Implementation of IPV6 in Linux or LAN

Arwinder Kaur, Gurpreet Singh
[1] Assistant professor(CSE)-CGC Landran , Mohali, [2] Assistant professor(CSE)-SSIET DERABSSI,
arwinder.cse@cgc.edu.in, gurpreet.dullat@gmail.com,

Abstract— The Internet Protocol (IP) is the most widely used communications protocol. Because it is the most pervasive communication technology, it is the focus of hundreds of thousands IT professionals. Right now, most of the world is using IPv4 (Internet Protocol version 4). The problem is that it does not allow for enough addresses. So the IETF has standardized the new Internet Protocol IPv6 as the successor of IPv4. In this paper built a basic IPv6 module in the Linux kernel. This implementation can be used as a testbed for conducting research into various issues related to deployment and performance of IPv6. And we also see how can we implement IPV6 in LAN

Keywords— IPV6 Features, IPV6 Addresses, Networking Subsystem for Linux and LAN.

INTRODUCTION

In the early 1990s, the IETF realized that a new version of IP would be needed, and the Task Force started by drafting the new protocol's requirements. IP Next Generation (IPng) was created, which then became IPv6 (RFC 1883). IPv6 is the second network layer standard protocol that follows IPv4 for computer communications across the Internet and other computer networks. IPv6 offers several compelling functions and is really the next step in the evolution of the Internet Protocol. These improvements came in the form of increased address size, a streamlined header format, extensible headers, and the ability to preserve the confidentiality and integrity of communications. The IPv6 protocol was then fully standardized at the end of 1998 in RFC 2460, which defines the header structure. IPv6 is now ready to overcome many of the deficiencies in the current IPv4 protocol and to create new

IPv6 has been designed as an evolutionary step from IPv4. This makes the transition from IPv4 to IPv6 easier, as IPv6 can be installed into the current IPv4 nodes as normal software up gradation. It is expected to be e client throughout the networking technology spectrum ranging from high-speed ATM networks to low-speed wireless networks. The protocol is also designed to be future-safe in the sense. An IPv6 address has a few differences from IPv4. The first is that it is in hexadecimal instead of decimal. The second is that it is split up into larger segments and more of them. The third is that it uses colons (:) rather than periods (.) to divide these segments.

In the end, one does not resemble the other. This is good because it prevents confusing the two. [1]

Features of IPV6

The most important features in IPV6 which are distinctive in comparison with IPV4.

1. *Simple Header Format***:** The IPv6 header is much simpler that that of IPv4. Considering the address size (128 bits), the IPv6 header is very small (40 bytes). All the IPv4 header ends that are not e actively used have been deleted. As far as possible bit ends have been avoided. Fields are aligned to facilitate extraction of maximum performance out of 64-bit architectures.

2. *Address Spaces***:** The addressing architecture of IPv6 is denied in [Hin95]. The main change has been the in the length of the address. The choices were 64 and 128 bits. Huitema in [C94] argues that 128 bits of address length is a safer choice and will provide with more than enough addresses. So the IPv6 addresses are standardized at 128 bits. As noted earlier, the actual efficiency of address allocation suffers, if we were to allocate addresses in such a manner to allow aggregation of routing information. But Huitema's calculations show that, even in the most pessimistic case we will have a total of 8x1017 addresses. The most optimistic estimate puts the total addresses at 2x1033. The preferred textual representation of an IPv6 address is of the form x:x:x:x:x:x:x:x. Each x designates the hexadecimal representation of a 16-bit quantity. Continuous run of zeros can be abbreviated by a \::". But it can appear only once. Embedded IPv4 addresses can be written in the well known dotted decimal notation, like x:x::a.b.c.d (IPv4 addresses always occupy the last 4 bytes.)

3. *Routing***:** The basic routing remains same as IPv4 routing under the CIDR. All the major routing protocols that support classless routing will work with a few, simple changes, like changing the address length to 128 bits from 32 bits.

IPv6 introduced a simple, but powerful routing feature in the form of a new routing header to be added after the IPv6 header in a packet. This header is identified by a value of 43 in the immediately preceding header. It allows the sender to choose a variant of router behavior by setting the routing data appropriately. This type is carried in the Routing Type elide. The intermediate nodes will check this type, and interpret the rest of the elides accordingly. On encountering a non-empty routing header of unknown type, the routers are required to send an ICMPv6 parameter problem message back to the source. Like all other extension headers, this header too contains the standard elides, Next Header and the Header Length.

4 *QoS Handling***:**

IPv4 supports only two simple QoS classes: low delay and high throughput. Even these two classes are not supported by typical routers. They processed the packets essentially in FIFO order. To implement support for more sophisticated QoS classes, the routers should use some sort of a priority queuing, instead of simple FIFO.

