Computer Networks
Delhi Technological university
Transport Layer:
UDP and TCP
Instructor: Divya Sethia

Objective

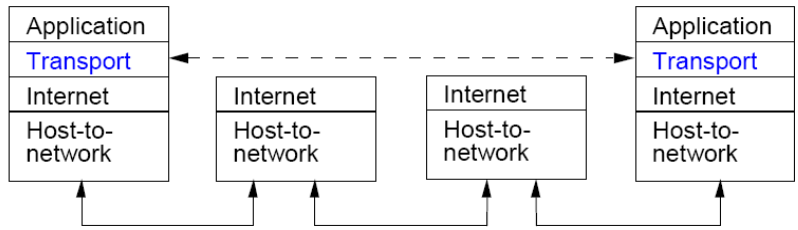
- Services
- UDP
- TCP

Divyashikha Sethia (DTU)

Services

-Provide a virtual end-to-end “message-pipe” for applications
-Two applications on different hosts can communicate messages to each other as if they are directly connected – the details of the underlying network are hidden.

- The transport layer is an end-to-end layer –nodes within the subnet do not participate in transport layer protocols – only the end hosts.
- As with other layers, transport layer protocols send data as a sequence of packets. In TCP/IP these packets are called ***segments.***



Divyashikha Sethia (DTU)

Issues

Two main issues addressed at the transport layer are

- Multiplexing
- Improving the service offered by network layer

Multiplexing

•The transport layer provides communication between two ***processes running on different*** hosts.
• Multiple processes communicating between two hosts – for example, there could be a FTP session and a Telnet session between the same two hosts.
•The transport layer provides a way to multiplex/demultiplex communication between various processes.
• The transport layer adds an address to each segment indicating the source and destination processes. These addresses need only be unique locally on a given host.
•In TCP/IP these transport layer addresses are called ***port-numbers.***
•In the Internet port numbers are 16-bits long. The port numbers between 0 and 1023 are reserved and called **well-known ports. These numbers are** assigned to well-known applications such as FTP, Telnet, and HTTP. For example HTTP is assigned well-known port number 80.

Multiplexing….

•***Applications based on client-server architecture –*** where one process, “the client,” requests services from a second process, “the server.”
•Example, a web browser implements the client side of HTTP and a web server implements the server side.
•The client initiates communication and the server responds.
•A client sending a request to a web server would use the destination port 80,and in IP would indicate the transport layer protocol is TCP (this is the transport layer protocol used by HTTP).
•The sending port number is picked from some other port numbers that are not being used for another session by the client.
•In this way if several different HTTP sessions are established between the two hosts, the sending port number distinguishes them.
•Thus, the 4-tuple: (sending port, sending IP address, destination port, destination IP address) uniquely identifies a process-to-process connection in the Internet.

Divyashikha Sethia (DTU)

Improving the Network layer service

- If Underlying network layer does not offer adequate service for an application, it is the job of the transport layer to improve this service. For example, suppose the network layer does not guarantee that every packet will arrive. Then the transport layer could provide a way to recover from lost packets and ensure that each packet is delivered correctly.

- As with the other layers, the service provided by the transport layer can have different characteristics. For example, it may be reliable or unreliable, connection-oriented or connectionless, etc. It could also try to guarantee a certain ***Quality of Service (QoS), i.e. a given data rate or delay.***

The service provided by the underlying network layer will determine:
1. How much work has to be performed at the transport layer to provide a certain service.
2. What types of service can even be provided

Divyashikha Sethia (DTU)

Transport layer services in the Internet

- The network layer in the Internet provides an unreliable, connectionless datagram service. IP tries to deliver each datagram to the destination, but makes no guarantees, that the data will arrive correctly, in order or even at all. This is sometimes described as a “best effort” service.

- In the Internet there are two transport layer protocols, each offering a different service.

i) UDP (User Datagram Protocol) provides an unreliable, connectionless transport layer protocol. The second transport layer protocol

ii) **TCP (Transport Control Protocol), provides a reliable, connection oriented** transport service. Some common applications and the corresponding transport layer protocol used are shown below:

Divyashikha Sethia (DTU)

Transport layer services in the Internet

Application protocol	Typical transport protocol
SMTP(e-mail)	TCP
DNS	UDP
HTTP (web browsing)	TCP
Internet Telephony	UDP
FTP	TCP
Telnet	TCP

9

Divyashikha Sethia (DTU)

User Datagram Protocol

•Transport-layer protocol (Layer 4)

•Connectionless service: provides application programs with ability to send and receive messages

•Allows multiple, application programs on a single machine to communicate concurrently

•No Acknowledgments, rate feedback or timers

•Same best-effort semantics as IP
- Message can be delayed, lost, or duplicated
- Messages can arrive out of order

