**TCP**

| **TCP** | **UDP** |
| --- | --- |
|  |  |  |
| **Connection** | TCP is a connection-oriented protocol. | UDP is a connectionless protocol. |
| **Ordering of data packets** | TCP rearranges data packets in the order specified. |  |
| **Speed of transfer** | The speed for TCP is slower than UDP. |  |
| **Reliability** | There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. | UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer. |
| **Header Size** | TCP header size is 20 bytes | UDP is faster because there is no error-checking for packets. |
| **Common Header Fields** | Source port, Destination port, Check Sum | There is no guarantee that the messages or packets sent would reach at all. |
| **Streaming of data** | Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries. | UDP Header size is 8 bytes. |
| **Weight** | TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. | Source port, Destination port, Check Sum |
| **Data Flow Control** | TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. | Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent. |
| **Error Checking** | TCP does error checking | UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP. |
| **Fields** | 1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port | UDP does not have an option for flow control |
| **Acknowledgement** | Acknowledgement segments | UDP does error checking, but no recovery options. |
| **Handshake** | SYN, SYN-ACK, ACK | 1. Length, 2. Source port, 3. Destination port, 4. Check Sum |
| **Checksum** | checksum | No Acknowledgment |
|  |  | No handshake (connectionless protocol) |
|  |  | to detect errors |
|  |  |  |

**IPv4 vs IPv6**

**Several fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:**

• *Fragmentation/Reassembly.* IPv6 does not allow for fragmentation and reassembly

at intermediate routers; these operations can be performed only by the source

and destination. If an IPv6 datagram received by a router is too large to be forwarded

over the outgoing link, the router simply drops the datagram and sends a

“Packet Too Big” ICMP error message (see below) back to the sender. The

sender can then resend the data, using a smaller IP datagram size. Fragmentation

and reassembly is a time-consuming operation; removing this functionality from

the routers and placing it squarely in the end systems considerably speeds up IP

forwarding within the network.

• *Header checksum.* Because the transport-layer (for example, TCP and UDP) and

link-layer (for example, Ethernet) protocols in the Internet layers perform checksumming,

the designers of IP probably felt that this functionality was sufficiently

redundant in the network layer that it could be removed. Once again, fast processing

of IP packets was a central concern. Recall from our discussion of IPv4

in Section 4.4.1 that since the IPv4 header contains a TTL field (similar to the

hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at

every router. As with fragmentation and reassembly, this too was a costly operation

in IPv4.

* *Options.* An options field is no longer a part of the standard IP header. However,

it has not gone away. Instead, the options field is one of the possible next

headers pointed to from within the IPv6 header. That is, just as TCP or UDP

protocol headers can be the next header within an IP packet, so too can an

options field. The removal of the options field results in a fixed-length, 40-

byte IP header

**Collision Domain vs Broadcast Domain-**

**Hub – 1 collision domain n 1 BD**

**Switch – Each port is a collision domain, 1 BD**

**Router – Each port is a collision domain, routers separate BD**

**PSH and the PUSH function**

When you send data, your TCP buffers it. So if you send a character it won't send it immediately but wait to see if you've got more. But maybe you want it to go straight on the wire: this is where the PUSH function comes in. If you PUSH data your TCP will immediately create a segment (or a few segments) and *push* them.  
But the story doesn't stop here. When the peer TCP receives the data, it will naturally buffer them**it won't disturb the application for each and every byte**. Here's where the PSH flag kicks in. If a receiving TCP sees the PSH flag it will immediately *push* the data to the application.

**URG and OOB data**

TCP is a stream-oriented protocol. So if you push 64K bytes on one side, you'll eventually get 64k bytes on the other. So imagine you push a lot of data and then have some message that says "Hey, you know all that data I just sent ? Yeah, throw that away". The gist of the matter is that once you push data on a connection you have to wait for the receiver to get all of it before it gets to the new data.  
This is where the URG flag kicks in. When you send urgent data, your TCP creates a special segment in which it sets the URG flag and also the urgent pointer field. This causes the receiving TCP to forward the urgent data on a separate channel to the application (for instance on Unix your process gets a SIGURG). This allows the application to process the data **out of band**.

### Selective acknowledgments[[edit](http://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&action=edit&section=14)]

Suppose a pure cumulative acknowledgment protocol if receiver cannot receives bytes 1,000 to 9,999 successfully, but failed to receive the first packet, containing bytes 0 to 999. Thus the sender may then have to resend all 10,000 bytes.

To solve this problem TCP employs the *selective acknowledgment (SACK)* option, which **allows the receiver to acknowledge discontinuous blocks of packets which were received correctly.** The acknowledgement can specify a number of *SACK blocks*, where each SACK block is conveyed by the starting and ending sequence numbers of a contiguous range that the receiver correctly received. In thi s example, the receiver would send SACK with sequence numbers 1000 and 9999. So sender retransmits only the first packet, bytes 0 to 999.

### Window scaling[[edit](http://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&action=edit&section=15)]

For more efficient use of high bandwidth networks, a larger TCP window size may be used. The TCP window size field controls the flow of data and its value is limited to between 2 and 65,535 bytes.

