<http://www.portknocking.org/view/primer/packets>

Recall that a **socket** **[look up socket on webopedia.com](http://www.webopedia.com/TERM/s/socket.html)** **[look up socket on FOLDOC](http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=socket)** is a combination of IP address and port. Information flows between two sockets. These sockets can be on the same computer, as in the case of inter-process communication (IPC), or on different computers, as in the case of network communication.

In order to transmit data reliably (more on this term later), computers divide the data up into smaller pieces. These pieces are called **packets** **[look up packet on webopedia.com](http://www.webopedia.com/TERM/p/packet.html)** **[look up packet on FOLDOC](http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=packet)** . Each packet contains header information that describes how the packet was formatted, where it came from and so on.

<http://www.portknocking.org/images/packet-headers-simple.png>

<http://www.portknocking.org/images/diagram-background.png>

=======================================

|  |  |  |
| --- | --- | --- |
| 7 | **Application** | e.g. HTTP, SMTP, SNMP, FTP, Telnet, SSH and Scp, NFS, RTSP etc. |
| 6 | **Presentation** | e.g. XDR, ASN.1, SMB, AFP etc. |
| 5 | **Session** | e.g. TLS, SSH, ISO 8327 / CCITT X.225, RPC, NetBIOS, ASP etc. |
| 4 | **Transport** | e.g. TCP, UDP, RTP, SCTP, SPX, ATP etc. |
| 3 | **Network** | e.g. IP/IPv6, ICMP, IGMP, X.25, CLNP, ARP, RARP, BGP, OSPF, RIP, IPX, DDP etc. |
| 2 | **Data Link** | e.g. Ethernet, Token ring, PPP, HDLC, Frame relay, ISDN, ATM, 802.11 Wi-Fi, FDDI etc. |
| 1 | **Physical** | e.g. wire, radio, fiber optic etc. |

|  |  |  |
| --- | --- | --- |
| 4 | **Application layer** | BGP, FTP, HTTP, HTTPS, IMAP, IRC, NNTP, POP3, RTP, SIP, SMTP, SNMP, SSH, SSL, Telnet, UUCP, Finger, Gopher, DNS, RIP, Traceroute, Whois, IMAP/IMAP4, Ping, RADIUS, BGP etc. |
| 3 | **Transport layer** | DCCP, OSPF, SCTP, TCP, UDP, ICMP etc. |
| 2 | **Network/Internet layer** | IPv4, IPv6, ICMP, ARP, IGMP etc |
| 1 | **Physical/ Data**  **Link layer** | Ethernet, Wireless (WAP, CDPD, 802.11, Wi-Fi), Token ring, FDDI, PPP, ISDN, Frame Relay, ATM, SONET/SDH, xDSL, SLIP etc.  RS-232, EIA-422, RS-449, EIA-485 etc. |

the basic function each of the TCP/IP layer is illustrated in the following figure.

<http://www.tenouk.com/Module42_files/image001.png>

A shown in figure 5, the four-layered structure of TCP/IP is seen in the way data handled as it passes down the protocol stack from the Application layer to the underlying physical network.  Each layer in the stack adds control information to ensure proper delivery.  This control information is called a **header** because it is placed in front of the data to be transmitted.  Each layer treats all of the information it receives from the layer above as **data**and places its own header in front of that information.  The addition of delivery information at every layer is called **encapsulation**.  Note that the **real data** that will be transmitted, seen or used at Application layer just a small portion of the whole packet.  When data is received, the opposite process happens.  Each layer strips off its header before passing the real data on the layer above.  As information flows back up the stack, information received from a lower layer is interpreted as both a header and data.

<http://www.tenouk.com/Module42_files/image002.png>

<http://www.tenouk.com/Module42_files/image003.png>

////////////////////////////////////////////////////////////////

Neworking access layer

The Network Access layer it is the lowest layer of the TCP/IP protocol hierarchy.  The protocols in this layer provide the means for the system to deliver data to the other device on a directly attached network.  It defines how to use the network to transmit an IP diagram.  Unlike higher-level protocols, it must know the details of the underlying network to correctly format the data being transmitted to comply with the network constraints.  The TCP/IP Network Access layer can encompass the function of all three lower layers of the OSI reference model Network layer, Data Link layer, and Physical layer.

Functions performed at this level include encapsulation of IP datagrams into the frames transmitted by the network and mapping of IP addresses to the physical addresses used by the network (provided by ARP protocol).  The network access layer is responsible for exchanging data between a host and the network and for delivering data between two devices on the same network.  Node physical addresses (MAC address) are used to accomplish delivery on the local network.

TCP/IP has been adapted to a wide variety of network types, including switching, such as X.21, packet switching, such as X.25, Ethernet, the IEEE 802.x protocols, frame relay, wireless etc.  For example, data in the network access layer encode EtherType (Ethernet) information that is used to demultiplex data associated with specific upper-layer protocol stacks.