•Application accepts full responsibility for errors

10

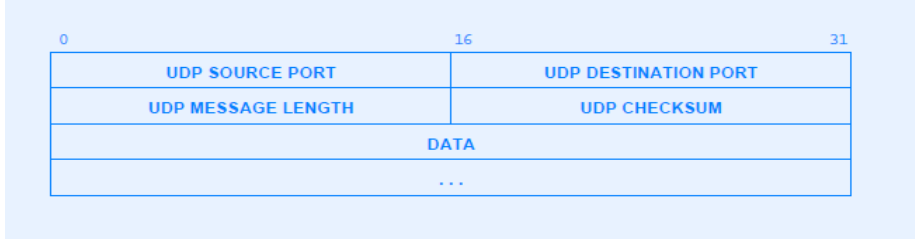
Divyashikha Sethia (DTU)

Added Benefit of UDP

The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines. It uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer.

Divyashikha Sethia (DTU)

UDP Message Format



•UDP port is a queue into which protocol software places arriving datagrams

•Source port is optional – when set implies where the replies must be sent to. If not used must be 0.
•Length – count in octet in UDP header + user data

•Checksum is optional and if it *contains zeroes, receiver does not* verify the checksum and omits it. – to reduce computational overhead.

• If the computed checksum is zero, then this field must be set to 0xFFFF

Divyashikha Sethia (DTU)

UDP Pseudo-Header

•Used when computing or verifying a checksum

•Temporarily prepended to UDP message for computing checksum

•Contains items from IP header

•Purpose is to guarantee that message arrived at correct destination

•Note: pseudo header is *not sent across Internet*

Divyashikha Sethia (DTU)

Components of Pseudo Header

• SOURCE ADDRESS and DESTINATION ADDRESS specify IP address of sending and receiving computers

•PROTO contains the Type from the IP datagram header (for UDP it is 17)

•To verify user extracts these fields from IP header, assemble them in pseudo-header format and recomputes checksum

Divyashikha Sethia (DTU)

Division Of Duties Between IP and UDP

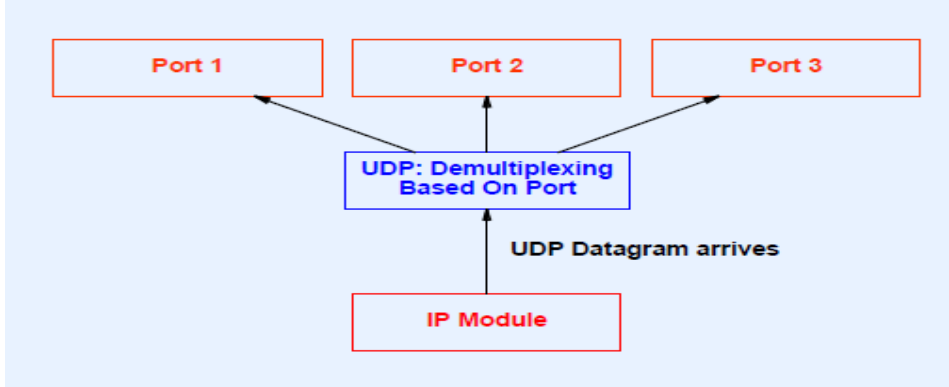
The IP layer is responsible for transferring data between a pair of hosts on an internet, while the UDP layer is responsible for differentiating among multiple sources or destinations within one host.

- IP header only identifies computer
- UDP header only identifies application programs

15

Divyashikha Sethia (DTU)

Demultiplexing based on port number



16

UDP Summary

•User Datagram Protocol (UDP) provides connectionless, best-effort message service

•UDP message encapsulated in IP datagram for delivery

•IP identifies destination computer; UDP identifies application on the destination computer

•UDP uses abstraction known as *protocol port numbers*

Examples Of Assigned UDP Port Numbers

Decimal	Keyword	UNIX Keyword	Description
0	-	-	Reserved
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
15	-	netstat	Network Status Program
17	QUOTE	qotd	Quote of the Day
19	CHARGEN	chargen	Character Generator
37	TIME	time	Time
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who Is
53	DOMAIN	nameserver	Domain Name Server
67	BOOTPS	bootps	BOOTP or DHCP Server
68	BOOTPC	bootpc	BOOTP or DHCP Client
69	TFTP	tftp	Trivial File Transfer
88	KERBEROS	kerberos	Kerberos Security Service
111	SUNRPC	sunrpc	Sun Remote Procedure Call
123	NTP	ntp	Network Time Protocol
161	-	snmp	Simple Network Management Protocol
162	-	snmp-trap	SNMP traps
512	-	biff	UNIX comsat
513	-	who	UNIX rwho Daemon
514	-	syslog	System Log
525	-	timed	Time Daemon

