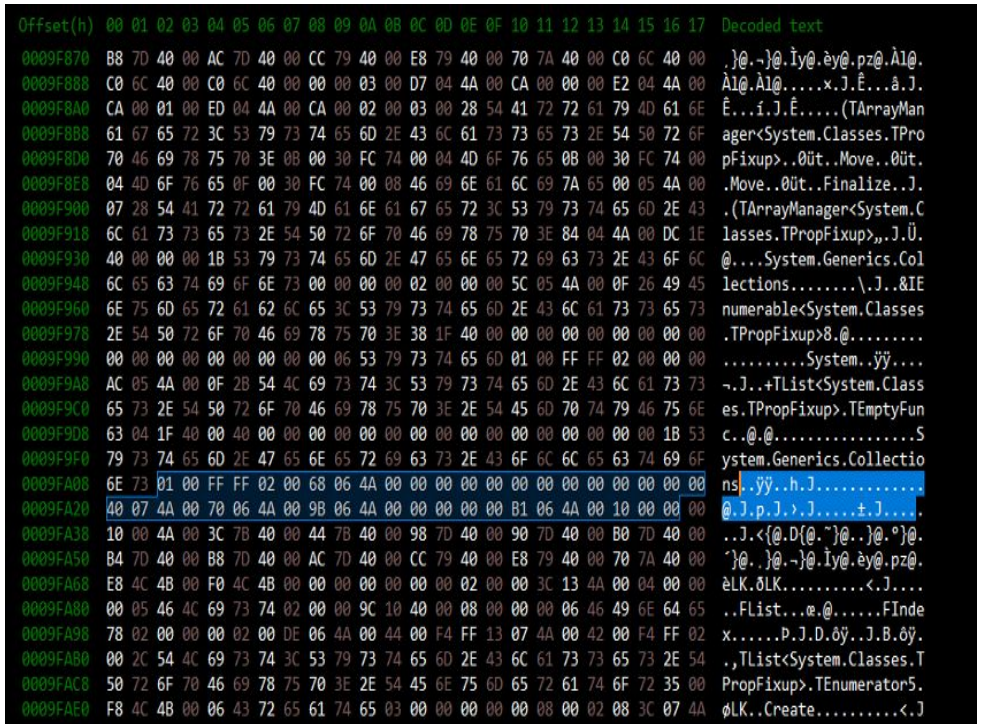
Byte Ordering

UNIT - 2

BYTE ORDERING



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	Decoded text
0009F870	B8	7D	40	00	AC	7D	40	00	CC	79	40	00	E8	79	40	00	70	7A	40	00	C0	6C	40	00	¸}@.¬}@.Ìy@.èy@.pz@.Àl@.
0009F888	C0	6C	40	00	C0	6C	40	00	00	00	03	00	D7	04	4A	00	CA	00	00	00	E2	04	4A	00	Àl@.Àl@.....×.J.Ê...â.J.
0009F8A0	CA	00	01	00	ED	04	4A	00	CA	00	02	00	03	00	28	54	41	72	72	61	79	4D	61	6E	Ê...í.J.Ê.....(TArrayMan
0009F8B8	61	67	65	72	3C	53	79	73	74	65	6D	2E	43	6C	61	73	73	65	73	2E	54	50	72	6F	ager<System.Classes.TPro
0009F8D0	70	46	69	78	75	70	3E	0B	00	30	FC	74	00	04	4D	6F	76	65	0B	00	30	FC	74	00	pFixup>..0üt..Move..0üt.
0009F8E8	04	4D	6F	76	65	0F	00	30	FC	74	00	08	46	69	6E	61	6C	69	7A	65	00	05	4A	00	.Move..0üt..Finalize..J.
0009F900	07	28	54	41	72	72	61	79	4D	61	6E	61	67	65	72	3C	53	79	73	74	65	6D	2E	43	.(TArrayManager<System.C
0009F918	6C	61	73	73	65	73	2E	54	50	72	6F	70	46	69	78	75	70	3E	84	04	4A	00	DC	1E	lasses.TPropFixup>„„.J.Ü.
0009F930	40	00	00	00	1B	53	79	73	74	65	6D	2E	47	65	6E	65	72	69	63	73	2E	43	6F	6C	@....System.Generics.Col
0009F948	6C	65	63	74	69	6F	6E	73	00	00	00	00	02	00	00	00	5C	05	4A	00	0F	26	49	45	lections.....\.J..&IE
0009F960	6E	75	6D	65	72	61	62	6C	65	3C	53	79	73	74	65	6D	2E	43	6C	61	73	73	65	73	numerable<System.Classes
0009F978	2E	54	50	72	6F	70	46	69	78	75	70	3E	38	1F	40	00	00	00	00	00	00	00	00	00	.TPropFixup>8.@.....
0009F990	00	00	00	00	00	00	00	00	00	06	53	79	73	74	65	6D	01	00	FF	FF	02	00	00	00	.....System..ÿÿ....
0009F9A8	AC	05	4A	00	0F	2B	54	4C	69	73	74	3C	53	79	73	74	65	6D	2E	43	6C	61	73	73	¬.J..+TList<System.Class
0009F9C0	65	73	2E	54	50	72	6F	70	46	69	78	75	70	3E	2E	54	45	6D	70	74	79	46	75	6E	es.TPropFixup>.TEmptyFun
0009F9D8	63	04	1F	40	00	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1B	53	c..@.@.............S
0009F9F0	79	73	74	65	6D	2E	47	65	6E	65	72	69	63	73	2E	43	6F	6C	6C	65	63	74	69	6F	ystem.Generics.Collectio
0009FA08	6E	73	01	00	FF	FF	02	00	68	06	4A	00	00	00	00	00	00	00	00	00	00	00	00	00	ns..ÿÿ..h.J.....
0009FA20	40	07	4A	00	70	06	4A	00	9B	06	4A	00	00	00	00	00	B1	06	4A	00	10	00	00	00	@.J.p.J.›.J.....±.J.....
0009FA38	10	00	4A	00	3C	7B	40	00	44	7B	40	00	98	7D	40	00	90	7D	40	00	B0	7D	40	00	..J.<{@.D{@.˜}@..}@.°}@.
0009FA50	B4	7D	40	00	B8	7D	40	00	AC	7D	40	00	CC	79	40	00	E8	79	40	00	70	7A	40	00	´}@..¸}@.¬}@.Ìy@.èy@.pz@.
0009FA68	E8	4C	4B	00	F0	4C	4B	00	00	00	00	00	00	00	02	00	00	3C	13	4A	00	04	00	00	èLK.ðLK.........<.J....
0009FA80	00	05	46	4C	69	73	74	02	00	00	9C	10	40	00	08	00	00	00	06	46	49	6E	64	65	..FList...œ.@.....FInde
0009FA98	78	02	00	00	00	02	00	DE	06	4A	00	44	00	F4	FF	13	07	4A	00	42	00	F4	FF	02	x.....Þ.J.D.ôÿ..J.B.ôÿ.
0009FAB0	00	2C	54	4C	69	73	74	3C	53	79	73	74	65	6D	2E	43	6C	61	73	73	65	73	2E	54	.,TList<System.Classes.T
0009FAC8	50	72	6F	70	46	69	78	75	70	3E	2E	54	45	6E	75	6D	65	72	61	74	6F	72	35	00	PropFixup>.TEnumerator5.
0009FAE0	F8	4C	4B	00	06	43	72	65	61	74	65	03	00	00	00	00	00	08	00	02	08	3C	07	4A	øLK..Create.....<.J

- An arrangement of bytes when data is transmitted over the network is called byte ordering.
- Different computers will use different byte ordering.
- When communication taking place between two machines byte ordering should not make discomfort.
- Generally an Internet protocol will specify a common form to allow different machines byte ordering. TCP/IP is the Internet Protocol in use.
- Two ways to store bytes : Big endian and little endian

Big-endian

- High order byte is stored on starting address and low order byte is stored on next address

Little-endian

- Low order byte is stored on starting address and high order byte is stored on next address

Byte ordering functions

Name of the function	Description
htons()	Host to network short
htonl()	Host to network long
ntohs()	Network to host short
ntohl()	Network to host long

Byte Ordering Functions

- Special functions are applied through routines to convert host's internal byte order representation to network byte order.

unsigned short htons()

- This function converts 16-bit (2-byte) data from host byte order to network byte order.

unsigned long htonl()

- This function converts 32-bit (4-byte) data from host byte order to network byte order.

unsigned short ntohs()

- This function converts 16-bit (2-byte) data from network byte order to host byte order.

unsigned long ntohl()

- This function converts 32-bit (4- byte) data from network byte order to host byte order.

SYSTEM CALLS & SOCKETS

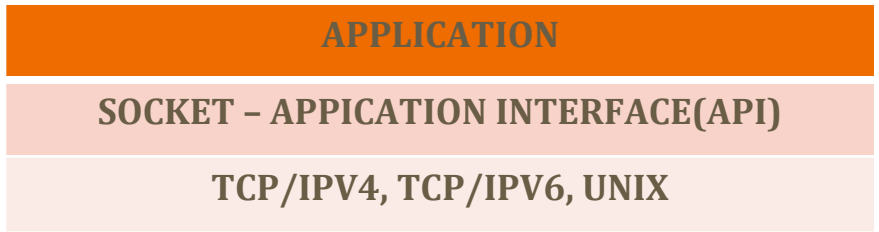
System calls

- System Calls – An interface between process and operating systems.

It provides

- The services of the operating system to the user programs via Application Program Interface(API).
- An interface to allow user-level processes to request services of the operating system.
- System calls are the only entry points into the kernel system.

SYSTEM CALLS & SOCKETS



Sockets

- Socket is an interface between applications and the network services provided by operating systems.
- Applications use sockets to send and receive the data.
- Socket provides IP address and port address

Socket Descriptors

- To perform file I/O, **file descriptor** is used.
- To perform network I/O, **socket descriptor** is used
- Each active socket is identified by its socket descriptor.
- The data type of a socket descriptor is **SOCKET**.
- Hack into the header file winsock2.h:
- **typedef u_int SOCKET**; /* u_int is defined as unsigned int */
- In UNIX systems, socket is just a special file, and socket descriptors are kept in the file descriptor table.
- The Windows operating system keeps a separate table of socket descriptors (named **socket descriptor table**, or SDT) for each process.

Socket creation

- The socket API contains a function socket() that can be called to create a socket.

```
#include <winsock2.h>
…
SOCKET s;
…
s = socket(AF_INET, SOCK_DGRAM,0);
```

The socket is not usable yet. We need to specify more information before using it for data transmission.