Priority

The router should be able to find out what kind of data a packet is carrying, before it can decide upon the special treatment or the priority, for that packet. This information is being encoded in the Topsfield of the IPv4 header, but as mentioned above, this allowed only two values to choose from, providing a crude classification. The IPv6 allocated 4 bits for denoting the priority class, allowing a much ner classification with 16 priority classes. They are split evenly between traffic that backs-down as a response to congestion and the traffic that does not respond to congestion. TCP traffic is an example for the rest kind, and UDP traffic is an example for the second kind. There is no relative priority denied between these two main subclasses. Different priority classes defined can be found in.

Flow ID

In some cases, knowing just the priority class may not. For example, some classes of traffic, like constant bit rate audio can be served better if, we can make some pre-assignment or reservation of resources. Reservation protocols like RSVP can reserve resources all along the path, but there should be a way to identify all the datagram's that should be allowed to use these reservations. So all the packets that are going to utilize these reservations, should present some identification to the routers. In other words we should have a way to associate a datagram to a logical. [3]

5. *MISCELLANEOUS FEATURES:*

Security & Authentication

The IPv6 provides framework for the implementation of authentication and security separately.

The Authentication Header (AH) provides for conveying the information relating to authentication. Though this header is designed to be algorithm independent, the use of keyed MD-5 algorithm is advocated. Authentication allows the destination to verify and be sure about the sender as well as the integrity of the datagram.

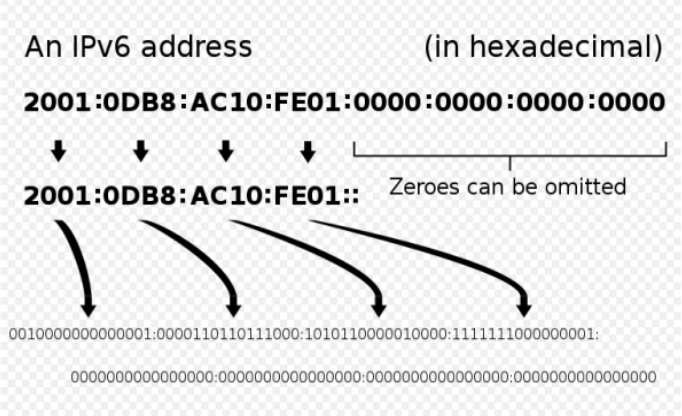
The other mechanism is called the Encapsulating the Security Payload (ESP). This provides the confidentiality. The standard DES encryption algorithm will be used to encrypt the message.

No specific key distribution algorithm is made standard as yet, but the AH allows both host-to-host and user-to-user keying.

These two mechanisms open up a lot of new possibilities like, corporate virtual networks that use public data carriers, electronic commerce, etc [2].

II ARCHITECTURE OF ADDRESSES

An IPv6 address has a few differences from IPv4. The first is that it is in hexadecimal instead of decimal. The second is that it is split up into larger segments and more of them. The third is that it uses colons (:) rather than periods (.) to divide these segments. In the end, one does not resemble the other. This is good because it prevents confusing the two.



Hexadecimal is better than decimal for a few reasons. For one, it takes up less space. The number “255″ is “FF” in hex; that is %50 smaller. Hexadecimal also relates more closely to binary. The number “1111″ is “15″ in decimal but “F” in hex. So “11111111″ is “FF” in hex, which is a much cleaner conversion than “255″ is. The downside is that humans typically think in decimal, so working with hex takes some getting used to.

IPv6 has a much larger address size. It has eight sections to it. IPv4 had half of that. The sections themselves are larger too: each one has four digits. IPv4 could only have three, and that only went to 255 tops. It should be obvious why this new address can afford all the room we have come to need. Hopeful it will for a long time. Here is a sample address.

FF00:00FF:0000:0000:0000:02f3:0000:0001.

At this point, you may begin to see the down side to IPv6: it is long and hard to remember. To make them more human readable, there are a few conventions to short address when you have a lot of zeros present. First of all, you can skip leading zeros, so “0001″ becomes “1″ and “02f3″ becomes “2f3″ (IPv4 did this too). Second, groups of all zeros can be abbreviated to “::” (a

double colon) once; you put nothing there. That makes ":0000:0000:0000:" reduce to "::" but ":0000:" cannot as it would be ambiguous. You can apply the first rule, making it ":0:" now. Putting all these rules into place gives us the following address.

FF00:FF::2f3:0:1

This address is about as long as an IPv4 number. Notice how the "00FF" shrank to "FF" and the "FF00″ did not. That was done intentionally to show the difference between leading and trailing zeros.

