Computer Networks - Important Topics Only

Satvik Gupta

May 15, 2023

Contents

Introduction

Data communication

It is the exchange of data between two or more devices via some transmission medium

Components of Effective Data Communication

- Delivery: The data should be delivered to the destination it was intended to.
- Accuracy
- Timeliness
- Jitter free

Components of Data communication system

- Sender
- Receiver
- Message
- Protocols
- Communication/Transmission medium

Types of Communication

- Simplex: Unidirectional communication.
- Half Duplex: Bidirectional communication but only one direction at a time.
- Full Duplex: Two simplex connections in opposite directions.

Physical Topology

It tells how systems are physically connected through links. It is a geometric representation of the network.

Bus Topology

Only one connection.

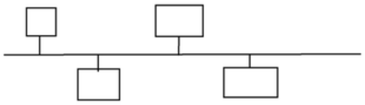


Figure 1: Bus Topology

Advantages

- Easy to install
- Cheap
- Easy to expand

Disadvantages

- Only one device can transmit at a time, which makes it low speed.
- Single point of failure - faulty cable can bring down the whole system.

Ring Topology

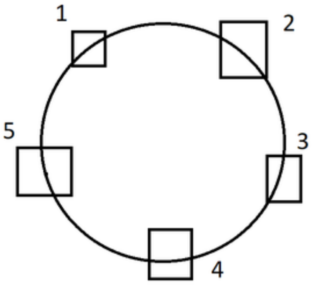


Figure 2: Ring Topology

Tokens are used to transfer data. Only one system can hold the token at a time. Token passing is done.

Advantages

- Cheap

Disadvantages

- Not easy to install.
- Not easy to expand.
- If one system/one link goes down the entire ring will go down.

Star Topology

Uses a central hub.

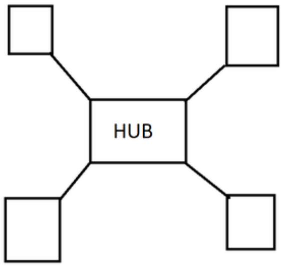


Figure 3: Star Topology

Advantages and disadvantages same as of any centralized system Hub can also be expensive.

Mesh Topology

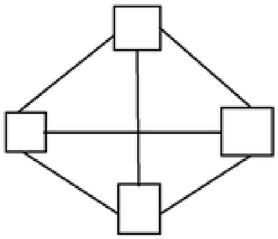


Figure 4: Mesh

Advantages

- Less traffic
- No single point of failure
- Messages can be sent directly without any routing

Disadvantages

- Cabling cost will be higher
- Maintenance cost will be higher.

Tree Topology

Tree structure.

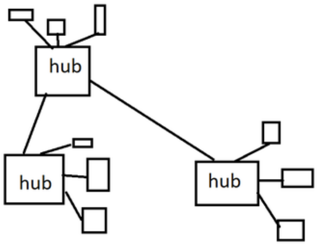
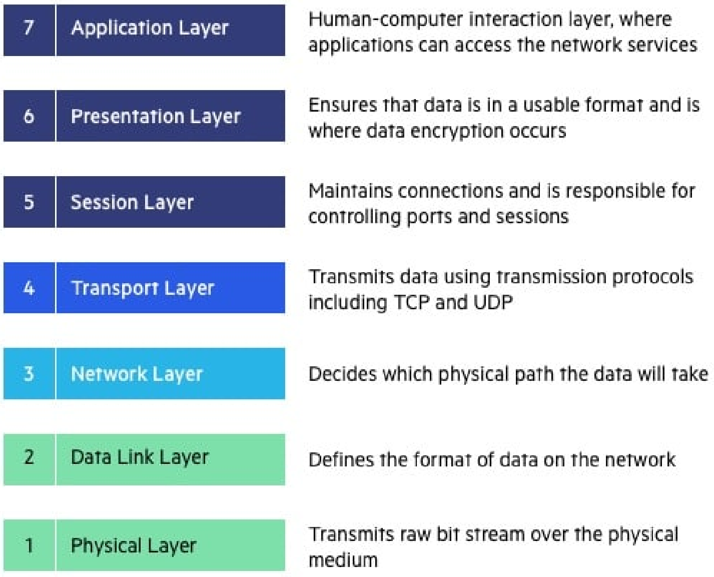


Figure 5: Tree Topology

OSI Model - Open Systems Interconnection

Given by ISO.



7	Application Layer	Human-computer interaction layer, where applications can access the network services
6	Presentation Layer	Ensures that data is in a usable format and is where data encryption occurs
5	Session Layer	Maintains connections and is responsible for controlling ports and sessions
4	Transport Layer	Transmits data using transmission protocols including TCP and UDP
3	Network Layer	Decides which physical path the data will take
2	Data Link Layer	Defines the format of data on the network
1	Physical Layer	Transmits raw bit stream over the physical medium

Figure 6: OSI

Data in layers:

S.No	Layer	Data	Responsibility	Protocols
1	Application Layer	Data	To allow access to network resources	Telnet, SMTP,DNS, HTTP
2	Presentation Layer	Data	To translate, encrypt and process the data	
3	Session Layer	Data	To establish, manage and terminate session	
4	Transport Layer	Segment	Process to Process msg delivery, error recovery	TCP, UDP (Port/Socket Address)
5	Network Layer	Packet	Move packet from source to destination.	IP, ARP, RARP, ICMP (Logical/IP Address)
6	Data Link Layer	Frame	Hop to hop delivery, organize the frames	IEEE 802 Std., TR,PPP (Physical/MAC Address)
7	Physical Layer	Bit	Transmit bits over a medium, provide mechanical and electrical specification	Transmission media

ARP - Address Resolution Protocol - Maps IP to MAC.

RARP - Reverse Address Resolution Protocol - Maps MAC to IP.

Physical Layer

It is responsible for moving physical bits. It defines:

- a transmission medium (wireless/wired)
- types of encoding to be used
- data rate
- synchronization of bits
- physical topology

Switched Networks

Large networks cannot have all nodes directly connected with each other. Therefore, to send data from one node to another, it has to be sent through other nodes.

Suppose there's a network with many nodes, and node A wants to send some data to node B. There are two ways of doing so.

1. **Packet Switching**

   Data is divided into small sized packets for transmission. This increased efficiency, reduces chances of lost data,etc.

   1. **Virtual Circuit** - Source establishes a (virtual) path that the data will follow. Each packet goes through the same route.
   2. **Datagram Switching -** Source doesn't decide any route. It sends each packet to the next nodes. Each node can decide where to forward the packets. Each packet may end up taking a different route. Packets may be delivered in a different order.

2. **Circuit Switching**

   A special path is set up for the transmission, and the intermediate nodes are already decided before the data transmission takes place. There is a dedicated path set up for the transmission of that packet.

3. **Message Switching -** Entire message is transferred between nodes. Each node stores the message, then decides where to forward it. This is also called *store and forward*.

Types of Communication on the basis of Connection

Connection-Oriented

- Similar to telephone.
- Establish a connection, communicate, and then release the connection.

Connectionless

- Like postal system
- Each packet has the destination address,
- Each packet is routed independently through the system.

Data Link Layer

Delay

Transmission Delay

The delay taken for the host to put the data onto the transmission line.

$$T_t = L/B$$

where L is the size of the data, and B is the bandwidth.

Propagation Delay

Time taken by the last bit of the data to reach the destination (after it has been transmitted from host to transmission media at the source.)

$$T_p = distance/velocity$$

Framing Techniques

One of the major issue in framing is to decide how to specify the start and end of a frame.

Flag (Character Stuffing/Byte Stuffing)

We use a special **flag byte** at the start and end of each frame. It is fixed so it can be recognized.

An issue with this is that the flag byte may occur "accidentally" in the data itself. This may cause the DLL to assume the frame has ended even when it has not.

To solve this, we use a special **ESC** byte, which is also fixed. Accidental flag bytes have the ESC sequence inserted before them, to tell the DLL that this FLAG is data and not the end of a frame.

If ESC occurs within the data "accidentally", we escape it with another ESC.

Examples

A Flag B –> A ESC Flag B

A ESC B –> A ESC ESC B

A ESC Flag B –> A ESC ESC ESC Flag B

A ESC ESC B –> A ESC ESC ESC ESC B

Doesn't work if the data isn't 8-bit.

Bit Stuffing

A special bit pattern denotes start and end of frames. Generally, this is taken to be `01111110`.

If the sender encounters the starting of this pattern in the data, it adds a 0 or a 1 before it ends so that the pattern never occurs. The receiver will do the opposite and remove the *stuffed* 0s or 1s.

For e.g, for `01111110`,

If the sender encounters a 0 followed by 5 consecutive 1's, it adds a 0 before continuing. This ensures that `01111110` never occurs in the data.

The receiver will *destuff* these extra zeroes on its end.

`01111110` –> `011111010`

`011011111111111111110010` –> `011011111011111011111010010`

Error Detection and Control

CRC (Cyclic Redundancy Check)

Data - k-bit Codeword - n-bit Divisor - (n-k+1) bits. Divisor should be mutually agreed between sender and receiver.

- Add (n-k) `0`s to the dataword.
- Divide dataword by divisor using *modulo-2 division*.
- Append the remainder found to the original dataword (without the extra zeroes)

At receiver's side:

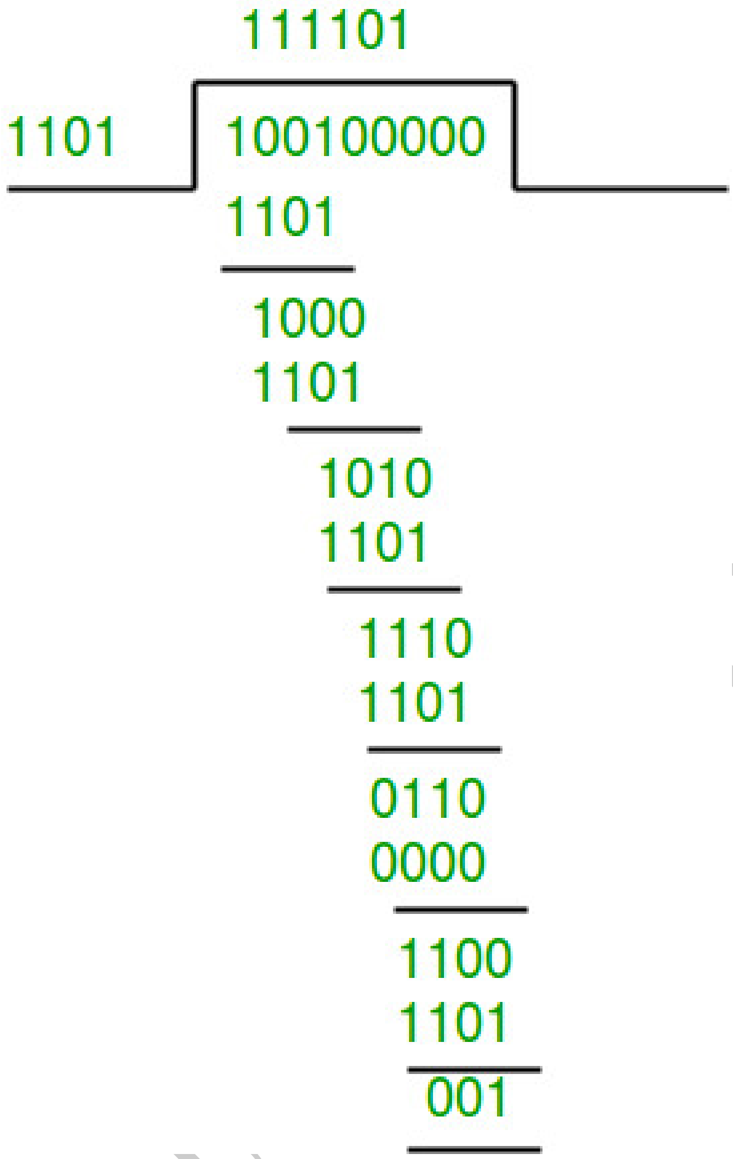
- Perform modulo-2 division of the received code-word and divisor.

- If the remainder is 0, the data is correct. Otherwise it's incorrect.

Modulo-2 Division

It's a method of dividing 2 binary numbers.