Well known UDP ports

Port	*Protocol*	*Description*
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

TCP (Transport Control Protocol)

•Stream orientation – transfer large volume of data – as stream of bits octets

•Virtual circuit connection – establish connection before transfer of data

•Buffered transfer – Efficient transfer collect sufficient data before transmitting

•Unstructured stream – no boundaries within data

•Full duplex connection

•Reliability

Divyashikha Sethia (DTU)

Providing Reliability

•Traditional technique**: Positive Acknowledgement with Retransmission (PAR)**

– Receiver sends *acknowledgement when data arrives*

– Sender starts timer whenever transmitting

– Sender retransmits if timer expires before acknowledgement arrives

Divyashikha Sethia (DTU)

Problem with simple PAR

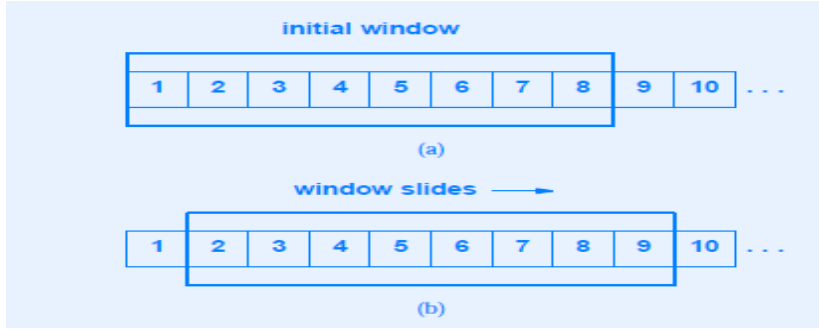
•A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.

•Problem is especially severe if network has long latency

Solution:
•Allow multiple packets to be outstanding at any time
•Still require acknowledgements and retransmission
•Known as sliding window

Divyashikha Sethia (DTU)

Sliding Windows



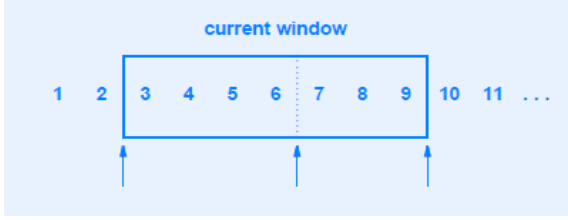
•Window size is fixed

•As acknowledgement arrives, window moves forward

•Keeps network completely saturated with packets, it obtains substantially higher throughput than

Divyashikha Sethia (DTU)

Sliding Window Used By TCP



•Measured in byte positions

•Bytes through 2 are acknowledged

•Bytes 3 through 6 not yet acknowledged

•Bytes 7 though 9 waiting to be sent

•Bytes above 9 lie outside the window and cannot be sent

Divyashikha Sethia (DTU)

Variable Windows Size and Flow Control

• TCP allows window size to vary over time

•Each ack that specifies how many octets have been received

•Also contains window advertisement that specifies how many octet of data receiver is ready to accept

Divyashikha Sethia (DTU)

TCP Ports, Connections, And Endpoints

•Endpoint of communication is application program

•TCP uses protocol port number to identify application

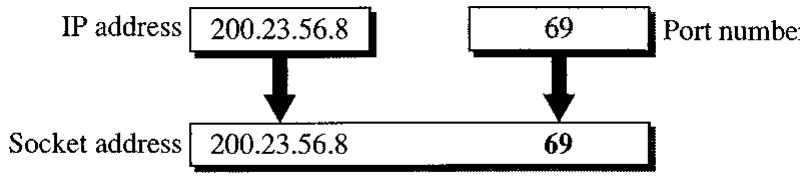
•UDP port is like a queue into which datagrams are places.

•TCP connection between two endpoints identified by four items

– Sender's IP address
– Sender's protocol port number
– Receivers IP address
– Receiver's protocol port number

•Given TCP port number can be shared by multiple connections on the same machine.

Socket

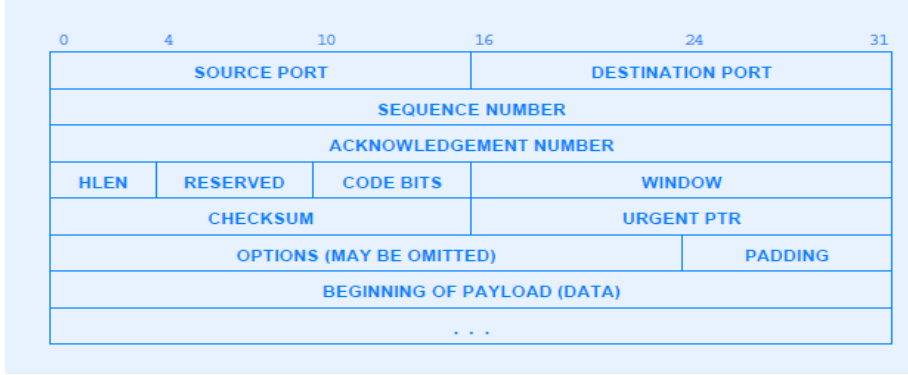


Passive And Active Opens

- Two sides of a connection
- One side waits for contact
  - A server program
  - Uses TCP's *passive open*
- One side initiates contact
  - A client program
  - Uses TCP's *active open*