Since the size field cannot be expanded, a scaling factor is used. The [TCP window scale option](http://en.wikipedia.org/wiki/TCP_window_scale_option), as defined in [RFC 1323](http://tools.ietf.org/html/rfc1323), is an option used to increase the maximum window size from 65,535 bytes to 1 gigabyte.

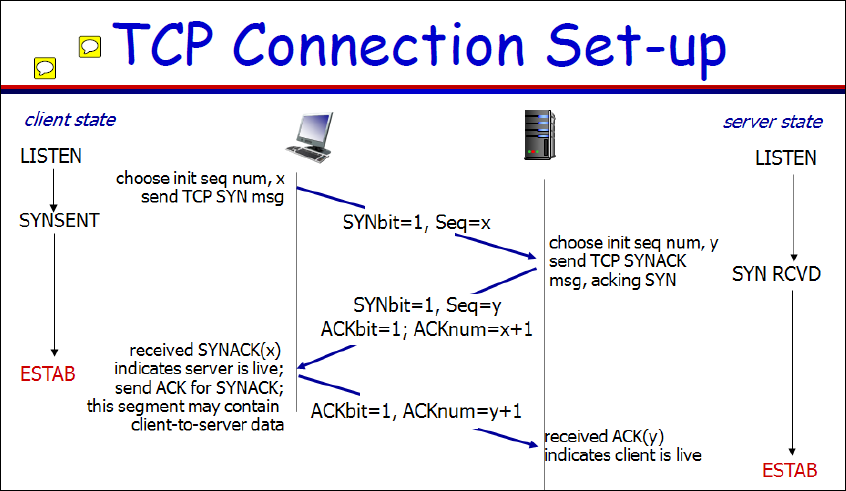
The window scale option is used only during the TCP 3-way handshake. The window scale value represents the number of bits to left-shift the 16-bit window size field.

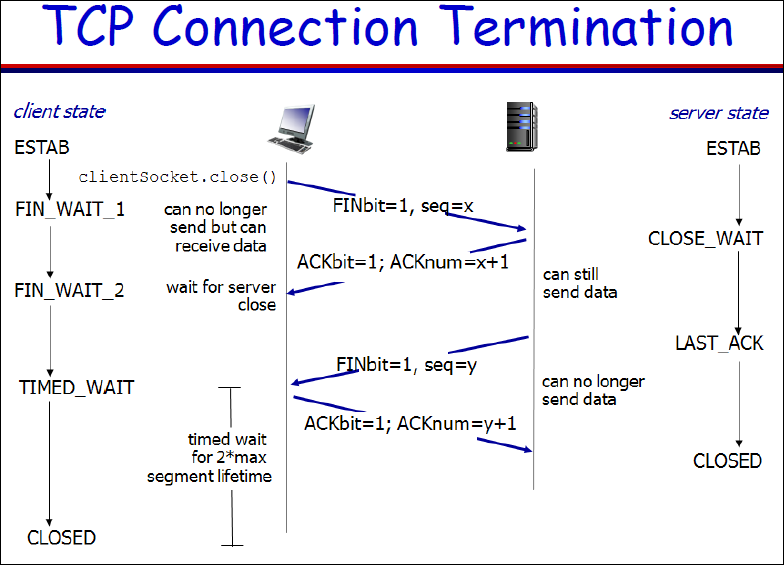
**Zero Window Probe**

This is called *closing the receive window*. Since the server's receive window is the client's send window, reducing its size to zero means the client cannot send any more data, as we saw at the end of the example in the previous topic. This situation continues until the client receives from the server a new acknowledgment with a non-zero *Window* field, which reopens the window. Then the client is able to send again.

The problem with this situation is that the client is dependent upon receipt of the “window opening” segment from the server. Like all TCP segments, this segment is carried over IP, which is unreliable. TCP is reliable only because it acknowledges sent data and retransmits lost data if necessary, but it can never ***guarantee*** that any particular segment gets to its destination. This means that when the server tries to re-open the window with an acknowledgment segment containing a larger *Window* field, it's possible that the client never gets the message. The client might conclude that a problem had occurred and terminate the connection.

To prevent this from happening, the client can regularly send special *probe* segments to the server. The purpose of these probes is to prompt the server to send back a segment containing the current window size. The probe segment can contain either zero or one byte of data, even when the window is closed. The probes will continue to be sent periodically until the window reopens, with the particular implementation determining the rate at which the probes are generated.





For Routing protocols – RIP, OSPF, BGP, Multicast and IGMP

<http://www-ee.uta.edu/online/wang/internet-routing.pdf>

**Router Architectrue**

The main components of the control plane are the routing protocols, the routing table, and other control or signaling protocols used to provision the dataplane. **The *data plane* is the packet forwarding path through a router or switch. The switching of the packets—or the forwarding plane—these days is done on specifically built hardware, or application-specific integrated circuits**

BGP

Read this –

<http://www.cisco.com/c/en/us/td/docs/ios/12_2sr/12_2srb/feature/guide/tbgp_c/tbrbover.html>

**BGP is a path vector routing protocol used to exchange routing information b/w autonomous systems**

**Each AS will different IGP To**

An autonomous system (AS) may have multiple BGP routers, each

communicating with a peer in an outside autonomous system. These

BGP routers will communicate among themselves via internal BGP

(iBGP) connections to guarantee that they have the consistent

information. An external BGP (eBGP) connection is used to

communicate with the peer in another autonomous system. These

connections may pass through intermediate routers between the two

BGP routers.