Types of Sockets

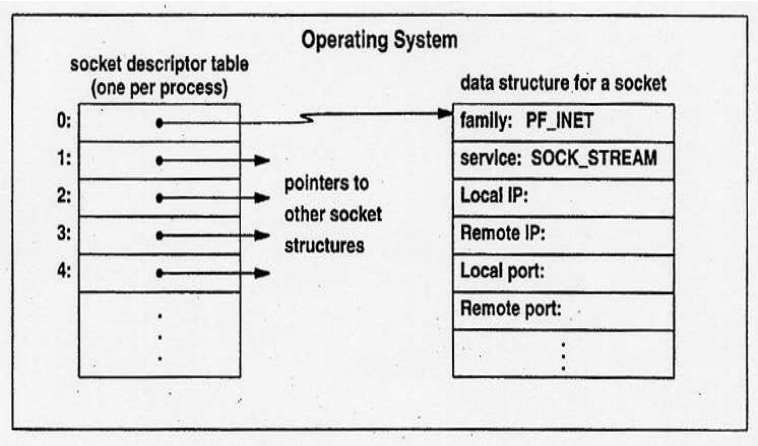
Under protocol family AF_INET

- **Stream socket**
  - Uses TCP for connection-oriented reliable communication
  - Identified by SOCK_STREAM
  - *s = socket(AF_INET, SOCK_STREAM, 0) ;*

- **Datagram socket**
  - Uses UDP for connectionless communication
  - Identified by SOCK_DGRAM
  - *s = socket(AF_INET, SOCK_DGRAM, 0) ;*

- **RAW socket**
  - Uses IP directly
  - Identified by SOCK_RAW

System Data Structures for Sockets



Data Structure for Sockets

- When an application process calls socket(), the operating system allocates a new data structure to hold the information needed for communication, and fills in a new entry in the process's socket descriptor table (**SDT**) with a pointer to the data structure.
- A process may use multiple sockets at the same time. The socket descriptor table is used to manage the sockets for this process.
- Different processes use different SDTs.
- The internal data structure for a socket contains many fields, but the system leaves most of them unfilled. The application must make additional procedure calls to fill in the socket data structure before the socket can be used.
- The socket is used for data communication between two processes (which may locate at different machines). So the socket data structure should at least contain the address information, e.g., **IP addresses, port numbers**, etc.

Functions used in client program

- socket()- create the socket descriptor
- connect()- connect to the remote server
- read(),write()- communicate with the server
- close()- end communication by closing socket descriptor

Socket()

- int socket(int domain, int type, int protocol)
- Returns a descriptor (or handle) for the socket

Domain

- protocol family PF_INET for the Internet

Type

- semantics of the communication
- SOCK_STREAM: Connection oriented
- SOCK_DGRAM: Connectionless

Protocol

- specific protocol UNSPEC: unspecified (PF_INET and SOCK_STREAM already implies TCP)

  E.g., TCP: sd = socket(PF_INET, SOCK_STREAM, 0);

  E.g., UDP: sd = socket(PF_INET, SOCK_DGRAM, 0);

Connect()

- int connect(int sockfd, struct sockaddr *server_address, socketlen_t addrlen)

Arguments

- socket descriptor, server address, and address size

- Remote address and port are in struct sockaddr

- Returns 0 on success, and -1 if an error occurs

Read(), Write() and Close()

Sending data

- write(int sockfd, void *buf, size_t len)
- Arguments: socket descriptor, pointer to buffer of data, and length of the buffer
- Returns the number of characters written, and -1 on error

Receiving data

- read(int sockfd, void *buf, size_t len)
- Arguments: socket descriptor, pointer to buffer to place the data, size of the buffer
- Returns the number of characters read (where 0 implies “end of file”), and -1 on error

Closing the socket

- int close(int sockfd)

Functions used in server program

- socket()- create the socket descriptor
- bind()- associate the local address
- listen()- wait for incoming connections from clients
- accept()- accept incoming connection
- read(),write()- communicate with client
- close()- close the socket descriptor

Bind()

- Bind socket to the local address and port
- int bind (int sockfd, struct sockaddr *my_addr, socklen_t addrlen)
- Arguments: socket descriptor, server address, address length.
- Returns 0 on success, and -1 if an error occurs.

Listen()

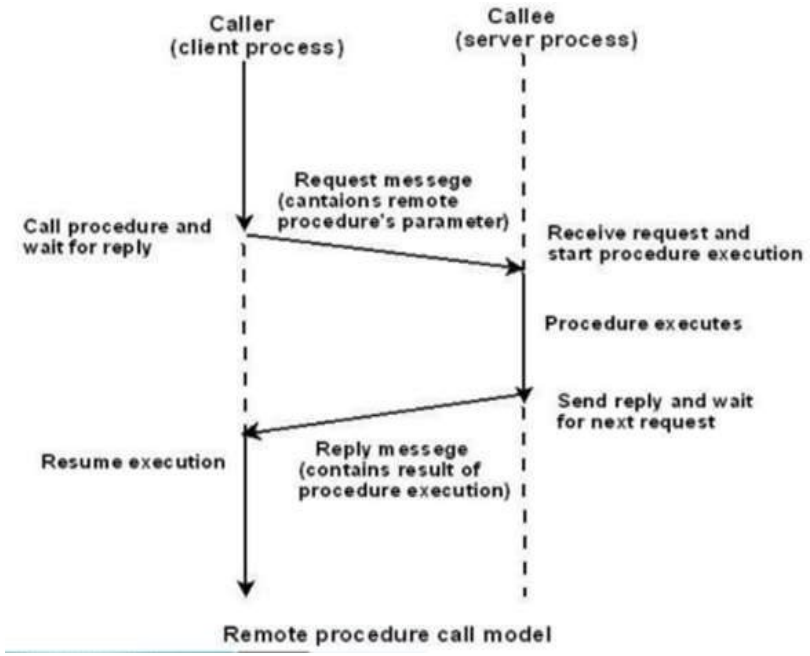
- Define the number of pending connections
  - int listen(int sockfd, int backlog)
    - Arguments: socket descriptor and acceptable backlog
    - Returns 0 on success, and -1 on error

Accept()

- int accept(int sockfd, struct sockaddr *addr, socketlen_t *addrlen)
- Arguments: socket descriptor, structure that will provide client address and port, and length of the structure.
- Returns descriptor for a new socket for this connection

- What happens if no clients are around?
  - The accept() call blocks waiting for a client
- What happens if too many clients are around?
  - Some connection requests don't get through
  - But, that's okay, because the Internet makes no promises

RPC – REMOTE PROCEDURE CALL

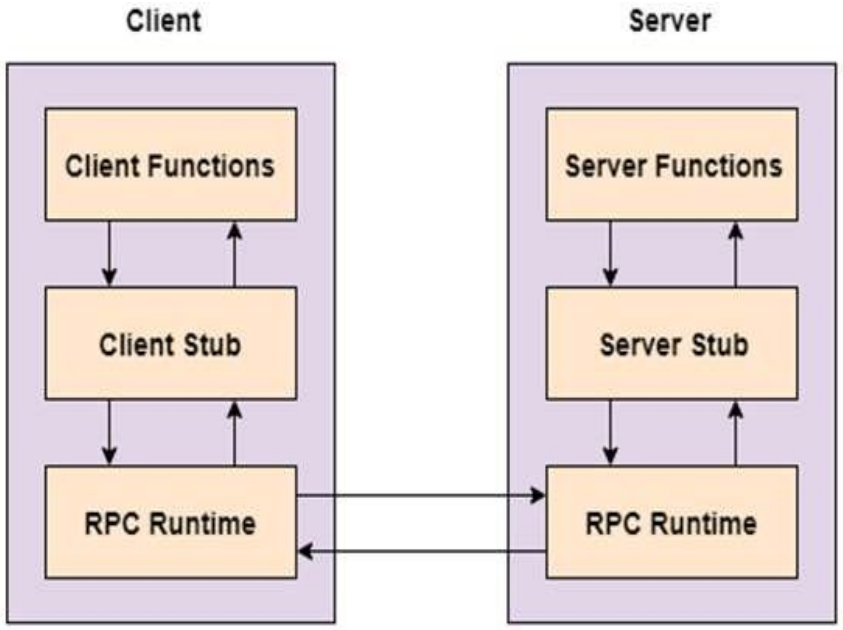


- A ***remote procedure call*** is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.
- RPC allows programs to call the procedure which is located on the other machines.
- Message passing is not visible to the programmer , so it is called as ***Remote Procedure call (RPC)***.
- RPC enables a procedure call that does not reside in the address space of the calling process.
- In RPC, the caller and the callee has *disjoint address* space, hence there is *no access to data and variables in the callers environment.*
- RPC performs a ***message passing scheme*** for information exchange between the caller and the callee process.

RPC Model(Remote Procedure Call)

- A client has a request message that the RPC translates and sends to the server.
- This request may be a procedure or a function call to a remote server.
- When the server receives the request, it sends the required response back to the client.
- The client is blocked while the server is processing the call and only resumed execution after the server is finished.

Client Server RPC Model



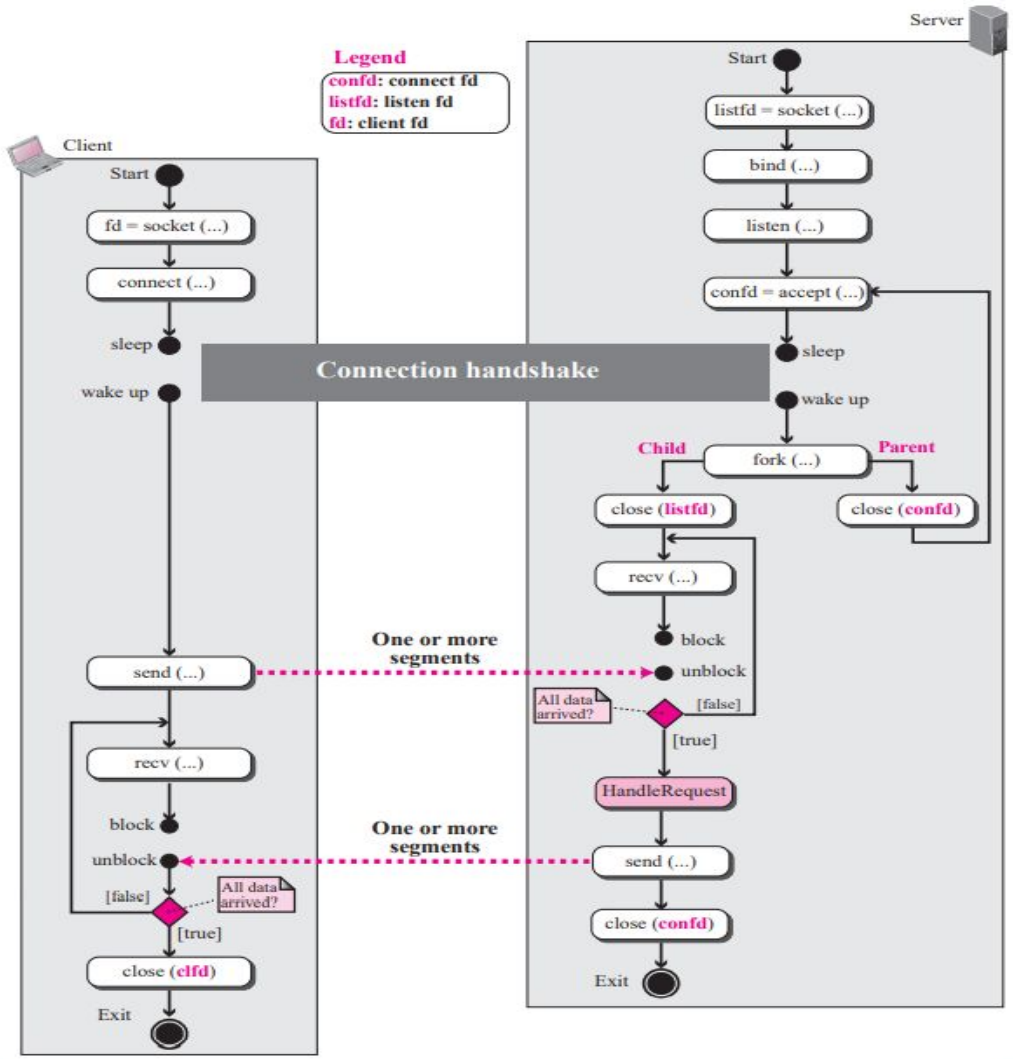
Sequence of events in a RPC

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

RPC Features

- Remote procedure calls support process oriented and thread oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimum in remote procedure calls.
- Remote procedure calls can be used in distributed environment as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.
- Ease of use, efficiency.

