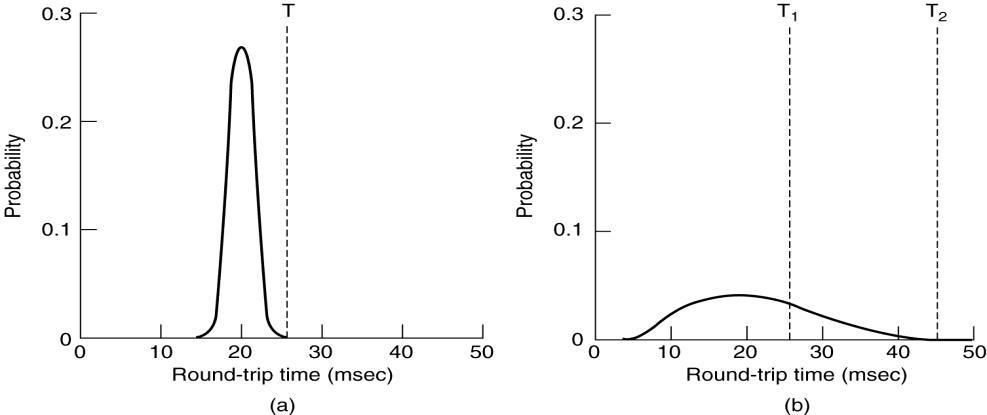
Silly window syndrome.

**TCP Timer Management**

* TCP uses multiple timers (at least conceptually) to do its work.
* The most important of these is the retransmission timer. When a segment is sent, a retransmission timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If, on the other hand, the timer goes off before the acknowledgment comes in the segment is retransmitted (and the timer started again).

The question that arises is: How long should the timeout interval be?

* This problem is much more difficult in the Internet transport layer than in the generic data link protocols, where the delay is very predictable.
* The solution is to use a highly dynamic statistical algorithm that constantly adjusts the timeout interval based on continuous measurements of network performance. This algorithm was proposed by Jacobson in 1988.



1. Probability density of ACK arrival times in the data link layer.
2. Probability density of ACK arrival times for TCP.

**Timeout**

For each connection, TCP maintains a variable, RTT, that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, M. It then updates RTT according to the formula.

RTT=α RTT+(1- α )M

Typically a = 7/8.

Software quality assurance and testing Page 38

Another smoothed variable, D, is the deviation, that is | RTT - M |.

D= α D +(1- α ) | RTT - M |

Timeout=RTT+4D

For each connection, TCP maintains a variable, *SRTT* (Smoothed Round-Trip Time), that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and also to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, *R.* It then updates *SRTT* according to the formula

*SRTT* = α *SRTT* + (1 − α) *R*

where α is a smoothing factor that determines how quickly the old values are forgotten. Typically, α = 7*/*8. This kind of formula is an **EWMA** (**Exponentially Weighted Moving** **Average**) or low-pass filter that discards noise in the samples.

Jacobson proposed making the timeout value sensitive to the variance in round-trip times as well as the smoothed round-trip time. This change requires keeping track of another smoothed variable, *RTTVAR* (Round- Trip Time variation) that is updated using the formula

*RTTVAR* = β *RTTVAR* + (1 − β) |*SRTT* − *R* |

This is an EWMA as before, and typically β = 3*/*4. The retransmission timeout,

*RTO*, is set to be

*RTO* = *SRTT* + 4 × *RTTVAR*

**11.Explain how TCP controls congestion? (April/May 2012)**

**TCP Congestion Control**

* Essential strategy :: The TCP host sends packets into the network without a reservation and then the host reacts to observable events.
* Originally TCP assumed FIFO queuing.
* Basic idea :: each source determines how much capacity is available to a given flow in the network.
* ACKs are used to ‘pace’ the transmission of packets such that TCP is “self-clocking”.