BGP defines five basic messages as follows:

Type Code Message Type Description

1 OPEN To initialize communication

2 UPDATE To advertise or withdraw routes

3 NOTIFICATION To inform an error and type of error

4 KEEPALIVE To test peer connectivity

5 REFRESH To request advertisement from peer

**BGP uses TCP** [RFC793] as its transport protocol. This eliminates the

need to implement explicit update fragmentation, retransmission,

acknowledgement, and sequencing. BGP **listens on TCP port 179**.

**BGP does**

**not require a periodic refresh of the routing table.** To allow local

policy changes to have the correct effect without resetting any BGP

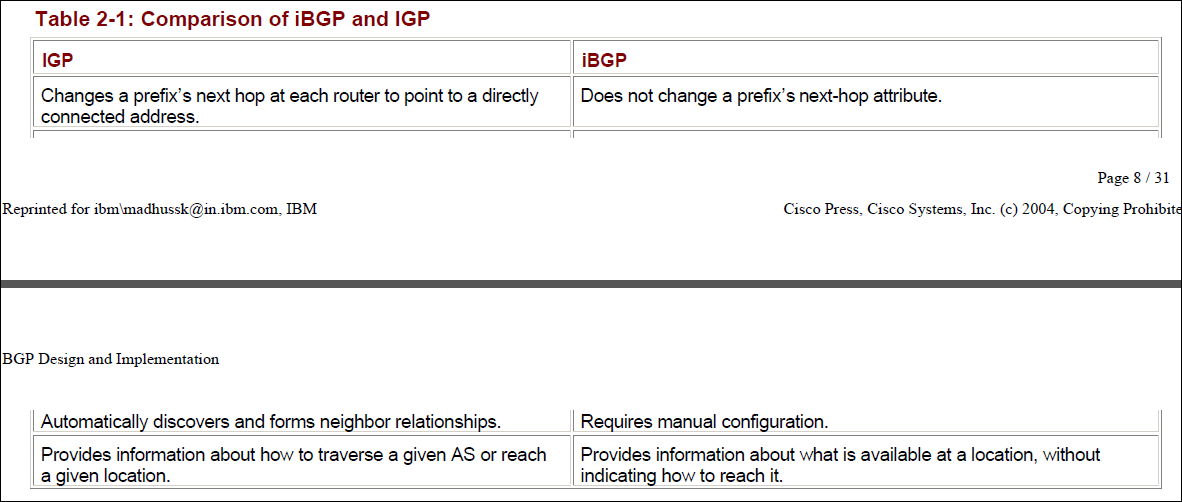
connections, a BGP speaker SHOULD either (a**) retain the current**

**version of the routes advertised to it by all of its peers for the**

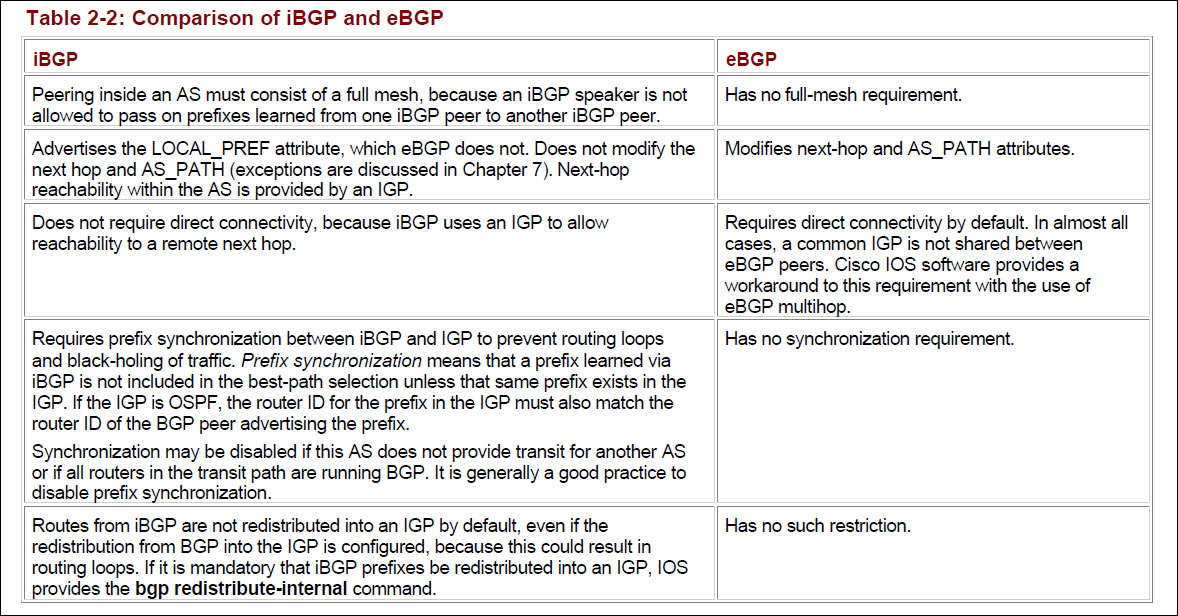
**duration of the connection, or (b) make use of the Route Refresh**

