LINUX SOCKET PART 16
Advanced TCP/IP -The TCP/IP Protocols Details

Menu

Network Story 1
Network Story 2
Network Story 3
Network Story 4
Network Story 5
Network Story 6
Socket Example 1
Socket Example 2
Socket Example 3
Socket Example 4
Socket Example 5
Socket Example 6
Socket Example 7
Advanced TCP/IP 1
Advanced TCP/IP 2
Advanced TCP/IP 3
Advanced TCP/IP 4
Advanced TCP/IP 5

My Training Period: hours

Note:
This is a continuation from Part IV series, Advanced TCP/IP Programming Tutorial. Working program examples if any compiled using gcc, tested using the public IPs, run on **Fedora Core 3**, with several times of update, as root or SUID 0. The Fedora machine used for the testing having the "No Stack Execute" disabled and the SELinux set to default configuration. This Module will concentrate on the TCP/IP stack and will try to dig deeper till the packet level.

The protocols: IP, ICMP, UDP and TCP

To fabricate our own packets, what we all need to know is the structures of the protocols that need to be included. We can define our own protocol structure (packets' header) then assign it with new values or we just assign new values for the standard built-in structures' elements. Below you will find detail information of the IP, ICMP, UDP and TCP headers. Unix/Linux systems provide standard structures for the header files, so it is very useful in learning and understanding packets by fabricating our own packet by using a struct, so we have the flexibility in filling the packet headers. We can always create our own struct, as long as the length of each field is correct. In building our program later on, note also the little endian (Intel x86) notation and the big endian based machines (some processor architectures other than Intel x86 such as Motorola). The following sections try to analyze header structures that will be used to construct our own packet in the program examples that follows, so that we know what values should be filled in and which meaning they have. The data types that we need to use are: unsigned char (1 byte/8 bits), unsigned short int (2 bytes/16 bits) and unsigned int (4 bytes/32 bits). Some of the information presented in the following sections might be a repetition from the previous one.

IP

The following figure is IP header format that will be used as our reference in the following discussion.

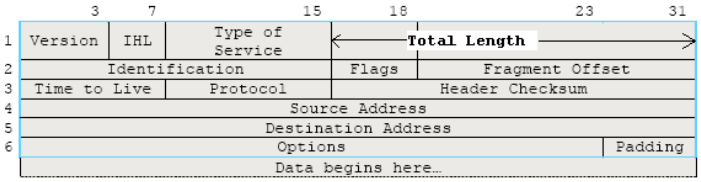

Figure 23: IP header format.

The following is a structure for IP header example. Here we try defining all the IP header fields.

```
struct ipheader {
 unsigned char          iph_ihl:4, ip_ver:4;
 unsigned char          iph_tos;
 unsigned short int     iph_len;
 unsigned short int     iph_ident;
 unsigned char          iph_flags;
 unsigned short int     iph_offset;
 unsigned char          iph_ttl;
 unsigned char          iph_protocol;
 unsigned short int     iph_chksum;
 unsigned int           iph_source;
 unsigned int           iph_dest;
};
```

The **Internet** Protocol is the **network** layer protocol, used for routing the data from the source to its destination. Every datagram contains an IP header followed by a transport layer protocol such as tcp or udp. The following Table is a list of the IP header fields and their information.

Element/field	Description
iph_ver	4 bits of the version of IP currently used, the ip version is 4 (other version is IPv6).

iph_ihl	4 bits, the ip header (datagram) length in 32 bits octets (bytes) that point to the beginning of the data. The minimum value for a correct header is 5. This means a value of 5 for the iph_ihl means 20 bytes (5 * 4). Values other than 5 only need to be set if the ip header contains options (mostly used for routing).
iph_tos	8 bits, type of service controls the priority of the packet. 0x00 is normal; the first 3 bits stand for routing priority, the next 4 bits for the type of service (delay, throughput, reliability and cost). It indicates the quality of service desired by specifying how an upper-layer protocol would like a current datagram to be handled, and assigns datagrams various levels of importance. This field is used for the assignment of Precedence, Delay, Throughput and Reliability. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput. Bits 0-2: Precedence. 111 - Network Control 110 - Internetwork Control 101 - CRITIC/ECP 100 - Flash Override 011 - Flash 010 - Immediate 001 - Priority 000 – Routine Bit 3: 0 = Normal **D**elay, 1 = Low Delay. Bits 4: 0 = Normal **T**hroughput, 1 = High Throughput. Bits 5: 0 = Normal **R**eliability, 1 = High Reliability. Bit 6-7: Reserved for Future Use.