**AIMD(Additive Increase / Multiplicative Decrease)**

* CongestionWindow (cwnd) is a variable held by the TCP source for each connection.
  + MaxWindow :: min (**CongestionWindow** , AdvertisedWindow)

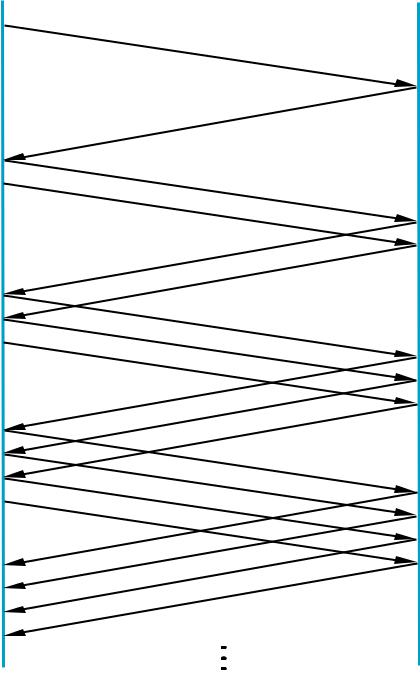
Software quality assurance and testing Page 39

* + EffectiveWindow = MaxWindow – (LastByteSent -LastByteAcked)
* **cwnd** is set based on the perceived level of congestion. The Host receives implicit(packet drop) or explicit (packet mark) indications of internal congestion.

**Additive Increase (AI)**

* Additive Increase is a reaction to perceived available capacity (referred to as **congestion avoidance** stage).
* Frequently in the literature, additive increase is defined by parameter **α** (where the default is **α** **= 1**).
* Linear Increase :: For each “cwnd’s worth” of packets sent, increase cwnd by 1 packet.
* In practice, **cwnd** is incremented fractionally for each arriving ACK.
  + **increment = MSS x (MSS /cwnd) cwnd = cwnd + increment**

Sourc Destinatio



Add one packet each RTT

**Multiplicative Decrease (MD)**

* Key assumption :: a dropped packet and resultant timeout are due to congestion at a router.

Software quality assurance and testing Page 40

* Frequently in the literature, multiplicative decrease is defined by parameter β (where the default is β = 0.5)

Multiplicate Decrease:: TCP reacts to a timeout by halving cwnd.

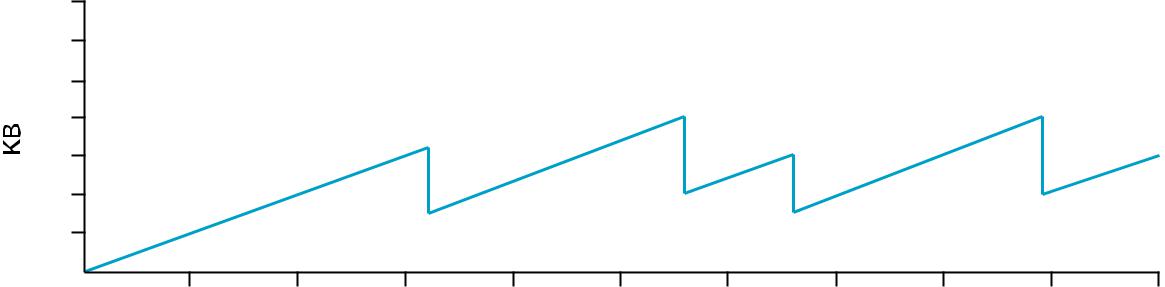
* Although defined in bytes, the literature often discusses cwnd in terms of packets (or more formally in MSS == Maximum Segment Size).

cwnd is not allowed below the size of a single packet.