**Network/Internet Layer**

The Internet layer is the heart of TCP/IP and the most important protocol.  This layer provides the basic packet delivery service on which TCP/IP networks are built.  The TCP/IP protocol at this layer is the **Internet Protocol**(IP- [RFC 791](http://www.ietf.org/)).  **All** protocols, in the layers above and below Internet layer, use the **Internet Protocol** to deliver data.  **All** TCP/IP data flows through IP, incoming and outgoing, regardless of its final destination.

The Internet layer is responsible for **routing** messages through **internetworks**.  Devices responsible for routing messages between networks are called **gateways** in TCP/IP terminology, although the term **router** is also used with increasing frequency.  In addition to the physical node addresses utilized at the network access layer, the IP protocol implements a system of **logical host** addresses called **IP addresses**.  The IP addresses are used by the**internet** and **higher layers** to identify devices and to perform internetwork routing.  As discussed in the previous Module the IP address may be a class or classless type.  The**Address Resolution Protocol (ARP)** enables IP to identify the physical address (Media Access Control, MAC) that matches a given IP address.  The physical address has been burnt on every NIC.  To make it readable for human being, the (domain) name is used instead of the IP address in normal operation.  The IP address and name resolution is done by Domain Name System (DNS).  In the implementation, UNIX/Linux uses BIND and Windows uses Domain Name Service (also DNS acronym).  The relationship is shown in the following figure.

<http://www.tenouk.com/Module42_files/image005.png>

**Internet Protocol (IP)**

The IP protocol functionalities include:

1. Defining the datagram, which is the basic unit of transmission in the Internet.
2. Defining the Internet addressing scheme, moving data between the Network Access layer and the Transport layer.
3. Routing datagrams to remote hosts.
4. Performing fragmentation and reassembly of datagrams.
5. **The Datagram**
7. Is a packet format defined by Internet Protocol.  The internet protocol delivers the datagram by checking the **Destination Address (DA)**.  This is an IP address that identifies the destination network and the specific host on that network.  If the destination address is the address of a host on the local network, the packet is delivered directly to the destination; otherwise the packet is passed to a gateway/router for delivery.  Gateways are devices that switch packets between the different physical networks.  Deciding which gateway to use is called **routing**.  IP makes the routing decision for each individual packet.  IP deals with data in chunks called **datagrams**.  The terms packet and datagram are often used interchangeably, although a packet is a data link-layer object and a datagram is a network layer object.  In many cases, particularly when using IP on Ethernet, a datagram and packet refer to the same chunk of data.  There's no guarantee that the physical link layer can handle a packet of the network layer's size.  If the media's MTU is smaller than the network's packet size, then the network layer has to break large datagrams down into packed-sized chunks that the data link layer and physical layer can digest.  This process is called **fragmentation**.  The host receiving a fragmented datagram reassembles the pieces in the correct order.
8. **Pv4 Datagram Format**
10. The following figure shows the IPv4 datagram header format.  It is 6 x 32 bits (word size) wide.

<http://www.tenouk.com/Module42_files/image006.png>

A brief field description:

|  |  |
| --- | --- |
| **Field** | **Description** |
| Version | The version of IP currently used. |
| IHL | IP Header Length (IHL) - datagram header length. Points to the beginning of the data. The minimum value for a correct header is 5. |
| Type of Service | Data in this field indicate the quality of service desired.  The effects of values in the precedence fields depend on the network technology employed, and values must be configured accordingly.  Format of the Type of Service field:   * Bits 0-2: Precedence   111 = Normal Control.  110 = Internetwork Control.  101 = CRITIC/ECP.  100 = Flash Override.  011 = Flash.  010 = Immediate.  001 = Priority.  000 = Routine.   * Bit 3 : Delay 0 = normal delay, 1 = low delay. * Bit 4 : Throughput 0 = normal throughput, 1 = high throughput. * Bit 5 : Reliability 0 = normal reliability, 1 = high reliability. * Bits 6-7: Reserved |
| Total Length | The length of the datagram in byte, including the IP header and data.  This field enables datagrams to consist of up to 65,535 bytes.  The standard recommends that all hosts be prepared to receive datagrams of at least 576 bytes in length. |
| Identification | An identification field used to aid reassembles of the fragments of a datagram. |
| Flags | If a datagram is fragmented, the MB bit is 1 in all fragments except the last.  This field contains three control bits:   * Bit 0: Reserved, must be 0. * Bit 1 (**DF**): 1 = **D**o not **f**ragment and 0 = May fragment. * Bit 2 (**MF**): 1 = **M**ore **f**ragments and 0 = Last fragment. |
| Fragment Offset | For fragmented datagrams, indicates the position in the datagram of this fragment. |
| Time-to-live | Indicates the maximum time the datagram may remain on the network. |
| Protocol | The 8 bits field of the upper layer protocol associated with the data portion of the datagram.  For a complete information please refer to RFC 1700 and the following is some of the protocol numbers:    **Decimal**              **Protocol**  1 ICMP (Internet Control Message)  2 IGMP (Internet Group Management)  4 IP (IP in IP -encapsulation)  5 ST (Stream)  6 TCP (Transmission Control)  17 UDP (User Datagram)  27                        RDP  (Reliable Data Protocol) |
| Header Checksum | A checksum for the header only. This value must be recalculated each time the header is modified. |
| Source Address | The IP address of the originated the datagram. |
| Destination Address | The IP address of the host that is the final destination of the datagram. |
| Options | May contain 0 or more options. |
| Padding | Filled with bits to ensure that the size of the header is a 32-bit multiple. |
| Table 1: IP datagram fields description. | |