Divyashikha Sethia (DTU)

TCP Segment Format



•Port numbers – identify application programs at ends of connection
•Sequence number – position in sender's byte stream of the data in the segment
•Ack number – octet number that source expects to receive next
•Hlen – Header length contains integer that specifies length of segment measured in 32-bit multiples (required since OPTION field is optional)

29

Divyashikha Sethia (DTU)

Flow Control and TCP Window

•Receiver controls flow by telling sender size of currently available buffer measured in bytes

•Called *window advertisement – sent in all types of segments*

•Each segment, including data segments, specifies size of window *beyond acknowledged byte*
•Window size may be zero (receiver cannot accept additional data at present). ACK with WIN = 0 is sent. Later when things are find sends same ACK no with a updated WIN value.
•Receiver can send additional acknowledgement later when buffer space becomes available
•Congestion – intermediate machines become overloaded.
•TCP uses sliding window mechanism to solve end-to-end flow control problem – indirectly helps recover from congestion

30

Divyashikha Sethia (DTU)

Code Bits In The TCP Segment Header

TCP segment can carry – data, ack or other depending on the purpose specifies by CODE Bits

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

31

Divyashikha Sethia (DTU)

Out of Band Data

•Send data out-of-band without waiting for program on the other end to consume octets already in the stream. –eg interrupts or aborts

•Data specified as urgent set URG code bit and URGENT POINTER field is valid.
- indicate a byte offset from the current sequence number at which urgent data are to be found
- used for Interrupts to be sent and sent immediately.

•PSH bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received. For improving efficiency

•SYN bit is used to establish connections

•FIN bit is used to release a connection

32

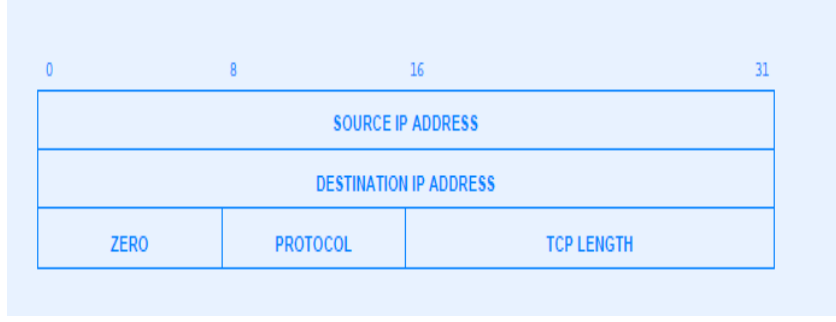
Divyashikha Sethia (DTU)

TCP Checksum

- Covers entire segment (header plus data)
- Required (unlike UDP)
- Pseudo header included in computation as with UDP

Divyashikha Sethia (DTU)

TCP Pseudo Header



Divyashikha Sethia (DTU)

TCP options

•Options field provides way to add extra facilities not covered by regular header.

•Most important option is maximum TCP payload host is willing to accept.

- During connection setup, each side can announce its maximum and see its partner's.
- If a host does not use this option, it defaults to a 536-byte payload
- Hosts are required to accept TCP segments of 536 + 20 = 556 bytes.
- The maximum segment size in the two directions need not be the same

Divyashikha Sethia (DTU)

Other TCP options

•**Window scale option** was proposed, allowing the sender and receiver to negotiate a window scale factor.

- For more efficient use of high bandwidth networks, larger TCP window size may be used. (max window size limited by the 16 bit window field in TCP header)

-a Window scale option allows sender and receiver to negotiate a window scale factor which allows both sides to shift the Window size field up to 14 bits to the left, thus allowing windows of up to 2^{30} bytes

Other TCP options….

•**Selective Repeat option**:
- If the receiver gets one bad segment and then a large number of good ones, the normal TCP protocol will eventually time out and retransmit all the unacknowledged segments, including all those that were received correctly (i.e., the go back n protocol).

-RFC 1106 introduced NAKs to allow the receiver to ask for a specific segment (or segments).

-After it gets these, it can acknowledge all the buffered data, thus reducing the amount of data retransmitted

Further read from Tanenbaum…

TCP Retransmission

•**Acknowledgements**:
- Receiver acknowledges the longest contiguous prefix of the stream that has been received correctly.
- Specifies seq no one greater than the highest octet position in the prefix received.