0	1	2	3	4	5	6	7
Precedence			D	T	R	0	0

	The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set. The type of service is used to specify the treatment of the datagram during its transmission through the internet system. The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.
iph_len	The total is 16 bits; total length must contain the total length of the ip datagram (ip and data) in bytes. This includes ip header, icmp or tcp or udp header and payload size in bytes. The maximum length could be specified by this field is 65,535 bytes. Typically, hosts are prepared to accept datagrams up to 576 bytes (whether they arrive whole or in fragments).
iph_ident	The iph_ident sequence number is mainly used for reassembly of fragmented IP datagrams. When sending single datagrams, each can have an arbitrary ID. It contains an integer that identifies the current datagram. This field is assigned by sender to help receiver to assemble the datagram fragments.
iph_flag	Consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used. The Control Flags: Bit 0: reserved, must be zero. Bit 1: (**DF**) 0 = May Fragment, 1 = **D**on't **F**ragment. Bit 2: (**MF**) 0 = Last Fragment, 1 = **M**ore **F**ragments.

0	1	2
0	DF	MF

ihp_offset	The fragment offset is used for reassembly of fragmented datagrams. The first 3 bits are the fragment flags, the first one always 0, the second the do-not-fragment bit (set by ihp_offset = 0x4000) and the third the more-flag or more-fragments-following bit (ihp_offset = 0x2000). The following 13 bits is the fragment offset, containing the number of 8-byte big packets already sent. This 13 bits field indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.
iph_ttl	8 bits, time to live is the number of hops (routers to pass) before the packet is discarded, and an icmp error message is returned. The maximum is 255. It is a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.
iph_protocol	8 bits, the transport layer protocol. It can be tcp (6), udp (17), icmp (1), or whatever protocol follows the ip header. Look in /etc/protocols or RFC 1700 for more. It indicates which upper-layer protocol receives incoming packets after IP processing is complete.
iph_chksum	16 bits, a checksum on the header only, the ip datagram. Every time anything in the datagram changes, it needs to be recalculated, or the packet will be discarded by the next router. It helps ensure IP header integrity. Since some header fields change, e.g., Time To Live, this is recomputed and verified at each point that the Internet header is processed.
iph_source	32 bits, source IP address. It is converted to long format, e.g. by inet_addr(). Can be chosen arbitrarily (as used in IP spoofing).
iph_dest	32 bits, destination IP address, converted to long format, e.g. by inet_addr(). Can be chosen arbitrarily.

Options	Variable. The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation. In some environments the security option may be required in all datagrams. The option field is variable in length. There may be zero or more options.
Padding	Variable. The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

Table 9: IP header fields description.

Fragmentation

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol (IP) are called intranet fragmentation. Fragmentation of an internet datagram is necessary when it originates in a local network that allows a large packet size and must traverse a local network that limits packets to a smaller size to reach its destination. An internet datagram can be marked "don't fragment". When the internet datagram is marked like that, it is not to be internet fragmented under any circumstances. If internet datagram that has been marked as "don't fragment" cannot be delivered to its destination without fragmenting it, it will be discarded instead.
The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the **identification** field to ensure that fragments of different datagrams are not mixed. The **fragment offset** field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The **more-fragments** flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.
The identification field is used to distinguish the fragments of one datagram from another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.
To fragment a long internet datagram, an internet protocol module (for example, in a gateway/router), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on an 8 bytes (64 bit) boundary (the second portion might not be an integral multiple of 8 bytes, but the first must be). The number of 8 byte blocks in the first portion is called NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.
This procedure can be generalized for an n-way split, rather than the two-way split described. To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the **same value** for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

ICMP

IP itself has no mechanism for establishing and maintaining a connection, or even containing data as a direct payload. **Internet Control Messaging Protocol** is merely an **addition** to IP to carry error, routing and control messages and data, and is often considered as a protocol of the network layer. The following is ICMP header format.

