

**Some of the common ports are:**

20 & 21 FTP

23 Telnet

25 SMTP (Simple Mail Transfer Protocol)

53 DNS

80 World Wide Web

110 POP3 (Post Office Protocol)

144 News

6000 X-Windows

**HTTP get vs post**

* GET - Requests data from a specified resource
* POST - Submits data to be processed to a specified resource

Some other notes on GET requests:

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions (128k bytes)
* GET requests should be used only to retrieve data

Some other notes on POST requests:

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length

POST /test/demo\_form.php HTTP/1.1  
Host: w3schools.com  
**name1=value1&name2=value2**

**TCP Header Format**

Each TCP header has ten required fields totaling 20 [bytes](https://www.lifewire.com/definition-of-byte-816252) (160 [bits](https://www.lifewire.com/definition-of-bit-816250)) in size.

They can also optionally include an additional data section up to 40 bytes in size.

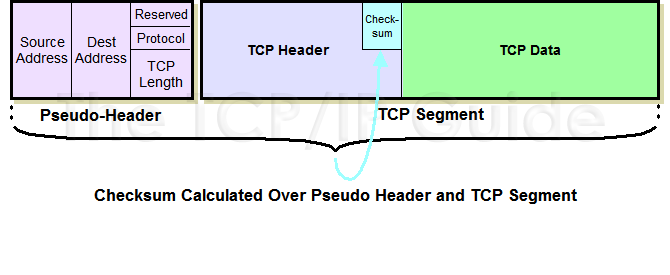
This is the layout of TCP headers:

1. Source TCP port number (2 bytes)
2. Destination TCP port number (2 bytes)
3. Sequence number (4 bytes)
4. Acknowledgment number (4 bytes)

the next sequence number the sender of the segment is expecting to receive

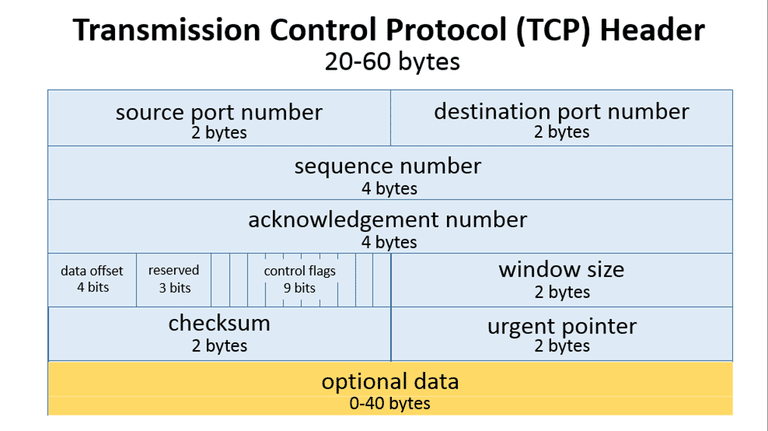
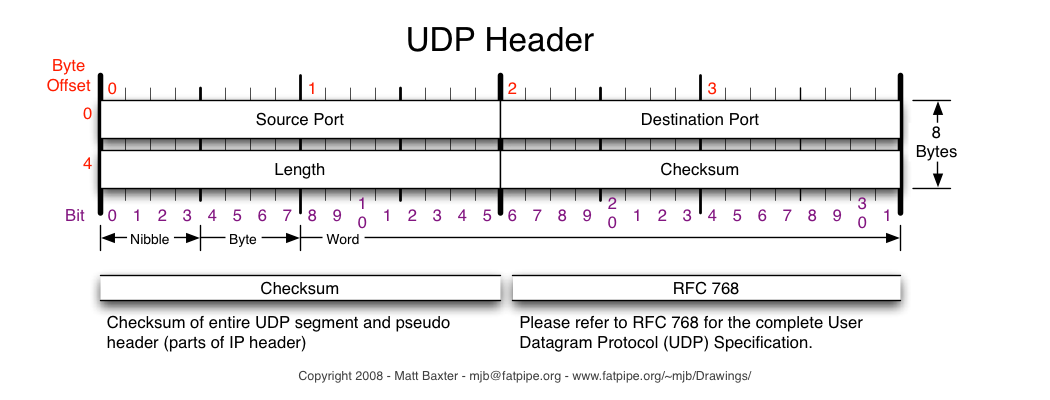
1. TCP data offset (4 bits)
2. Reserved data (3 bits)
3. Control flags (up to 9 bits)
4. Window size (2 bytes)
5. TCP [checksum](https://www.lifewire.com/what-does-checksum-mean-2625825) (2 bytes)