It follows the rules and logic of normal division, with subtraction step replaced by bitwise XOR.



Flow Control

To make sure receiver receives all the data.

Stop & Wait ARQ (Automatic Repeat ReQuest)

- Sender sends a frame and waits for ACK (acknowledgement) for the frame from receiver.
- Receiver receives the frame. If the frame is correct, receiver sends ACK.
- If the frame is corrupt, receiver drops the frame and does nothing.
- Sender waits a certain amount of time for ACK from receiver. After this, it times out and resends the frame.
- ACK message may also get lost, then the sender will assume the original frame was corrupted or lost. It will retransmit, which means the receiver may get duplicate data.
- To avoid this, frames are numbered.

- We only need to differentiate between a frame and its immediate successor. I.e, we need to differentiate between frame x and x+1. Therefore, 1 bit sequence number is enough. If the first frame is 0, the second frame is 1, the third is again 0, and so on.

Formulas

Efficiency = η = Useful Time/ Total Time

$$= \frac{1}{1+2(\frac{T_p}{T_t})}$$

$$= \frac{1}{1+2a}$$

where a = T_p/T_t.

where T_p is propagation time and T_t is transmissiontime

Throughput = η * Bandwidth

Go Back N

- Sliding Window Protocol
- Receiver window Size = 1
- Sender Window Size = $2^m - 1$
- Sequence numbers for frames – $[0, 1, ...2^m - 1]$, 0 and $2^m - 1$ inclusive.
- Window Size = WS/W
- We send up to W frames at a time, and keep them in memory until the receiver ACKs them.
- Receiver only receives one frame at a time.
- Receiver can send a single ACK for many frames. For eg if sender sent 7,8,9 and receiver received all, it can simply send ACK 10.
- If receiver receives wrong frame (e.g receiver was waiting for frame 3 and frame 4 came), or a corrupted frame, it stays silent.
- Sender's timer will timeout. Sender will resend all frames in the window.

For e.g, if WS=3 and sender has sent 1,2,3,4,5,6 and timer for 3 times out (1 and 2 ACKed successfully), sender will send 3,4,5 again.

Efficiency = $\eta = \frac{WS}{1+2a}$

where $a = T_p/T_t$

For maximum efficiency (100% usage),

- WS = 1+2a
- No. of bits needed for sequence number = $\lceil log(1+2a) \rceil$

Selective Repeat (SR)

- Only lost/corrupted frames are resent.
- Sender window size = Receiver Window Size
- Window Size = $2^m/2$
- Receiver buffers frames that are within its window range. Others are dropped. For e.g, if receiver's window is waiting for frames 3,4,5 and sender sends 6, it will be dropped.
- ACK is only sent after frames are received in order. If receiver window is 3,4,5 and we receive 4,5 - 4,5 will be stored and buffered, but no ACK will be sent.

  Instead, receiver will send a negative acknowledgement (NACK) for 3 - NACK 3.
- This tells the sender that receiver hasn't received 3.
- Sender will resend 3 (only 3).

- When 3 is received, receiver will send ACK 6.
- If we had received 3 in the beginning, we would have immediately sent ACK 4. This would also make the receiver move its window to 4,5,6.

Example case:

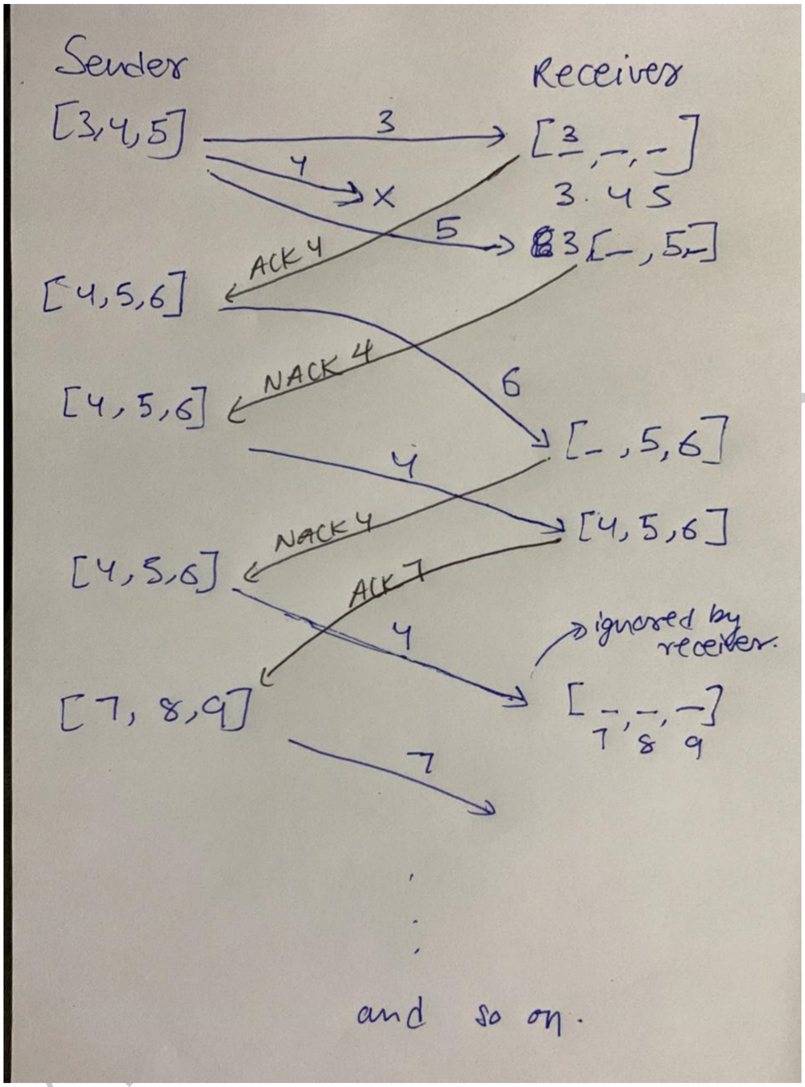


Figure 7: Selective Repeat Example

Data Encoding Techniques

NRZ-Unipolar

1 - +ve

0 - 0

NRZ-Polar

1 - +ve

0 - -ve

NRZ-I

Differential Encoding.

1 - Signal transition at start (high-to-low or low-to-high)

0 - No signal transition at start

Manchester

Always has a mid-bit transition:

1 - Low to High

0 - High to Low

The start of the bit may also have a transition, if needed according to the bit's value.

For eg, if the bit is 1 (which means we need a low to high transition in the middle), and the interval starts with a high value, we will transition to low at the start.

(Eg - Encoding 11)

Differential Manchester

Mid-bit transition is only for clocking purposes.

1 - Absence of transition at the start.

0 - Presence of transition at the start.

Bipolar Encoding

1 - Alternating +1/2, -1/2 voltages

0 - 0 voltages

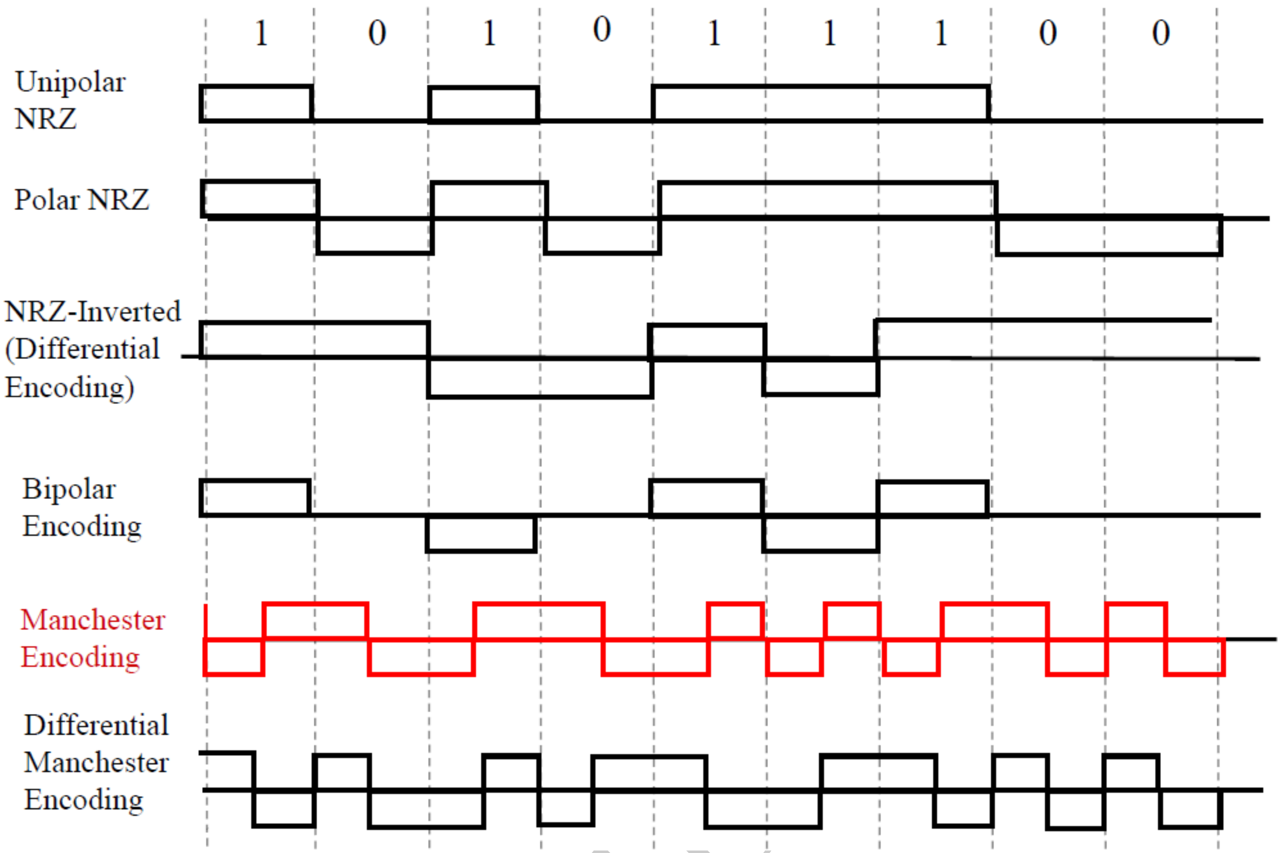

1
0
1
0
1
1
1
0
0
Unipolar NRZ
Polar NRZ
NRZ-Inverted (Differential Encoding)
Bipolar Encoding
Manchester Encoding
Differential Manchester Encoding

Figure 8: Data Encoding Techniques

Media Access Sublayer

The data link layer is divided into two sublayers.

1. **Media Access Control (MAC)** - Defines the access method for each LAN.
2. **Logical Link Control (LLC)** - Flow control, Error control, etc.

Framing is handled by both.

Media Access Control and Multiple Access Protocols

Handle how multiple nodes can communicate on a single link.

Random Access/ Contention Methods

- All nodes are considered equal.
- No scheduled transmission.
- Transmission occurs randomly.
- Nodes compete for access.

Pure Aloha

- Each node sends a frame when it has a frame to send.
- Obviously, we will have collisions in case 2 nodes decide to send a frame together.
- Aloha expects the receiver of the frame to send ACK for the frame.

- Vulnerable time for Aloha is $2 * T_t$. This is the time frame in which collisions can happen.

  For eg, A sent a frame at 12:05

  Let transmission time = 5 minutes.

  B wants to send a frame. But it cannot send a frame until 12:10, because till 12:10 A will be transmitting its frame. A collision will occur if B sends before 12:10.

  Similarly, if C had earlier sent a frame anytime after 12:00, A's frame will collide with it.

  Therefore, the vulnerable time is 12:00 - 12:10, which is 10 minutes = twice of transmission time.
- In case a collision occurs, the node waits a random amount of time before retransmitting. How much time to wait is explained in the flowchart below.
- Maximum number of attempts are fixed.This value is called K_{max} For eg, if max attempts = 15, if a node has transmitted the same frame 15 times and always gotten collision, it will abort and try again some time later.
- K_{max} is generally set to be 15.

Efficiency of Pure Aloha

$$S = G.e^{-2G}$$

where G is the the average number of frames created by the *entire system* (all nodes combined), during the transition time of a single frame.