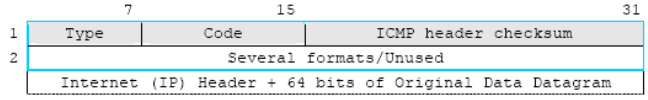

Figure 24: ICMP header format.

The following example is a structure that tries to define the ICMP header. This structure defined for Echo or Echo Reply Message.

```
struct icmpheader {
 unsigned char      icmph_type;
 unsigned char      icmph_code;
 unsigned short int icmph_chksum;
 /* The following data structures are ICMP type specific */
 unsigned short int icmph_ident;
 unsigned short int icmph_seqnum;
}; /* total ICMP header length: 8 bytes (= 64 bits) */
```

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. The following Table lists all the information for the previous structure element (the ICMP header's fields).

Element/field	Description
icmph_type	The message type, for example 0 - echo reply, 8 - echo request, 3 - destination unreachable. Look in for all the types. For each type of message 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. For further details, refer to the standard.
icmph_code	This is significant when sending an error message (unreach), and specifies the kind of error. Again, consult the include file for more. 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.
icmph_chksum	The checksum for the ICMP header + data. Same as the IP checksum. Note: The next 32 bits in an ICMP packet can be used in many different ways. This depends on the ICMP type and code. The most commonly seen structure, an ID and sequence number, is used in echo requests and replies, but keep in mind that the header is actually more complex.

icmph_ident	An identifier to aid in matching requests/replies; may be zero. Used to echo request/reply messages, to identify the request.
icmph_seqnum	Sequence number to aid in matching requests/replies; may be zero. Used to identify the sequence of echo messages, if more than one is sent.

Table 10: ICMP header fields description.

The following is an example of the ICMP header format as defined in the above structure for Echo or Echo Reply Message.

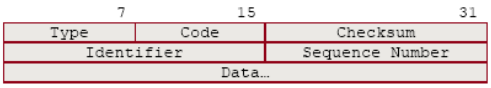

Figure 25: An example of IP header format for Echo or Echo Reply Message.

The description:

Field	Description
Type	8 - For echo message; 0 - for echo reply message.
Code	0.
Checksum	The checksum is the 16-bit ones' 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. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.
Identifier	If code = 0, an identifier to aid in matching echoes and replies, may be zero.
Sequence Number	If code = 0, a sequence number to aid in matching echoes and replies, may be zero. The data received in the echo message must be returned in the echo reply message. The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply. Code 0 may be received from a gateway or a host.

Table 11: IP header fields for Echo or Echo Reply Message description.

UDP

The **User Datagram Protocol** is a transport protocol for sessions that need to exchange data. Both transport protocols, UDP and TCP provide 65535 (2^{16}) different standard and non standard source and destination ports. The destination port is used to connect to a specific service on that port. Unlike TCP, UDP is not reliable, since it doesn't use sequence numbers and stateful connections. This means UDP datagrams can be spoofed, and might not be reliable (e.g. they can be lost unnoticed), since they are not acknowledged using replies and sequence numbers. The following figure shows the UDP header format.

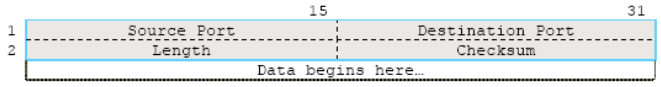

Figure 26: UDP header format.

As an example, we can define a structure for the UDP header as follows.

```
struct udpheader {
 unsigned short int udph_srcport;
 unsigned short int udph_destport;
 unsigned short int udph_len;
 unsigned short int udph_chksum;
}; /* total udp header length: 8 bytes (= 64 bits) */
```

A brief description:

Element/field	Description

udph_srcport	The source port that a client binds to, and the contacted server will reply back to in order to direct his responses to the client. It is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.
udph_destport	The destination port that a specific server can be contacted on.
udph_len	The length of udp header and payload data in bytes. It is a length in bytes of this user datagram including this header and the data. (This means the minimum value of the length is eight.)
udph_chksum	The checksum of header and data, see IP checksum. It is the 16-bit one's complement of the one's complement sum of a **pseudo header** (shown in the following figure) of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as used in TCP. If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

Table 12: UDP header fields description.

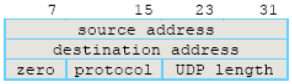

Figure 27: UDP pseudo header format.