TCP CLIENT –SERVER Communication



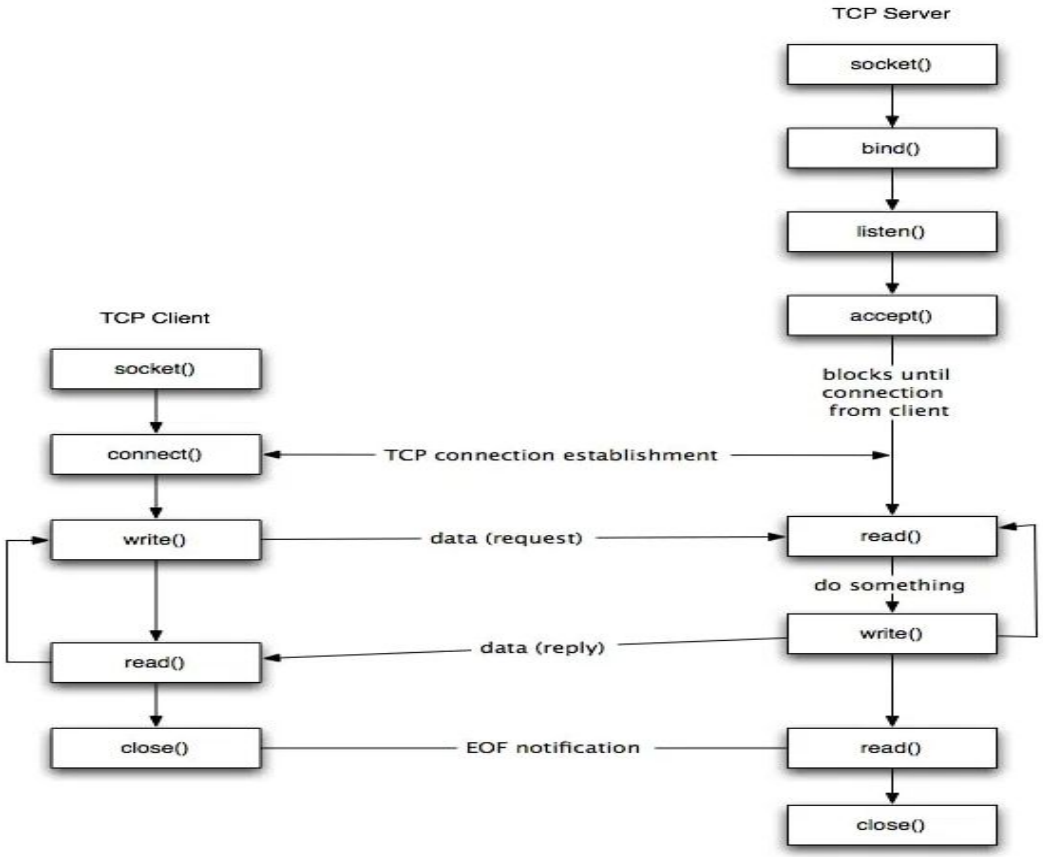
Flow diagram for connection-oriented, concurrent communication

Communication Using TCP

Accept function

- This function is a blocking function; when it is called, it is blocked until the TCP receives a connection request (SYN segment) from a client.
- The accept function then is unblocked and creates a new socket called the connect socket that includes the socket address of the client that sent the SYN segment.
- After the accept function is unblocked, the server knows that a client needs its service.
- To provide concurrency, the server process (parent process) calls the fork function.
- This function creates a new process (child process), which is exactly the same as the parent process.
- After calling the fork function, the two processes are running concurrently, but each can do different things

Sequence of function for client-server communication



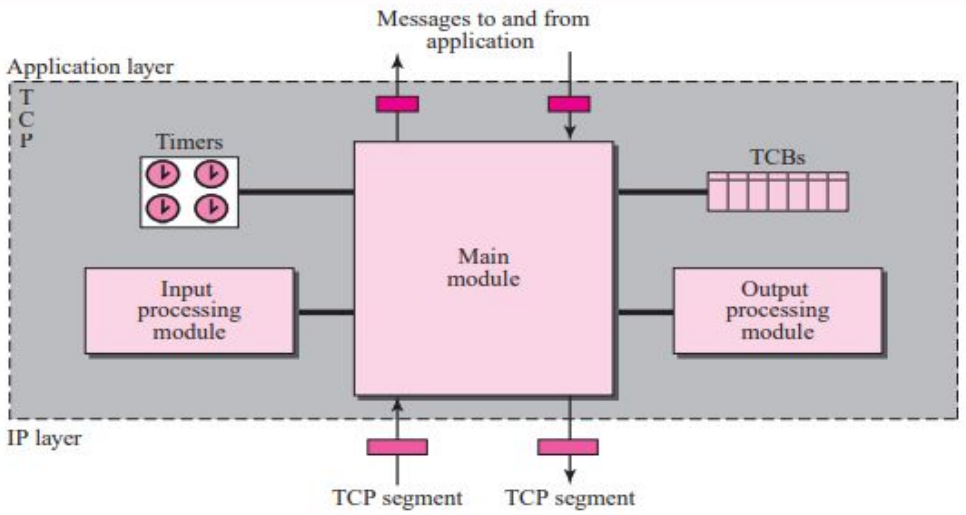
Steps to create a client using TCP/IP API

- Create a socket using the socket() function.
- Initialize the socket address structure as per the server and connect the socket to the address of the server using the connect();
- Receive and send the data using the recv() and send() function
- Close the connection by calling the close() function.

Steps to create a server using TCP/IP API

- Create a socket using the socket() function.
- Initialize the socket address structure and bind the socket to an address using the bind() function.
- Listen for connections with the listen() function.
- Accept a connection with the accept() function system call. This call typically blocks until a client connects to the server.
- Receive and send data by using the recv() and send() function.
- Close the connection by using the close() function.

TCP Package - Input, Output Processing Module

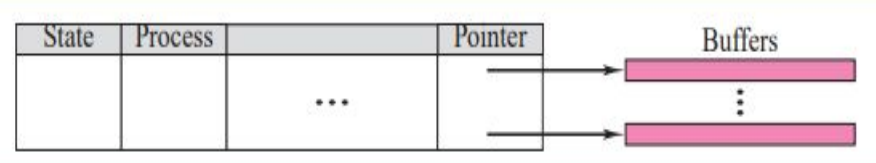


TCP package

TCP Package

- TCP is a stream-service, connection-oriented protocol with an involved state transition diagram.
- It uses flow and error control.
- It is so complex because actual code includes tens of thousands of lines.

Transmission Control Blocks (TCBs)



TCBs

- TCP is a connection-oriented transport protocol.
- A connection may be open for a long period of time.
- To control the connection, TCP uses a structure to hold information about each connection.
- This is called a transmission control block (TCB).

Fields in TCB

- ✔ State
- ✔ Process
- ✔ Local IP address
- ✔ Local port number
- ✔ Remote IP address
- ✔ Remote port number
- ✔ Interface
- Local window
- Remote window
- Sending sequence number
- Receiving sequence number
- Sending ACK number
- Round-trip time
- Time-out values
- Buffer size
- Buffer pointer

TCP Package

Timers

- TCP needs to keep track of its operations.
- Three software modules
  - ✔ Main module
  - ✔ ***An input processing module***
  - ✔ ***An output processing module***

Input Processing Module

- The Input processing module handles all the details which is required for the processing of data or an acknowledgement received when **TCP** is in the **ESTABLISHED STATE.**
- This module sends an **ACK** if needed.
- Takes care of the window size
- Performs Error checking and so on.

TCP Package

Output Processing Module

- The output processing module handles all the details needed to send out data received from application program when TCP is in the **ESTABLISHED STATE**.

- This module handles retransmission time-outs, persistent time-outs and so on.

UDP – USER DATAGRAM PROTOCOL

- ✔ Connectionless
- ✔ Unreliable transport protocol
- ✔ Located between application layer and network layer in the TCP/IP protocol suite
- ✔ Process to process communication using port numbers

Limitations of UDP

- There is no flow control mechanism.
- There is no acknowledgement for received packets.
- Does not provide error control to some extent.

Why would a process want to use UDP when it is powerless?

- ✔ It is simple protocol with minimum overhead.
- ✔ Application which use small messages to be sent without reliability in that case UDP is the best.
- ✔ For small messages less no. of interactions is required between sender and receiver for UDP compared to TCP.

UDP Services

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Domain	Domain Name Service (DNS)
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)

Well Known Ports used in UDP

Process to Process communication

- UDP provides process-to-process communication using sockets, a combination of IP addresses and port numbers. Several port numbers used by UDP.

Connectionless services

- Each datagram is independent even it comes from same source and delivered in same destination.
- The datagrams are not numbered.
- No connection establishments is done.
- Cannot send a stream of data to UDP, It will chop them into different packets related user datagrams.

UDP Services

Flow control

- No Flow Control and no window mechanism.
- The receiver may overflow with incoming messages.

- UDP should provide for this service, if needed.