**AIMD(Additive Increase / Multiplicative Decrease)**

* It has been shown that AIMD is a necessary condition for TCP congestion control to be stable.
* Because the simple CC mechanism involves timeouts that cause retransmissions, it is important that hosts have an accurate timeout mechanism.
* Timeouts set as a function of average RTT and standard deviation of RTT.
* However, TCP hosts only sample round-trip time once per RTT using coarse-grained clock.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 70 |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |
| 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 8.0 | 9.0 | 10.0 |



Time (seconds)

**Typical TCP Sawtooth Pattern**

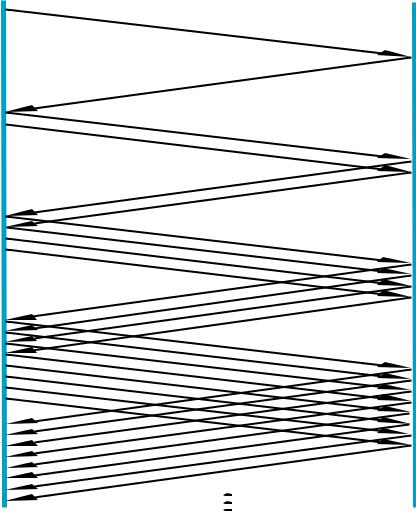
**Slow Start**

* Linear additive increase takes too long to ramp up a new TCP connection from cold start.
* Beginning with TCP Tahoe, the slow start mechanism was added to provide an initial exponential increase in the size of **cwnd**.

Remember mechanism by: **slow start prevents a slow start. Moreover, slow start is** **slower than sending a full advertised window’s worth of packets all at once.**

Software quality assurance and testing Page 41

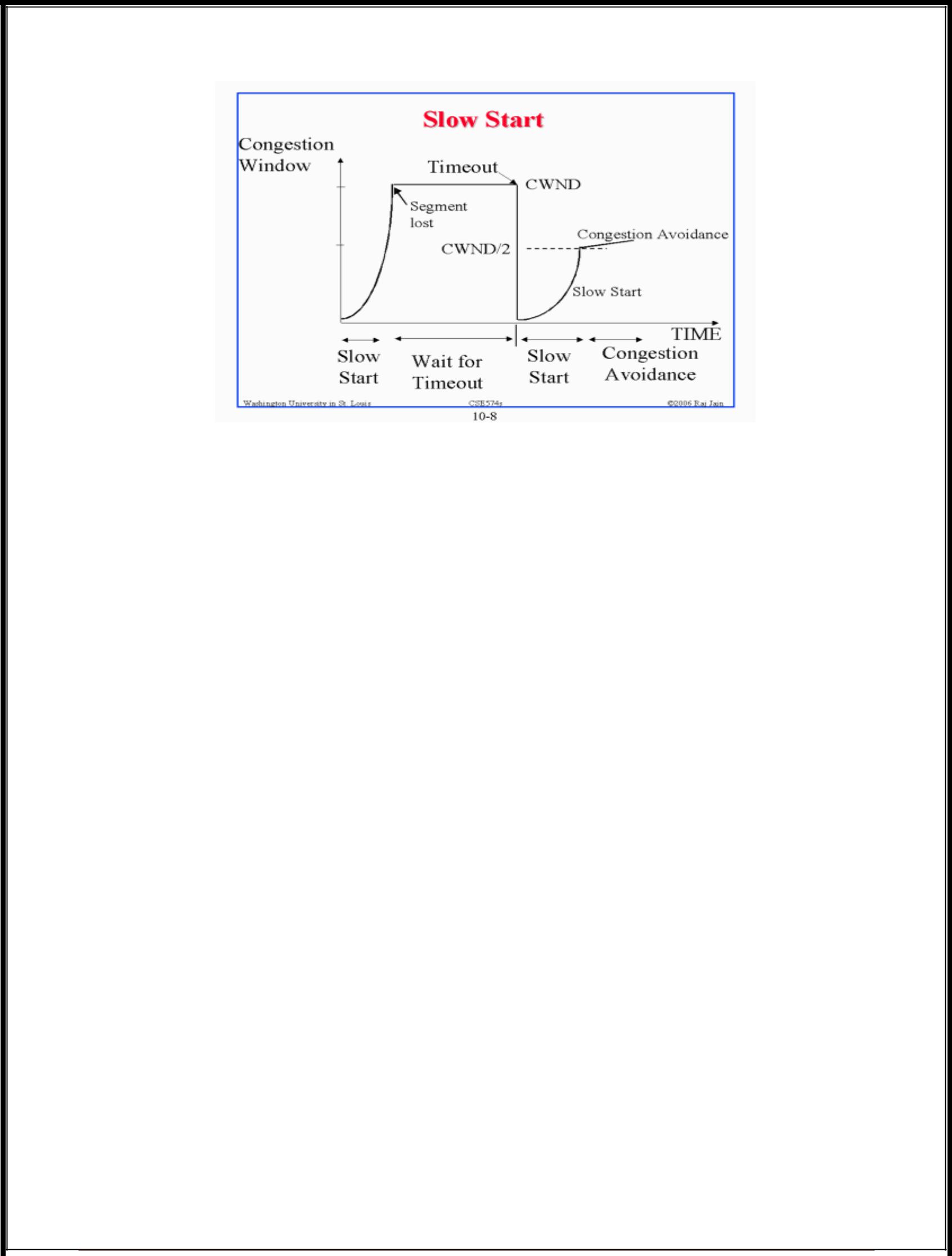
* The source starts with cwnd = 1.
* Every time an ACK arrives, cwnd is incremented.
* cwnd is effectively doubled per RTT “epoch”.
* Two slow start situations:
  + At the very beginning of a connection **{cold start**}.
  + When the connection goes dead waiting for a timeout to occur (i.e, the advertized window goes to zero!)