III. Header Comparison between IPV6 & IPV4 [4]

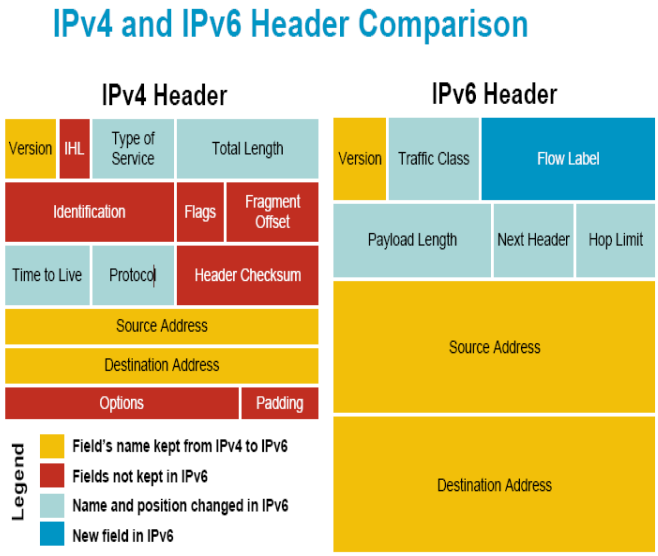


Fig.1 Header format

1. Networking Implementations

There are two major styles of network protocol management in unixen, BSD Sockets and streams BSD sockets were introduced with 4.2 BSD in 1983, and streams with SVR3 in 1986. While streams provide a very nice API to add more protocols and device drivers, they tend to lose on performance when compared to sockets.

2. Networking API

In order to use networking protocols among other things, 4.2 BSD introduced the socket API. SVR3 also introduced another API called TLI3. However they are independent of networking implementation inside the kernel. A TLI API actually can be provided over sockets implementation and a socket API can be implemented on streams.

3. *Linux*

Linux implements the networking subsystem based on BSD sockets. The networking architecture of Linux is fairly object oriented. It is designed to support many different communication protocols. A typical Linux kernel can be running TCP/IP, IPX, DECNET, UNIX, and many more communication protocols simultaneously. The IPv6 implementation in Linux is still under development. The current stable kernel, Linux 2.4, is missing many IPv6 extensions and does not confirm to all drafts and RFCs. This project actually is a study of the Linux kernel. [5]

V. PROCESS OF IMPLEMENTATION

A. Socket Communication:

Communication using sockets follows the client server model. The server process creates a socket and listens on the socket. For a protocol such as TCP/IPv6, the client process creates a socket and connects to the server socket. Once the connection is established, the processes can send messages to each other using **send** or **write** system call. After the data has been written on the socket, the following actions take place within the kernel.

B. Network Buffers

Network buffers are control structures with a block of shared memory which store incoming or outgoing packets. One of the main use of buffers is to avoid unnecessary copy of data. In the kernel a complete copy of the packet is done only a minimum required number of times, once from the user-space to the kernel-space and then from the kernel-space to the interface. In between, protocols look at the whole packet only if it is needed. [6]

C. Protocols

When a socket is to be created, it is supplied with an argument to identify its type. Various types of sockets supported by the Linux kernel are listed in the socket man page5. Socket type specifies the communication semantics of the socket. INET6 support three socket types. Stream sockets,

Datagram sockets and raw sockets. Stream sockets are used for TCPv6, datagram sockets are used for UDPv6 and raw sockets are to send IPv6 packets directly6. The socket API specifies some basic.

The sendmsg and rcvmsg functions are used to send packets from the protocol family layer to the socket layer, and receive packets from the socket layer to the protocol family layer respectively. INET6 declares two proto_ops structures. One is inet6_stream_ops for stream sockets, and the other one is inet6_dgram_ops for datagram and raw sockets. These are shown in appendix.

```
struct proto_ops {

int family;

...
```

```
int (*sendmsg) (struct kiocb *iocb, struct socket *sock,

struct msghdr *m, int total_len);

int (*recvmsg) (struct kiocb *iocb, struct socket *sock,

...

};
```

D. Packet

_When a packet arrives on the interface, the device driver receives it. After processing the packet it has to place the packet in the input queue of the appropriate protocol. Various data link level protocols define ways of identifying network protocols. For example, IEEE 802.3 identifies IPv6 packets with a constant 0x86DD (hex) in the Ethernet header. The constant in Linux is called ETH_P_IPV6 and is defined in include/linux/if_ether.h

```
packet_type

struct packet_type

{

unsigned short type; /* This is really htons(ether_type) */

struct net_device *dev; /* Device on which packet can be recieved

NULL is a wildcard here */

int (*func) (struct sk_buff *, struct net_device *,

struct packet_type *);

/* Packet Type Handler. */

void *data;

/* Private to the packet type */

struct packet type *next;

};
```