TCP Checksum is a 16-bit field in TCP header used for error detection. It is computed over the TCP segment (might plus some padding) and a 12-byte TCP pseudo header created on the fly.



1. Urgent pointer (2 bytes)
2. TCP optional data (0-40 bytes)

TCP's urgent mode is just a notification from the sender to the receiver that urgent data has been sent, along with the sequence number of the final byte of urgent data.

**UDP Header Format**

Because UDP is significantly more limited in capability than TCP, its headers are much smaller.

A UDP header contains 8 bytes, divided into the following four required fields:

1. Source port number (2 bytes)
2. Destination port number (2 bytes)
3. Length of data (2 bytes)
4. UDP checksum (2 bytes)

It is easy to generate these ICMP errors using UDP.

Broadcasting and multicasting only apply to UDP, where it makes sense for an

application to send a single message to multiple recipients.

Here are some quick info.

**SSL** (Secure Sockets Layer)

Transport Layer Security (**TLS**)

**Transport Layer Security** (**TLS**) and its predecessor, **Secure Sockets Layer** (**SSL**), both frequently referred to as "SSL"

SSL/TLS are protocols used for encrypting information between two points. It is usually between server and client, but there are times when server to server and client to client encryption are needed. For the purpose of this blog, I will focus only on the negotiation between server and client.

For SSL/TLS negotiation to take place, the system administrator must prepare the minimum of 2 files: **Private Key and Certificate**. When requesting from a Certificate Authority such as Symantec Trust Services, an additional file must be created. This file is called **Certificate Signing Request**, generated from the Private Key. The process for generating the files are dependent on the software that will be using the files for encryption.

For a list of the server softwares Symantec has, have a look at: [Symantec CSR Generation](https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=AR235&actp=GETTING_STARTED)

Note that although certifcates requested from Certificate Authorities such as Symantec are inherently trusted by most clients, additional certificates called Intermediate Certificate Authority Certificates and Certificate Authority Root Certificates may need to be installed on the server. This is again server software dependent. There is usually no need to install the Intermediate and Root CA files on the client applications or browsers.

Once the files are ready and correctly installed, just start the SSL/TLS negotiation by using the secured protocol.  On browser applications it is usually [https://www.yourwebsite.com](https://www.yourwebsite.com/).

Remember to use **your** secured website address. Above is just a sample address.

That will start the SSL/TLS negotiation:

**Keys and Secrets during RSA SSL negotiation**

The following is a standard SSL handshake when RSA key exchange algorithm is used:

1. **Client Hello**  
   - Information that the server needs to communicate with the client using SSL.  
   - Including SSL version number, cipher settings, session-specific data.
2. **Server Hello**  
   - Information that the client needs to communicate with the server using SSL.  
   - Including SSL version number, cipher settings, session-specific data.  
   - Including Server’s Certificate (Public Key)
3. **Authentication and Pre-Master Secret**  
   - Client **authenticates** the server certificate. (e.g. Common Name / Date / Issuer)  
   - Client (depending on the cipher) creates the pre-master secret for the session,  
   - **Encrypts** with the server's public key and sends the encrypted pre-master secret to the server.
4. **Decryption and Master Secret**  
   - Server uses its private key to **decrypt** the pre-master secret,  
   - Both Server and Client perform steps to generate the master secret with the agreed cipher.
5. **Generate Session Keys**  
   - Both the client and the server use the master secret to generate the session keys,  which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session
6. **Encryption with Session Key**  
   - Both client and server exchange messages to inform that future messages will be encrypted.