• Retransmission designed for Internet environment
– Delays on one connection vary over time
– Delays vary widely between connections

•Fixed value for timeout will fail
– Waiting too long introduces unnecessary delay
– Not waiting long enough wastes network bandwidth with unnecessary retransmission

•Retransmission strategy must be adaptive

Adaptive Retransmission

•TCP keeps *estimate of round-trip time (RTT)* on each connection

•Round-trip estimate derived from observed delay between sending segment and receiving acknowledgement

•Timeout for retransmission based on current round-trip estimate

Difficulty with Adaptive Retransmission

•**Acknowledgment ambiguity** : Segment is retransmitted. Since both carry same seq no, difficult to determine if acknowledgement is for original/ retransmitted segment.

•Assumption Ack belongs to original – makes RTT and hence timeout longer

- Next time TCP sends segment , the larger RTT results in longer timeout
- Possible that ack is result of one or more retransmissions
- Again associate RTT with the original results in RTT growing further

Difficulty with Adaptive Retransmission

•Assumption Ack belongs to most recent retransmission – makes RTT and hence timeout smaller

- If end to end delay increases , timer expires due to small RTT and retransmits
- First ack associated again with retransmission
- Further RTT reduced
- Stable position RTT observed to be smaller than correct RTT

Partridge / Karn Scheme†

•Solves the problem of associating ACKs with correct transmission

•Specifies ignoring round-trip time samples that correspond to retransmissions

•Separates timeout from round-trip estimate for retransmitted packets

•†Also called *Karn's Algorithm*

Divyashikha Sethia (DTU)

Partridge / Karn Scheme†

•Doubles retransmission timer value for each retransmission without changing round-trip estimate

•Resets retransmission timer to be function of round-trip estimate when ACK arrives for non-retransmitted segment

Divyashikha Sethia (DTU)

Flow control and congenstion

•Congestion occurs when there is severe delay caused by overloaded intermediate routers in the network.

•Datagrams enqeued for transmission and if queue exceed drop off.

•Retransmissions further aggravate congestion due to increased delay – ***congestion collapse***

•Routers watch queue lengths and use ICMP source quench to inform hosts of congestion

•TCP can help reduce congestion by reducing transmission rate on delay using two techniques:
- Slow-start - Multiplicative decrease

• Sender chooses effective window smaller than receiver's advertised window if congestion detected

Response to Congestion

•Besides receiver advertised window , TCP maintains congestion window

•Allowed_window = min (recevier_adver, congestion_win)

Multiplicative Decrease Congestion Recovery

•In steady state (no congestion), the congestion window is equal to the receiver's window

•When segment lost (retransmission timer expires), reduce congestion window by half

•Never reduce congestion window to less than one maximum sized segment

•Reducing congestion window reduces traffic injected by TCP into connection.

Divyashikha Sethia (DTU)

Recover when congestion end

•Reverse multiplicative decrease and double congestion when traffic is normal?

•Results in unstable system between no traffic and congestion

47

Divyashikha Sethia (DTU)

Slow Start

•Used when starting traffic or when recovering from congestion

•Self-clocking startup to increase transmission rate rapidly as long as no packets are lost

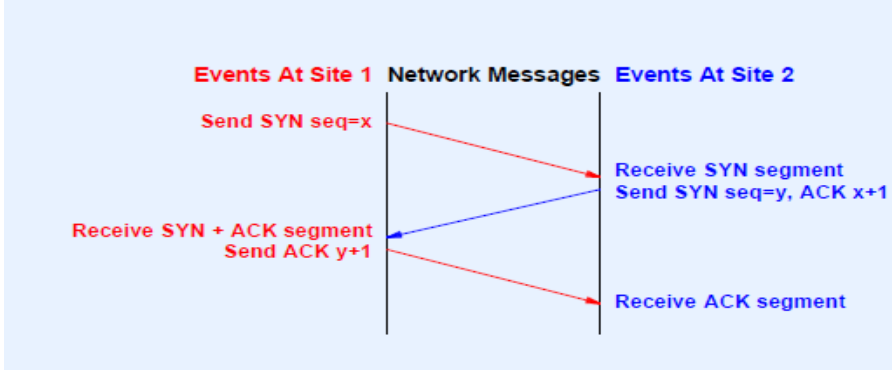
•When starting traffic, initialize the congestion window to the size of a single maximum sized segment

•Increase congestion window by size of one segment each time an ACK arrives without retransmission

48

Divyashikha Sethia (DTU)