For eg, if T_t is 1ms, G is number of frames produced per millisecond.

$S_{max} = 0.184$ at $G = 1/2$.

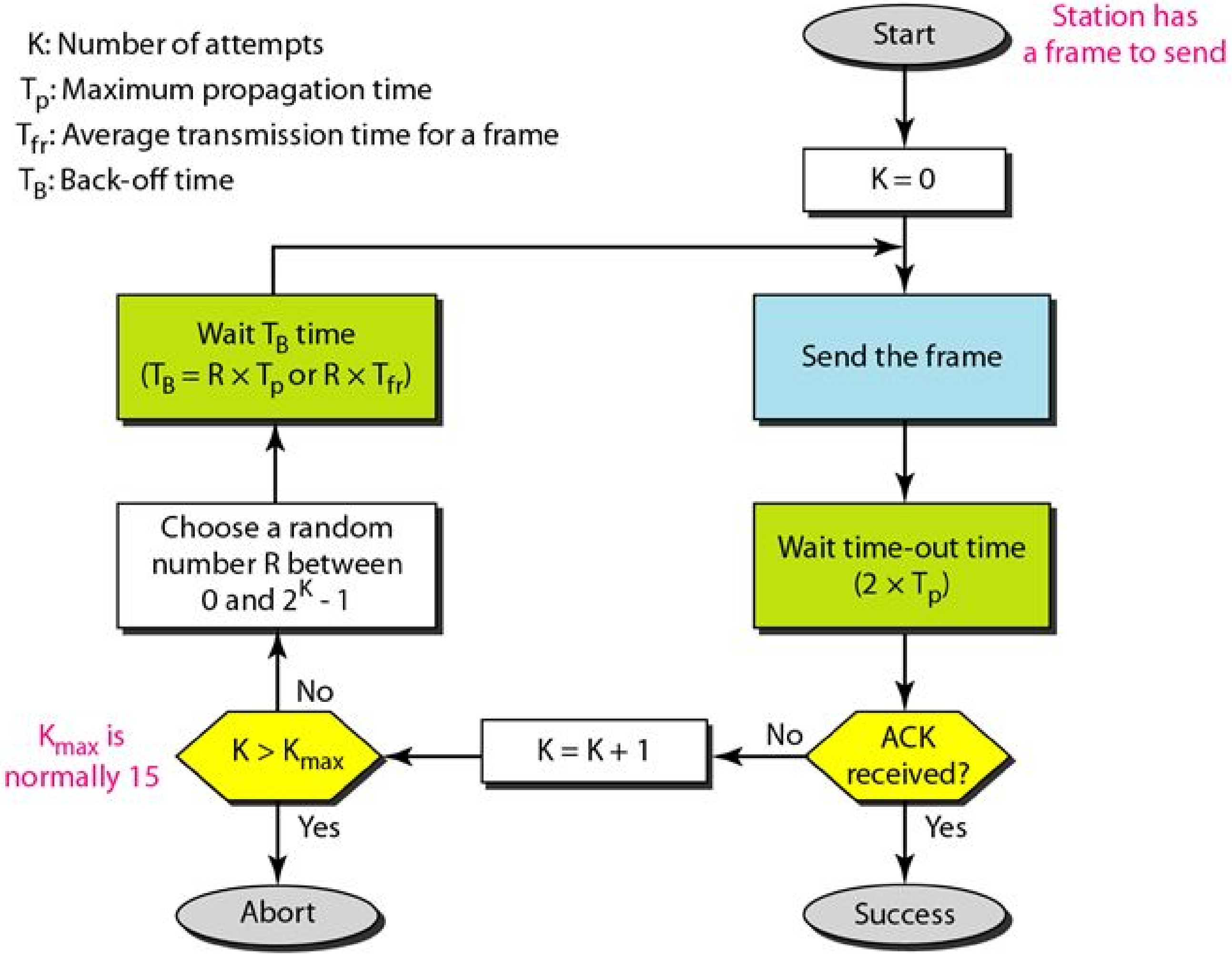

K: Number of attempts
T_p: Maximum propagation time
T_fr: Average transmission time for a frame
T_B: Back-off time
Start
Station has a frame to send
K = 0
Wait T_B time (T_B = R × T_p or R × T_fr)
Send the frame
Choose a random number R between 0 and 2^K - 1
Wait time-out time (2 × T_p)
No
K_max is normally 15
K > K_max
K = K + 1
No
ACK received?
Yes
Yes
Abort
Success

Figure 9: Flowchart for Aloha

The procedure of choosing a random number between 0 and $2^K - 1$, incrementing the value of K, and waiting an amount of time based on R, is called the **backoff procedure.**

Slotted Aloha

- Same as Aloha, but time is divided into slots.
- Frames can be sent *only at the beginning* of a time slot.
- Vulnerable Time = T_t

Efficiency of Slotted Aloha

$$S = G.e^{-G}$$

$S_{max} = 0.368$ at $G = 1$.

CSMA (Carrier Sense Multiple Access)

- Each node will sense the medium before sending.
- If the medium is idle, send the data. Otherwise wait.
- Collisions may still occur due to propagation delay.

For e.g, if A sent a frame at 12:01, and propagation time from A to D is 2 minutes. If D checks the medium at 12:02, it will find it idle and send the frame. A's frame and D's frame will then collide.

- Vulnerable time = T_p (Max propagation time).

Persistence Methods for CSMA

Persistence methods decide when and how to send data after sensing medium.

1-Persistent

- Continuously Sense the medium.
- As soon as the medium is idle, send the data immediately.

Non-Persistent

1. If medium is idle, send the data immediately.
2. If medium busy, wait a random amount of time, then sense the medium again and repeat from step 1.

- Less efficient, as the channel may remain idle in the random waiting time.
- Less chance of collision.

P-Persistent

Uses a value p that is fixed by the network administrator for each node. Different nodes have different values of p

1. Continuously check medium till idle.
2. If idle:
    - Generate random number r
    - If $r < p$, then transmit the data.
    - Else:
        - Wait for a time slot, and then check the line.
        - If line is idle, go to step 2.
        - If the line is busy, act as if a collision occurred, and follow the backoff procedure.

CSMA with Collision Detection (CSMA/CD)

- Continue sensing medium for time= $2 * T_p$ after transmission.
- This will help us sense potential collisions.
- If a collision is detected, the node that detected the collision will send a jamming signal to the access medium. After that, backoff procedure will be applied.
- Condition to detect collision:

  $$T_t = 2 * T_p$$

  The transmission time for the frame should be long enough that we can detect collisions while we are transmitting it.
- Efficiency of CSMA/CD is

  $$\eta = \frac{1}{1 + 6.44a}$$

  where $a = T_p/T_t$.

Network Layer

- Moves packet from source to destination
- Uses routers, bridges, switches.
- Deals with IP addresses.

Performs the following functions:

- Routing
- Fragmentation
- Congestion Control

IP addresses have two parts - network ID, and host ID. Network ID represents which network the IP address is part of, i.e, which organization controls it. Host ID represents which *computer* (or mobile,printer,etc.) in that network the IP address belongs to.

For e.g, Google has many servers. They will all have the same Network ID, but different Host IDs.

Classful Addressing

IPv4 addresses were divided into various classes for easier addressing purposes.

An IPv4 address is a 32-bit address. For ease of representation, it is divided into 4 octets. Each octet contains 8-bits.

(8)(8)(8)(8)

It allows us to write the IP address in the following format - 192.168.10.1 Here, 192 is the first octet, 168 is the second, and so on. All are represented in decimal. This is called Dotted Decimal Representation.

Class A

In this, the first bit of the first octet is always fixed as 0.

```
0 _ _ _ _ _ _ _ . (octet 2) . (octet 3) . (octet 4)
```

The range of the first octet becomes `00000000` to `01111111`, i.e. 0 to 127.

The 1st octet represents the Network ID (7-bits).

The 2nd, 3rd and 4th octet represent the Host ID (24-bits).

Number of IP Addresses in Class A = 2^{31}, which is half of all IPv4 Addresses in the world.

Total number of Networks = $2^7 - 2 = 126$. Range is [1,126].

The network IDs 00000000 and 01111111 are reserved and unused. They aren't given to any organization.

Total number of hosts in each Network = $2^{24} - 2$

Host with values 0.0.0 and 255.255.255 (first and last hosts) are also reserved and unused.

The host with value 0.0.0 is used to represent the entire network. So, for a network with networkID = 28, 28.0.0.0 is used to represent the entire network (and not any particular host in the network).

Host with value 255.255.255 is used to represent Direct Broadcast Address. If anyone wants to send a particular message to *all* the hosts in a network, they will use this. For a network with networkID =28, 28.255.255.255 represents its direct broadcast address.

Default Mask for class A is 255.0.0.0

For any IP address, performing bitwise AND of the IP Address, and the Default Mask for its class will give us the network that IP address belongs to.

For eg, let the IP address be 28.12.34.1

First bit of first octet is 0, so we know it's a class A Address.

Bitwise AND with 255.0.0.0

```
00011100.00001100.00100010.00000001 (28.12.34.1)
11111111.00000000.00000000.00000000
-----------------------------------
00011100.00000000.00000000.00000000

= 28.0.0.0
```

28.0.0.0 represents the network that the IP address 28.12.34.1 belongs to.

Class B

First octet has first 2 bits fixed as `10`

`1 0 _ _ _ _ _ _ . (octet 2) . (octet 3) . (octet 4)`

First 2 octets represent the Network ID.

Last 2 octets represent the Host ID.

Range is (128-191)

Number of IP Addresses = 2^{30}, 25% of all IPv4 addresses in the world.

Number of networks = 2^{14}

Number of hosts in each network = $2^{16} - 2 = 65534$

First and last hosts are excluded here, as they were in class A.

Class A excluded first and last networks also. Class B excludes first and last hosts only. All networks in Class B are available for use.

Default Mask - 255.255.0.0.

Class C

First octet has first 3 bits fixed as `110`

`1 1 0 _ _ _ _ _ . (octet 2) . (octet 3) . (octet 4)`

First 3 octets represent the Network ID.

Last octet represent the Host ID.

Range is (192-223)

Number of IP Addresses = 2^{29}, 12.5% of all IPv4 addresses in the world.

Number of networks = 2^{21}

Number of hosts in each network = $2^{8} - 2 = 254$

First and last hosts are excluded here, as they were in class A.

Class A excluded first and last networks also. Class C excludes first and last hosts only. All networks in Class C are available for use.

Default Mask - 255.255.255.0.

Class D

First Octet has first 4 bits reserved as `1110`

`1 1 1 0 _ _ _ _ . (octet 2) . (octet 3) . (octet 4)`

Range is (224-239)

Number of IP Addresses = 2^{28}

In class D, all IP addresses are reserved. There is no network ID and no host ID.

These IP addresses are only meant to be used for multicasting, group email, broadcasting, etc.

Class E

First Octet has first 4 bits reserved as `1111`

`1 1 1 1 _ _ _ _ . (octet 2) . (octet 3) . (octet 4)`

Range is (240-255)

Number of IP Addresses = 2^{28}

All IP addresses are reserved for military purposes.

Classless Addressing

This divides 32-bit IP addresses into BlockID and HostID.

BlockID is similar to NetworkID.

Format - x.y.z.w/n

n represents the number of bits used to represent the blockID.

It also represents the number of 1-bits that the mask for this IP address should have.

For e.g - 128.225.1.1/10

This means the first 10 bits represent the block ID, and rest 22 bits specify host ID.

The mask for this network would be - `11111111.11000000.00000000.00000000`, (10 times 1, then all 0)

To find the network, we do bitwise AND of IP address and mask

```
10000000.11100001.00000001.00000001
11111111.11000000.00000000.00000000
-----------------------------------
10000000.11000000.00000000.00000000

= 128.192.0.0
```

The network that IP address belongs to is 128.192.0.0/10

Rules for Classless Addressing

- Addresses should be contiguous
- Number of addresses must be in power of 2.
- 1st address of every block must be evenly divisible by block size (number of hosts)

  Block Size = number of hosts = 2^{32-10} = $2^{2}2$ First address of block is same as network's IP address = 128.192.0.0/10 = 10000000.11000000.00000000.00000000 = 10000000110000000000000000000000

  We don't need to divide. We can just check that the last 22 (22 is number of bits used to specify host) bits are 0.

  i.e, the 22 least significant bits (LSBs) should be 0, which is true in our case.