Slow Start Add one packet per ACK

* However, in the second case the source has more information. The current value of cwnd can be saved as a **congestion threshold.**
* This is also known as the “slow start threshold” **ssthresh**.

Software quality assurance and testing Page 42



**Fast Retransmit**

* Coarse timeouts remained a problem, and Fast retransmit was added with **TCP** **Tahoe**.
* Since the receiver responds every time a packet arrives, this implies the sender will see duplicate ACKs.

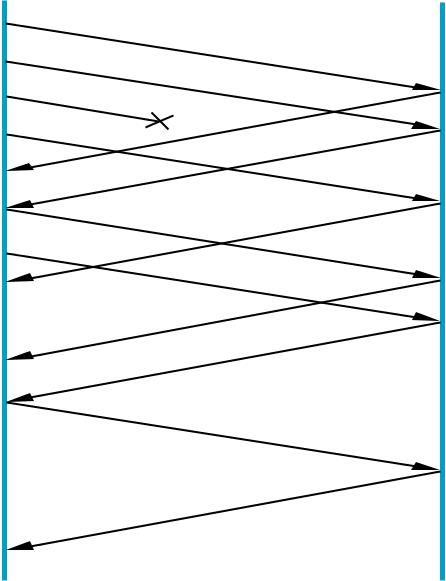
Basic Idea:: use **duplicate ACKs** to signal lost packet.

Upon receipt of **three** duplicate ACKs, the TCP Sender retransmits the lost packet.

* Generally, fast retransmit eliminates about half the coarse-grain timeouts.
* This yields roughly a 20% improvement in throughput.
* Note – fast retransmit does not eliminate all the timeouts due to small window sizes at the source.