Note that in the IP packet we just have the source and destination IP addresses.  There is no source and destination port numbers here which is set in UDP or TCP header.

**Internet Control Message Protocol (ICMP and ICMPv6)**

Is part of the **Internet layer** and **uses the IP datagram delivery facility** to sends its messages.  ICMP sends messages that perform control, error reporting, and informational functions for TCP/IP.  The RFC document for ICMP is RFC 792.  The following figure is the ICMP header format.

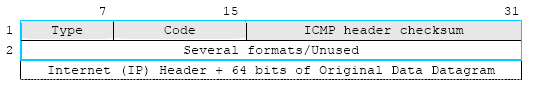


Figure 10: The ICMP Header Format.

A brief description:

|  |  |
| --- | --- |
| **Field** | **Description** |
| Type | Messages can be error or informational messages.  Error messages can be Destination unreachable, Packet too big,Time exceed, Parameter problem.  The possible informational messages are, Echo Request, Echo Reply, Group Membership Query, Group Membership Report and Group Membership Reduction.  A summary of message Types are listed below.    0:  Echo Reply.  3:  Destination Unreachable.  4:  Source Quench.  5:  Redirect.  8:  Echo.  11:  Time Exceeded.  12:  Parameter Problem.  13:  Timestamp.  14:  Timestamp Reply.  15:  Information Request.  16:  Information Reply. |
| Code | For each type of message as listed above, several different codes are defined.  An example of this is the Destination Unreachable message, where possible messages are: no route to destination, communication with destination administratively prohibited, not a neighbor, address unreachable, port unreachable.  The code and its means forDestination Unreachable message is listed below.    0 = net unreachable.  1 = host unreachable.  2 = protocol unreachable.  3 = port unreachable.  4 = fragmentation needed and DF set.  5 = source route failed. |
| Checksum | The 16-bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type.  For computing the checksum, the checksum field should be zero. |
| Second word  (Several formats/unused | Several formats that match with certain IP header fields/depend on the Type and Code fields. |
| Table 2: ICMP datagram fields description. | |

The usage examples of the ICMP (together with IP) are listed below:

1. Flow control: When datagrams arrive too fast for processing, the destination host or intermediate gateway sends an ICMP Source Quench Message back to the sender.  This tells the source to temporarily stop sending datagrams.
2. Detecting unreachable destinations: When a destination is unreachable, the system detecting the problem sends an ICMP Destination Unreachable Message to the datagrams source.  If the unreachable destination is a network or host, the message is sent by an intermediate gateway.  But if the destination is an unreachable port, the destination host sends the message.
3. Redirecting routes: A gateway sends the ICMP Redirect Message to tell a host to use another gateway, presumably because the other gateway is a better choice.  This message can only be used when the source host is on the same network as both gateways.
4. Checking remote hosts: A host can send the ICMP Echo Message to see if a remote system's internet protocol is up and operational. When a system receives an echo message, it sends the same packet back to the source host (e.g. PING command).

Other message types include:

1. Information Request or Information Reply Message.
2. Timestamp or Timestamp Reply Message.
3. Parameter Problem Message.
4. Time Exceeded Message.

Unless otherwise noted under the individual format descriptions as explained above, the values of the Internet Protocol (IP) header fields for the ICMP are as follows:

|  |  |
| --- | --- |
| **IP Field** | **Description** |
| Version | 4. |
| IHL | Internet header length in 32-bit words. |
| Type of Service | 0. |
| Total Length | Length of internet header and data in octets. |
| Identification, Flags,  Fragment Offset | Used in fragmentation. |
| Time to Live | Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse. |
| Protocol | ICMP = 1. |
| Header Checksum | The 16 bit one's complement of the one's complement sum of all 16 bit words in the header.  For computing the checksum, the checksum field should be zero.  This checksum may be replaced in the future. |
| Source Address | The address of the gateway or host that composes the ICMP message.  Unless otherwise noted, this can be any of a gateway's addresses. |
| Destination Address | The address of the gateway or host to which the message should be sent. |
| Table 3: IP fields description when used with ICMP.  ////////////////////////////////////////////////////////////////////////////////////////////////////////// | |

<http://www.laneye.com/network/ethernet-network-packet-holding-an-ip-packet.gif>

///////////////////////////////////////////////////////////////////////////////////////////////

<https://www.segger.com/embos-ip-structure.html>

<https://www.segger.com/admin/uploads/imageBox/embosip_struct.gif>

**Application layer**

The API Layer is the interface between embOS/IP and the user application. It uses the embOS/IP API to transmit data over an TCP/IP network. The embOS/IP API provides functions in Berkley socket style, such as connect(), bind(), listen(), etc.