Subnetting

Subnetting is the process of dividing a large network into smaller *subnets*. This helps in organization, security, fixing bugs, etc.

An organization may divide it's network into separate subnets for finance department, legal department, etc.

We do this by dividing the hosts.

NetworkID REMAINS UNCHANGED IN SUBNETTING, ALWAYS.

Subnetting in Classful Addressing

Suppose we have a network with the IP `200.10.20.0`. This is a Class C IP address.

It has a total of 254 hosts. We want to divide it into 2 equal subnets.

- Network ID is 200.10.20. This will remain unchanged.
- The last octet is used to specify the host. Let's say we want 2 subnets - S_1 and S_2.
- We will subnet by fixing the first bit in the last octet as either 1 or 0.

S_1

For the last octet, we will prefix the first bit for S_1 as `1`.

S_1 will have IP addresses of the form `200.10.20.1 _ _ _ _ _ _ _`.

Range => 200.10.20.128 - 200.10.20.255

Usable IP addresses $= 2^7 - 2 = 126$

Network IP address for S_1 is 200.10.20.128.

Broadcast IP address is 200.10.20.255

S_2

For the last octet, we will prefix the first bit for S_2 as `0`.

S_2 will have IP addresses of the form `200.10.20.0 _ _ _ _ _ _ _`.

Range => 200.10.20.0 - 200.10.20.127

Usable IP addresses $= 2^7 - 2 = 126$

Network IP address for S_1 is 200.10.20.0.

Broadcast IP address is 200.10.20.127

Thus, if the network receives a packet meant for 200.10.20.50, it will be sent to S_2.

A packet meant for 200.10.20.165 will be sent to S_1.

Subnet Mask

Since we are indirectly using 1 extra bit to specify Network ID now, our subnet mask will also have 1 extra bit added to it.

Class C's Subnet mask is 255.255.255.0

Now, it will be `255.255.255.1 _ _ _ _ _ _ _`.

We added 1 in the place where we prefixed an extra bit.

Subnet mask for our network is now 255.255.255.128

Our total number of usable IP addresses has become 126+126= 252. Earlier, it was 254.

Due to subnetting, we have lost the use of 2 IP addresses.

Number of usable IP addresses = Number Of Original Usable IP Addresses - $n * 2$

where n is the number of subnets.

Subnetting in Classless Addressing

The procedure is mostly the same as for subnetting in classful addressing. The only extra thing that needs to be done is to change the value of n.

For.eg,

128.192.0.0/10 - let this be our network.

First 10 bits specify block. Last 22 bits specify host.

10000000.11000000.00000000.00000000. The bold part represents block ID.

To subnet into two equal subnets, we fix the first host-bit.

Subnet 1 will have IP addresses of the form 10000000.111*0*_ _ _ _ ._ _ _ _ _ _ _ _ ._ _ _ _ _ _ _ _. (We prefixed 0)

Subnet 2 will have IP addresses of the form 10000000.111*1*_ _ _ _ ._ _ _ _ _ _ _ _ ._ _ _ _ _ _ _ _. (We prefixed 1)

To find out IP Addresses for Subnet1 and Subnet2, we will have to increase the value of n by 1, since now we are using an extra bit to find out block ID.

Subnet1 IP Address = 128.192.0.0/11

Subnet2 IP Address = 128.224.0.0/11

Accordingly, mask will be changed.

Variable Length Subnet Masking (VLSM)

In case we want subnets of different sizes, we use this technique. It works for both classless and classful addressing in similar ways. We'll explain using classful here.

Suppose we have a network with IP 200.10.20.0.

We want to divide it into 3 networks - S_1, S_2, S_3. S_1 should have 50% of all hosts. S_2 and S_3 should have 25% each.

We will subnet in the following way:

S_1

IP Addresses of the form - 200.10.20.0 _ _ _ _ _ _ _ (1 bit prefixed)

Range - 200.10.20.0 - 200.10.20.127

Usable IP Addresses - 126

S_2

IP Addresses of the form - 200.10.20.1 0 _ _ _ _ _ _ (2 bits prefixed)

Range - 200.10.20.128 - 200.10.20.191

Usable IP Addresses - 62

S_3

IP Addresses of the form - 200.10.20.1 1 _ _ _ _ _ _ (2 bits prefixed)

Range - 200.10.20.192 - 200.10.20.255

Usable IP Addresses - 62

For S_1, we prefixed only 1 bit. For S_2 and S_3 we prefixed two bits. We can see that the S_1 has roughly double the number of IP addresses that S_2 and S_3 each have.

We divided the network into two equal halves - S_1, and let's call the other part S'.

Then, we further divided S' into S_2 and S_3.

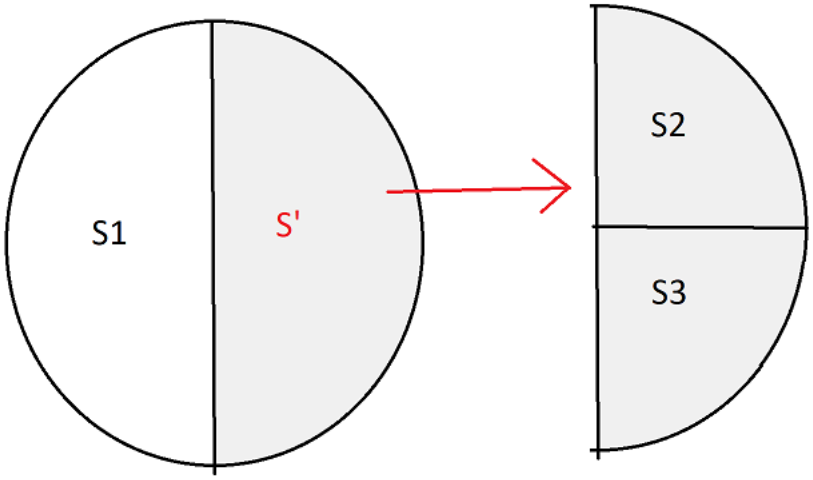


Figure 10: VLSM

Header Formats for IP Protocols

Whenever a packet is sent using IP (IPv4 or IPv6), it includes data (payload), as well as a header. The header doesn't contain the actual data, but it contains metadata such as destination, priority, source, etc.

IPv4 Header Format

- IPv4 is a connectionless datagram service.
- Header size is between 20-60 bytes.
- The total datagram size is at max 64 KB, or 65535 bytes.
- The payload size is maximum 65515 bytes. This happens when the header size is 20 bytes. In case the header size is larger than 20, the max payload size will be decreased accordingly.

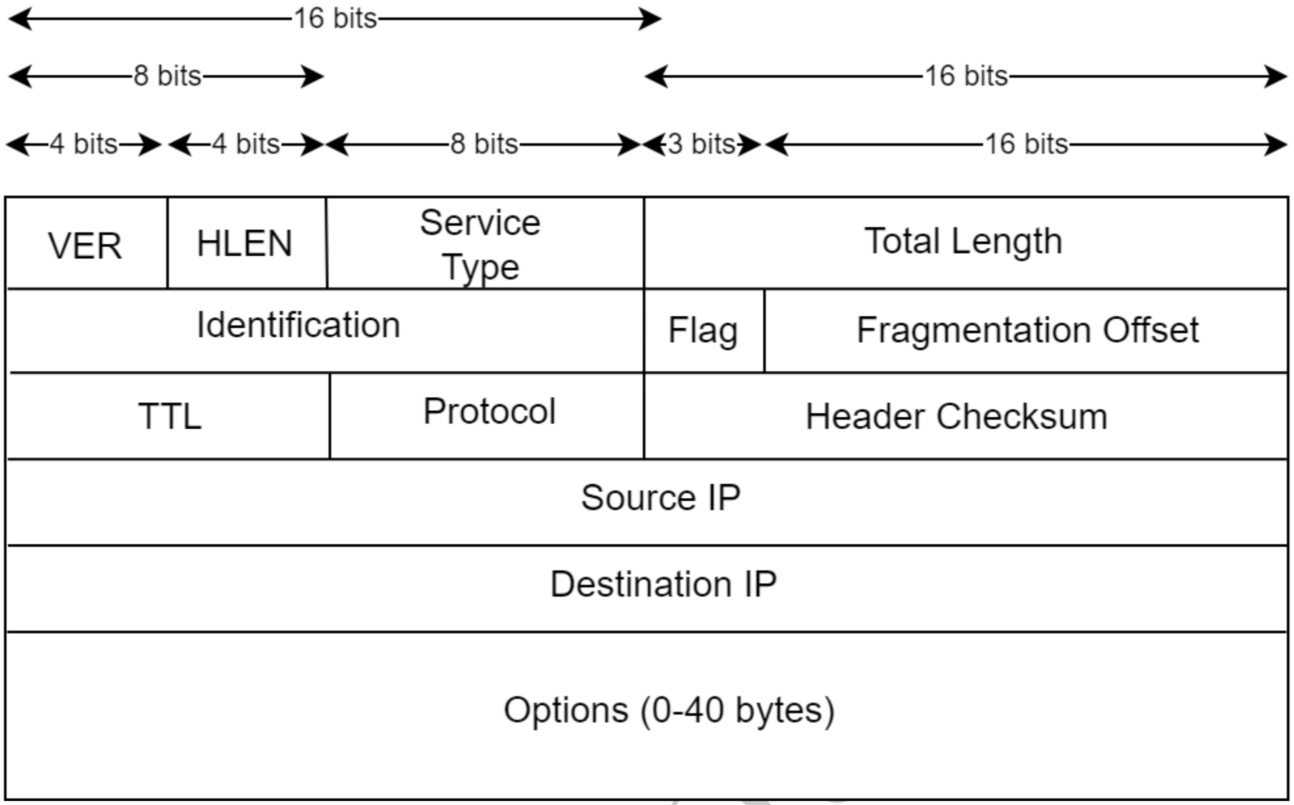


Figure 11: IPv4 Header Format

VER

- Stands for version. It is a 4-bit value. It contains the value of version, i.e, which IP version is being used.
- Almost all transmissions either use IPv4 or IPv6.
- Version for IPv4 is 4 = 0100
- Version for IPv6 is 6 = 0110

HLEN

- Contains the header-length.
- 4-bit value.
- Uses a factor of 4.

Header size = HLEN*4.

As the minimum header size for IPv4 is 20 bytes, HLEN can never be 0,1,2,3 or 4.

Type of Service

- Also known as DSCP (Differentiated Service Code Point)
- 8-bit value
- Contains different values specifying the type of service we wish to use.

The 8 bits are:

`[P][P][P][D][T][R][C][0]`

The first 3 bits (P) are used to set the precedence, or priority of the packet.

D - Delay.

0 means normal delay. 1 tells router this packet needs minimal delay.

T- Throughput

0 - normal

1 - maximize

R - Reliability

0 - normal

1 - maximize

C - Cost

0 - normal

1 - minimize

The last value is reserved as 0. It's fixed for future use. Only one bit out of D,T,R and C can be 1 in a packet. More than 1 cannot be 1. For eg - 0011 is not valid.

Total Length

Contains total length of the packet. 16-bit value.

TTL (Time to Live)

- 8-bit value
- Source sets it to max value (255), or it can also be set as (max number of routers between source and destination)*2.
- At each node it encounters (router/switch/etc.), the value is reduced by 1.
- When it becomes 0, the packet is dropped.
- This helps in case a packet is getting stuck in loops in the network, causing congestion.

Protocol

- 8-bit
- Tells which protocol is being used, TCP,UDP, etc.

Header Checksum

- Contains the checksum value for the header.
- 16-bit
- Only IP header fields are used while calculating checksum, actual data isn't used. This is because higher level protocols such as TCP and UDP use their own checksums for the data, so it isn't required for IP to do it as well.
- Since fields like TTL can change, header checksum is recalculated at each router.

Source IP

- Contains IP address of the source
- 32-bit

Destination IP

- Contains IP address of destination.
- 32-bit

Fragmentation is done for packets that are larger in size than the permitted size.