Software quality assurance and testing Page 43

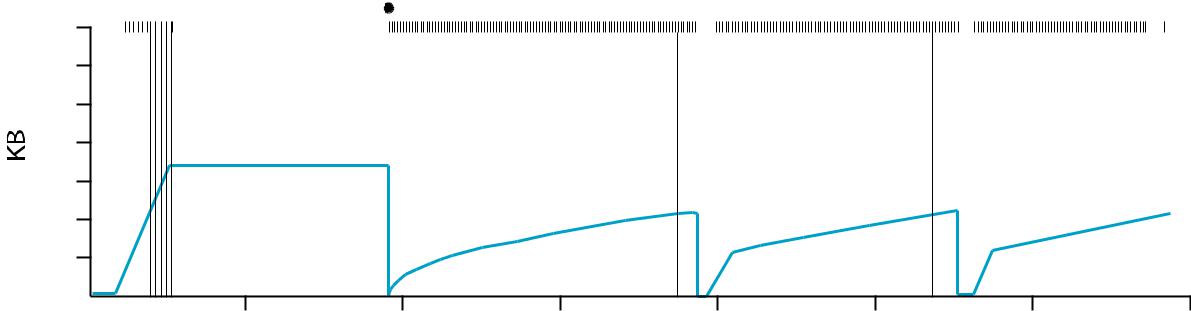
|  |  |
| --- | --- |
| Sender | Receiver |
| Packet 1 |  |
| Packet 2 |  |
| Packet 3 | ACK 1 |
|  |
| Packet 4 | ACK 2 |
| Packet 5 | ACK 2 |
|  |
| Packet 6 |  |
|  | ACK 2 |
|  | ACK 2 |



Retransmit packet 3

ACK 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Fast Retransmit Based on three duplicate ACKs | | | | |  |
| 70 |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 |



Time (seconds)

TCP Fast Retransmit Trace

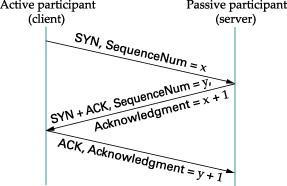
**Fast Recovery**

* Fast recovery was added with **TCP Reno**.
* Basic idea:: When fast retransmit detects three duplicate ACKs, start the recovery process from congestion avoidance region and use ACKs in the pipe to pace the sending of packets.

After Fast Retransmit, half cwnd and commence recovery from this point using linear additive increase ‘primed’ by left over ACKs in pipe.

**Three-way TCP Handshake**

Software quality assurance and testing Page 44



**Adaptive Retransmissions**

RTT:: Round Trip Time between a pair of hosts on the Internet.

* How to set the TimeOut value (RTO)?

– The timeout value is set as a function of the expected RTT.

– Consequences of a bad choice?

**Original Algorithm**

* Keep a running average of RTT and compute TimeOut as a function of this RTT.

– Send packet and keep timestamp ts .

– When ACK arrives, record timestamp ta .

SampleRTT = ta - ts

Compute a weighted average:

EstimatedRTT = α x EstimatedRTT + (1- α) x SampleRTT Original TCP spec: α in range (0.8,0.9)

TimeOut = 2 x EstimatedRTT

Software quality assurance and testing Page 45

**Karn/Partidge Algorithm**

An obvious flaw in the original algorithm:

Whenever there is a retransmission it is impossible to know whether to associate the ACK with the original packet or the retransmitted packet.

**Karn/Partidge Algorithm**

* 1. Do not measure SampleRTT when sending packet more than once.
  2. For each retransmission, set TimeOut to double the last TimeOut.

{ Note – this is a form of exponential backoff based on the believe that the lost packet is due to congestion.}

**Jacobson/Karels Algorithm**

The problem with the original algorithm is that it did not take into account the variance of SampleRTT.

Difference = SampleRTT – EstimatedRTT

EstimatedRTT = EstimatedRTT +(δ x Difference)

Deviation = δ (|Difference| - Deviation)

where δ is a fraction between 0 and 1.

TCP computes timeout using both the mean and variance of RTT

TimeOut = µ x EstimatedRTT + Φ x Deviation

where based on experience µ = 1 and Φ = 4.