**Transport layer**

The transport layer provides end-to-end communication services for applications. The two relevant protocols of the transport layer protocol are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP is a reliable connection-oriented transport service. It provides end-to-end reliability, resequencing, and flow control. UDP is a connectionless transport service.

**Internet layer**

All protocols of the Transport layer use the Internet Protocol (IP) to carry data from source host to destination host. IP is a connectionless service, providing no end-to-end delivery guarantees. IP datagrams may arrive at the destination host damaged, duplicated, out of order, or not at all. The Transport layer is responsible for reliable delivery of the datagrams when it is required. The IP protocol includes provision for addressing, type-of-service specification, fragmentation and reassembly, and security information.

**Link layer**

The Link layer provides the implementation of the communication protocol used to interface to the directly-connected network. A variety of communication protocols have been developed and standadized. The most commonly used protocol is Ethernet (IEEE 802.3). In this version of embOS/IP is only Ethernet supported.

## Encapsulation

The four layers structure is defined in RFC 1122. The data flow starts at application layer goes over the transport layer, the network layer, and the link layer. Every protocol adds an protocol-specific header and encapsulates the data and header from the layer above as data. On the receiving side the data will be extracted in the complementary direction. The opposed protocols do not know which protocols on the layers above and below are used. The following illustration shows the encapsulation of data within an UDP datagram within an IP packet.

<https://www.segger.com/admin/uploads/imageBox/embosip_encapsulation.gif>

//////////////////////////////////////////////////////////////////////////////////////////

<https://tournasdimitrios1.files.wordpress.com/2011/02/tcpdump-encapsulation.jpg>

///////////////////////////////////////////////////////////////////////

<http://www.unixwiz.net/techtips/iguide-ipsec.html>

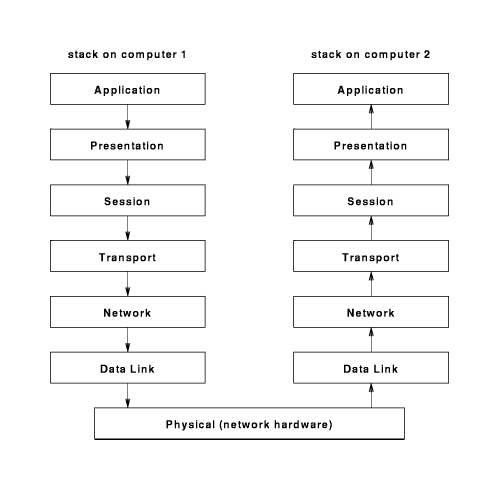
///////////////////////////////////////////////////////////////////////////////////////////

<http://www.cs.buap.mx/~dpinto/networking/osistack.html>

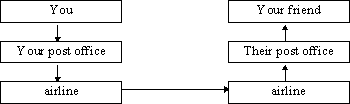
Standards are extremely important in the field of networking. The whole purpose of a network is to allow different computers to communicate. If the computers do not follow a common standard, they will not be able to communicate.

One of the most important standards in networking is the International Standards Organization Open System Interface, known as ISO OSI. OSI is both a standard and a network architecture model. As a standard, it is not very often followed. Some networks are OSI standard networks, but most are not. OSI has been more popular in Europe. The Internet Protocol, probably the most commonly used network protocol, is not an OSI standard protocol. The OSI model, on the other hand, has proved to be a popular model for network architectures. The Internet Protocol does not exactly follow the OSI model, but it is relatively close.

The OSI model divides the many networking functions into seven different layers. This is often called the *OSI stack*. Each layer provides functions or services for the layer above it. Each layer calls upon services provided by the layer below it. Each layer is supposed to be independent of the other layers so that a change in one layer will not affect other layers. (This is not always the case.) The layers are implemented in each computer on the network (although some intermediate nodes may not support all of the upper layers.) Each layer communicates with its peer layer in another computer. Although the logical communication is between peer layers on different computers, the actual flow of information is down the protocol stack on the sending computer and then up the protocol stack on the receiving computer. When a layer wants to send something to its peer layer in another computer, it calls a function in the layer below it to actually send the data. Only the lowest layer actually sends bits to another computer.



An analogy to the OSI layers can be found in sending a package to a friend in another state. The desired communication is between you and your friend. Because you cannot actually hand the package to your friend, you give it to the post office. The post office gives it to an airline, which actually transfers the package to your friend's city. The airline then gives the package to the post office which then gives it to your friend.