**extension [RFC2918].**

**IBGP vs IGP**

****

**IBGP vs EBGP**

****

**Why is BGP needed?**

**Different AS run different IGP, follow different policies. To exchange routing information between AS, BGP is needed**

**Why IBGP needed?**

**IGPs used today are Distance vector (RIP,IGRPP), Link State( OSPF, IS-IS), Hybrid (EIGRP)**

**IGPs are designed to periodically exchange the routing tables (prefix tables). Internet has over 10000 prefixes. Periodic refresh of this will lead to network instability and resource consumption.**

**Why does IBGP use full mesh?**

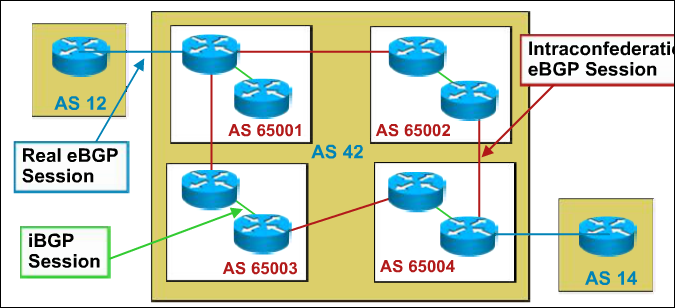
The AS\_PATH attribute is used by eBGP primarily for loop prevention for INTER-AS routing, however in case of iBGP this attribute cannot be used to prevent looping and hence the iBGP speaker is not allowed to PASS-ON the prefixes learned from one iBGP peer to another, so as to prevent looping in iBGP case. Becasue of this restriction peering inside an AS must consist of a FULL MESH, each iBGP speaker must peer with the remaining iBGP speakers within the AS.

**How does BGP pick routes**

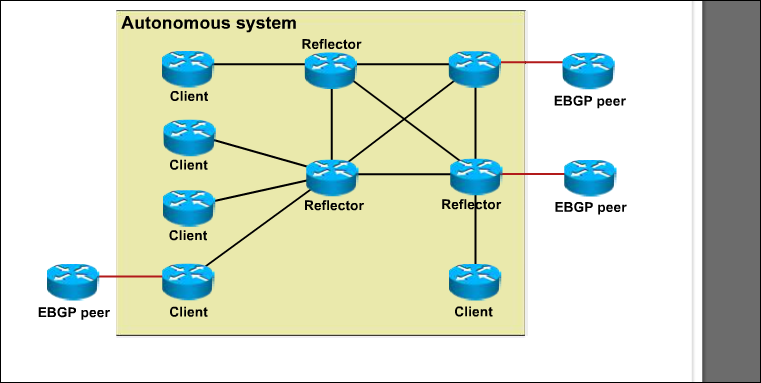
1. **Local PREF – preference as to where outgoing traffic goes**
2. **Shotest AS path- approx of latency**
3. **Lowest MED – multi exit discriminator**
4. **Minimum IGP distance to next hop**
5. **EBGP from speaker with lowest ID**
6. **IBGP from speaker with lowest ID**

**IBGP Scalablility –**

1. **BGP Confederations – BGP members split into smaller member AS within an AS and are assigned private AS number from 64000 to 65535. eBGP sessions between members called intraconfederation BGP**

****

1. **Route reflectors – Instead of connecting the IBGP routers in a full mesh they are connected through the route reflectors. all route updates goes through route reflectors which are connected to clients( IBGP peers).**

****

**MPLS**

**Labels inserted between**

**Read Section 4.3 in CN: system’s approach Peterson and davie**

**Pre MPLS**

Customers leased ATM links and Frame Relay links or used leased lines and built their own private network over it. Because the routers of the provider supplied a Layer 2 service toward the Layer 3

customer routers, the separation and isolation between different customer networks were guaranteed. These kinds of networks are referred to as *overlay networks*

**MPLS Applications**

1. **Load Balancing – Explicit Routing**
2. **VPN**

**Need for MPLS**

By using MPLS with IP, you can extend the possibilities of what you can transport. Adding labels to the packet enables you to carry other protocols than just IP over an MPLS-enabled Layer 3 IP backbone, similarly to what was previously possible only with Frame Relay or ATM Layer 2 networks. MPLS can transport IPv4, IPv6, Ethernet, High-Level Data Link Control (HDLC), PPP, and other Layer 2 technologies.