**PONDICHERRY UNIVERSITY QUESTIONS**

**2 MARKS**

1.Write the relationship between transport and network layer?**(April/May 2013)** **(Pg.No.2)(Qn. No.2)**

2.What is Transmission delay?**(Nov/Dec 2012) (Pg.No.3)(Qn. No.8)** 3.What is round-trip time?**(Nov/Dec 2012) (Pg.No.4)(Qn. No.9)**

4.What are the different types of Multiplexing?**(Nov/Dec 2014) (Pg.No.5)(Qn. No.13)**

5.What is meant by demultiplexing?**(April/May 2014) (Pg.No.5)(Qn. No.14)**

Software quality assurance and testing Page 46

6.Define UDP?**(Nov/Dec 2011)(Nov/Dec 2013) (Pg.No.5)(Qn. No.18)** 7.What is the drawback of UDP?**(April/May 2012) (Pg.No.5)(Qn. No.19)** 8.What is Datagram? **(Nov/Dec 2011) (Pg.No.6)(Qn. No.20)**

9.Define TCP/IP? **(Nov/Dec 2011) (Pg.No.6)(Qn. No.23)**

10.List out the services provided by TCP?**(Nov/Dec 2012) (Pg.No.7)(Qn. No.24)** 11.Discuss the TCP connections needed in FTP?**(Nov/Dec 2014) (Pg.No.7)(Qn. No.25)** 12.What is the use of option field in TCP?**(April/May 2012) (Pg.No.7)(Qn. No.26)**

13.What do you mean by receive window?**(April/May 2014) (Pg.No.7)(Qn. No.27)**

14. Why an Application developer would ever choose to build an application over UDP rather than over TCP?**(Nov/Dec 2012) (Pg.No.8)(Qn. No.28)**

**11MARKS**

**Nov/Dec 2011**

1.Discuss about socket programming with TCP and UDP? **(Nov/Dec 2011) (Pg.No.14)(Qn.**

**No.3)**

2.Briefly explain transport layer services? **(Nov/Dec 2011) (Pg.No.8)(Qn. No.1)**

3.Explain TCP/IP protocol? **(Nov/Dec 2011) (Pg.No.34)(Qn. No.3)**

4.What is the need for congestion control mechanisms? Explain with any one algorithm in detail? **(Nov/Dec 2011) (Pg.No.25)(Qn. No.7)**

**April/May 2012**

1.What are the services provided by the transport layer to the upper layers? Explain?

**(April/May 2012) (Pg.No.8)(Qn. No.1)**

2.Explain how TCP controls congestion? **(April/May 2012) (Pg.No.42)(Qn. No.11)**

3.Describe the Major components of TCP control algorithm? **(April/May 2012)** **(Pg.No.42)(Qn. No.11)**

**Nov/Dec 2012**

1.Describe the principle of congestion control? **(Nov/Dec 2012) (Pg.No.34)(Qn. No.7)**

**April/May 2013**

1.Explain briefly about the TCP segment structure? **(April/May 2013) (Pg.No.10)(Qn.**

**No.3)**

**Nov/Dec 2013**

Software quality assurance and testing Page 47

1.Briefly explain transport layer services? **(Nov/Dec 2013) (Pg.No.8)(Qn. No.1)**

2.What is Multiplexing? Discuss it. **(Nov/Dec 2013) (Pg.No.24)(Qn. No.6)** 3.Discuss about flow control? **(Nov/Dec 2013) (Pg.No.22)(Qn. No.5)**

**April/May 2014**

1.Explain briefly about the UDP segment structure? **(April/May 2014) (Pg.No.31)(Qn.**

**No.8)**

3.Explain in detail about the principle of congestion control? **(April/May 2014)** **(Pg.No.28)(Qn. No.7)**

**Nov/Dec 2014**

1.Explain Multiplexing and Demultiplexing applications? **(Nov/Dec 2014) (Pg.No.24)(Qn.**

**No.6)**

2.Write in detail about congestion control? **(Nov/Dec 2014) (Pg.No.25)(Qn. No.7)**

Software quality assurance and testing Page 48