You do not have to worry about how to find your friend's house in the distant city. That is the responsibility of the post office in that city. You just have to specify your friend's address. The post office doesn't have to be concerned with how to fly an airplane. That is the responsibility of the airline. Each layer assumes that the layer below it will provide certain functions. Each layer provides additional functionality.

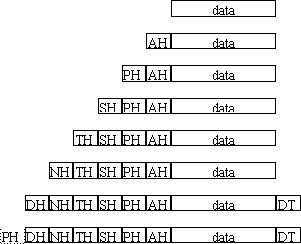
The seven layers of the OSI model are:

|  |  |  |
| --- | --- | --- |
| **layer** | **purpose** | **example** |
| application | Provides network services. | X.400 email, HTTP, FTP, telnet |
| presentation | Converts the data to the representation used by the local computer. May exchange bytes of integers to resolve Big Endian - Little Endian problems. |  |
| session | Establishes sessions. |  |
| transport | Multiplexes data streams from different applications. Directs packets to the correct user on a computer. This is the first end-to-end layer. May also provide error correction. | Transport Control Protocol (TCP) |
| network | Finds a route for packets to take through the network. Directs packets to the correct computer | Internet Protocol (IP) |
| data link - logical data link | Detects and corrects any errors on the link. Provides flow control. |  |
| data link - media access control | For local area networks where all computers share a communications media, this layer determines which node is allowed to transmit. | Ethernet, Token Ring |
| physical | Defines the characteristics of the physical connections, such as type of wire, plug shape, how is a zero or one bit represented and what voltages are used. This is the only layer that actually sends bits to another computer. | Sonet, RS-232C |

The OSI model was developed over 20 years ago. Much has been learned about networking in the past 20 years. If the OSI model was developed today, it would probably be somewhat different. The data link layer would probably be split into two distinct layers. The session layer has very little to do and would probably be eliminated.  Some people have complained humorously about the[OSI model](http://www.europa.com/~dogman/osi/). The Internet protocol follows a model which is similar to the OSI model. Because the Internet was designed to run on top of existing different networks, it does not define the lower layers.

|  |  |  |  |
| --- | --- | --- | --- |
| **layer** | **purpose** | **OSI equivalent** | **example** |
| Application | Provides network services. | Application | HTTP, FTP, Telnet |
| Transport | Multiplexes data streams from different applications. May also provide error correction. | Transport | TCP, UDP |
| Internet | Routing. | Network | IP |
| Network Interface | Provides access to the Data Link and lower protocols. The IP stack does not define the lower levels. | Data Link | Ethernet |

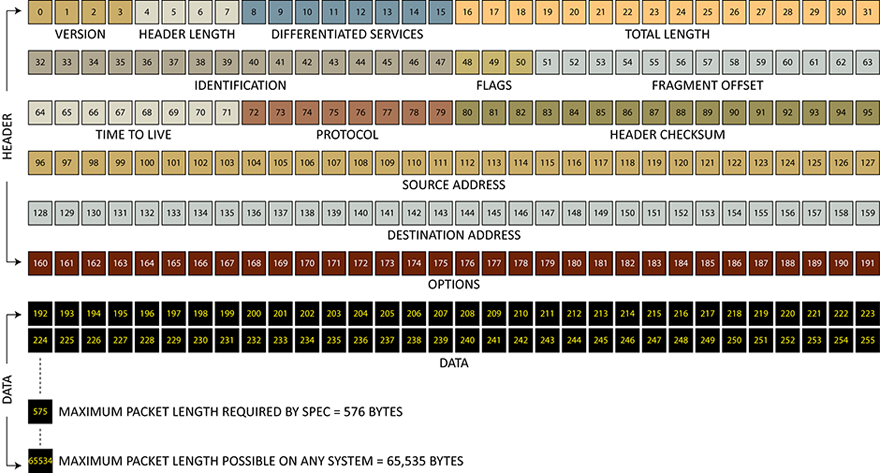
Each layer communicates with its peer layer by prefixing the data from the above layer with a header.



////////////////////////////////////////////////////////////////////

<http://essayweb.net/miscellany/datatransmission.shtml>

The basic structure of a IP packet is shown below:



Each box represents a single bit, numbered starting from bit 0. As can be seen, it contains two broad sections, the data itself (or payload), and a header section consisting of meta data (information describing the data itself, addresses, etc.). The total length of an IP packet is variable. The general rules for the packet size are:

* All packets must contain a header with contains a certain minimum number of fields. All the fields shown in the diagram above are required, except the field marked "Options". Since the total length of the required fields is 160 bits (20 bytes), the **minimum length of an IP packet is 20 bytes**. Such a minimum-length packet will contain no data (a header-only packet is perfectly legal for IP), and it will not contain the optional part of the header.
* The maximum length possible for an IP packet is determined by the "Total Length" field of the header shown above. This field is described in more detail later, but here were can see that this field is 16 bits in length, therefore the maximum value it can contain is 216, or 65536. Since zero is explicitly allowed, therefore the maximum possible length is really 65536 - 1 = 65,535. The unit of "Total Length" is bytes, so **the maximum size possible for a IP packet is 65,535 bytes** or 524,280 bits.
* In practice, many systems do not allow packet sizes that large. For this reason, the specification calls for some minimum required limit, which is 576 bytes. So any networking software written and any networking hardware manufactured **must allow packet sizes of at least 576 bytes**, in order to meet the spec. In reality, most software and hardware allows much larger packet sizes, though they may not always allow the maximum possible size (65,535 bytes).
* Regardless of packet size, it is always possible for the Data-Link Layer to break down large IP packets into smaller ones, or combine small packets into larger chunks. These chunks are called "frames", so there is no confusion with the packet itself. The Data-Link Layer at the other end will convert the frames back into IP packets, so the IP packet size is preserved at the other end.

Here's a more detailed description of the header fields.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | The version of IP used to create the packet. Currently, we use IPv4, so it contains the value 4. This is a 4-bit field, so values of 0 to 15 are allowed. | | |
| **Header Length** | Also called the Internet Header Length (IHL), it simply describes the length of the packet's header, in units of 32-bit words. This is a 4-bit field, so values of 0 to 15 are possible. Since headers must contain all the required fields, the minimum possible header length is 160 bits (5 words). So this field will contain a minimum value of 5. The maximum value is 15, which corresponds to 15 x 32 = 480 bits, or 60 bytes. That is the maximum header size possible. This field can be used as an offset to read the data. Simply multiply the value by 32, count off that many bits from the start of the packet, and start reading data. | | |
| **Differentiated Services** | This 8-bit field was originally known as the TOS (Type of Service) field. It's basically a way for the host to express a preference for how it wants that packet handled - fast and less reliable, slower but more reliable, using a more expensive route, using a cheaper route, etc. It's not been much implemented so far, but will probably be important in IPv6 when there's a lot of real time traffic involved (audio/video stuff). The 8 bits of this field are apportioned as shown below. | | |
|  | 0 - 2 | Precedence, or priority: 3 bits, 8 possible values from very high to very low. |
| 3 | Delay: two values possible, normal delay or low delay |
| 4 | Throughput: two values possible, normal or high throughput |
| 5 | Reliability: two values possible, normal or high reliability |
| 6 | Cost: two values possible, normal cost or minimize cost |
| 7 | Undefined |
| Some switches may read the information in these bits, and route traffic accordingly. Others may totally ignore this information and handle all packets in exactly the same manner. This is therefore a feature of how a particular network is implemented. | | |
| **Total Length** | This 16-bit field defines the total length of the packet (header + data). The unit is bytes. It can hold values from 0 to 65535 in theory. Legally, the minimum value is 20 (minimum header plus no data). Legally, the maximum must be at least 576, but it can be much larger up to a limit of 65535. This means that the maximum possible size for any IP packet is 65535 bytes, or half a megabit. In practice, it may be much smaller, depending upon the software/hardware generating the packets. However, the maximum allowable must be at least 576 bytes to meet specs. | | |
| **Identification** | This 16-bit field is not always used. Its intended purpose was originally as a unique ID or identifier for different fragments of an IP packet. It's not much used for that purpose today. Some people suggest that it could be used to add information to prevent source address spoofing. | | |
| **Flags** | This is a 3-bit field which contains flags that help manage fragmentation. Fragmentation is described in more detail in a section below, but for now, remember that IP packets can be fragmented if they are too large to go through lower layers (which may have packet size limitations imposed by their own protocol). Therefore, information is needed that helps those layers to fragment these packets. This information is provided via the Flags field and the Fragment Offset field. The 3 bits reserved for the Flags are used as follows. | | |
|  | 0 | Reserved bit. Not used. Must be a 0. |
| 1 | two possible values, 1 means "don't fragment", which lower layers understand as "just trash this packet if it's too big to send without fragmentation". 0 means "okay to fragment". |
| 2 | two possible values: these values are actually set by the lower layers if fragmentation is needed. A value of 1 means "more fragments", meaning that this packet is a fragment of a larger packet, and more packets are to follow. A value of 0 means that this is the last fragment of the series. So if a large IP packet is fragmented into 5 smaller IP packets, the first 4 will have the "more fragments" flag set and the 5th one won't. |