**The feature whereby any Layer 2 frame is carried across the MPLS backbone is called *Any Transport over MPLS* (AToM).**

**Better IP over ATM Integration**

One solution was to implement IP over ATM according to the well-known RFC 1483, “Multiprotocol Encapsulation over ATM Adaptation Layer 5,” which specifies how to encapsulate multiple routed and bridged protocols over ATM adaptation Layer (AAL) 5. In this solution, all ATM circuits had to be manually established, and all mappings between IP next hops and ATM endpoints had to be manually configured on every ATM-attached router in the network

**MPLS operation**

The destination IP address of all IP packets entering the ingress LSR will be looked up in the IP forwarding table. All these addresses belong to a set of prefixes that are known in the routing table as *BGP prefixes*. Many BGP prefixes in the routing table have the same BGP next-hop address, namely one egress LSR. All packets with a destination IP address for which the IP lookup in the routing table recurses to the same BGP next-hop address will be mapped to the same FEC. As already mentioned, all packets that belong to the same FEC get the same label imposed by the ingress LSR.

Consider the example of plain IPv4-over-MPLS, which is the simplest example of an MPLS network. Plain IPv4-over-MPLS is a network that consists of LSRs that run an IPv4 Interior Gateway Protocol (IGP) (for example, Open Shortest Path First [OSPF], Intermediate System-to-Intermediate System [IS-IS], and Enhanced Interior Gateway Routing Protocol [EIGRP]). The ingress LSR looks up the destination IPv4 address of the packet, imposes a label, and forwards the packet. The next LSR (and any other intermediate LSR) receives the labeled packet, swaps the incoming label with an outgoing label, and

forwards the packet. The egress LSR pops the label and forwards the IPv4 packet without labels on the outgoing link. For this to work, adjacent LSRs must agree on which label to use for each IGP prefix. Therefore, each intermediate LSR must be able to figure out with which outgoing label the incoming label should be swapped. This means that you need a mechanism to tell the routers which labels to use when forwarding a packet. Labels are local to each pair of adjacent routers. Labels have no global meaning across the network. For adjacent routers to agree which label to use for which prefix, they need some form of communication between them; otherwise, the routers do not know which outgoing label

needs to match which incoming label. A label distribution protocol is needed. You can distribute labels in two ways:

n Piggyback the labels on an existing IP routing protocol

n Have a separate protocol distribute labels

**Problem with Piggybacking**

Link state routing protocols (such as IS-IS and OSPF) do not function in this way. Each router originates link state updates that are then forwarded unchanged by all routers inside one area. The problem is that for MPLS to work, **each router needs to distribute a label for each IGP prefix—even the routers that are not originators of that prefix**. Link state routing protocols need to be enhanced in an intrusive way to be able to do this. The fact that a router needs to advertise a label for a prefix it does not originate is counterintuitive to the way link state routing protocols work anyway

**IS-IS**

1. IS-IS is very different than other network routing protocols because it runs natively on Layer 2 of the OSI Reference Model. What does that mean? Unlike the IP routing protocols like RIP, OSPF and BGP, IS-IS does not need valid interface addressing information to transmit a message
2. Level 1 , Level 2 and Level 1-2 areas
3. Like OSPF, maintains links state database
4. Based on Dijkstra’s Shortest Path first algorithm
5. Directly passes the routing information to MAC layer unlike OSPF which uses Ip

**Problems with IS-IS and advantages of OSPF**

"Why not simply adopt the IS-IS OSI routing protocol as your new routing protocol?"

IS-IS was being developed at the same time that we were undertaking development of OSPF. Looking at the

emerging IS-IS protocol, we saw a number of barriers to its becoming a TCP/IP routing

protocol. We mention a few of these problems by way of example. First, IS-IS ran

directly over the link layer, which we thought was the wrong choice for a TCP/IP routing protocol, as explained next. At the time, IS-IS had no way to fragment LSAs, with a

router having to originate all of its routing data within a single LSA. In some TCP/IP

environments in which routers import many external routes, such a design would be

unworkable. IS-IS had an area routing scheme, but one that did not allow any shortcuts

between areas that we thought were necessary for an Internet routing protocol. Also,

IS-IS made no attempt to align fields in their packet formats, making life more difficult

for protocol implementors

**RIP (port 520 through UDP)**

In this protocol, routing table update messages are exchanged

between neighbors periodically (every 30 seconds).

The routing table in each router contains the following information:

- The IP address of the destination network;

- The IP address of the next hop router;

- The output port to the next hop;

- The neighboring router that was the source of this information;

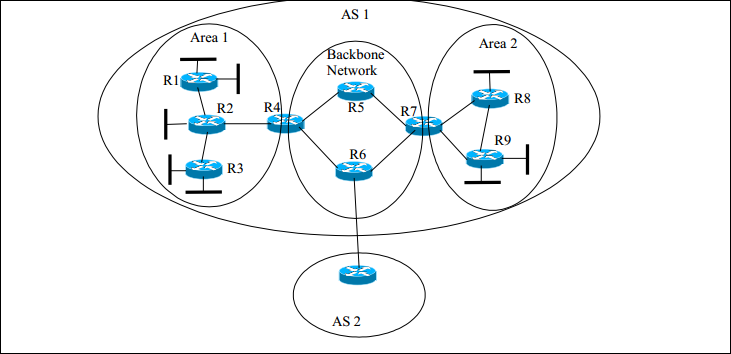
- Age of this information.