The packet is broken down into many smaller packets, then sent over the network. It is reassembled at the source.

The IPv4 header contains values to identify the fragments.

Identification Bits

This is a 16-bit unique packet ID. It identifies a group of fragments that belong to a single IP datagram.

Flag

3-bit value.

`[R][D][M]`

The first bit is reserved as 0.

The second bit (D) stands for **Do not Fragment**. If this bit is 1, no node will try to fragment this bit. But, if some node doesn't allow a packet of a large size, and we set D=1, the node may drop that packet altogether.

The third bit (M) stands for **More Fragments**

If M=0, either this packet is the last packet in its datagram, ot it's the only fragment.

If M=1, it means more fragments are coming after this packet.

Fragment Offset

This represents the number of data bytes ahead of this particular fragment, i.e, the position of this fragment in the original unfragmented datagram.

It uses a factor of 8.

I.e, number of bytes ahead = Fragment Offset * 8.

For eg, if a datagram is broken into 4 fragments of 80 bytes each (excluding the header length).

Fragment offset value for

1st fragment - 0 (No bytes ahead of it)

2nd fragment - 10 (80 bytes of data ahead of it. 80/8=10)

3rd fragment - 20

4th fragment - 30

Options

These contain optional headers and metadata. For example, they may be:

Record Route

Tells the nodes to record the route this packet has taken in the header. Can record upto 9 router addresses.

Source routing

The source defines the route that the packet will take.

Users cannot set source routing, only routers are allowed to do this.

Padding

This is added in case the header size is not in multiple of 4.

We need to store the header length in HLEN, which uses a factor of 4.

So, if the header length is 21, we will add 3 bytes of padding to make it 24. Then HLEN will store 24/4 = 6.

IPv6 Header

- IPv6 uses 128-bit IP addresses instead of 32-bit.
- Only source can fragment packets. Intermediate nodes cannot.
- Header length is fixed at 40 bytes.

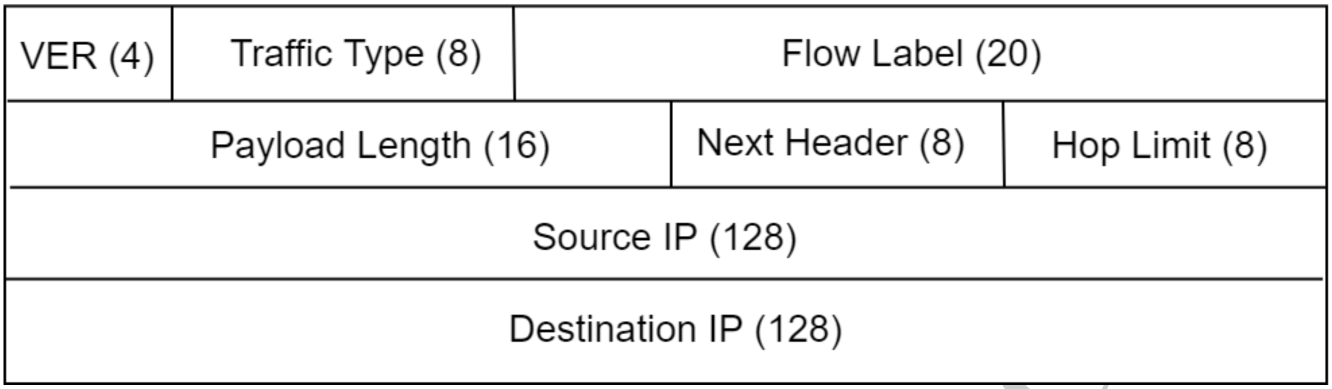


Figure 12: IPv6 Header. Length of fields written in brackets.

VER is the same as in IPv4.

Traffic Type is the same as Type of Service in IPv4.

Flow Label

20-bit value.

For continuous data that travels in a flow, such as video streaming, or live updates, we use flow labels. Even large files may be sent using flows. A source and a destination may have multiple flows occurring between them.

For e.g, if you're downloading two files together from Google Drive, they may be sent through 2 different flows. So, we give them different flow labels to help identify which flow is which.

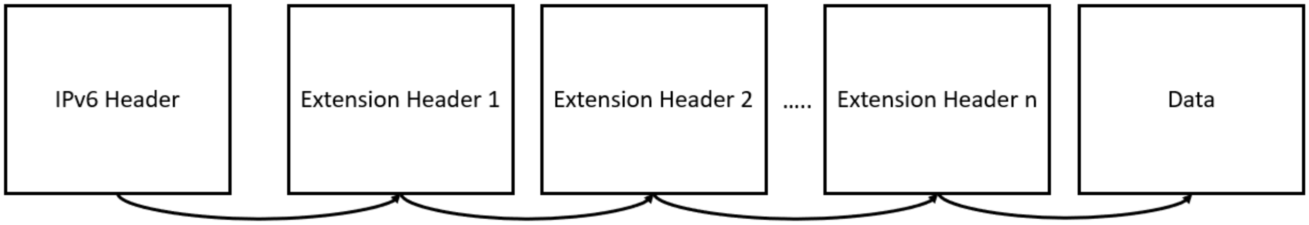
Payload Length

Length of the payload

Next Header

IPv6 uses extension headers instead of options. Metadata for routing, authentication, fragmentation, etc. are set in special extension headers.

This field is a 8-bit field that contains the type of the extension header (if present), that comes immediately after the IPv6 header. Each extension header contains its own "Next Header" field. The extension headers are thus chained together like this.



In some cases, this field is also used to indicate protocols in upper layers, such as TCP or UDP.

Hop Limit

Same as TTL in IPv4 Header

Source IP and **Destination IP** are 128-bit IPv6 addresses for the source and destination respectively.

Extension Headers

Extension headers may be used for many purposes. Some common extension headers used in IPv6 are:

- Routing headers, used if source wants to determine the route the packet should take.
- Authentication headers, used for security purposes.
- Fragmentation Header, used for fragments.

Routing Protocols

Routing Protocols are used to decide the route a particular packet will take to its destination.

Distance Vector Routing (DVR)

- Each node maintains table of minimum distance to *every* other node in the network.
- Information is shared only between neighbors (directly connected). Each node will share its routing table to its immediate neighbours.
- Update may be **periodic** or **triggered**.
- Count to Infinity problem may occur in this.
  - Three nodes - X—A—B. X-A = 1, A-B=1, X-A=2
  - If link between X and A breaks, then the following problem will occur. A will set its distance to X as infinity.
  - B will send its routing table to A. A will think B has found another path to X with cost 2. It will update its routing table so that X-A=2+1=3
  - A will send new table to B. B will think the cost to get to X (through A) has increased to 3. It will update its table so that X-B=4.
  - Similarly, B will again send the new table to A. A will update the X-A value to 5. This process will continue forever.
- Bellman Ford is used to calculate distance tables.

Link State Routing

- Router sends information about its neighbors to the entire network through flooding.
- Uses Dijsktra to calculate routing tables.
- Hello messages are sent to discover neighbor nodes.

Transport Layer

Responsibilities

- **Port to Port Delivery/ Process to Process Delivery.** It must deliver the data from the sender application (for e.g, a browser) to the receiver application (the server of the website the user opened).

- **Segmentation** - Break down the data into smaller parts that can be sent over the network, and reassemble it at the receiver side.

- **Multiplexing and Demultiplexing** - A single device generally only has a single connection to the Internet (or any other network). If multiple applications are using that connection, multiplexing and demultiplexing of data is done.

- **Connection Management**

- **Reliability** - All the data must be delivered to the receiver correctly. No lost/corrupted data should be delivered. Lost/corrupted data must be detected and resent. **UDP doesn't provide reliability.**

- **Order** - Data must be sent in order. If the data was broken down into 4 parts, they must arrive in the correct order. **UDP doesn't provide in-order delivery.**
- **Error Control** - Checksums are used for this. Receiver verifies the checksum that the sender sent.
- **Congestion Control**
- **Flow Control**

Socket Address and Port Numbers

Socket Addresses are made up of an IP address, and a 16-bit port number. They are used to uniquely identify a TCP/UDP/ any other transport layer protocol connection.

A particular computer will have an IP address, but many applications running on it may need to access the Internet. When data packets arrive, the OS must be able to figure out which data packet belongs to which application.

A unique port number is assigned to each application. Only one application may use a particular port at a given time. For e.g, two applications cannot listen on port 3000 at the same time.

Port Number Types

There are three categories of port numbers:

1. **Well Known/System Ports** - These are the ports for most commonly used networking tasks. For example, a web-server will always listen for HTTP requests on Port 80, and HTTPS on port 443. Typing www.google.com in a browser is the same as typing www.google.com:443, because the browser knows that the standard port for HTTPS is 443.

   Range for these is 0-1023.

2. **Registered/User Ports** - Organizations and Applications can have specific ports reserved for their use. They can register these with IANA (Internet Assigned Numbers Authority). For e.g, Xbox Live has 3074 port number reserved for it.

   Range for these is 1025-49151.

3. **Dynamic/Ephemeral Ports** - These are the rest of the ports left in the range. When an application needs to use a transport layer protocol, it requires a port number. The OS will generally assign it any random port number that is currently not in use.

   Range for these is 49152-65535

IP Address is used to differentiate one machine from another. Port Numbers are used to differentiate different applications on the same machine.

TCP (Transmission Control Protocol)

TCP (Transmission Control Protocol) is a popular transport layer protocol. Most applications, such as HTTP, HTTPS, Email (SMTP/IMAP), etc. use TCP.

Characteristics

Characteristics of TCP are:

1. **Byte Streaming** - Application layer continuously sends data to the transport layer. TCP breaks it down into bytes, and packages several bytes into a single **segment**. Multiple segments are created and sent to the receiver.
2. **Connection-Oriented** - TCP establishes a connection with the receiver first, using a 3-way handshake. All future communication between sender and receiver occurs over this connection.
3. **Full Duplex** - Two-way communication can happen, at the same time.
4. **Piggybacking** - Sender sends data to receiver, and receiver must send back acknowledgement for that data. In most cases these days, communication occurs both ways, i.e, if A sends data to B, B also sends data to A.

Thus, instead of B sending acknowledgements separately to A, B will attach the acknowledgement along with the data it has to send A.

5. **Needs buffers** - Sending and receiving processes may not operate at same speed, therefore TCP needs sender and receiver buffers for storage.

TCP Header

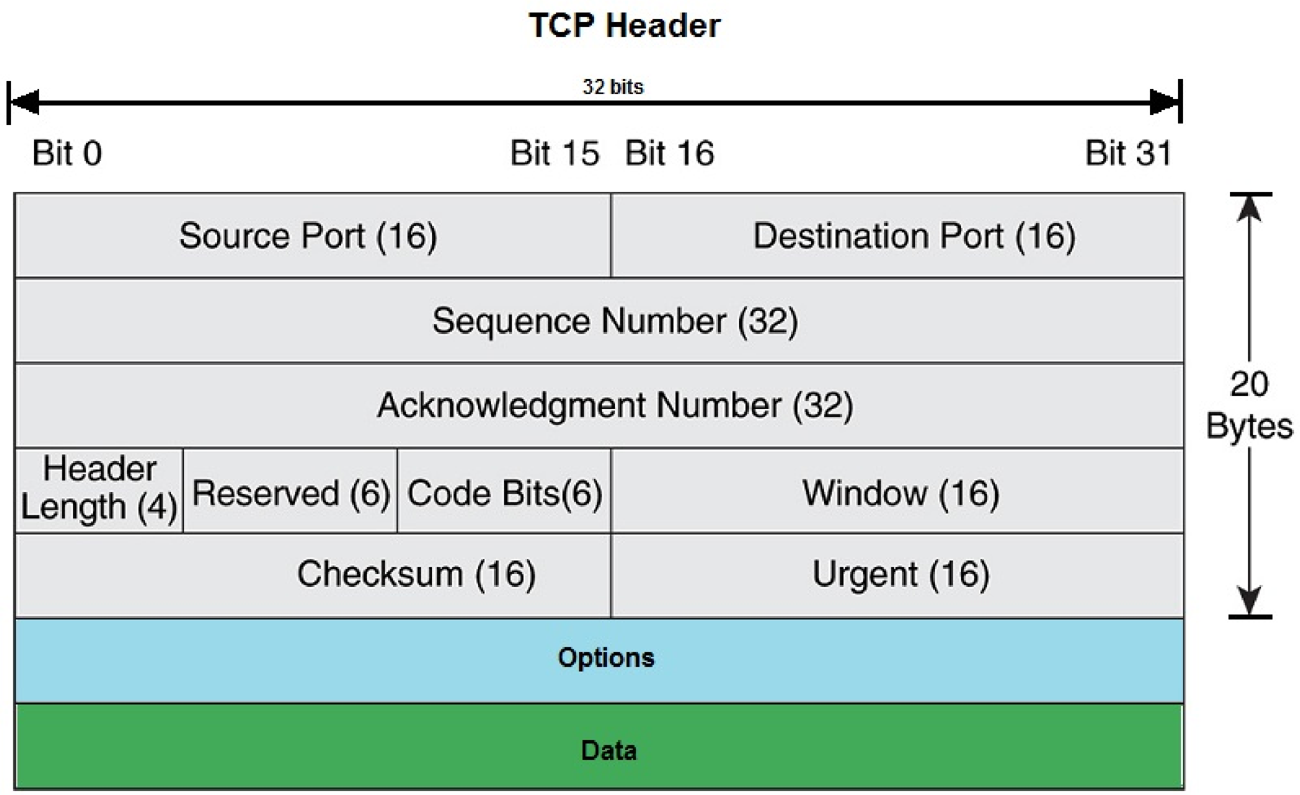


Figure 13: TCP Header

TCP Header is attached to each TCP Segment. It contains the following fields. It's size can vary from 20-60 bytes.