| **Fragment Offset** | This 13-bit field tells the host at the receiving end how to re-assemble a packet that was fragmented. The unit is a block of 8 bytes. Since it can have values from 0 to 8191 (13 bits), it can provide a maximum offset of 8191 x 8 = 65528 bytes. This is sufficient to cover the maximum possible length of an IP packet (65535 bytes minus header). This field is again used by lower layers, in case a large IP packet needs to be fragmented. Each fragment is stamped with a fragment offset, which is used to re-assemble the original packet at the other end. For example, a fragment offset of 107 means that the data contained in this fragment belongs to position 107 x 8 = 856, measured from the beginning of the original packet. | | |
| **Time to Live (TTL)** | This 8-bit field specifies exactly what it says: how long a packet should "live". The units are seconds, so values can be 0 to 255. However, 0 is not a legal value, and any value less than 1 is rounded up to 1. So the legal values are 1 to 255 seconds. The purpose of this field is to prevent a packet from being forwarded forever in a circle, since routes to hosts aren't always known exactly in advance. Every time a packet encounters a switch or router, the TTL is decremented by 1 before it's passed on. When it hits 0, the packet is discarded and no longer forwarded. The switch/router that decrements it to 0 sends an ICMP message back to the sender informing him that the package was discarded. This feature can be used to implement traceroutes. | | |
| **Protocol** | This 8-bit field defines the protocol used for the data or payload of the packet. It can have values from 0 to 255, each of which specifies a certain Transport Layer protocol. The Internet Assigned Numbers Authority [maintains a list](http://en.wikipedia.org/wiki/List_of_IP_protocol_numbers) which assigns a specific number to a certain protocol. For example, a value of 6 means [TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol), or Transport Control Protocol, which is commonly used for the data in TCP packets. A value of 1 means [ICMP](http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) (Internet Control Message Protocol), which is used for standard messages used for housekeeping/control functions in networks. | | |
| **Header Checksum** | This is simply a checksum calculated for the header portion of the packet, which is used for error checking. Note that the data is not included in calculating the header checksum. Integrity of the data is the responsibility of the data protocol used. For example, TCP has its own separate checksum for verifying the data integrity. Each switch/router along the route calculates a checksum for the header and compares it against the value in this field. If they don't match, it requests re-transmission of the packet. Note that since each switch changes the header (by decrementing the TTL field), it must calculate and embed a new checksum in each packet before forwarding it. | | |
| **Source Address** | This is usually an IPv4 address in binary format. The address field is 32 bits, and can be thought of as a series of four 8-bit fields. Each 8-bit field can contain a value from 0 to 255. Therefore, an IPv4 address of the form 202.134.227.153, for example, can be contained by first converting each of its four parts into binary, then concatenating them together. So it would be: 202 = 11001010; 134 = 10000110; 227 = 11100011; 153 = 10011001. So the value of source address would be obtained by concatenating these four binary numbers: 11001010100001101110001110011001. Note that because of [NAT](http://en.wikipedia.org/wiki/Network_address_translation) (Network Address Translation) the source address might not be the address of the actual host where the packet originated, but rather the address of the NAT machine. Replies will therefore be sent back to the NAT machine, which will forward them to the appropriate host. | | |
| **Destination Address** | This is the IPv4 address of the destination machine, again in binary format as described for the Source Address field. | | |
| **Options** | This is an optional part of the header, and is hardly ever used in IPv4. Since the minimum header length is 20 bytes (with no options) and the maximum header length is 60 bytes (determined by the maximum value possible in the header length field), these options can use up to a maximum of 40 bytes. The first 2 bytes (16 bits) are reserved for the options header, and the remainder for the options data. The bit assignment for the header is shown below. | | |
| Copied | 1 bit | Should options be copied into all fragments if the packet is fragmented. 1 means yes. |
| Class | 2 bits | 4 possible values. Generally used to categorize the options. 0 means "control options" 2 is for "debugging and measurement options", while 1 and 3 are currently not used. |
| Number | 5 bits | The option number, uniquely identifies each option contained in this part of the header. A maximum of 32 options are possible. |
| Length | 8 bits | The length of the option in bits. Includes the length of this field as well. |
| Data | Variable | Any data used by the options. The length is variable, with the constraint that the total optional part of the header can't be more than 40 bytes. Simple options might not have any data associated with them. |
| **Data** | The actual data or payload of the packet. This is not part of the header and not included in the header checksum. The data can be any of the Transport Layer protocols, as defined in the Protocol field of the header. Usually over the Internet, the data will be a Layer 4 packet, either a TCP packet or a UDP packet. | | |