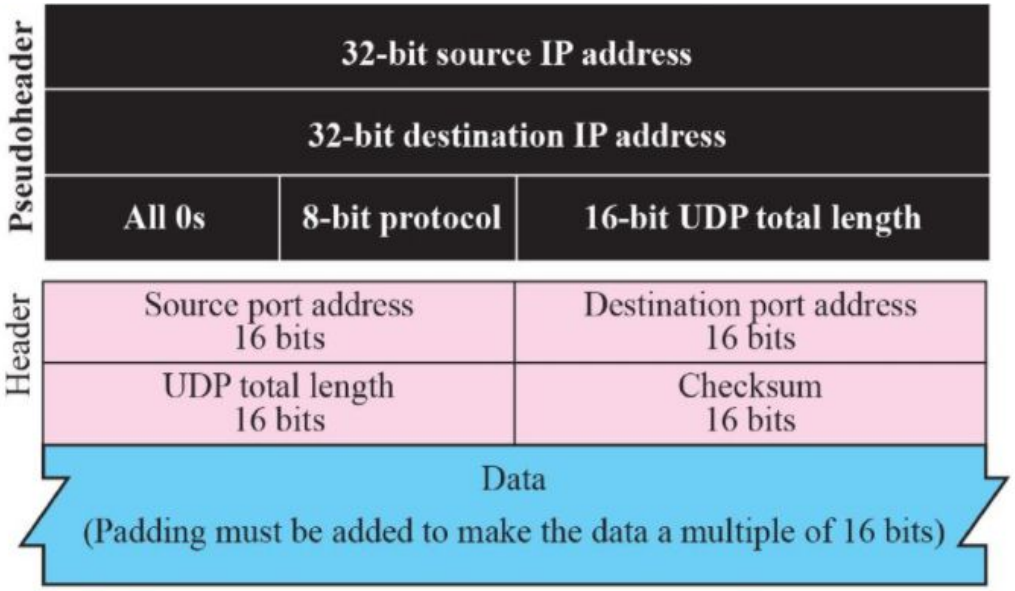
Error Control

- No Error control mechanism except for checksum.
- Sender is unknown about the loss and duplication.
- When error detects in receiver the datagram is discarded.

Congestion control

- Does not provide congestion control and has an assumption that the packets are small and sporadic so they cant create congestion.

UDP Services



Pseudo header for checksum calculation

Checksum

- Three sections: a pseudo header, the UDP header, and the data coming from the application layer.

Pseudo header

- It is a part of the IP header
- Encapsulated with some fields with 0's
- Protocol field to differentiate between UDP and TCP
- The value of the protocol field is 17. If it is changed then the packet gets discarded at receiver end.

UDP header

Data

- communication from the application layer

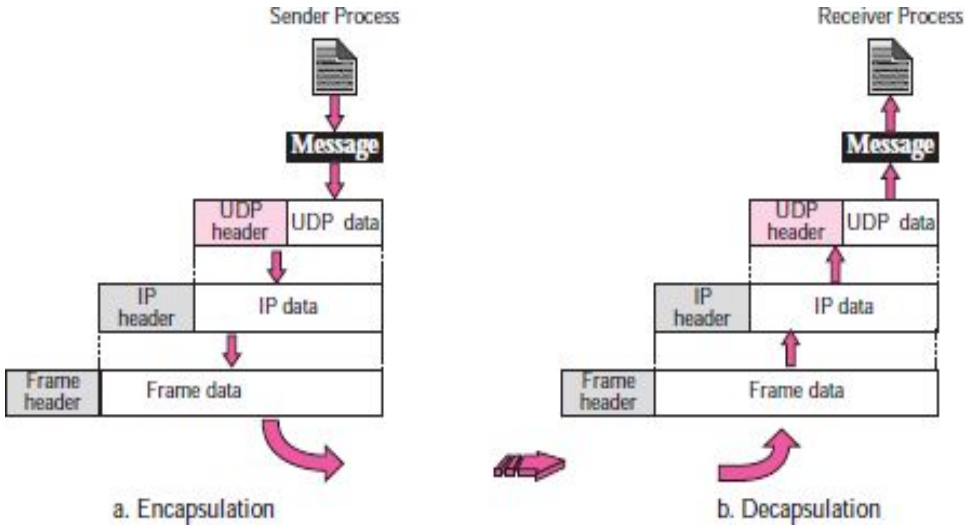
UDP Services

Encapsulation and Decapsulation

Encapsulation

- A process send message through UDP
- Along with a pair of socket address and the length of data.
- UDP receives data and add UDP header then pass to IP with socket.
- IP add its own header using value 17 in protocol field.
- Indicating UDP protocol.
- The IP datagram is then passed to the data link layer.
- The data link layer receives the IP datagram and passes it to the physical layer.
- The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

Decapsulation

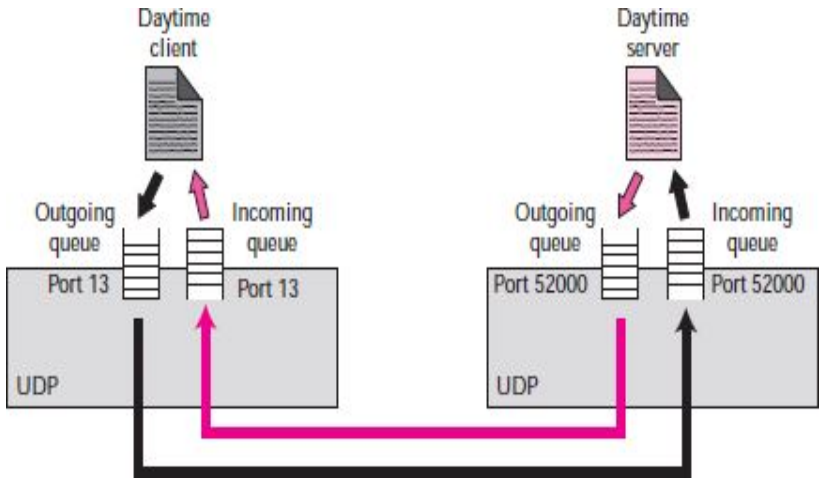


At the destination host

- ✔ The physical layer decodes the signals and pass it to the data link layer.
- ✔ The data link layer uses the header (and the trailer) to check the data.

 If there is no error

- ✔ The header and trailer are dropped , datagram is passed to IP.
- ✔ The header is dropped and the user datagram is passed to UDP with the sender and receiver IP addresses.
- ✔ Checksum is to check the entire user datagram.
- ✔ The header is dropped and the application data along with the sender socket address is passed to the process.
- ✔ The sender socket address is passed to the process in case it needs to respond to the message received.



Queuing in UDP

Queuing in UDP

- At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process
- Process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming queue. The queues opened by the client are, in most cases, identified by ephemeral port numbers. The queues function as long as the process is running. When the process terminates, the queues are destroyed.
- The client process can send messages to the outgoing queue by using the source port number specified in the request

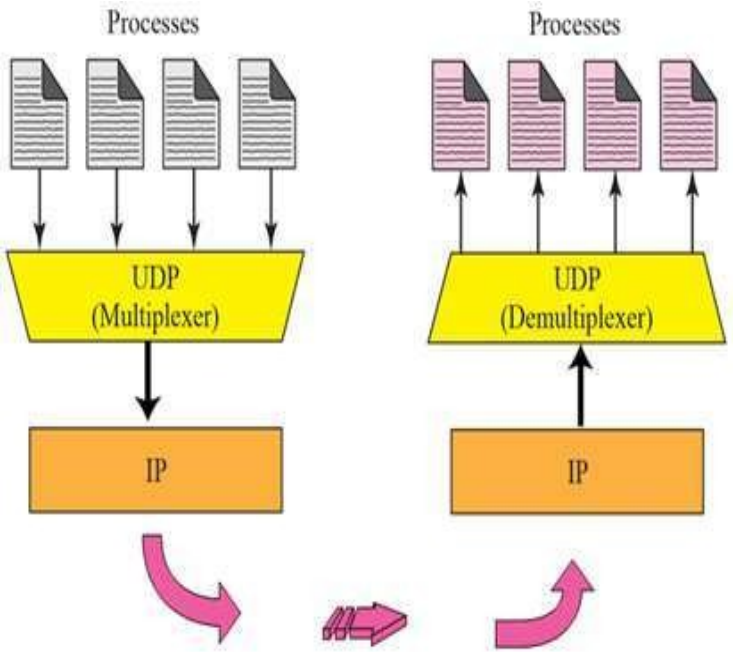
Queuing in UDP

- The client process can send messages to the outgoing queue by using the source port number specified in the request

- UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow

- It happens the operating system can ask the client process to wait before sending any more messages

Multiplexing & Demultiplexing



Multiplexing

- ✔ Many to one relationship.
- ✔ UDP accepts messages from different process and differentiate by port numbers.
- ✔ Adds a header and then sends to the IP

Demultiplexing

- ✔ One to many relationship.
- ✔ UDP receives the user datagram from IP and drops the header then sends the message to appropriate process based on port numbers.

UDP Features

- ✔ Connectionless services
- ✔ Lack of error control
- ✔ Lack of congestion control

Connectionless service

- ✔ Preferable for small message which fits in a single datagram.
- ✔ The overhead to establish and close a connection may be significant whereas in TCP it takes 9 packets for exchanges between client and server to achieve the above goal.
- ✔ Provides less delay

Lack of error control

- UDP does not provide error control.
- Provides unreliable service.
- In reliable service the transport layer needs to take care of the lost packet by resending it. So there will be a uneven delay between different parts of the message delivered.

Lack of congestion control

- UDP does not provide congestion control.
- UDP does not provide additional traffic in error prone network.
- TCP leads to creation of congestion or additional congestion in network by resending packets several times when a packet are lost.

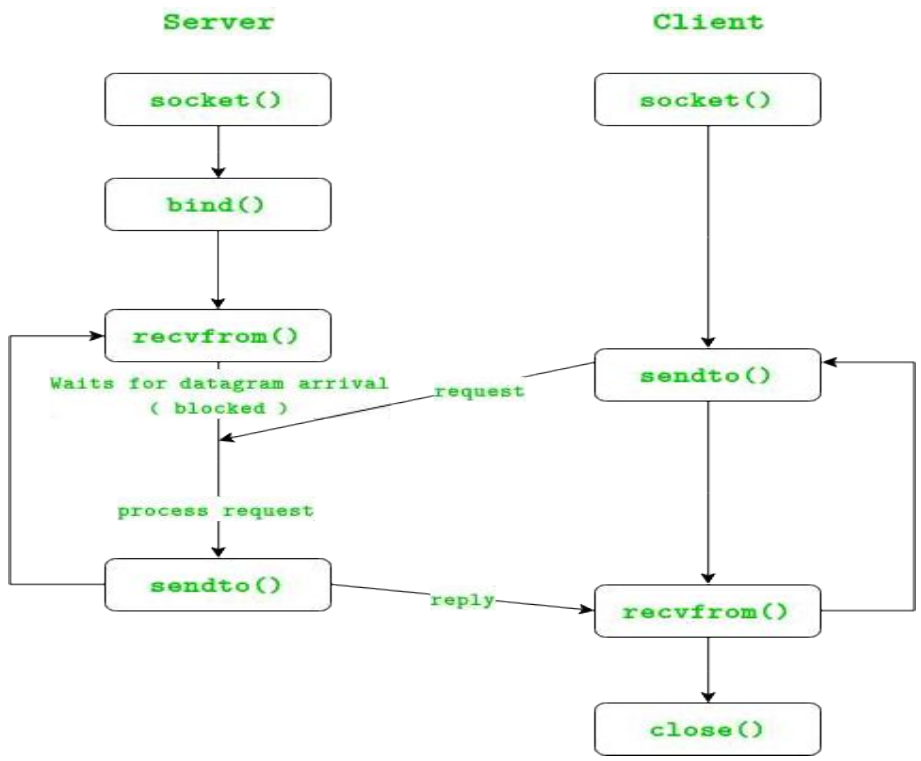
UDP Applications

- Used for simple request response communication when size of data is less hence there is lesser concern about flow and error control.