Source Port

Port Number for the source (sender) application.

Destination Port

Port Number for the destination (receiver) application.

Sequence Number

Each segment is given a sequence number based on the data sent before it. When a TCP connection is established, a random sequence number is generated.

For e.g, let the initial sequence number be 500.

Let's say A sends B 200 bytes of data, in 4 segments of size 50 bytes. The sequence number will be 500 for the first segment, 550 for the second segment, 600 for the third, and so on.

Acknowledgment Number

This is used to let the sender know that the packet that it sent has been received. It is set as the value of the next sequence number that the receiver is expecting.

For e.g, if A sends B data with sequence number 500, that has 50 bytes in it. Thus, B receives bytes number 500,501,502....549. Now, it expects byte number 550. Thus, it will set Acknowledgement Number as 550.

Header Length/Data Offset

Length of the TCP Header. Uses a factor of 4, same as IPv4 Header's HLEN field

Reserved

Next 6 bits are reserved for future use. 2 of them have already been defined as CWR and ECE.

Code-Bits/Flag Bits

There are 6 Flag Bits, each with their own purpose.

URG (Urgent)

If this bit is set to 1, it means that this segment contains urgent data. Location of the urgent data is set in the urgent pointer.

ACK

Indicates that this message contains an acknowledgement, i.e, the value of the acknowledgement number is significant.

PSH (Push)

When receiver is receiving data, it will generally buffer some amount of data before sending it to the application layer. PSH field indicates that the receiver should stop buffering and push whatever data it has to the application layer.

RST (Reset)

Used to Reset the TCP connection.

SYN (Sync)

Used to sync sequence numbers. Only the first packet sent from each end should have this value set.

FIN (Finish)

Used to terminate the connection.

Window Size

Specifies the size of the receiver's window, i.e, the current amount of data it is willing to receive. A will tell its window size to B, and B will tell its window size to A.

Checksum

Used for error-checking.

Urgent Field

If the segment contains urgent data, this field tells *where* the urgent data is located. It contains the sequence number of the *last* urgent byte. For eg, if A sent bytes number 500-549, and bytes 500-520 are urgent, the urgent field will contain the value 520.

Options

Contains optional fields, such as timestamps, window scale, and **maximum segment size (MSS)**. MSS is the maximum size of **one** single segment that the receiver is willing to accept. It is separate from window size, which may be larger, as a window can contain multiple segments.

Padding

In case the total header size is not a multiple of 4, we add empty zeroes to make it so, so that we can store it in the header length field.

TCP Connection Establishment

A 3-way handshake is used to establish a TCP connection. This process occurs before any actual data is sent. The 3 steps are:

1. **SYN** - Sender will send the receiver a connection request. It will send a randomly generated sequence number,its port number, its window size, etc. It will set the SYN flag as 1, indicating that it wants to set up a connection.

   Let's say A sent a connection request to B, with the sequence number 3000 (random), and window size as 1200 bytes. This will let B know that A only can only receive 1200 bytes of data (until it empties its buffer again)

2. **SYN-ACK** - The receiver will respond to the sender's request. It will send a response, with SYN field as 1. It will also send an acknowledgement (in the same response), and tell its own window size to the sender.

   B will reply to A. It will generate a random sequence number, say 5000. It will set the ACK and SYN flag. It will also set the acknowledgement number to 3001 (since A's sequence number was 3000.) It sends A its window size, say 800 bytes.

3. **ACK** - Sender will acknowledge the response. After this, sender and receiver will begin exchanging actual data. SYN flag is 0 in this.

   A will send B a response with sequence number 3001, and Acknowledgment number 5001. ACK will be set as 1. SYN flag will be 0.

   Sequence numbers are not consumed if PURE ACK is sent. If a segment contains only ACK, and not data, and it uses sequence number x, then the next segment can also use sequence number x.

After this, A and B can start exchanging data. Both A and B will reserve some resources (memory, RAM, etc.) for this A-B TCP connection.

A will not send more than 800 bytes to B, and B will not send more than 1200 bytes to A. A will send sequence numbers 3002,3003,3004. . . ., and B will send sequence numbers 5001,5002,5003. . . and so on.

TCP Connection Termination

4-way handshake.

1. **FIN from Client**- Client wants to close the connection. Client will send server a segment with FIN bit as 1. (Server may also choose to close the connection)

   Client will enter **FIN_WAIT_1** state. In this state, the client waits for an ACK from the server for this FIN segment. This is also called **Active Close** state.

2. **ACK From Server**- Server will receive the FIN segment, and send an ACK to the client. Server now enters a **Close Wait (Passive Close)** state. Server will release any buffer resources, because client has said it doesn't want to send any more data to server. (Server may still have data to send to the client).

   When the client receives the ACK from server, it enters **FIN_WAIT_2** state. In this state, the client is waiting for the server to send a segment with FIN bit set as 1 (i.e, client is waiting for server to also close the connection.)

3. **FIN from Server** - Server can send any pending data, and then it will send a segment with FIN bit as 1. Server now enters **LAST_ACK** state. In this state, the server only expects to receive one last ACK from the client (for the FIN segment server just sent). After receiving the ACK, Server will release all resources for this connection, and the connection will be closed.

4. **ACK from Client** - Client will receive the server's FIN segment, and send an ACK for it. Client will enter **TIME_WAIT** state. In this, the client waits in case the final ACK was lost. If the final ACK was lost, the server will timeout and resend the FIN message. If the client receives any FIN message in the **TIME_WAIT** state, it will resend the ACK. If it doesn't, client assumes the last ACK was successfully delivered, and it will close the connection.

   The amount of time to be waited varies, but it's generally 30s or 1 min.

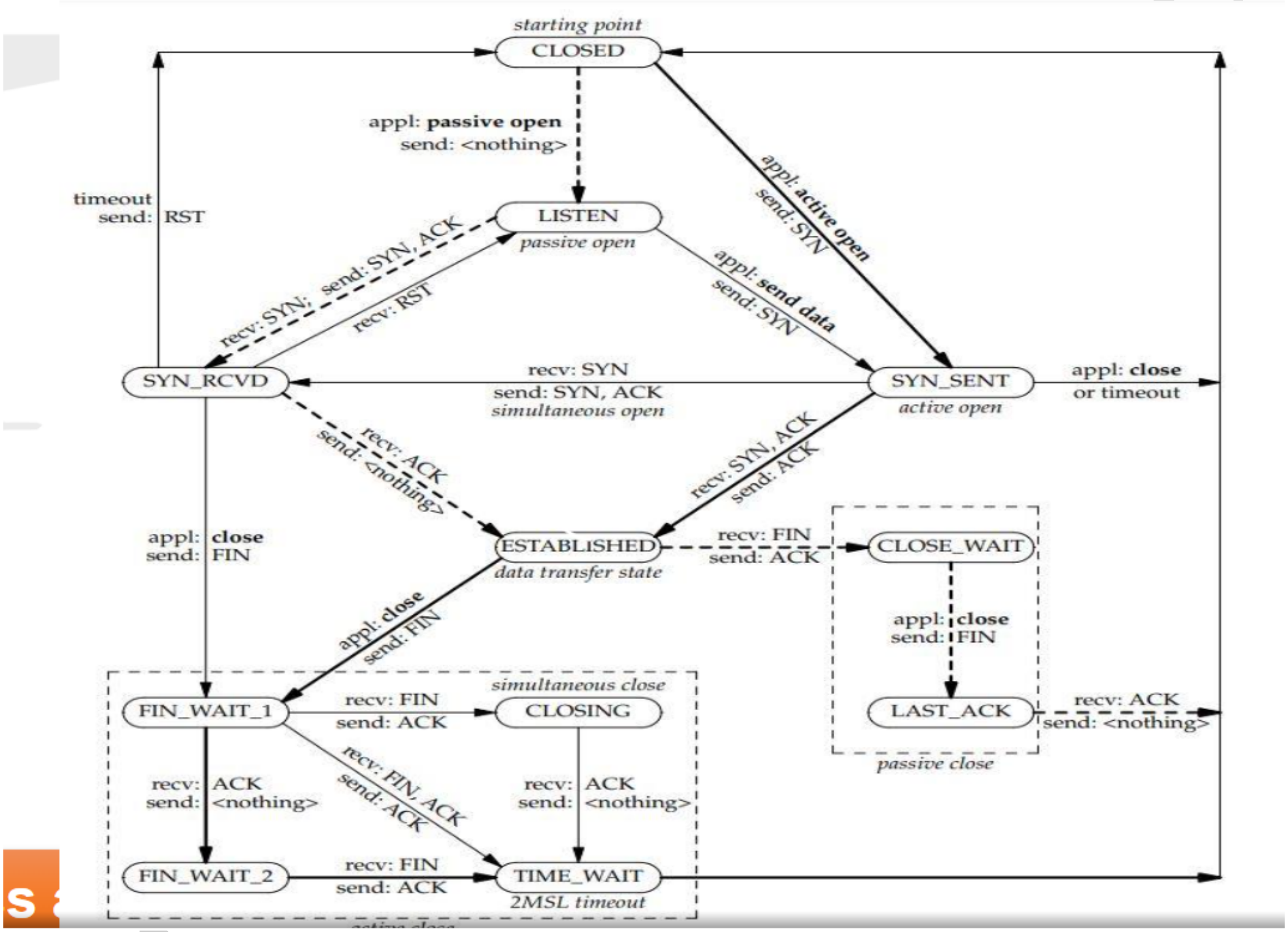


Figure 14: TCP State Diagram

Dashed lines are for server, solid for client.

TCP Congestion Control

Congestion Window is used for congestion control. Size of congestion window changes throughout the TCP connection. We first increase it, to send more data in less time. In case congestion occurs while increasing, we again decrease it.

Concept of MSS (Maximum Segment Size) is used.

Congestion Control in TCP has 3 phases:

1. **Slow Start Phase (Exponential Growth)** - In this, the congestion window is increased exponentially. Initially, the window size is 1 MSS. Then it becomes 2 MSS, then 4, then 8, then 16, and so on, until the *slow*

start threshold. Slow Start Threshold is determined as

$$(ReceiverWindowSize/MSS)/2$$

This gives max number of segments in slow start phase (not their size.)

2. **Congestion Avoidance Phase (Linear Growth)** - Congestion Window grows linearly. If it at x MSS , it becomes $x + 1$, then $x + 2$, and so on.

   This continues until congestion window size becomes equal to receiver window size. After that, we keep congestion window size as constant.

3. **Congestion Detection** - Congestion is detected in this phase, and we change window size to accommodate it.