As you can see from the table, the IP packet contains all data needed for transmitting a packet from a host to a destination. Therefore, an IP packet is the smallest unit that is independently routable end-to-end, meaning from the client computer to the host computer, and vice versa.

### Fragmentation of IP Packets

Since IPv4 traffic can go over a variety of networks, including both WAN and LANs (such as ethernet), the issue of packet size becomes important. When designing the protocol, the designers might have picked an arbitrarily small packet size to make sure it never got fragmented, because it was smaller than the[MTU](http://en.wikipedia.org/wiki/MTU_%28networking%29) (Maximum Transmission Unit) of the common networking infrastructures. However, this would have been inefficient, because smaller packet size decreases data density, since a proportionately larger fraction of the bandwidth is spent in simply transmitting headers.

Instead, it was decided to allow for a system of fragmentation, so that larger packets could be fragmented and re-assembled only as needed, for the particular type of network on which they would be used.

As mentioned above, packets are first generated from data by the Transport layer (TCP packets or UDP packets). These packets are generated without any concerns about fragmentation or MTU size (since the Transport layer is not network-aware, so it doesn't even know what an MTU is), so they are almost always fragmented by the Network layer when it repackages them into IP packets.

|  |  |  |
| --- | --- | --- |
| **Media** | **MTU (bytes)** | **Comments** |
| Ethernet v2 | 1500 | The vast majority of Ethernet implementations are Ethernet v2. |
| Ethernet 802.3 | 1492 |
| Ethernet Jumbo Frames | 1500-9000 |
| 802.11 | 2272 |  |
| 802.5 | 4464 |  |
| FDDI | 4500 |  |
| Internet IPv4 | 68+ | These are minimum values, practically they are much higher. |
| Internet IPv6 | 1280+ |
| MTU Sizes for Common Media | | |

However, since the Network layer is aware of MTU restrictions, the size of the IP packets it generates can be finely tuned.

Over a LAN with a relatively homogenous structure, fragmentation is not much of a problem. The host knows the MTU of its own interface (and possibly the MTU of other hosts around it, through handshaking). It can set the packet size so it doesn't exceed the MTU. Most LANs use some standard protocol like ethernet, which has some specified minimum MTU size. So packet size can be based on that.

The table on the right shows MTU sizes for a variety of media. Consider Ethernet v2, which is the commonest Ethernet standard used. Chances are, the computer you are using right now is connected to a LAN running Ethernet. The LAN will probably have a router or gateway to the Internet (could be as simple as the cable modem from your local cable company, or a DSL modem). Ethernet v2 has a MTU of 1500 bytes. If you wanted to guarantee that there would be no fragmentation at this level, you would make sure that no packet exceeded 1500 bytes. Remember, the 1500 bytes includes both data and header.

On the Internet, a packet encounters a wide range of networks and equipment, and the MTU size may vary from hop to hop. This makes it difficult to choose a large enough packet size to minimize overhead, and still stay below the MTU of all hops along the route. There are procedures for discovering the MTU size of all hops along the route ([Path MTU Discovery](http://en.wikipedia.org/wiki/Path_MTU_discovery)) in order to optimize packet size before transmission. The procedure is to basically send a series of packets of smaller and larger sizes with the "don't fragment" flag set. If a packet is too big for a switch/router to forward without fragmentation, it will simply drop the packet and send back an ICMP "Fragmentation Needed" message, which contains its MTU value. In this way, the smallest MTU size along a route can be discovered, and the packet size set accordingly.

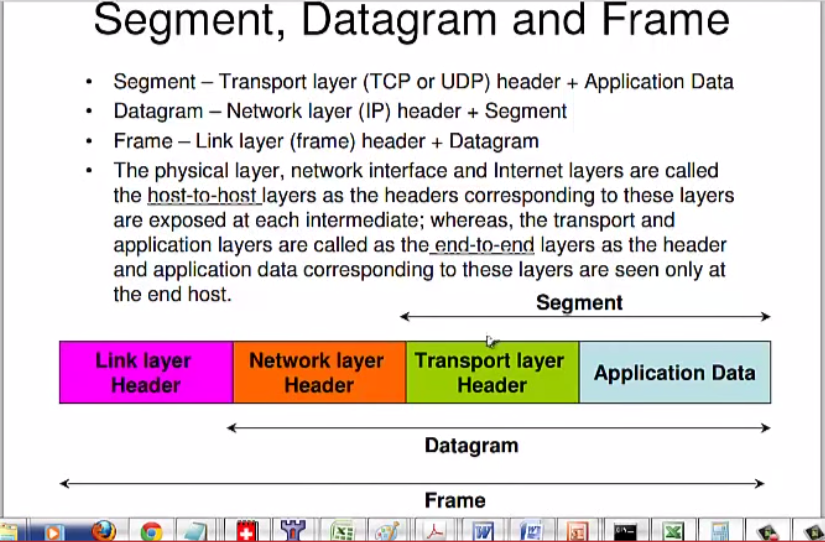
However, many network routers are configured incorrectly and do not send back ICMP errors. This might be deliberate practice out of fear of DDoS attacks. So Path MTU Discovery methods are not foolproof. Since overheads caused by fragmentation are an immensely important source of bandwidth loss across the Internet, these issues are an active area of work and research.

Continue on to [Page 2](http://essayweb.net/miscellany/datatransmission2.shtml) of this article, which talks about encapsulation and TCP packets.

Game networking expert

<http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/>

<https://technet.microsoft.com/en-us/library/cc700820.aspx>



What thing you should understand after this artcle

to be able to play games we need to send information across to the internet.but long time age there was a lot and iso win the battle , the iso is not a actual physic , but a ? . look at it

so able to to know what thimgs slow us we first go deep and and find out what information are contain in there.one of the model which actuallt exits is called TCP/IP this is refecening ISO model.

A game program network expert have a dicuss on about

Frpm the above any we notoic that some information are are needed include