NEAR EAST UNIVERSTY

Faculty of Engineering

Department of Computer Engineering

Internet Protocol Version 6

Graduation Project

Com-400

Student :Syed Taimoor Hussain(20002214)

Supervisor : Asst.Prof Firudin Muradov

Lefkoşa-2004

ACKNOWLEDGMENT

I want to thank Ass.Prof. Firudin Muradov to be my adviser under his guidance I successfully overcome difficulties and learn a lot about Internet Protocol Version 6 . In each discussion he explained my questions patiently and I felt my quick progress from his advices.

I also want to thank my friends for there help: Irfan, rehan, dawood . Being with them make my time full of fun .

Finally I want to thank my family especially my parents. Without their endless support and love for me I would never achieve my current position

Abstract

This project presents the Internet Protocol Version 6 (IPv6) in detail which will also cover architectures and design of IPv6.

It describes the OSI model, TCP, comparison with IPv4 new features in version6 and focuses on the basic knowledge of networking concepts and terminology used in data communication.

It defines how to assign and configure the IPs to different network topologies and it also informs how IPv6 functions during data communication.

This project helps to understand the problems that can occur in IPv6 and debugging of these problems by using commands.

The project also provides security issues available in IPv6 firewalls, and their implementation, Encryption, authentication.

Table of Contents

INTRODUCTION

IPv6 is a new layer 3 transport protocol which will supersede IPv4 (also known as IP). IPv4 was designed long time ago (RFC 760 from January 1980) and since its inception, there have been many requests for more addresses and enhanced capabilities. Major changes in IPv6 are the design of the header, including the increase of address size from 32 bits to 128 bits Because layer 3 is responsible for end-to-end packet transport using packet routing based on addresses, it must include the new IPv6 addresses (source and destination), like IPv4.For more information about the IPv6 history take a look at older IPv6 related RFCs listed e.g. at Switch.

On any IP header, the first 4 bits are reserved for protocol version. So theoretically a protocol number between 0 and 15 is possible:

4: is already used for IPv4

5: is reserved for the Stream Protocol (STP, RFC 1819) (which never really made it to the public) The next free number was 6. Hence IPv6 was born

The first IPv6 related network code was added to the Linux kernel 2.1.8 in November 1996 by Pedro Roque Because of lack of manpower, the IPv6 implementation in the kernel was unable to follow the discussed drafts or newly released RPCs. In October 2000, a project was started in Japan, called USAGI, whose aim was to implement all missing or outdated IPv6 support in Linux. It tracks the current IPv6 implementation in Free BSD made by the KAME project. From time to lime they create snapshots against current vanilla Linux

IPv6 defines address types based on some leading bits, which are hopefully never going to be broken in the future (unlike IPv4 today and the history of class A, B, and C).Also the number of bits are separated into a network part (upper 64 bits) and a host part (lower 64 bits), to facilitate auto-configuration

This interfaces are normally named site. The name sit is a shortcut for Simple Internet Transition. This device has the capability to encapsulate IPv6 packets into IPv4 ones and tunnel them to a foreign end point has a special meaning and cannot be used for dedicated tunnels.

IPv6 firewalling is important, especially if using IPv6 on internal networks with global IPv6 addresses. Because unlike at IPv4 networks where in common internal hosts are protected automatically using private IPv4 addresses like RFC 1918 / Address Allocation for Private Internets or APIPA / Automatic Private IP Addressing, in IPv6 normally global addresses are used and someone with IPv6 connectivity can reach all internal IPv6 enabled nodes

Unlike in IPv4 current versions doesn't allow to bind a server socket to dedicated IPv6 addresses so only any or none are valid. Because this can be a security issue, check the Access Control List (ACL)

Chapter 1

NETWORKING

1.1 COMPUTER NETWORK

A network can consist of two computers connected together on a desk or it can consist of many Local Area Networks (LANs) connected together to form a Wide Area Network (WAN) across a continent. The key is that two or more computers are connected together by a medium and are sharing resources- These resources can be files, printers, hard
drives, or CPU number-crunching power.

1.1.1 NETWORK HARDWARE

There is no generally accepted taxonomy into which all computer networks fit but two dimensions stand out as important: transmission technology and scale. We will now examine each of these in turn.

Broadly speaking, there are two types of transmission technology:

a. Broadcast networks.

b. Point-to-point networks.