- It is suitable protocol for multicasting as UDP supports packet switching.

- Following implementations uses UDP as a transport layer protocol
  - ✔ NTP (Network Time Protocol)
  - ✔ DNS (Domain Name Service)
  - ✔ BOOTP, DHCP.
  - ✔ NNP (Network News Protocol)
  - ✔ Quote of the day protocol
  - ✔ TFTP, RTSP, RIP, OSPF.
  - ✔ UDP is null protocol if you remove checksum field.

UDP CLIENT – SERVER AND PACKAGES



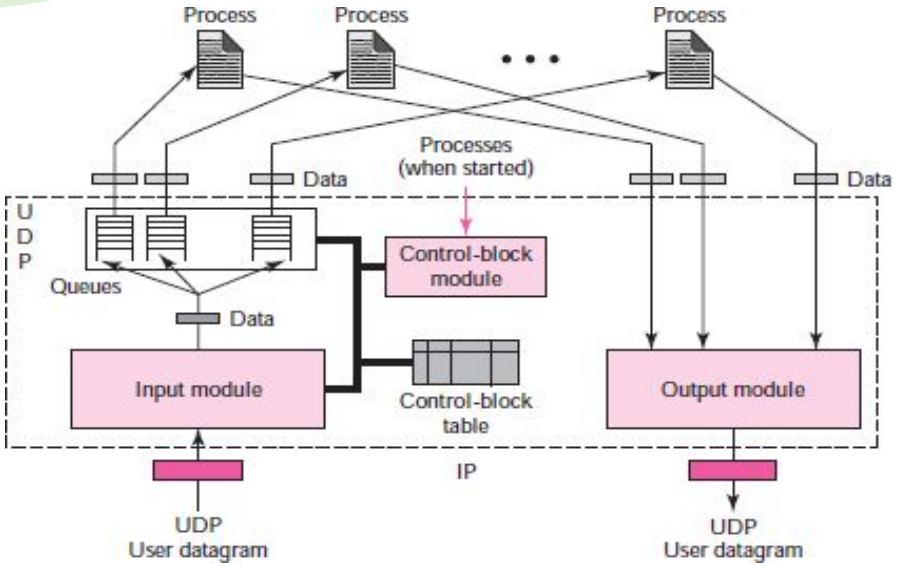
Server Processing using UDP

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

Client Processing using UDP

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

UDP Package



UDP design

The UDP package involves five components

Control-Block Table

- UDP has a control-block table to keep track of the open ports.
- Each entry in this table has a minimum of four fields: the state, which can be FREE or IN-USE, the process ID, the port number, and the corresponding queue number.

Input Queues

- UDP package uses a set of input queues, one for each process. In this design, we do not use output queues.

UDP Package

```
1  UDP_Control_Block_Module (process ID, port number)
2  {
3      Search the table for a FREE entry.
4      if (not found)
5          Delete one entry using a predefined strategy.
6      Create a new entry with the state IN-USE
7      Enter the process ID and the port number.
8      Return.
9  } // End module
```

Control-Block Module

- The control-block module is responsible for the management of the control-block table.
- When a process starts, it asks for a port number from the operating system.
- The operating system assigns well- known port numbers to servers and ephemeral port numbers to clients.
- The process passes the process ID and the port number to the control-block module to create an entry in the table for the process.

UDP Package

```
1  UDP_INPUT_Module (user_datagram)
2  {
3       Look for the entry in the control_block table
4       if (found)
5       {
6            Check to see if a queue is allocated
7            If (queue is not allocated)
8                 allocate a queue
9            else
10                enqueue the data
11      } //end if
12      else
13      {
14           Ask ICMP to send an "unreachable port" message
15           Discard the user datagram
16      } //end else
17
18      Return.
19 } // end module
```

Input Module

- The input module receives a user datagram from the IP. It searches the control-block table to find an entry having the same port number as this user datagram.

- If the entry is found, the module uses the information in the entry to enqueue the data. If the entry is not found, it generates an ICMP message.

UDP Package

```
1  UDP_OUTPUT_MODULE (Data)
2  {
3      Create a user datagram
4      Send the user datagram
5      Return.
6  }
```

Output module

- The output module is responsible for creating and sending user datagrams.

Example

State	Process ID	Port Number	Queue Number
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	
FREE			
IN-USE	4,652	52,012	38
FREE			

The first activity is the arrival of a user datagram with destination port number 52,012. The input module searches for this port number and finds it. Queue number 38 has been assigned to this port, which means that the port has been previously used. The input module sends the data to queue 38. The control-block table does not change.

Example

After a few seconds, a process starts. It asks the operating system for a port number and is granted port number 52,014. Now the process sends its ID(4,978) and the port number to the control-block module to create an entry in the table. The module takes the first FREE entry and inserts the information received. The module does not allocate a queue at this moment because no user datagrams have arrived for this destination.

State	*Process ID*	*Port Number*	*Queue Number*
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	
IN-USE	4,978	52,014	
IN-USE	4,652	52,012	38
FREE			

Example

A user datagram now arrives for port 52,011. The input module checks the table and finds that no queue has been allocated for this destination since this is the first time a user datagram has arrived for this destination. The module creates a queue and gives it a number (43).

Control-Block Table after Example 14.10

State	*Process ID*	*Port Number*	*Queue Number*
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	43
IN-USE	4,978	52,014	
IN-USE	4,652	52,012	38
FREE			

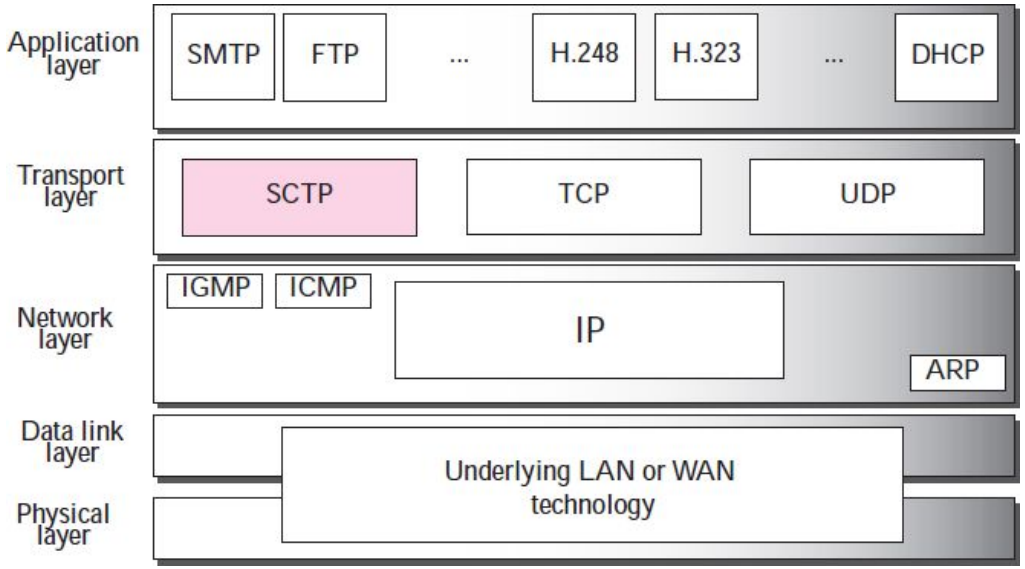
Example

After a few seconds, a user datagram arrives for port 52,222. The input module checks the table and cannot find an entry for this destination. The user datagram is dropped and a request is made to ICMP to send an unreachable port message to the source.

Stream Control Transmission Protocol (SCTP)

- SCTP is designed as a general-purpose transport layer protocol that can handle multimedia and stream traffic, which are increasing every day on the Internet.

- It is a new reliable, message-oriented transport-layer protocol.



Relationship of SCTP to the other protocols in the Internet protocol suite

Stream Control Transmission Protocol (SCTP)

Comparison between UDP, TCP, and SCTP

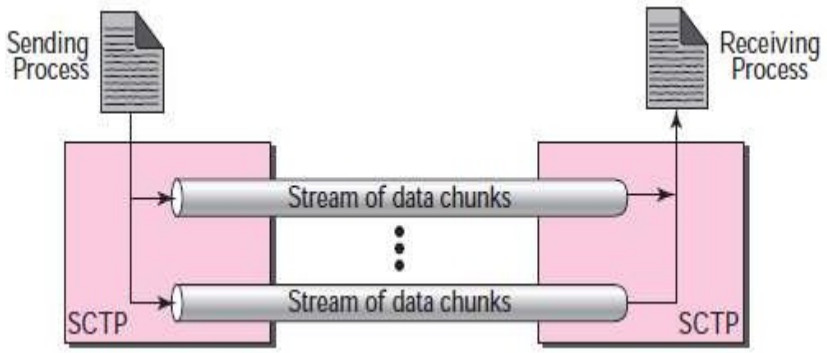
UDP	TCP	SCTP
Message - oriented protocol	Byte-oriented protocol	Best features of UDP and TCP
UDP conserves the message boundaries	No preservation of the message boundaries	Preserves the message boundaries along with detection of lost data, duplicate data, and out-of-order data
UDP is unreliable	TCP is a reliable protocol	SCTP is a reliable message oriented Protocol
Lacks in congestion control and flow control	TCP has congestion control and flow control mechanisms	It has congestion control and flow control mechanisms

SCTP is a *message-oriented, reliable* protocol that combines the best features of UDP and TCP.

SCTP Services

Protocol	Port Number	Description
IUA	9990	ISDN over IP
M2UA	2904	SS7 telephony signaling
M3UA	2905	SS7 telephony signaling
H.248	2945	Media gateway control
H.323	1718, 1719, 1720, 11720	IP telephony
SIP	5060	IP telephony

Some SCTP applications



Multiple-stream concept

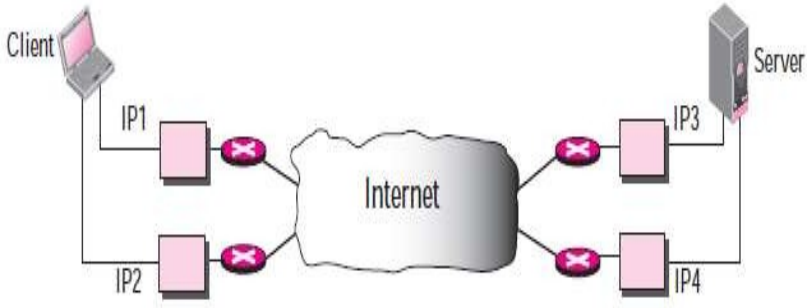
Process-to-Process Communication

- SCTP uses all well-known ports in the TCP space

Multiple Streams

- SCTP allows multi stream service in each connection called as association
- If one of the streams is blocked, the other streams can still deliver their data

SCTP Services



Multihoming concept

Multihoming

- The sending and receiving host can define multiple IP addresses in each end for an association.
- when one path fails, another interface can be used for data delivery without interruption.
- This fault-tolerant feature helps in sending and receiving a real-time payload such as Internet telephony.