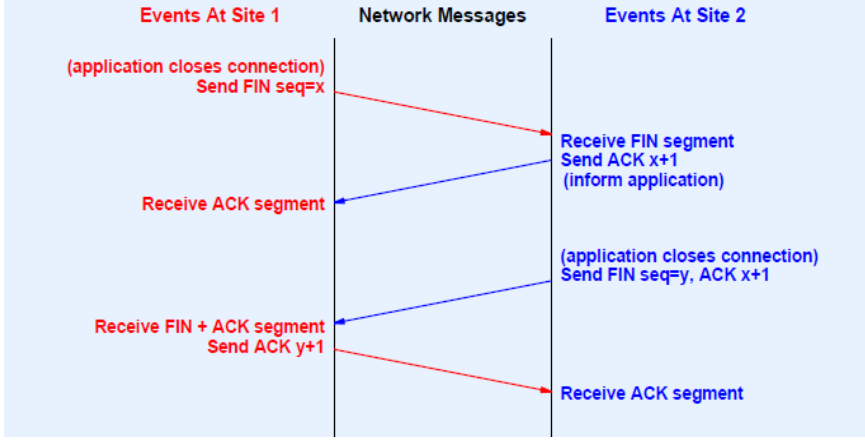
3 Way Handshake



- First segment of handshake sets SYN bit in code field
- Second segment set both SYN and ACK bit in code field
- Site2 : Passive Site1: Active

Divyashikha Sethia (DTU)

3-Way Handshake For Connection shutdown



Connection Establishment

•The initial sequence number on a connection is not 0.

•A clock-based scheme is used, with a clock tick every 4 μsec.

•For additional safety, when a host crashes, it may not reboot for the maximum packet lifetime to make sure that no packets from previous connections are still roaming around the Internet somewhere.

SYN Flooding Attack

• Malicious attacker sends large number of SYN segments by faking source IP addresses in the datagrams

•Server assumes different clients are issuing active open and allocates resources

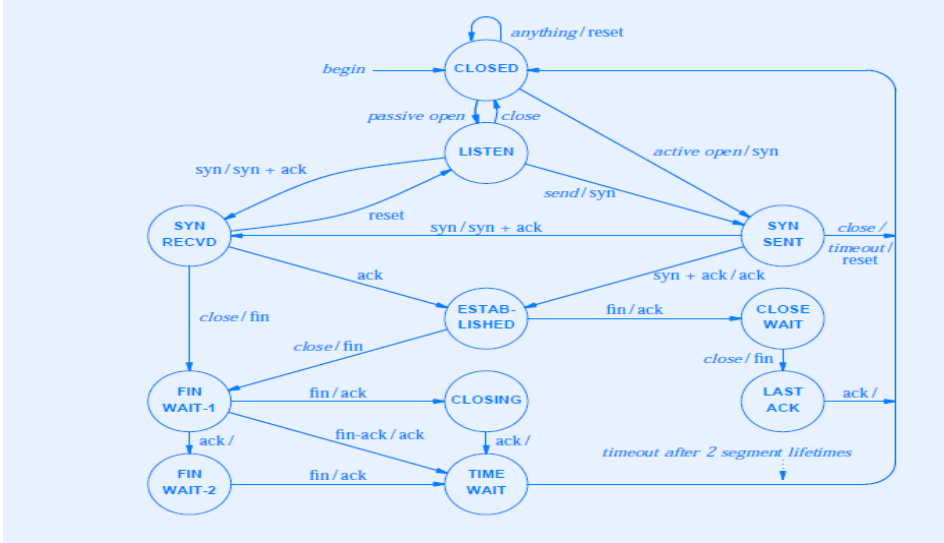
•TCP server sends SYN + ACK segments to fake client

•Server may run out of resource and eventually crash

•Security attack known as **denial-of-service attack.**

Divyashikha Sethia (DTU)

TCP Finite State Machine



•Label indicate the input that caused transition followed by output if any

Divyashikha Sethia (DTU)

TCP Finite State Machine

•TIMED WAIT state
-Handles problem with unreliable delivery
-Maximum Segment lifetime (MSL) max time an old segment can remain alive in internet
-To avoid having segment from previous connection interfere with current one, moves to this state
-Remains in this state for twice MSL before deleting its record of the connection.
-If any duplicate segments arrive for connection during timeout interval they are rejected
-However if it receives any last ack that was lost TCP acks valid segments and restarts the timer.