The routing table may also contain the subnet mask and the hop count

to the destination network

.

**OSPF (uses IP directly)**

****

OSPF is a **link-state protocol. Each router actively tests the status of its link to each of its neighbors**. The link-state is defined by the network authority. It can be speed, delay, throughput, reliability, or a

combination of these parameters. The router then sends its link-state information to its neighbors, which then propagates it throughout the autonomous system. Thus, **the link-state information is multicasted** to all the routers that participate in the process. Each router takes this link-state information and builds a complete routing table.

While RIP uses UDP with well-known port number 520, OSPF uses IP directly. The protocol ID for OSPF is 89

**IGMP**

To participate in a multicast event over the Internet, a receiving host must first inform the local multicast router its intention to join the multicast. The local multicast router then contacts other multicast routers in the Internet to pass the membership information and establish the multicast route. The local router also needs to keep track of any host(s) that remains as member(s) of the multicast group.

The protocol used for the host and the local router to communicate the multicast group membership information is called Internet Group Management Protocol (IGMP). IGMP uses IP datagrams to carry its

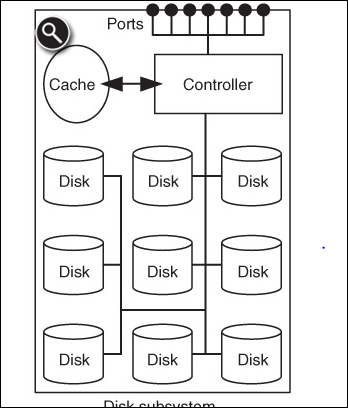
messages. The protocol ID in the IP header for IGMP is 2

**Storage Area Networks**

SAN systems do not require server connections; they are LAN-free backup systems. They do not need to be housed in the same box as servers, nor are they required to be from the same manufacturing companies as servers. Rather than putting data storage directly on the network, the SAN solution puts data storage network devices *between* storage subsystems and the servers. SANs can be built as switched- and/or shared-access networks

Servers are connected to the connection port of the disk subsystem using standard I/O techniques such as Small Computer System Interface (SCSI), Fibre Channel or Internet SCSI (iSCSI) and can thus use the storage capacity that the disk subsystem provides (Figure 2.1). The internal structure of the disk subsystem is completely hidden from the server, which sees only the hard disks that the disk subsystem provides to the server

The connection ports are extended to the hard disks of the disk subsystem by means of internal I/O channels (Figure 2.2). In most disk subsystems there is a controller between the connection ports and the hard disks. The controller can significantly increase the data availability and data access performance with the aid of a so-called RAID procedure. Furthermore, some controllers realise the copying services instant copy and remote mirroring and further additional services. The controller uses a cache in an attempt to accelerate read and write accesses to the server.

****

**RAID**

**RAID** is a storage technology that combines multiple [disk drive](http://en.wikipedia.org/wiki/Disk_drive) components into a logical unit for the purposes of data redundancy and performance improvement

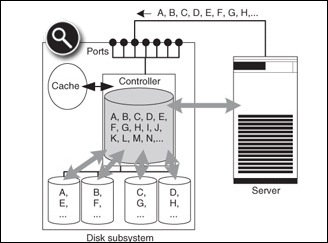
RAID has two main goals: to increase performance by striping and to increase fault-tolerance by redundancy. **Striping distributes the data over several hard disks and thus distributes the load over more hardware. Redundancy means that additional information is stored so that the operation of the application itself can continue in the event of the failure of a hard disk**

A server that is connected to a RAID system sees only the virtual hard disk; the fact that the RAID controller actually distributes the data over several physical hard disks is completely hidden to the server

One factor common to almost all RAID levels is that they store redundant information. If a physical hard disk fails, its data can be reconstructed from the hard disks that remain intact. The defective hard disk can even be replaced by a new one during operation if a disk subsystem has the appropriate hardware. Then the RAID controller reconstructs the data of the exchanged hard disk. This process remains hidden to the server apart from a possible reduction in performance: the server can continue to work uninterrupted on the virtual hard disk. The Hamming code permits the correct recreation of the data, even if individual bits are changed on the hard disk. If the system is looked after properly you can assume that the installed physical hard disks will hold out for a while. Therefore, for the benefit of higher performance, it is generally an acceptable risk to give access by the server a higher priority than the recreation of the data of an exchanged physical hard disk

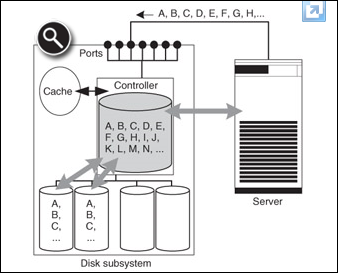
**RAID 0**

the server writes the blocks A, B, C, D, E, etc. onto the virtual hard disk one after the other

****

**RAID 1**

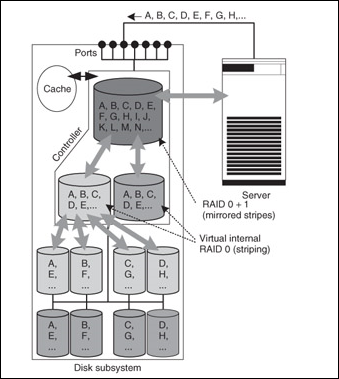
The basic form of RAID 1 brings together two physical hard disks to form a virtual hard disk by mirroring the data on the two physical hard disks. If the server writes a block to the virtual hard disk, the RAID controller writes this block to both physical hard disks (Figure 2.10). The individual copies are also called mirrors.