([Wikipedia: Transport Layer Security](http://en.wikipedia.org/wiki/Transport_Layer_Security))

Tools such as OpenSSL can be used check the SSL/TLS negotiations:

OpenSSL s\_client -connect [www.symantec.com:443](http://www.symantec.com:443/) -state -ssl3  
Loading 'screen' into random state - done  
CONNECTED(000001C0)  
SSL\_connect:before/connect initialization  
SSL\_connect:SSLv3 write client hello A  
SSL\_connect:SSLv3 read server hello A  
depth=2 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign, Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Certification Authority - G5

SSL\_connect:SSLv3 read server certificate A  
SSL\_connect:SSLv3 read server done A  
SSL\_connect:SSLv3 write client key exchange A  
SSL\_connect:SSLv3 write change cipher spec A  
SSL\_connect:SSLv3 write finished A  
SSL\_connect:SSLv3 flush data  
SSL\_connect:SSLv3 read finished A  
---  
Certificate chain  
 0 s:/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=Delaware/businessCategory=Private Organization/serialNumber=2158113/C=US/postalCode=94043/ST=California/L=Mountain View/street=350 Ellis Street/O=Symantec Corporation/OU=Corp Mktg & Comms - Online Exp/CN=www.symantec.com

There it is. SSL and SSL Negotiation summarized. Mission complete.

The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate (although TLS 1.0 does incorporate a mechanism by which a TLS implementation can back down to SSL 3.0).

1. SSL means a “by port” explicit connection to a port that expects to the session to start with security negotiation
2. TLS means a “by protocol” connection where the program will connect “insecurely” first and use special commands to enable encryption (implicit).
3. Use of either could result in a connection encrypted with *either* SSL v3 or TLS v1.0+, based on what is installed on the server and what is supported by your program.
4. *Both methods of connection (implicit and explicit) result in equally secure communications*.

**MTU**

There is a limit on the size of the frame for both Ethernet

encapsulation and 802.3 encapsulation. This limits the number of bytes of data to 1500 and 1492, respectively. This characteristic of the link layer is called the *MTU,* its **m**aximum **t**ransmission **u**nit. Most types of networks have an upper limit.

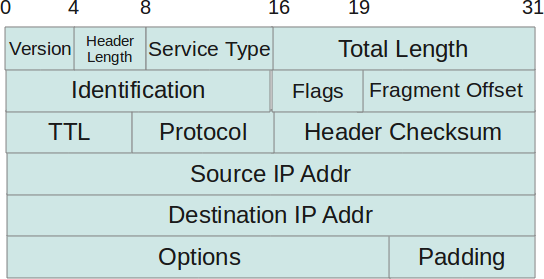
If IP has a datagram to send, and the datagram is larger than the link layer's MTU, IP performs *fragmentation,* breaking the datagram up into smaller pieces (fragments), so that each fragment is smaller than the MTU.

**IP**

IP provides an unreliable, connectionless datagram delivery service.

By *unreliable* we mean there are no guarantees that an IP datagram successfully gets to its destination. IP provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IP has a simple error handling algorithm: throw away the datagram and try to send an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