Full-Duplex Communication

- SCTP has sending and receiving buffer, hence packets are sent in both directions.

SCTP Services

Connection-Oriented Service

- A connection is called an association in SCTP
- Steps to send and receive data in SCTP
  1. The two SCTP's establish an association between each other.
  2. Data are exchanged in both directions.
  3. The association is terminated.

Reliable Service

- It uses an acknowledgment mechanism to check the safe and sound arrival of data.

SCTP Services

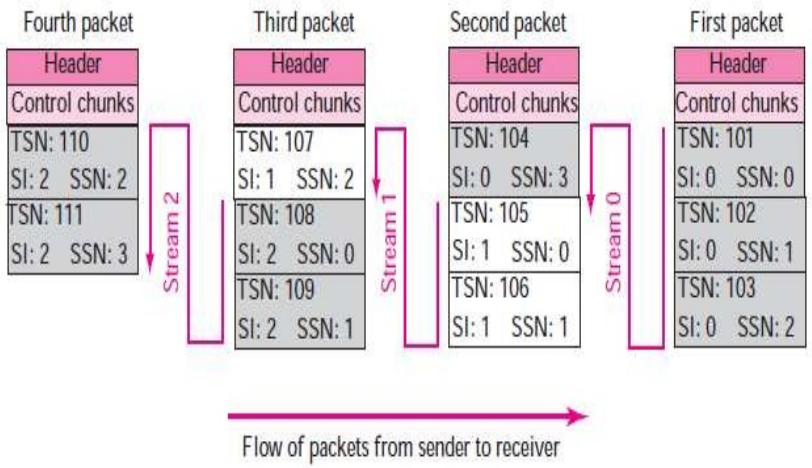
Transmission Sequence Number (TSN)

- Unit of data in SCTP is called the data chunk.
- Data transfer in SCTP is controlled by numbering the data chunks.
- SCTP uses a TSN to number the data chunks with 32 bits long number which is randomly initialized between 0 and $2^{32} - 1$.
- Each data chunk carry their TSN in its header.

Stream Identifier (SI)

- Each stream in SCTP needs to be identified using a SI.
- Each data chunk carry SI in its header.
- when it arrives at the destination, it is placed in order in its stream.
- The SI is a 16-bit number starting from 0.

SCTP Services



Packet, data chunks, and streams

Stream Sequence Number (SSN)

- When a data chunk arrives at the destination SCTP, it is delivered to the appropriate stream in the proper order.
- In addition to an SI, SCTP defines a SSN in each data chunk in each stream.

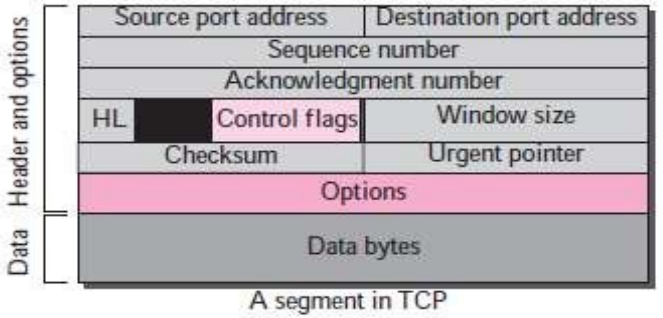
Packets

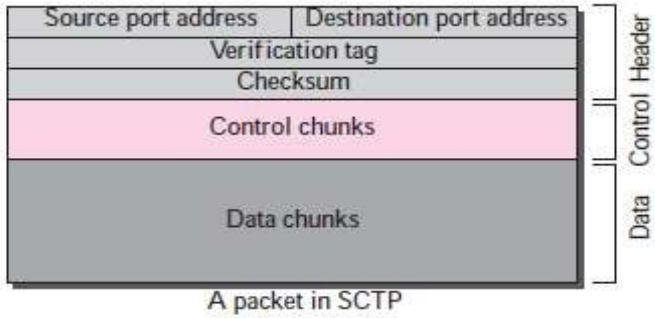
- Data are carried as data chunks, control information as control chunks.
- Several control chunks and data chunks can be packed together in a packet.

SCTP Features

Comparison between a TCP segment and an SCTP packet

TCP segment	SCTP packet
Control information is part of the header	Control information is included in the control chunks
Data is treated as one entity	Carry several data chunks, each can belong to a different stream
Options section exist separately	Options are handled by defining new chunk types
Mandatory part of header is 20 bytes	General header is only 12 bytes
Checksum is 16 bits	Checksum is 32 bits
Combination of IP and port addresses define a connection	Verification tag is an association identifier
Includes one sequence number in the header	Includes several different data chunks
Some segments carry control information	Control chunks never use a TSN, IS, or SSN number, they are used for data chunks only





SCTP Features

Acknowledgment Number

- SCTP acknowledgment numbers are chunk-oriented refer to TSN
- Control information is carried by control chunks, which do not need a TSN
- Control chunks are acknowledged by another control chunk of the appropriate type

Flow Control

- SCTP implements flow control to avoid overwhelming receiver

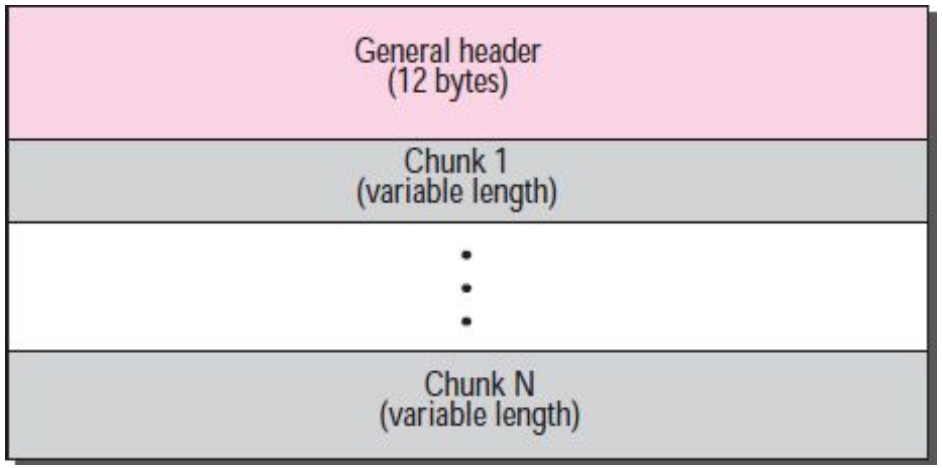
Error Control

- TSN numbers and acknowledgment numbers are used for error control.

Congestion Control

- SCTP implements congestion control to determine how many data chunks can be injected into the network

SCTP Features

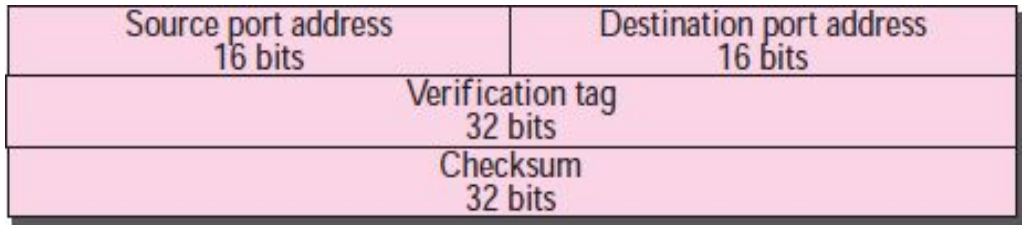


SCTP packet format

- Main Parts are
  - General header
  - Chunks – set of blocks

- Types of chunks
  - Control chunks - controls and maintains the association
  - Data chunks - carries user data

SCTP Packet Format



General header

General Header

- Defines the end points of each association to which the packet belongs.
- Guarantees for a packet belongs to a particular association.
- Preserves the integrity of the contents of the packet.

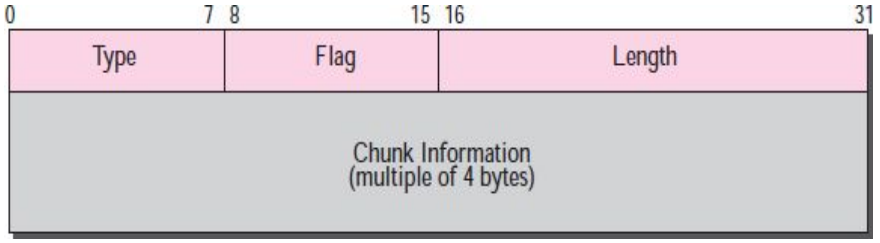
 There are four fields in the general header

- **Source port address:** 16-bit field defines the port number of the sender process
- **Destination port address:** 16-bit field defines the port number of the receiving process
- **Verification tag:** Number that matches a packet to an association
  - It serves as an identifier for the association
  - Separate verification used for each direction in the association.
- **Checksum:** 32-bit field contains a CRC-32 checksum

SCTP Packet Format

Type	Chunk	Description
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Abort an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk
9	ERROR	Reports errors without shutting down
10	COOKIE ECHO	Third packet in association establishment
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk
14	SHUTDOWN COMPLETE	Third packet in association termination
192	FORWARD TSN	For adjusting cumulating TSN

Types of Chunks

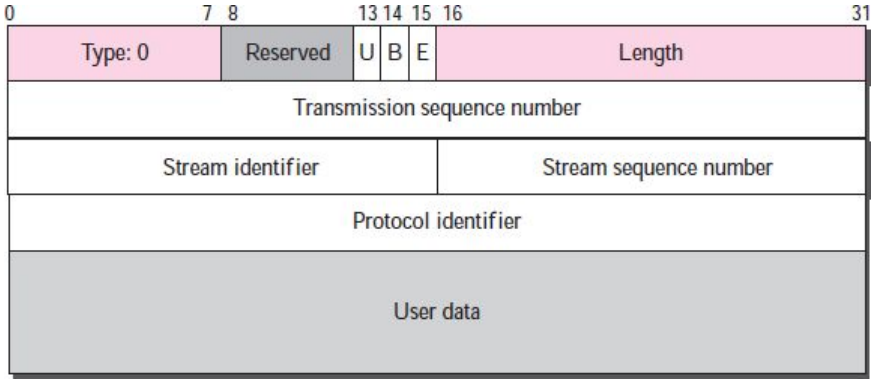


Common layout of a chunk

Chunks

- Control information or user data are carried
- First three fields are common to all chunks
  - **Type:** 8-bit field define up to 256 types of chunks(few have been defined, rest are reserved for future use)
  - **Flag:** 8-bit field defines special flags that a particular chunk may need.
  - **Length:** 16-bit field defines the total size of the chunk, in bytes, including the type, flag, and length fields
- Information field depends on the type of chunk.
- SCTP requires the information section to be multiples of 4 bytes
  - If not, padding bytes (eight 0s) are added at the end of the section