   There are two ways in which congestion can be detected:

   1. **Time-Out** - When timer times out before we receive an ACK. Congestion in this case is Severe.
   2. **3-ACK**- Sender receives 3 duplicate ACKs for the same segment. Congestion in this case is light.

      For e.g, if sender sent packets 1,2,3,4 and 5. Receiver received packet 1 and sent ACK 2 (Original ACK). Packet 2 was lost. Receiver received packet 3, and again sent ACK 2 (because it hasn't received packet 2). Receiver received packet 4, and again sent ACK 2. Similarly for packet 5. Sender will thus receive 4 ACKS - 1 original, and 3 duplicate acknowledgments for packet 2.

Reaction in Congestion Detection

Time-Out

1. Slow Start Threshold is set as half of current window size. For e.g, if current window size is 16 MSS, slow start threshold will be 8.
2. Congestion window is reset to be equal to 1 MSS.
3. Slow Start Phase is resumed

3-ACK

1. Slow Start Threshold is set as half of current window size.
2. Congestion window is set equal to slow start threshold
3. Congestion Avoidance phase is resumed

TCP Timers

Retransmission Timer

- TCP start timer after each transmission. If an ACK is not received before this timer runs out, the segment is retransmitted.
- The amount of time it waits is called RTO (Retransmission Timeout)
- RTO is calculated using RTT, there are many ways to do so.

Time-Wait Timer

- Takes care of late packets
- Never close a TCP connection immediately. Wait for 2*LT, so that any delayed packets can arrive.

Keep-Alive Timer

- Used to close idle connections.
- Periodically check connections, and close them if no reply.
- After keep-alive time duration, server will send 10 probe messages with gap of 75 seconds. If no reply, the connection is closed.
- Keep-alive time duration is generally 2 hours.

Persistent Timer

- Suppose receiver's buffer is full, so it sends an ACK to sender with window-size=0
- Sender understands that it cannot send more data as receiver buffer is full, and it waits.
- Receiver processes the data in buffer and empties it. Now it has space, so it sends another ACK to server with window-size = some non-zero value. Suppose this ACK gets lost.
- Now, sender is waiting for receiver to empty its buffer, and receiver is waiting for sender to send data. This is a **deadlock**.
- To prevent this, persistent timer is use. When sender receives a packet with window size=0, it will start a persistent timer.
- After that timer goes off, it will send a probe with only 1 byte of new data. The receiver will receive this probe and send its new window size.
- If the new window size is non-zero, the sender will start transmitting data. If it is still zero, the sender will start the persistent timer again and wait.

UDP

UDP (User Datagram Protocol) is another transport layer service. It's popular applications include DNS, VoIP, etc.

Characteristics

- Connectionless
- Unreliable
- Messages may be delivered out of order.
- Less overhead, as header is very small.
- Faster than TCP.

Header

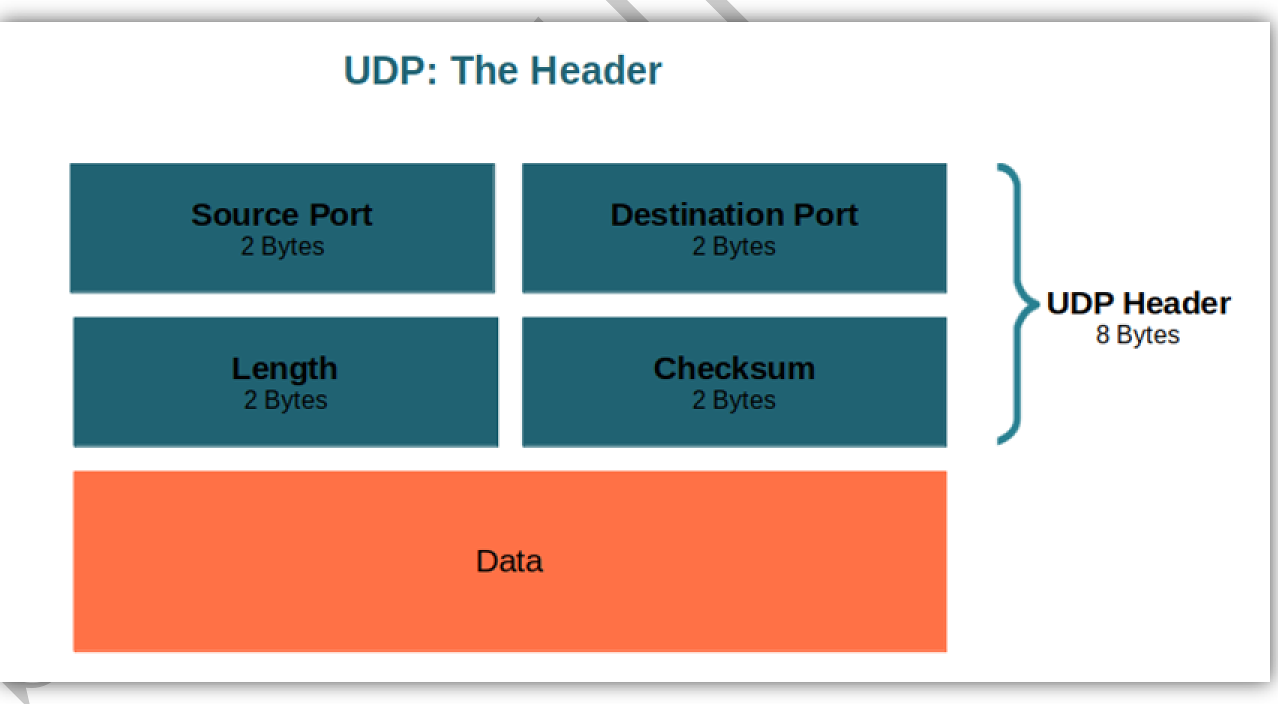


Figure 15: UDP Header

- Header size is fixed at 8 bytes.
- Length contains total length (header+data)
- Maximum length of UDP datagram is 2^{16} bytes (including header)

- Checksum field is optional in IPv4, mandatory in IPv6.

UDP Applications

- **Query-Response Protocol** (One query-One reply, no need to make a connection as we only need one reply). For e.g - DNS
- **Speed** - When we need high-speed applications. For e.g. - Online games, VoIP.
- **Broadcasting/Multicasting**- Eg, RIP (Routing Information Protocol), Distance Vector Routing. Nodes share routing tables after every 30 seconds, to *all other nodes*. If we use TCP, the node will have to establish connections with all other nodes, which will be time consuming.
- **Continuous Streaming**- E.g, Skype/YouTube.
- **Stateless** - Don't save information about the connecting clients.

TCP vs UDP

TCP	UDP
Connection-Oriented	Connectionless
Reliable	Unreliable
In-order Delivery	Delivery may be out of order
Error Control is Mandatory	Error control is optional
Slow	Fast
More Overhead	Less overhead
Flow Control, Congestion Control	No flow control or congestion control
HTTP, FTP	DNS,BOOTP,DHCP

Application Layer

Enables users (human or s/w) to access the network. It is responsible for providing services to the user.

Paradigms

- Provides services to the user
- To use the internet we need 2 application programs that communicate with each other using the application layer.
- Communication uses a logical connection, i.e, the 2 application programs assume that there is an imaginary direct connection between them. In reality, the communication happens through various layers (Transport, Network, Data-Link, etc.)

There are 2 types:

- **Client-Server** - Server program provides service to the client program. It is the most popular method today. Server runs continuously. Client creates a connection to the server using the Internet and requests a particular service.
- **Peer to Peer** - Gaining popularity in recent times. Both communicating programs have equal responsibility and power. No program needs to be always running. A computer can even provide and receive services at the same time.

File Transferring

FTP (File Transfer Protocol)

- It's an application layer protocol used for transferring (uploading and downloading) files over the Internet.
- It uses TCP under the hood.
- It solves problems such as different systems having different ways of representing and storing files.
- It establishes 2 connections between 2 hosts:
  - One connection is used for control information (commands and responses). This uses TCP port 21. It remains active during the FTP session.

  - The other connection is used for actual data transfer. This uses TCP port 20. It closes and opens for each file transfer.

Security in FTP

SSL (Secure Sockets Layer) can be added to FTP (between FTP and TCP) to make it more secure. This makes it SSL-FTP.

Email

- Used to send messages over the Internet
- Used to be plain text, now can include images, videos, files, etc.
- Actual message transfer is done using a message transfer agent (MTA). Client must have client MTA to send mail, server must have server MTA to receive mail.

SMTP

SMTP (Simple Mail Transfer Protocol) is the formal protocol that defines MTA client and server applications. SMTP defines how commands and responses must be sent back and forth. It is used twice, once between sender and sender's mail server, and second between sender's mail server and receiver's mail server. Mail is transferred in 3 phases - Connection Establishment, Mail Transfer, and Connection Termination.

Message Access Agents - POP and IMAP

- SMTP is not involved in pulling mail to client. It is only a push protocol and pushes mail from client to server.
- Message Access Agents such as POP (Post Office Protocol) and IMAP (Internet Mail Access Protocol) are used to pull messages from the server.

POP3

- Simple and limited.
- Client opens connection to server on port 110. It sends its username and password.
- It can access mail messages in delete mode or keep mode. In delete mode, once a message is downloaded from the server, the server will delete it. The user can keep a local copy. In keep mode, the server side copy will be kept intact.

IMAP4

- More powerful and complex.
- User can check email header before downloading the entire message.
- User can search email
- Partially downloadable email
- Different mailboxes, folders, hierarchy of folders can be created.

DNS

DNS stands for domain name system. It is used to convert domain names, such as google.com, to IP addresses, such as 8.8.8.8. Special DNS servers are used for this. Many of them exist due to the large number of websites on the internet that need DNS services.

DNS Server Hierarchy

Root DNS Server

This is the highest level in the hierarchy. Many of these exist, and are operated by very few organizations (around 13). When a root DNS server is asked to provide the IP of a website, it doesn't provide the IP directly. Instead, it provides the IP of the correct TLD DNS server that will contain the needed website's IP.

Top-Level Domain (TLD) DNS Server

TLDs are used for a particular domain ending - for e.g, .com, .edu, .in, etc. Each will have a different TLD server. The TLD will point the query towards the correct authoritative DNS server that contains the website's IP.

Authoritative DNS Server

Authoritative DNS servers contain a broad list of domain names and their IP addresses. The authoritative DNS server will return back a website's IP address to us.

DNS Name Hierarchy

Suppose we take the domain - www.example.com

A fully qualified domain name (FQDN) always has a . at the end, even if we don't write it. So our domain becomes www.example.com.

This can be divided into 4 parts, from right to left.

1. **The root level domain** - Represented by the dot at the end, this is the highest hierarchy level. It is used to denote the root level DNS server.
2. **TLD** - The TLD in our case is *com*
3. **Second-level domain**- This is the website's name, for e.g a business name. In our case it is *example*
4. **Sub-domain** - Here, *www* is a subdomain. We can also create more subdomains such as shop.example.com, cloud.example.com, images.example.com, etc.

Session Layer

Functions of Session Layer

- Dialog Control - allow systems to enter into full/half duplex dialog
- Managing tokens
- Synchronization

Design Issues

- Establishing sessions between 2 machines - opening, closing and maintaining a semi-permanent dialogue.
- Enhanced services (checkpoints and tokens)

RPC (Remote Procedure Call)

RPC is a protocol that works in session layer of OSI model, and application layer of TCP/IP model.

- RPC is when a client calls a service on a server (or any other network computer), as if it is calling a function on its own local system.
- Can work on TCP/UDP both, but prefers UDP.
- Uses authentication to verify client's identity.

Client -> Client Side API —– RPC ——> Server -> Local function call -> Return value to client.

- Client doesn't see the OSI layers or the network calls. To the client, it is simply calling a function.
- **Stubs** are used to convert data to different formats, as client and server may use different formats for data.
- The client stub takes the parameters for the RPC call and puts them into the message. This is called **parameter marshalling**. It also puts the name or number of the procedure to be called.
- The receiver stubs receives the message, unpacks it and gives it to the receiver application. The receiver application calls the requested procedure with the given parameters. The result is packet into a message by the receiver stub, and send to the client.
- The client stub receives the response, unpacks it, and gives it back to the client application.