### IP Header



* ***Protocol Version(4 bits)*** : This is the first field in the protocol header. This field occupies 4 bits. This signifies the current IP protocol version being used. Most common version of IP protocol being used is version 4 while version 6 is out in market and fast gaining popularity.
* ***Header Length(4 bits)****:* This field provides the length of the IP header. The length of the header is represented in 32 bit words. This length also includes IP options (if any). Since this field is of 4 bits so the maximum header length allowed is 60 bytes. Usually when no options are present then the value of this field is 5. Here 5 means five 32 bit words ie 5 \*4 = 20 bytes.
* ***Type of service(8 bits)*** : The first three bits of this field are known as precedence bits and are ignored as of today. The next 4 bits represent type of service and the last bit is left unused. The 4 bits that represent TOS are : minimize delay, maximize throughput, maximize reliability and minimize monetary cost.
* ***Total length(16 bits)***: This represents the total IP datagram length in bytes. Since the header length (described above) gives the length of header and this field gives total length so the length of data and its starting point can easily be calculated using these two fields. Since this is a 16 bit field and it represents length of IP datagram so the maximum size of IP datagram can be 65535 bytes. When IP fragmentation takes place over the network then value of this field also changes. There are cases when IP datagrams are very small in length but some data links like ethernet pad these small frames to be of a minimum length ie 46 bytes. So to know the exact length of IP header in case of ethernet padding this field comes in handy.
* ***Identification(16 bits)***: This field is used for uniquely identifying the IP datagrams. This value is incremented every-time an IP datagram is sent from source to the destination. This field comes in handy while reassembly of fragmented IP data grams.
* ***Flags(3 bits)***: This field comprises of three bits. While the first bit is kept reserved as of now, the next two bits have their own importance. The second bit represents the ‘Don’t Fragment’ bit. When this bit is set then IP datagram is never fragmented, rather its thrown away if a requirement for fragment arises. The third bit represents the ‘More Fragment’ bit. If this bit is set then it represents a fragmented IP datagram that has more fragments after it. In case of last fragment of an IP datagram this bit is not set signifying that this is the last fragment of a particular IP datagram.
* ***Fragment offset(13 bits)***: In case of fragmented IP data grams, this field contains the offset( in terms of 8 bytes units) from the start of IP datagram. So again, this field is used in reassembly of fragmented IP datagrams.
* ***Time to live(8 bits)***: This value represents number of hops that the IP datagram will go through before being discarded. The value of this field in the beginning is set to be around 32 or 64 (lets say) but at every hop over the network this field is decremented by one. When this field becomes zero, the data gram is discarded. So, we see that this field literally means the effective lifetime for a datagram on network.
* ***Protocol(8 bits)***: This field represents the transport layer protocol that handed over data to IP layer. This field comes in handy when the data is demultiplex-ed at the destination as in that case IP would need to know which protocol to hand over the data to.
* ***Header Checksum(16 bits)***: This fields represents a value that is calculated using an algorithm covering all the fields **in header** (assuming this very field to be zero). This value is calculated and stored in header when IP data gram is sent from source to destination and at the destination side this checksum is again calculated and verified against the checksum present in header. If the value is same then the datagram was not corrupted else its assumed that data gram was received corrupted. So this field is used to check the integrity of an IP datagram.
* ***Source and destination IP(32 bits each)***: These fields store the source and destination address respectively. Since size of these fields is 32 bits each so an IP address os  maximum length of 32 bits can be used. So we see that this limits the number of IP addresses that can be used. To counter this problem, IP V6 has been introduced which increases this capacity.
* ***Options(Variable length)***: This field represents a list of options that are active for a particular IP datagram. This is an optional field that could be or could not be present. If any option is present in the header then the first byte is represented as follows :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| copy flag | option class | | option num | | | | |

* In the description above, the ‘copy flag’ means that copy this option to all the fragments in case this IP datagram gets fragmented. The ‘option class’ represents the following values : 0 -> control, 1-> reserved, 2 -> debugging and measurement, and 3 -> reserved. Some of the options are given below :

|  |  |  |  |
| --- | --- | --- | --- |
| class | number | length | description |
| 0 | 0 | – | end of option list |
| 0 | 1 | – | no operation |
| 0 | 2 | 11 | security |
| 0 | 3 | var. | loose source routing |
| 0 | 9 | var. | strict source routing |
| 0 | 7 | var. | record route |
| 0 | 8 | 4 | stream id |
| 2 | 4 | var. | INTERNET time stamp |

* ***Data***: This field contains the data from the protocol layer that has handed over the data to IP layer. Generally this data field contains the header and data of the transport layer protocols. Please note that each TCP/IP layer protocol attaches its own header at the beginning of the data it receives from other layers in case of source host and in case of destination host each protocol strips its own header and sends the rest of the data to the next layer.

### Subnetting

You are allocated the class C network 192.168.123.0. (For illustration, this address is actually from a range that is not allocated on the Internet.) This means that you can use the addresses 192.168.123.1 to 192.168.123.254 for your 150 hosts.  
  