TCP

The **Transmission Control Protocol** is the mostly used transport protocol that provides mechanisms to establish a reliable connection with some basic authentication, using connection states and sequence numbers. The following is a TCP header format.

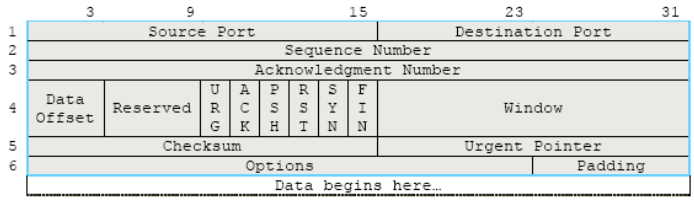

Figure 28: TCP header format.

And a structure example for the TCP header's field.

```
struct tcpheader {
 unsigned short int   tcph_srcport;
 unsigned short int   tcph_destport;
 unsigned int     tcph_seqnum;
 unsigned int     tcph_acknum;
 unsigned char    tcph_reserved:4, tcph_offset:4;
 unsigned char    tcph_flags;
 unsigned short int   tcph_win;
 unsigned short int   tcph_chksum;
 unsigned short int   tcph_urgptr;
};
/* total tcp header length: 20 bytes (= 160 bits) */
```

A brief description:

Element/field	Description
tcph_srcport	The 16 bits source port, which has the same function as in UDP.
tcph_destport	The 16 bits destination port, which has the same function as in UDP.
tcph_seqnum	The 32 bits sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1. It is used to enumerate the TCP segments. The data in a TCP connection can be contained in any amount of segments (= single tcp datagrams), which will be put in order and acknowledged. For example, if you send 3 segments, each containing 32 bytes of data, the first sequence would be (N+)1, the second one (N+)33 and the third one (N+)65. "N+" because the initial sequence is random.
tcph_acknum	32 bits. If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent. Every packet that is sent and a valid part of a connection is acknowledged with an empty TCP segment with the ACK flag set (see below), and the tcph_acknum field containing the previous tcph_seqnum number.
tcph_offset	The segment offset specifies the length of the TCP header in 32bit/4byte blocks. Without tcp header options, the value is 5.
tcph_reserved	4 bits reserved for future use. This is unused and must contain binary zeroes.
tcph_flags	This field consists of six bits flags (left to right). They can be ORed. TH_URG - Urgent. Segment will be routed faster, used for termination of a connection or to stop processes (using telnet protocol).

	TH_ACK - Acknowledgement. Used to acknowledge data and in the second and third stage of a TCP connection initiation. TH_PSH - Push. The systems IP stack will not buffer the segment and forward it to the application immediately (mostly used with telnet). TH_RST - Reset. Tells the peer that the connection has been terminated. TH_SYN - Synchronization. A segment with the SYN flag set indicates that client wants to initiate a new connection to the destination port. TH_FIN - Final. The connection should be closed, the peer is supposed to answer with one last segment with the FIN flag set as well.
tcph_win	16 bits Window. The number of bytes that can be sent before the data should be acknowledged with an ACK before sending more segments.
tcph_chksum	The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros. It is the checksum of pseudo header, tcp header and payload. The pseudo is a structure containing IP source and destination address, 1 byte set to zero, the protocol (1 byte with a decimal value of 6), and 2 bytes (unsigned short) containing the total length of the tcp segment. The checksum also covers a 96 bit pseudo header (shown in the following figure) conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.
tcph_urgptr	Urgent pointer. Only used if the TH_URG flag is set, else zero. It points to the end of the payload data that should be sent with priority.

Table 13: TCP header fields description.

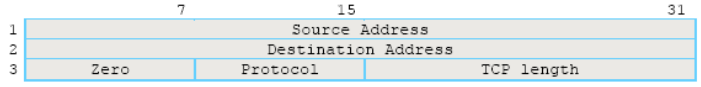

Figure 29: TCP pseudo header format.

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

More reading and digging:

1. Check the best selling C / C++, Networking, Linux and Open Source books at Amazon.com.
2. GCC, GDB and other related tools.

| Winsock & .NET | Winsock | < TCP/IP Stack & RAW Socket | Linux Socket Index | The RAW Socket Examples > |