Divyashikha Sethia (DTU)

TCP Urgent Data

- Segment with urgent bit set contains pointer to last octet of urgent data
- Urgent data occupies part of normal sequence space
- Urgent data can be retransmitted
- Receiving TCP should deliver urgent data to application "immediately" upon receipt

55

Divyashikha Sethia (DTU)

Comparison of UDP and TCP

UDP	TCP
Between apps. and IP	Between apps. and IP
Packets called datagrams	Packets called segments
Unreliable	Reliable
Checksum optional	Checksum required
Connectionless	Connection-oriented
No flow control	Flow control
No congestion control	Congestion control

56

Divyashikha Sethia (DTU)

TCP Summary

•Major transport service in the Internet

•Connection oriented

•Provides end-to-end reliability

•Uses adaptive retransmission

•Includes facilities for flow control and congestion avoidance

•Uses 3-way handshake for connection startup and shutdown

Divyashikha Sethia (DTU)

Remote Procedure Call

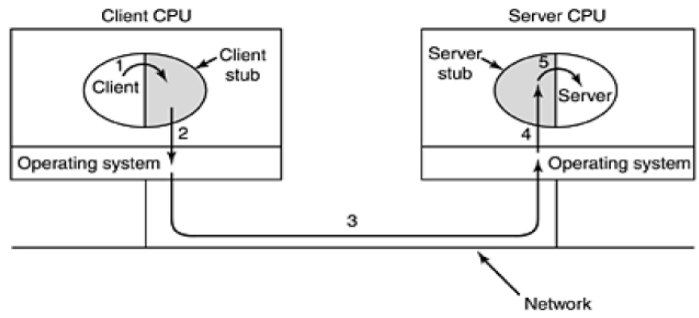
•sending a message to a remote host and getting a reply back is a lot like making a function call in a programming language. In both cases you start with one or more parameters and you get back a result. This observation has led people to try to arrange request-reply interactions on networks to be cast in the form of procedure calls

• allowing programs to call procedures located on remote hosts. When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the programmer. This technique is known as **RPC**

•calling procedure is known as the client and the called procedure is known as the server,

•client program must be bound with a small library procedure, called the **client stub, that represents the server procedure in the** client's address space. Similarly, the server is bound with a procedure called the **server stub.** These procedures hide the fact that the procedure call from the client to the server is not local.

Remote Procedure Call



Steps

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.

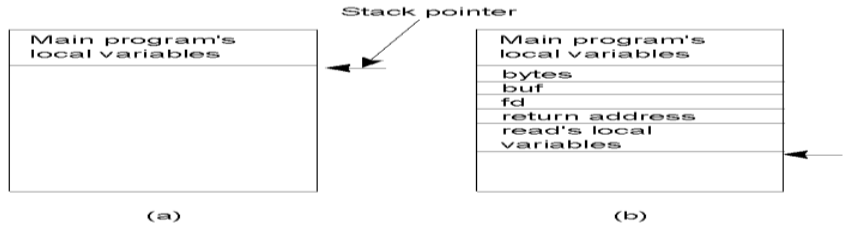
Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called **marshaling.**

Step 3 is the kernel sending the message from the client machine to the server machine.

Step 4 is the kernel passing the incoming packet to the server stub.

Step 5 is the server stub calling the server procedure with the unmarshaled parameters. The reply traces the same path in the other direction.

Conventional Procedure Call



Count = read (fd, buf, bytes)

a) Parameter passing in local procedure call: stack before call to read

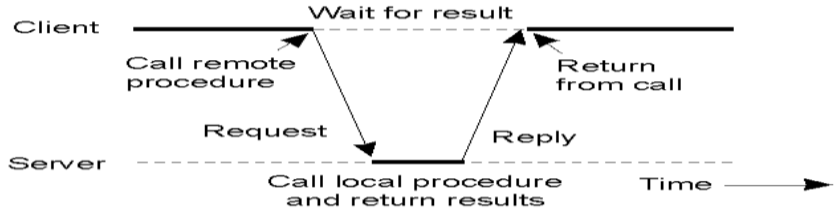
b) The stack while the called procedure is active

- caller pushes parameters onto stack in order, last one first so that first param is accessible first
- Procedure after computation puts return value in a register, removes return address, and transfers control back to caller.
- caller then removes parameters from stack, returning stack to original state

61

Divyashikha Sethia (DTU)

Client and Server Stubs



•Read does a system call by pushing the parameters onto the stack

•For a read implemented as RPC (e.g., one that will run on the file server's machine), a different version of read, called a client stub, is put into the library.

•It does a call to the local operating system.

•Instead of OS giving data, it packs the parameters into a message and requests that message to be sent to the server.

•After call to send, client stub calls receive, blocking until reply comes back.

62

RPC –server side

•Server's OS passes it up to server stub (equivalent of a client stub) which transforms requests coming in over network into local procedure calls.

•Server stub calls receive and remains blocked waiting for incoming messages.

•Server stub unpacks parameters from message and then calls server procedure usual way using the stack as if it is being called directly by client, the parameters and return address are all on stack.

•After work completion server returns result to caller
- Eg case of read, server will fill buffer, pointed to by the second parameter, with the data. This buffer will be internal to the server stub.