Two addresses that cannot be used in your example are 192.168.123.0 and 192.168.123.255 because binary addresses with a host portion of all ones and all zeros are invalid. The zero address is invalid because it is used to specify a network without specifying a host. The 255 address (in binary notation, a host address of all ones) is used to broadcast a message to every host on a network. Just remember that the first and last address in any network or subnet cannot be assigned to any individual host.

The following shows how a class B network address could effectively split into 254 separate virtual class C networks:

nnn.nnn.sss.hhh

nnn = network portion of the address

sss = subnet portion of the address

hhh = host portion of the address

**ARP (Address Resolution Protocol)**

Assuming an Ethernet, the sending host must convert the 32-bit IP address into a 48-bit

Ethernet address. A translation is required from the *logical* Internet address to its

corresponding *physical* hardware address. This is the function of ARP.

ARP is intended for broadcast networks where many hosts or routers are connected to a

single network.

ARP sends an Ethernet frame called an *ARP request* to every host on the network. This is

called a *broadcast.* We show the broadcast in Figure 4.2 with dashed lines. The ARP

request contains the IP address of the destination host (whose name is bsdi) and is the

request "if you are the owner of this IP address, please respond to me with your hardware

address."

The fundamental concept behind ARP is that the network interface has a hardware address (a 48-bit value for an Ethernet or token ring interface). Frames exchanged at the hardware level must be addressed to the correct interface. But TCP/IP works with its own addresses: 32-bit IP addresses. Knowing a host's IP address doesn't let the kernel send a frame to that host. The kernel (i.e., the Ethernet driver) must know the destination's hardware address to send it data. The function of ARP is to provide a dynamic mapping between 32-bit IP addresses and the hardware addresses used by various network technologies.

Point-to-point links don't use ARP. When these links are configured (normally at bootstrap time) the kernel must be told of the IP address at each end of the link. Hardware addresses such as Ethernet addresses are not involved.

ICMP (**Internet Control Message Protocol**)

is often considered part of the IP layer. It communicates error messages and other conditions that require attention. ICMP messages are usually acted on by either the IP layer or the higher layer protocol (TCP or UDP). Some ICMP messages cause errors to be returned to user processes.

An ICMP packet may be sent when a router is experiencing congestion or when a destination host is unavailable.

**Ping Program**

The program sends an ICMP echo request message to a host, expecting an ICMP echo reply to be returned. Ping also measures the round-trip time to the host, giving us some indication of how "far away" that host is. Ping also gives us an opportunity to examine the IP record route and timestamp options.

The ping program is the basic connectivity test between two systems running TCP/IP. It uses the ICMP echo request and echo reply messages and does not use a transport layer (TCP or UDP). The Ping server is normally part of the kernel's ICMP implementation.

**Traceroute**

Traceroute is an indispensable tool when working with a TCP/IP network. Its operation is simple: send UDP datagrams starting with a TTL of 1, increasing the TTL by 1, to locate each router in the path. An ICMP time exceeded is returned by each router when it discards the UDP datagram, and an ICMP port unreachable is generated by the final destination.

We ran examples of traceroute on both LANs and WANs, and used it to examine IP

source routing. We used loose source routing to see if the route to a destination is the same as the return route from that destination.

**IP Rounting**

The operation of IP routing is fundamental to a system running TCP/IP, be it a host or

router. The routing table entries are simple: up to 5 flag bits, a destination IP address (host, network, or default), a next-hop router IP address (for an indirect route) or a local interface IP address (for a direct route), and a pointer to a local interface to use.

Host entries have priority over network entries, which have priority over default entries. A search of this routing table is made for every IP datagram that the system generates or forwards, and can be updated by either a routing daemon or ICMP redirects. By default a system should never forward a datagram unless it has specifically been configured to do so.

Static routes can be entered using the route command, and the newer ICMP router

discovery messages can be used to initialize and dynamically update default entries. Hosts can start with a simple routing table that is updated dynamically by ICMP redirects from its default router.

**DNS (Domain Name System)**

We've mentioned that the well-known port numbers for DNS name servers are UDP port 53 and TCP port 53. This implies that the DNS supports both UDP and TCP. But all the examples that we've watched with tcpdump have used UDP.