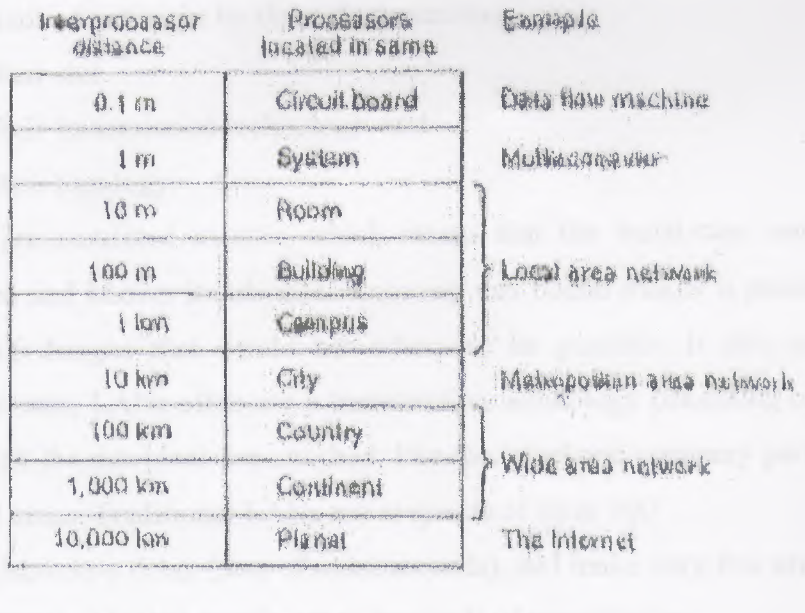
a. Broadcast Networks:

Broadcast Networks have a single communication channel that is shared by all the machines on the network- Short messages, called packets in certain contexts, sent by any machine are received by all the others. An address field within the packet specifies for whom it is intended. Upon receiving a packet, a machine checks the address field If the packet is intended for itself, it processes the packet; if the packet is intended for some other machine, it is just ignored.

As an analogy, consider someone standing at the end of a corridor with many rooms off it and shouting "Watson, come here. I want you." Although the packet may actually be received (heard) by many people, only Watson responds. The others just ignore it Another example is an airport announcement asking all flight 644 passengers to report to gate 12.Broadcast systems generally also allow the possibility of addressing a packet to all destinations by using a special code in die address field. When a packet with this code is transmitted, it is received and processed by every machine on the network. This mode of operation is called broadcasting. Some broadcast systems also support transmission to a subset of the machines, something known as multicasting. One possible scheme is to reserve one bit to indicate multicasting. The remaining n-l address bits can hold a group number. Each machine can "subscribe" to any or all of the groups. When a packet is sent to a certain group, it is delivered to all machines subscribing to that group.

b. Point-To-Point:

In contrast, point-to-point networks consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths are possible, so routing algorithms play an important role in point-to-point networks. As a general rule (although there are many exceptions) smaller, geographically localized networks tend to use broadcasting, whereas larger networks usually are point-to-point.



Interprocessor distance	Processors located in same	Example
0.1 m	Circuit board	Data flow machine
1 m	System	Multicomputer
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1,000 km	Continent	
10,000 km	Planet	The Internet

Figure 1 Classification of Interconnected Processors by scale

An alternative criterion for classifying networks is their scale, In Fig. 1-2 we give a classification of multiple processor systems arranged by their physical size. At the top are data flow machines, highly parallel computers with many functional units all working on the same program. Next come the multicomputers systems that communicate by sending messages over very short, very fast' buses. Beyond me multicomputers are the true networks, computers that communicate by exchanging messages over longer cables. These can be divided into local, metropolitan, and wide area networks. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is a well-known example of an internetwork Distance is important as a classification metric because different techniques are used at different scales.

1.1.2 Local Area Network (LAN)

A communications network connecting a group of computers, printers, and other devices located within a relatively limited area (for example, a building). A LAN allows any connected device to interact with any other on the network. Local area networks, generally called LANs, are privately owned networks within a single building or campus of up to a few kilometers in size. They are widely used to connect personal computers and workstations in company offices and factories to
share resources (e.g., printers) and exchange information. LANs are distinguished from other kinds of networks by three characteristics:

1. Their size
2. Their transmission technology, and
3. Their topology.

LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management, LANs often use a transmission technology consisting of a single cable to which all the machines are attached, like the telephone company party lines once used in rural areas. Traditional LANs run at speeds of 10 to 100
Mbps, have low delay (tens of microseconds), and make very few errors- Newer LANs may operate at higher speeds, up to hundreds of megabits/sec.