SCTP Packet Format



DATA chunk

Data Chunk

- Carries the user data
- A packet may contain zero or more data chunks
- Common fields
  - Type field has a value of 0
  - Flag field has 5 reserved bits and 3 defined bits
- U - signals unordered data
- B - beginning bit of fragmented message
- E - end bit of fragmented message

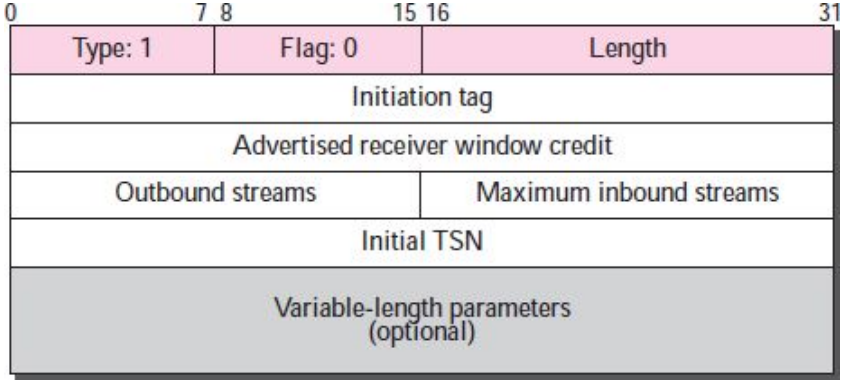
- TSN - Sequence number initialized in an INIT chunk for one direction and in the INIT ACK chunk for the opposite direction

- SI - all chunks of same stream in one direction have same stream identifier

- Protocol identifier: 32-bit field used by the application program to define the type of data which is ignored by SCTP

- User data: carries the actual user data
  - No chunk can carry data belonging to more than one message
  - A message can be spread over several data chunks
  - Must have at least one byte of user data, can't be empty
  - If the data cannot end at a 32-bit boundary, padding must be added

SCTP Packet Format

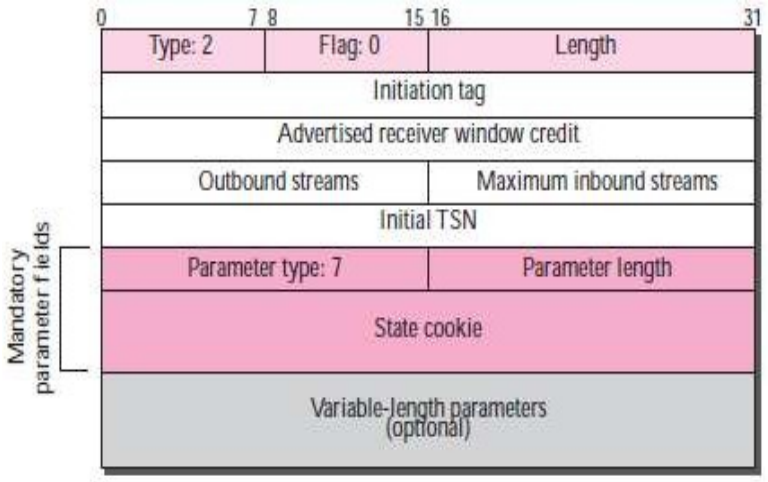


INIT chunk

➢ ***INIT* (Initiation chunk)**

- First chunk sent by an end point to establish an association
- Cannot carry any other control or data chunks
- Verification tag for this packet is 0
- Common fields
  - Type field has a value of 1
  - Flag field is 0
  - Length field value is minimum of 20
- Initiation tag
  - 32-bit field defines the value of the verification tag for packets traveling in the opposite direction
  - Tag is same for all packets traveling in one direction in an association
  - Random number between 0 and $2^{32} - 1$
- Advertised receiver window credit:
  - 32-bit field used in flow control
  - Defines the initial amount of data in bytes that the sender of the INIT chunk can allow
- Outbound stream: 16-bit field defines the number of streams an initiator of the association suggests for streams in outbound direction.
- Maximum inbound stream: 16-bit field defines the maximum number of streams an initiator of the association can support in inbound direction
- Initial TSN: initializes TSN in the outbound direction
- Variable-length parameters: optional parameters to define the IP address of sending end point, multihome, cookie state etc.

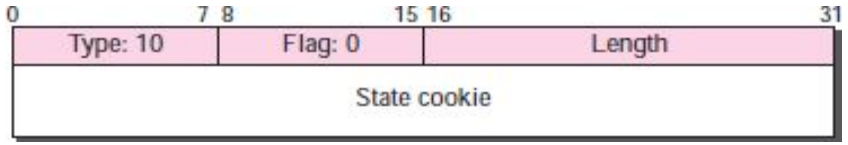
SCTP Packet Format



INIT ACK chunk

- ***INIT ack***(initiation acknowledgment chunk)
- Second chunk sent during association establishment
- Value of the verification tag is the value of the initiation tag of INIT chunk.
- The parameter of type 7 defines the state cookie sent by the sender of this chunk
- Initiation tag field in this chunk initiates the value of the verification tag for future packets traveling from the opposite direction.

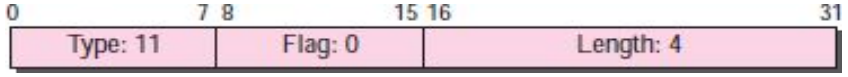
SCTP Packet Format



COOKIE ECHO chunk

Cookie echo

- Third chunk sent during association establishment that carry user data too.
- Sent by the end point that receives an INIT ACK chunk.
- Chunk of type 10.

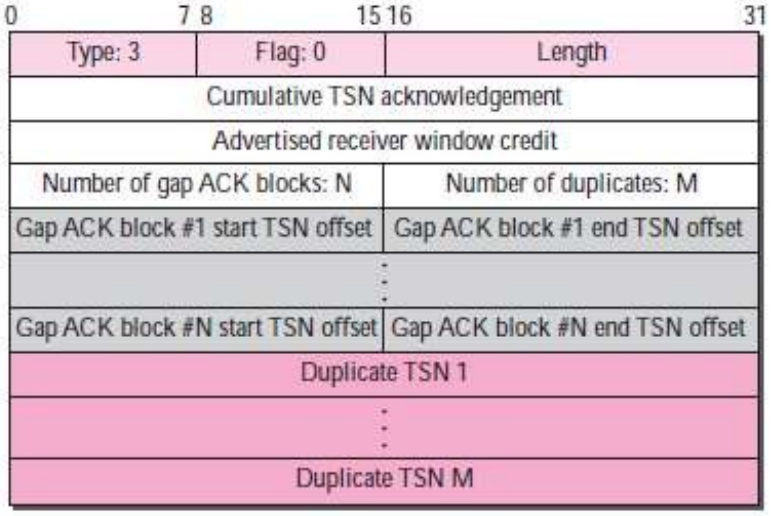


COOKIE ACK

COOKIE ACK

- fourth and last chunk sent during association establishment with data chunk too.
- sent by an end point that receives a COOKIE ECHO chunk.
- chunk of type 11.

SCTP Packet Format



SACK chunk

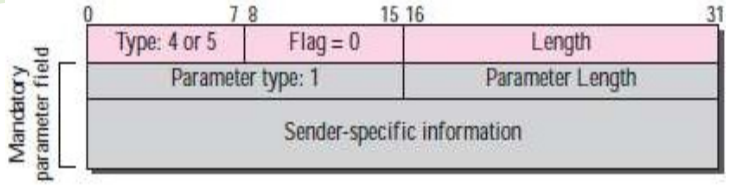
SACK(selective ACK chunk)

- Acknowledges the receipt of data packets
- Common fields
  - Type field has 3
  - Flag bits are set to 0s
- Cumulative tsn acknowledgment: 32-bit field defines the tsn of the last data chunk received in sequence
- Advertised receiver window credit: 32-bit field that have updated value for the receiver window size
- Number of gap ACK blocks: 16-bit field defines the number of gaps in the data chunk received after the cumulative

TSN

- Number of duplicates: 16-bit field defines the number of duplicate chunks following the cumulative TSN
- Gap ACK block start offset: 16-bit field gives the starting TSN relative to the cumulative TSN
- Gap ACK block end offset: 16-bit field gives the ending TSN relative to the cumulative TSN
- Duplicate tsn: 32-bit field gives the tsn of the duplicate chunk.

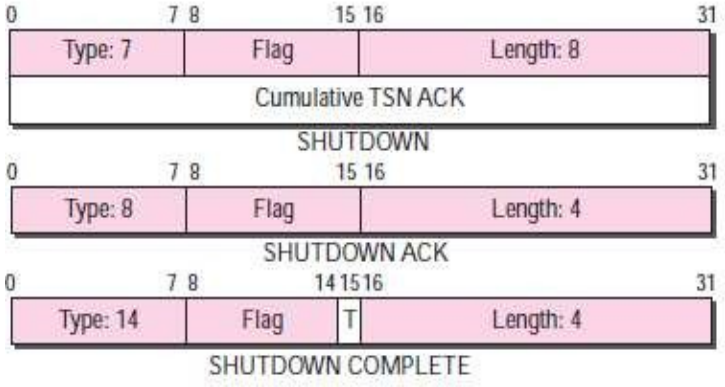
SCTP Packet Format



HEARTBEAT and HEARTBEAT ACK chunks

HEARTBEAT and HEARTBEAT ACK

- First has a type of 4 and the second a type of 5
- Used to periodically probe the condition of an association
- An end point sends a HEARTBEAT chunk, peer responds HEARTBEAT ACK if it is alive
- Parameter fields provide sender-specific information like address and local time
- Same is copied into the HEARTBEAT ACK chunk.



SHUTDOWN, SHUTDOWN ACK, and SHUTDOWN COMPLETE chunks

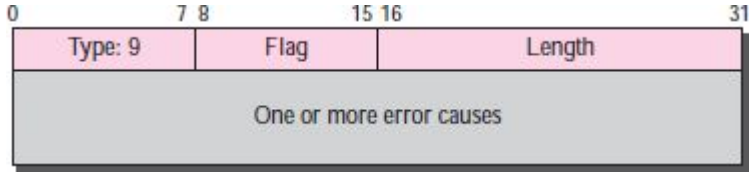
SHUTDOWN, SHUTDOWN ACK, and SHUTDOWN COMPLETE

- Used for closing an association
- Shutdown
  - Type 7 is eight bytes in length
  - Second four bytes define the cumulative TSN
- SHUTDOWN ACK: type 8 is four bytes in length.
- Shutdown complete
  - Type 14 is 4 bytes long
  - T flag is 1 bit flag shows that the sender does not have a TCB table

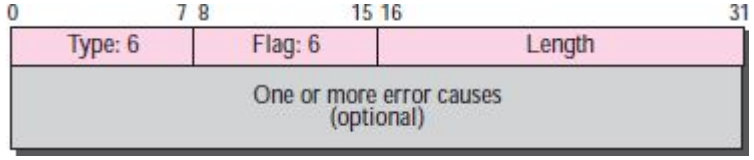
SCTP Packet Format

Code	Description
1	Invalid stream identifier
2	Missing mandatory parameter
3	State cookie error
4	Out of resource
5	Unresolvable address
6	Unrecognized chunk type
7	Invalid mandatory parameters
8	Unrecognized parameter
9	No user data
10	Cookie received while shutting down

Errors



ERROR chunk



ABORT chunk

ERROR

- Sent when an end point finds some error in a received packet.
- It does not imply the aborting of the association.