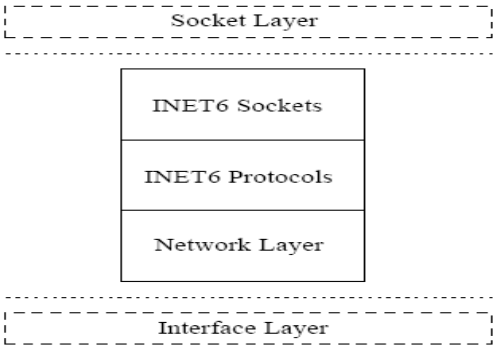
VI. INET6

Although INET6 is a family of protocols which sends IPv6 packets to the interface layer, neither IPv6 is completely a part of the INET6 framework, nor is the entire INET6 framework a part of

IPv6. In this section we look at organization of various protocols in INET6.

INET6 Layer

The entire INET6 family together with its protocols can be seen as a protocol layer in the kernel.



INET6 sockets are sockets belonging to the INET6 family. As mentioned before INET6 supports stream, datagram and raw sockets. INET6 protocols represent various IPv6 packet types. Extension headers are also treated as different protocol inside INET6 layer. The network layer take packets from INET6 protocols, add the IPv6 header to them, and send them to the interface layer. This layer is also responsible for receiving incoming packets from the interface layer and sending them
to INET6 protocols.

E. Proto Structure

The proto structure is the transition point between the socket layer and the protocol layer. It specifies functions which are used to communicate with the proto_ops structure in the socket layer. INET6 declares separate proto structures for TCP, UDP and RAW sockets.[7]

proto	Description	file
rawv6_prot	RAW proto structure	net/ipv6/raw.c
tcpv6_prot	TCP proto structure	net/ipv6/tcp_ipv6.c
udpv6_prot	UDP proto structure	net/ipv6/udp.c

VII. LOCAL SETUP FOR IPV6

In any campus network consists of a backbone Ethernet, to which all individual department Ethernets are connected via IPv4 routers. We chose two machines, one on CSE net and second on EE net, to act as IPv6 routers. We joined these two machines with a bidirectional tunnel comprising of two unidirectional IPv6/IPv4 tunnels (v6tunl). We also have one simple host on each of the two networks. Addresses assigned to these machines are shown in the following table. Constituent machines of 6Bone

Constituent Machines of 6Bone		
Name	IPV4 Address	Site Local IPV6 Address
CSE	192.168.25.112	FEC9:6164:6F752:6555:1:0000:COOF:00D9
EE	192.168.52.16	FEC9:6164:6F72:6555:1:0000:0080:4800:39bE

VIII. UTILITIES IN LINUX FOR IPV6

THE /PROC _LE SYSTEM INTERFACE

The /proc _le system started as a pseudo _le system under VFS, with the main aim to provide a _le system interface to the virtual memory of the active processes. In this _le system, which is traditionally mounted on /proc, a process is represented as a _le under /proc directory with its process ID forming the name. For example, reading the _le /proc/43, from the beginning, is equivalent to reading the virtual memory of the process with ID 43 starting at the virtual address zero. The process can be manipulated in a variety of ways by executing appropriate ioctl commands on the _le descriptor obtained by opening the corresponding _le in the /proc _le system. Thus this interface forms a superset of the process manipulating system call ptrace. [8]

CONCLUSION:

We have successfully implemented the basic IPv6 in Linux, as a new protocol family. An enhanced version of the popular BSD API is also provided. Applications could access the protocol module through the raw socket interface of this API and could communicate with each other. We have also implemented a minimal IPv6 router. This router uses CIDR style of routing and could forward packets between IPv6 networks. We provided the functionality of source routing also, which is going to aid in experimenting and conducting research into issues like mobility etc. We could use the existing routing infrastructure by implementing a tunnel driver that tunnel IPv6 packets in IPv4 routing domains. This helps in connecting island of IPv6 networks through the IPv4 networks.

REFERENCES

[1] http://www.networkworld.com/subnets/cisco/121908-ch1-ipv6-security.html

[2] Thesis submitted on IPV6 implementaion by Jay Ram in IIT Kanpur.

[3] file:///C:/Documents%20and%20Settings/Administrator/Desktop/ipv6/An%20Introduction%20to%20IPv6.htm.

[4] Introduction to IPV6 by Philip Smit <pfs@cisco.com>.

[5] Study of IPV6 implementatoin by Emaunela LIns and Saurabh Aggarwal.

[6] Tigran Aivazian. Linux kernel 2.4 internals. Available at http://tldp.org/guides.html.

[7] Conta and S. Deering. RFC 2463: Internet control message protocol (icmpv6) for the

internet protocol version 6 (ipv6) speci_cation.

[8] Thesis submitted on IPV6 implementaion by Jay Ram in IIT Kanpur.