Various topologies are possible for broadcast LANs- An arbitratiou mechanism is needed to resolve conflicts when two or more machines wani to transmit simultaneously. The arbitration mechanism may be centralized or distributed. IEEE 802.3, popularly called Ethernet, for example, is a bus-based broadcast network with decentralized control operating at 10 or 100 Mbps- Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

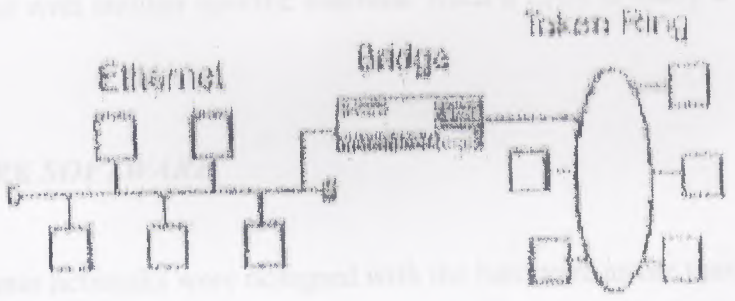


Figure 2 Local Area Network in a building

A second type of broadcast system is the ring. In a ring, each bit propagates around on its own, not waiting for the rest of the packet to which it belongs. Typically, each bit circumnavigates the entire ring in the time it takes to transmit a few bits, often before the complete packet has even been ransmitted. Like all other broadcast systems, some rule is needed for arbitrating simultaneous accesses to the ring. Various methods are in use and will be discussed later in this book. IEEE 802.5 (the IBM token ring), is a popular ring-based LAN operating at 4 and 16 Mbps.

Broadcast networks can be farther divided into static and dynamic, depending on how the channel is allocated. A typical static allocation would be to divide up time into discrete intervals and run a round robin algorithm, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so most systems attempt to allocate me channel dynamically (i.e., on demand)

Dynamic allocation methods for a common channel are either centralized or decentralized, m the centralized channel allocation method, there is a single entity, for example a bus arbitration unit, which determines who goes next. It might do this by accepting requests and making a decision according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether or not to transmit. You might think that this always leads to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

The other kind of LAN is built using point-to-point lines. Individual lines connect a specific machine with another specific machine. Such a LAN is really a miniature wide area network.

1.1.3 NETWORK SOFTWARE

The first computer networks were designed with the hardware as the main concern and the software as an afterthought. This strategy no longer works. Network software is now highly structured.

Protocol Hierarchies

To reduce their design complexity, most networks are organized as a series of layers or levels, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network.However in all networks, the purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

Layer n on one machine carries on a conversation with layer won another machine.The rules and conventions used in this conversation are collectively known as the layer n protocol.Basically a protocol is an agreement between the communicating parties on how communication is to proceed.

We have five-layer network illustrated here. The entities comprising die corresponding layers on different machines are called peers. In other words, it is the peers that communicate using the protocol.

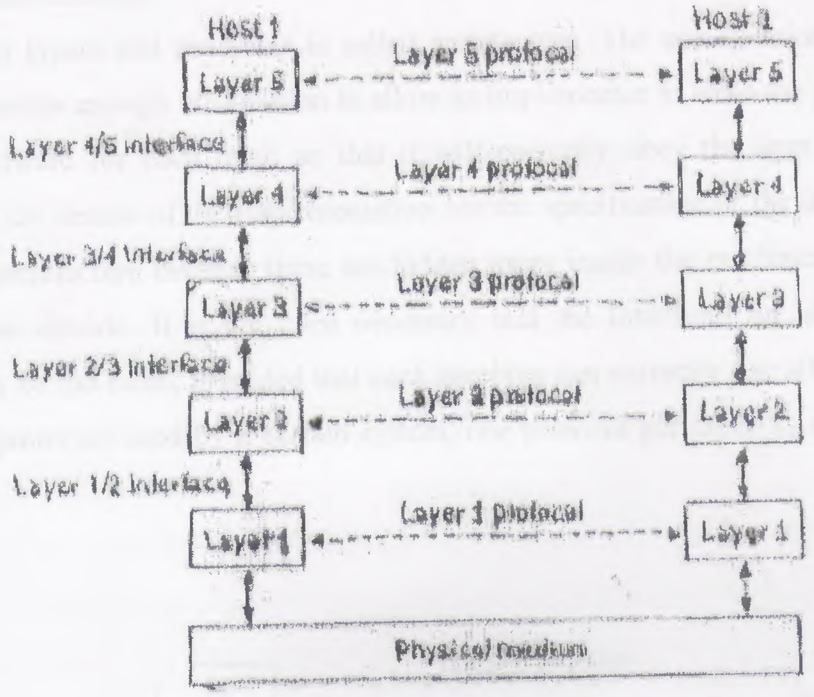


Figure 3 Layers, protocols, and interfaces

In reality, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the physical medium through which actual communication occurs.

Between each pair of adjacent layers there is an interface. The interface defines which primitive operations and services the lower layer offers to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clean-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels), because all that is required of the new implementation is that it offers exactly the same set of services to its upstairs neighbor as the old implementation did.