****

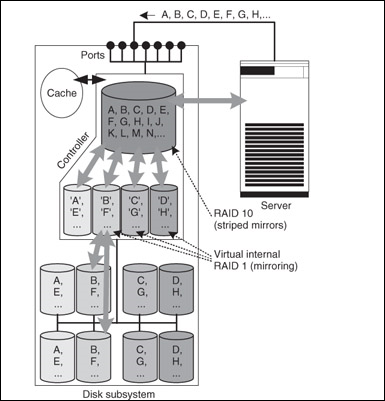
**RAID 0+1 and RAID 10**

RAID 0+1 and RAID 10 each represent a two-stage virtualisation hierarchy. Figure 2.11 shows the principle behind RAID 0+1 (mirrored stripes). In the example, eight physical hard disks are used. The RAID controller initially brings together each four physical hard disks to form a total of two virtual hard disks that are only visible within the RAID controller by means of RAID 0 (striping). In the second level, it consolidates these two virtual hard disks into a single virtual hard disk by means of RAID 1 (mirroring); only this virtual hard disk is visible to the server.

**RAID 0+1**

****

**RAID 10**

****

**RAID 4 and 5**

RAID controller with a suitably large cache can hold frequently changed parity blocks in the cache after writing to the disk, so that the next time one of the data blocks in question is changed there is no need to read in the parity block. In both cases the I/O load is now lower than in the case of RAID 10. In the example only five physical blocks now need to be written instead of eight as is the case with RAID 10.

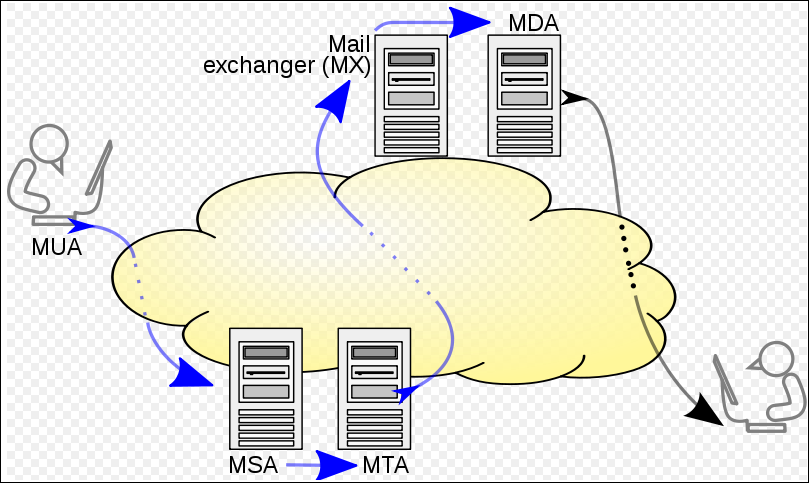
RAID 4 wrote the parity on a single disk

RAID 5 distributed this all over the disk

**RAID 2 and 3 – old and discarded**

**SMTP –**

Email is submitted by a mail client (MUA, [mail user agent](http://en.wikipedia.org/wiki/Mail_user_agent)) to a mail server (MSA, [mail submission agent](http://en.wikipedia.org/wiki/Mail_submission_agent)) using SMTP on[TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) port 587. MSA delivers the mail to its mail transfer agent (MTA, [mail transfer agent](http://en.wikipedia.org/wiki/Mail_transfer_agent)). Often, these two agents are just different instances of the same software launched with different options on the same machine. The boundary MTA has to locate the target host. It uses the [Domain name system](http://en.wikipedia.org/wiki/Domain_name_system) (DNS) to look up the mail exchanger record (MX record) for the recipient's domain (the part of the [email address](http://en.wikipedia.org/wiki/Email_address) on the right of @).Once the MX target accepts the incoming message, it hands it to a [mail delivery agent](http://en.wikipedia.org/wiki/Mail_delivery_agent) (MDA) for local mail delivery. An MDA is able to save messages in the relevant [mailbox](http://en.wikipedia.org/wiki/Email_mailbox) format. Once delivered to the local mail server, the mail is stored for batch retrieval by authenticated mail clients (MUAs). Mail is retrieved by end-user applications, called email clients, using [Internet Message Access Protocol](http://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) (IMAP)

****

**Firewalls**

**Stateful Packet inspection**

A stateful inspection packet firewall tightens up the rules for TCP traffic by

creating a directory of outbound TCP connections, as shown in Table 22.2. There is

an entry for each currently established connection.The packet filter will now allow

incoming traffic to high-numbered ports only for those packets that fit the profile of

one of the entries in this directory

**HTTP**

**HTTP Request Message**

Below we provide a typical HTTP request message:

GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

HTTP/1.1 200 OK

Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

(data data data data data ...)