- When server stub gets control back after the call has completed, it packs result (the buffer) in a message and calls send to return it to the client.

•Server stub does a call to receive again, to wait for next incoming request.

RPC client receiving results

•Client's OS sees that it is addressed to client process (client stub)

•Message is copied to waiting buffer and client process unblocked.

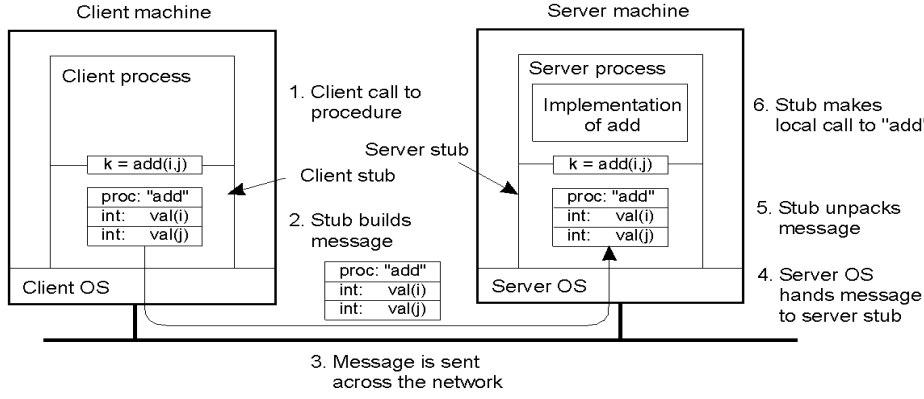
•Client stub inspects message, unpacks result, copies it to its caller, and returns the usual way.

•Caller gets control following the call to read, and gets data available unaware of fact that the work was done remotely

Parameter Passing

- Passing Value Parameters
- Passing Reference Parameters
- Parameter Specification and Stub Generation

Passing Value Parameters (1)



Steps involved in doing remote computation through RPC

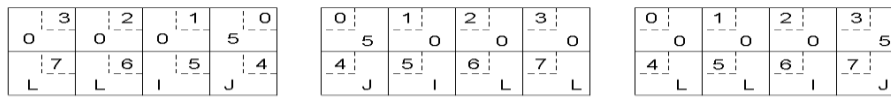
Packing parameters into a message is called **parameter marshaling**.

Passing Value Parameters(2) Issues

•Different representation for numbers, characters, and other data items. Eg: EBCDIC character code, whereas IBM personal computers use ASCII

•Incorrect interpretation of characters
Format of code: Intel Pentium, number their bytes from right to left, whereas others, such as the Sun SPARC, number them the other way.

Passing Value Parameters (3)



(a) (b) (c)

Procedure with two parameters, an integer and a four-character String (5, JILL)

Each parameter requires one 32-bit word (each box is a byte)

a) Original message on the Pentium

Message transferred byte for byte (first byte sent is first received)

Intel Pentium number their bytes from right to left – little endian

b) The message after receipt on the SPARC

Sparc numbers bytes from left to right – big endian

c) The message after being inverted. The little numbers in boxes indicate the address of each byte 5 interpreted as 83,886,080 (5×2^{24})

=>integer is 5 and the string is "LLIJ“ since integers are reversed by different byte ordering, but strings are not.

Passing References (1)

•Pointer is meaningful only within address space of process in which it is being used
eg: Address 1000 on server might be in middle of program text

•Solution – avoid pointers call-by-reference replaced by copy/restore.
- If client stub knows parameter point to array of characters and length of array - copy array into message and send it to server
- server stub then calls the server with a pointer to this array
- Changes made by server using the pointer (e.g., storing data into it) directly affects message buffer inside server stub
- When finished original message can be sent back to client stub, which copies it back to client
- Optimization: if buffer an input parameter (eg write) then copying can be eliminated since it does not have to be sent back.

Passing References (2)

•pointer to an arbitrary data structure such as a complex graph:

- pass pointer to server stub and special code in server to handle such pointers
eg: send back request to client to provide referenced data

Issues

• Pointers
  - standard calling sequence of call-by-reference has been replaced by copyrestore
  - complex structure like graph
  - complex data structure size

• Different representations on machines

•Global variables

•UDP used for RPC. If packet size large then set TCP connection

Advantages

•Like the common communications between the portions of an application, the development of the procedures for the remote calls is quite general.
•The code re-writing / re-developing effort is minimized.

Disadvantage

•RPC is only interaction based. This does not offer any flexibility in terms of hardware architecture.

•RPC is not a standard

References

• chapter 12, 13: TCP/IP volume 1 by Douglas Comer, Fourth Edition

•http://www.networkcomputing.com/netdesign/ip101.html

http://www.tcpipguide.com/free/t_toc.htm

THANKS