Issues in RPC

- **Binding** - How does the client know who to call, what the procedure name is, etc.?

  This has 2 solutions:

  - **Dynamic Binding** - Find the server when RPC is called (at runtime)
  - **Naming and Locating** - Server offering a service exports an interface for it, and registers the interface with the system. Client must import an interface before communication.

- Different formats of data - solved by stubs.

- How to pass parameters - solved by parameter marshalling.

Presentation Layer - Security

One of the major functions of the presentation layer is security and cryptography.

Types of Cryptosystems

Symmetric

AKA conventional cryptography/shared-key systems/secret-key systems.

Sender and receiver share the same key, which is used both for encryption and decryption.

The shared key must be kept private. Anyone in possession of the key can read encrypted messages.

The notation $K_{a,b}$ is used to denote a secret-key shared by A and B.

Asymmetric

AKA Public-key cryptography.

The keys for encryption and decryption are different, but form a unique pair. The key for decryption can only decrypt the data encrypted with its pair key.

Key for encryption - K_E.

Key for decryption - K_D.

One of the keys is made public, and the other one kept private.

The notation K_A^+ is used to denote a public key belonging to A, and K_A^- denotes a private key belonging to A.

If Bob wants to send a message to Alice, he should encrypt it using Alice's public key. Since Alice is the only person who possessed the corresponding private key, only she can decrypt the message.

Types of Ciphers

Ciphers are algorithms used to change *plaintext* to *ciphertext*. Plaintext is our original information. Ciphertext is it's encrypted form. A cipher does character-to-character, or bit-to-bit transformations. A code, on the other hand, transforms entire words.

Substitution Ciphers

Each letter is replaced, or substituted, by another letter. It can also be done for groups of letters, for e.g, two at a time.

A simple example is the Caesar cipher, in which the letters of the alphabet are shifted a fixed number of positions. For e.g, if we shift by 2,

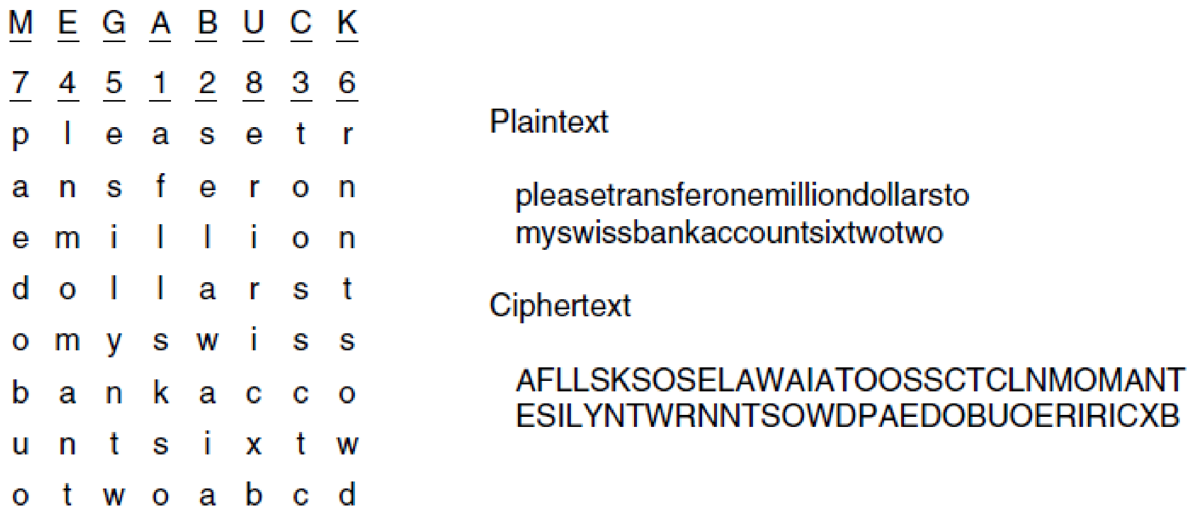
A becomes C, B becomes D, C becomes E, etc.

Modern substitution ciphers are much more complex, and tough to break.

Transposition Ciphers

In substitution ciphers, we changed the letters/bits, but kept their order the same. In transposition ciphers, we will keep the letters the same, but change their order.

One simple transposition cipher is the columnar transposition, shown here.



M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

Figure 16: image-20230506141817783

Here, MEGABUCK is the secret key that must not be shared with anyone except the receiver. We write the plaintext rowwise, with the number of columns being the length of the key.

We write the ciphertext columnwise. We start with the column whose key is lowest. Here, lower is determined by position in alphabet. Thus, we write out column A -> B->C->E->G->K->M.

RSA

An asymmetric encryption algorithm named after its inventors - Rivest, Shamir and Adleman.

Based on the fact that prime factorization of very large numbers is a difficult and time-consuming process.

Steps

1. Take 2 very large prime numbers - p and q.
2. Calculate

$$n = p * q$$

$$z = (p-1) * (q-1)$$

3. Choose d such that d is relatively prime to z.
4. Compute e such that

$$(e * d)\%z = 1$$

Now, the number d can be used for decryption, and e for encryption.

One of these is kept private, and the other is made public.

Usage

Let the message to be sent be m. Here, m is interpreted simply as a binary number.

1. Divide m into fixed length blocks, m_i, such that:

$$0 \leq m_i \leq n$$

Each m_i is also interpreted as a binary number.

2. The sender calculates
$$c_i = (m_i^e)\%n$$
All such c_i are calculated and concatenated into a single variable c.
3. c is sent to the receiver.
4. The receiver calculates
$$y_i = (c_i^d)\%n$$

Based on the properties of modulus, and the way we have chosen e and d, we can easily see that $y_i = m_i \forall i$.

This way, the receiver is able to reconstruct the message.

Properties of RSA

- RSA is secure because no method exist to (efficiently) find prime factors of large numbers.
- RSA itself is also computationally expensive, around 100-1000x slower than DES.
- It's generally used to securely share session keys, and then those session keys are used in a (faster) encryption algorithm, such as AES or DES.

Securely sending messages (Secure Channels)

Securely sending messages has the following problems to solve.

- **Confidentiality**

  No one else other than the intended recipient should be able to read the message.
- **Integrity**

  The recipient should have a way to be sure that the contents of the message weren't tampered.
- **Authentication**

  Both parties should have a way to be confident that they are sending messages to the right person.

Digital Signatures

Confidentiality and Integrity needs to be maintained in secure channels.

- Alice needs to be sure that Bob cannot alter a message and claim that Alice sent it.
- Bob needs to be able to prove that a message indeed came from Alice, and that she cannot deny having sent it.

Digital Signatures are used for this. The document is signed using the sender's public key, which uniquely ties the sender to the message.

- Alice sends a message m to Bob. She encrypts it with *her* private key to create a ***signature***. The signature and the original message are sent to Bob.
  - If she wants to keep the message content a secret, she can encrypt the entire thing using Bob's public key.
  - The message will then be $K_B^+(m, K_A^-(m))$, where $K_A^-(m)$ is the signature.
- Message arrives at Bob.
  - If it's secret, he first decrypts it using his private key.
- He decrypts the signature using Alice's public key, and matches it with m. If the decrypted signature and m match, then he can be sure the message was sent from Alice and is untampered with.

- Alice cannot claim she never sent the message, or sent a different message, because Bob has the signed version of m, and only Alice could have signed it, since only she possesses her private key.
- Bob cannot claim Alice sent a modified message, because he would have to prove that Alice signed the modified message as well.

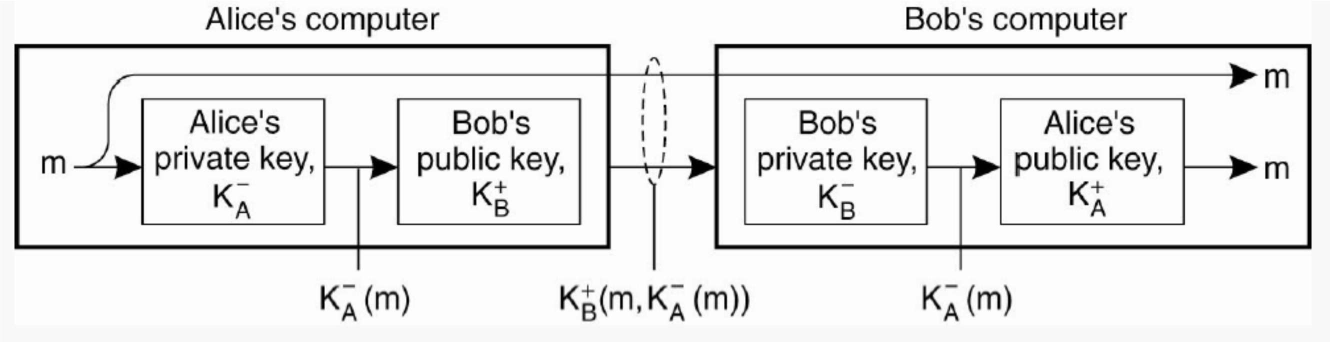

Alice's computer
Bob's computer
m
Alice's private key, K_A^-
Bob's public key, K_B^+
Bob's private key, K_B^-
Alice's public key, K_A^+
m
m
$K_A^-(m)$
$K_B^+(m, K_A^-(m))$
$K_A^-(m)$

Figure 17: Digitally signing messsages

Issues with this scheme

- This remains valid only as long as Alice's private key remains private. If the key is stolen or leaked, Alice will have to generate a new key, and all messages signed using the previous key will then become worthless.
- If the message is long, encrypting the entire message may be computationally expensive.

A solution for the second problem is a **message digest**.

Message Digest

It's a fixed length string h that's computed from a message m of arbitrary length, using a hash function H.

If m is changed to m', then it's hash $H(m')$ will not be the same as before $(H(m))$. Thus, modifications will easily be detected.

Instead of signing m, Alice signs $H(m)$, which becomes the signature.

The message sent to Bob is now $K_B^+(m, K_A^-(H(m)))$, where $K_A^-(H(m))$ is the signature.

On Bob's end, Bob will hash the entire message himself, decrypt the signature, and compare the hashes. If they match, all is good.

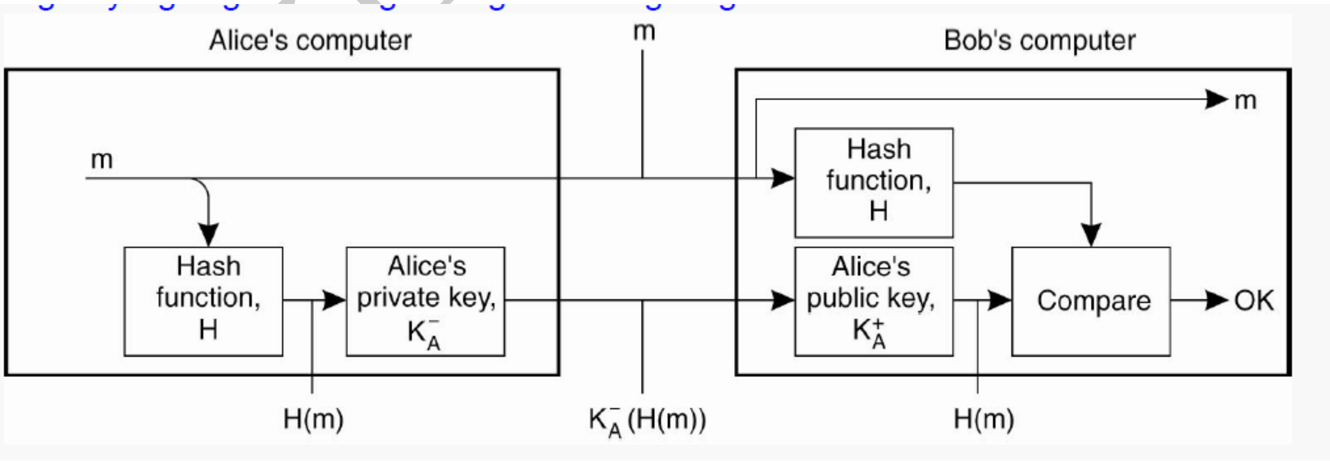


Figure 18: Digitally signing messages using digests