Some common status codes and associated

phrases include:

• 200 OK: Request succeeded and the information is returned in the response.

• 301 Moved Permanently: Requested object has been permanently moved;

the new URL is specified in Location: header of the response message. The

client software will automatically retrieve the new URL.

400 Bad Request: This is a generic error code indicating that the request

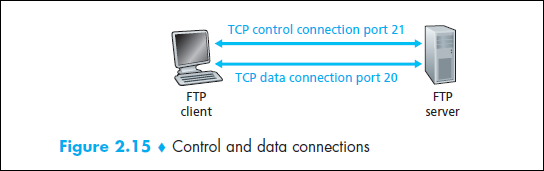
could not be understood by the server.

• 404 Not Found: The requested document does not exist on this server.

• 505 HTTP Version Not Supported: The requested HTTP protocol

version is not supported by the server.

**FTP**

****

**Virtualization**

**By using virtual networks, test and production servers can run simultaneously and use same IP addresses.**

**// quickSort.c**

#include <stdio.h>

void quickSort( int[], int, int);

int partition( int[], int, int);

void main()

{

int a[] = { 7, 12, 1, -2, 0, 15, 4, 11, 9};

int i;

printf("\n\nUnsorted array is: ");

for(i = 0; i < 9; ++i)

printf(" %d ", a[i]);

quickSort( a, 0, 8);

printf("\n\nSorted array is: ");

for(i = 0; i < 9; ++i)

printf(" %d ", a[i]);

}

void quickSort( int a[], int l, int r)

{

int j;

if( l < r )

{

// divide and conquer

j = partition( a, l, r);

quickSort( a, l, j-1);

quickSort( a, j+1, r);

}

}

int partition( int a[], int l, int r) {

int pivot, i, j, t;

pivot = a[l];

i = l; j = r+1;

while( 1)

{

do ++i; while( a[i] <= pivot && i <= r );

do --j; while( a[j] > pivot );

if( i >= j ) break;

t = a[i]; a[i] = a[j]; a[j] = t;

}

t = a[l]; a[l] = a[j]; a[j] = t;

return j;

}

**Merge Sort**

#include<stdio.h>

#define MAX 50

void mergeSort(int arr[],int low,int mid,int high);

void partition(int arr[],int low,int high);

int main(){

    int merge[MAX],i,n;

    printf("Enter the total number of elements: ");

    scanf("%d",&n);

    printf("Enter the elements which to be sort: ");

    for(i=0;i<n;i++){

         scanf("%d",&merge[i]);

    }

    partition(merge,0,n-1);

    printf("After merge sorting elements are: ");

    for(i=0;i<n;i++){

         printf("%d ",merge[i]);

    }

   return 0;

}

void partition(int arr[],int low,int high){

    int mid;

    if(low<high){

         mid=(low+high)/2;

         partition(arr,low,mid);

         partition(arr,mid+1,high);

         mergeSort(arr,low,mid,high);

    }

}

void mergeSort(int arr[],int low,int mid,int high){

    int i,m,k,l,temp[MAX];

    l=low;

    i=low;

    m=mid+1;

    while((l<=mid)&&(m<=high)){

         if(arr[l]<=arr[m]){

             temp[i]=arr[l];

             l++;

         }

         else{

             temp[i]=arr[m];

             m++;

         }

         i++;

    }

    if(l>mid){

         for(k=m;k<=high;k++){

             temp[i]=arr[k];

             i++;

         }

    }

    else{

         for(k=l;k<=mid;k++){

             temp[i]=arr[k];

             i++;

         }

    }

    for(k=low;k<=high;k++){

         arr[k]=temp[k];

    }

}

**Difference between Macros and Inline functions**

Inline functions provides following advantages over macros.

* Macro interpreted by preprocessor, inline by Compiler
* Since they are functions so type of arguments is checked by the compiler whether they are correct or not.
* There is no risk if called multiple times. But there is risk in macros which can be dangerous when the argument is an expression.
* They can include multiple lines of code without trailing backlashes.
* Inline functions have their own scope for variables and they can return a value.
* Debugging code is easy in case of Inline functions as compared to macros.

# Difference Between Global , extern global , static , static global

1. Defines a global variable, allocating memory for it which will be accessible by all files in the program.

2. Declares a global variable which is defined in another file. No memory is allocated. This is how global variables defined in one file can be used in another.

3. static local variables retain their values on repeated calls into the function unlike other local variables.

4. These global variables can only be accessed from within the file. Using the static keyword you can declare global variables with the same name in multiple files in the same program.

5. static functions are functions that are only visible to other functions in the same file.

**Volatile**

It tells the compiler that the value of the variable may change at any time--without any action being taken by the code

## Proper use of volatile

A variable should be declared volatile whenever its value could change unexpectedly. In practice, only three types of variables could change:

1. Memory-mapped peripheral registers

2. Global variables modified by an interrupt service routine

3. Global variables accessed by multiple tasks within a multi-threaded application