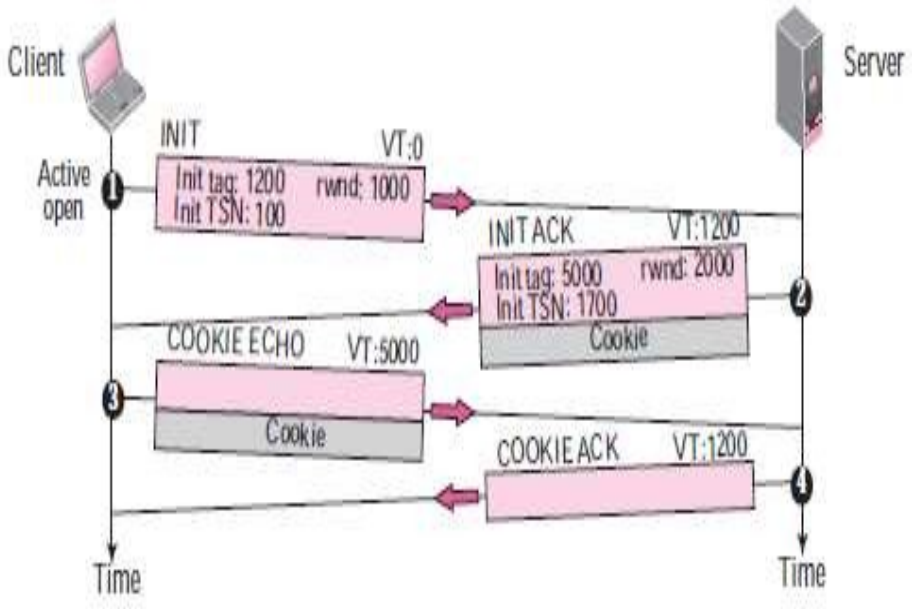
ABORT

- Sent when an end point finds a fatal error and needs to abort the association.

FORWARD TSN

- This is a chunk recently added to the standard to inform the receiver to adjust its cumulative TSN

SCTP Client/Server(Association)



Four-way handshaking

Association Establishment

- ***Four-way handshake***
  1. First packet has INIT chunk sent by client
     - Verification tag is 0
     - Rwnd is advertised in a SACK chunk
     - Inclusion of a DATA chunk in the third and fourth packets
  2. Second packet has INIT ACK chunk sent by server
     - Verification tag is the initial tag field in the INIT chunk
     - Initiates the tag to be used in the other direction
     - Defines the initial TSN and sets the servers' rwnd
  3. Third packet has COOKIE ECHO chunk sent by client
     - Echoes the cookie sent by the server
     - Data chunks are included in this packet
  4. Fourth packet has COOKIE ACK chunk sent by server
     - Acknowledges the receipt of the COOKIE ECHO chunk
     - Data chunks are included with this packet.

SCTP Client/Server(Association)

Number of Packets Exchanged

- Number of packets exchanged is four(3 for TCP)
- Allows the exchange of data in the third and fourth packets, so it is efficient

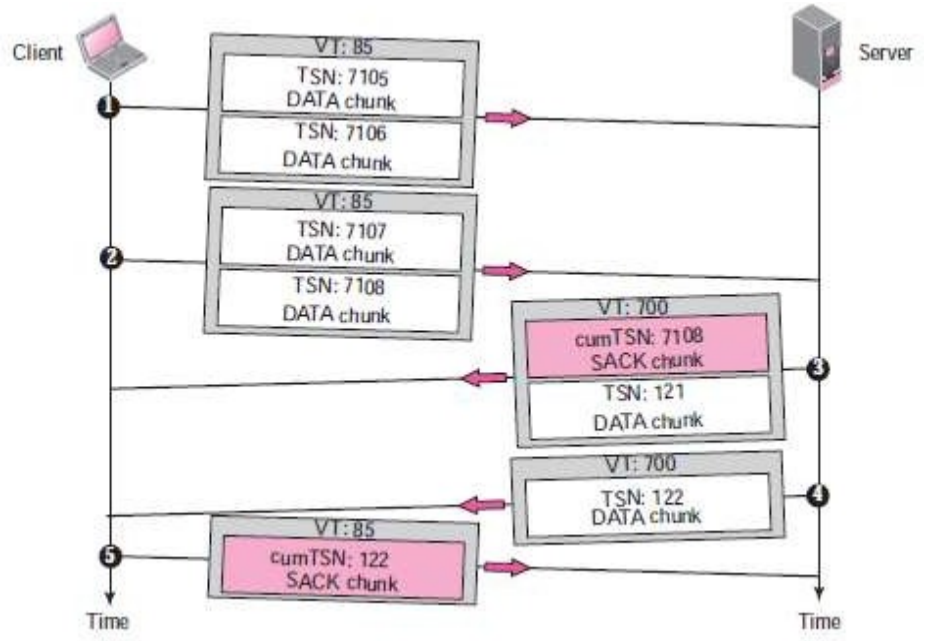
Verification tag

- It is a common value carried in all packets traveling in one direction in an association
- Blind attacker cannot inject a random packet into an association
- A packet from an old association cannot show up in an incarnation

Cookie

- Cookie is sent with the second packet to the address received in the first packet
- If the sender of the first packet is an attacker, the server never receives the third packet
- If the sender of the first packet is an honest client, it receives the second packet, with the cookie

SCTP Client/Server(Association)



Simple data transfer

Data transfer

- Purpose of an association is to transfer data between two ends.
- Once association is established, bidirectional data transfer can take place.
- SCTP supports piggybacking.
- Each message coming from the process is treated as one unit and inserted into a DATA chunk.
- Each DATA chunk formed by a message or a fragment has one TSN and acknowledged by SACK chunks.

SCTP Client/Server(Association)

Multihoming Data Transfer

- Allows both ends to define multiple IP addresses for communication.
- One address is **primary address,** rest are alternative addresses.
- The primary address is defined during association establishment.
- Primary address of the destination is used by default for data transfer, if it is not available, one of the alternative addresses is used.
- SACK is sent to the address from which the corresponding SCTP packet originated.

Multistream delivery

- TSN numbers are used to handle data transfer whereas delivery of the data chunks are controlled by SIs and SSNs.
- Two types of data delivery in each stream
  - Ordered: SSNs define the order of data chunks in the stream.
  - Unordered: U flag is set, it delivers the message carrying the chunk to the destination application without waiting for the other messages.

SCTP Client/Server(Association)

Fragmentation

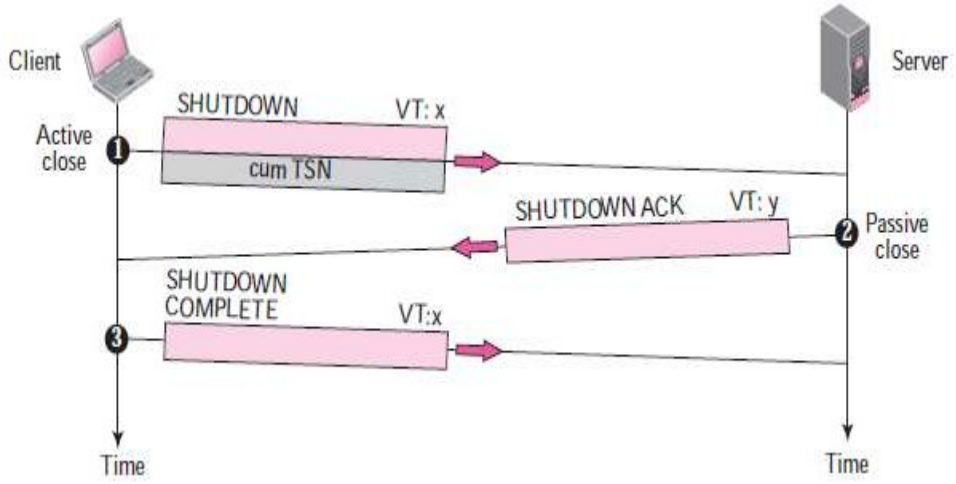
- SCTP preserves the boundaries of the message when creating DATA chunk from a message

- If the total size exceeds the MTU, the message needs to be fragmented

- Steps for fragmentation
  - Message is broken into smaller fragments to meet the size requirement
  - DATA chunk header is added to each fragment that carries a different TSN
  - All header chunks carry the same SI, SSN, payload protocol identifier and U flag
  - B and E are assigned as
    A. First fragment: 10
    B. Middle fragments: 00
    C. Last fragment: 01

- Fragments are reassembled at the destination

SCTP Client/Server(Association)

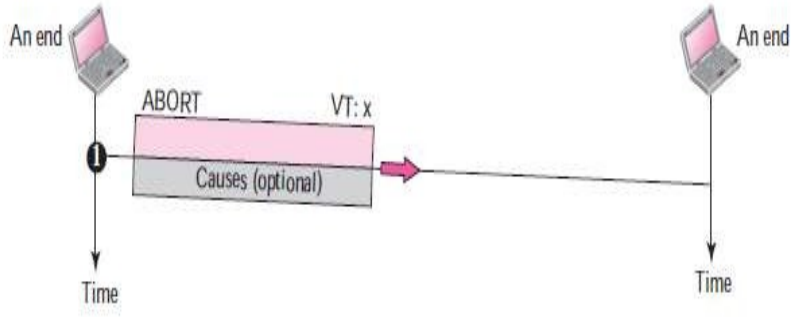


Association termination

Association Termination (Graceful termination)

- Either client or server involved in exchanging data can close the connection
- SCTP does not allow a “half closed” association, i.e. if one end closes the association, the other end must stop sending new data
- If not, the data in the queue are sent and the association is closed
- Association termination uses three packets
  - SHUTDOWN
  - SHUTDOWN ACK
  - SHUTDOWN COMPLETE

SCTP Client/Server(Association)



Association abortion

Association abortion

- Association in SCTP can be aborted based on request by the process at either end or by SCTP
- A process may wish to abort the association if the process receives wrong data from the other end, going into an infinite loop etc.
- Server may wish to abort since it has received an INIT chunk with wrong parameters, requested resources are not available after receiving the cookie, the operating system needs to shut down etc.
- For abortion process either end can send an abort chunk to abort the association