A set of layers and protocols is called architecture. The specification of architecture must contain enough information to allow an implementer to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces are part of the architecture because these are hidden away inside the machines and not visible from the outside. It is not even necessary mat the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols. A list of protocols used by a certain system, one protocol per layer, is called a **protocol stack.**

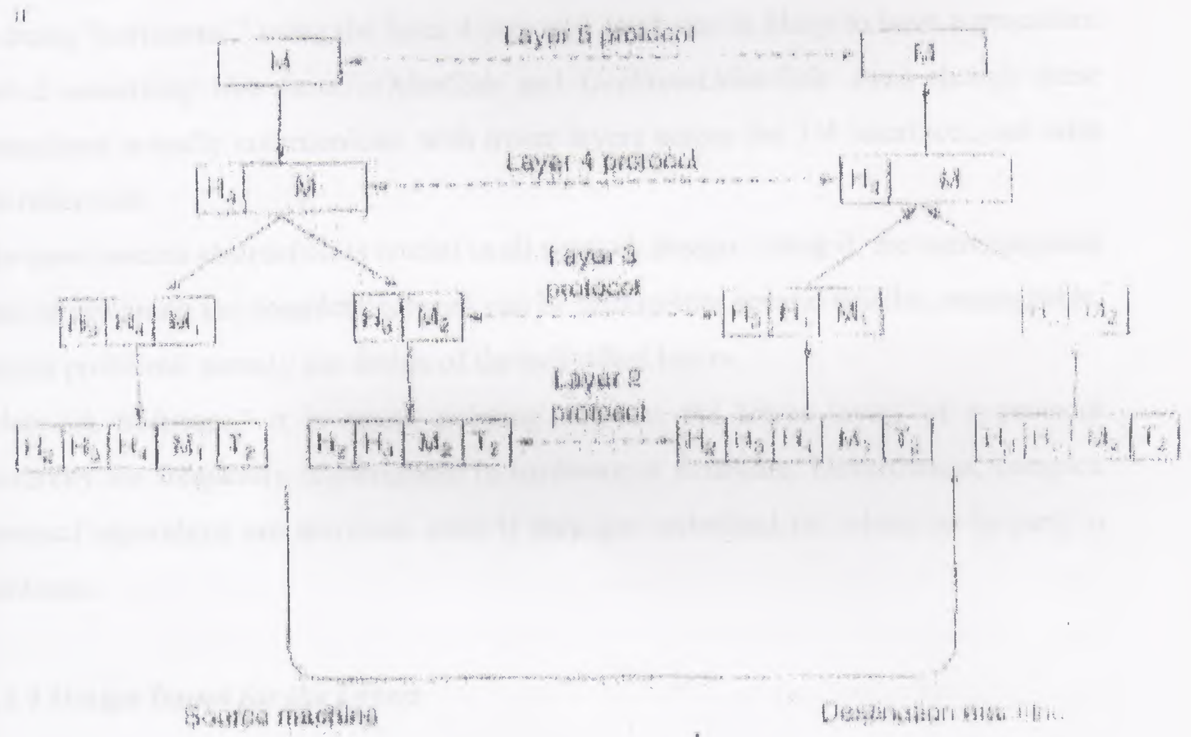


Figure 4 Example information flow supporting virtual communication in layer 5

Now consider a more technical example: how to provide communication to me top layer of the five-layer network . A message, M, is produced by an application process running in layer 5 and given to layer 4 for transmission. Layer 4 puts a header in front of the message to identify me message and passes the result to layer 3. The header includes control information, such as sequence numbers, to allow layer 4 on the destination machine to deliver messages in the right order if the lower layers do not maintain sequence. In some layers, headers also contain sizes, times, and other control fields.

In many networks, there is no limit to the size of messages transmitted in me layer 4 protocol, but there is nearly always a limit imposed by the layer 3 protocol. Consequently, layer 3 must break up the incoming messages into smaller

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds to only a header to each piece, but also a trailer, and gives the resulting unit to layer I for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have a procedure called something like *SendToOtherSide* and *GetFromOtherSide*. even though these procedures actually communicate with lower layers across the 3/4 interfaces, not with the other side.

The peer process abstraction is crucial to all network design. Using it, the unmanageable task of designing the complete network can be broken into several smaller, manageable, design problems, namely the design of the individual layers.

"Network Software," it is worth pointing out that the lower layers of a protocol hierarchy are frequently implemented in hardware or firmware. Nevertheless, complex protocol algorithms are involved, even if they are embedded (in whole or in part) in hardware.

1.1.4 Design Issues for the Layers

Some of the key design issues mat occur in computer networking are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer needs a mechanism for identifying senders and receivers. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to talk. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Another set of design decisions concerns the rules for data transfer, hi some systems, data only travel in one direction (simplex communication). In others they can travel in either direction, but not simultaneously (half-duplex communication). In still others they travel in both directions at once (full-duplex communication). The protocol must also determine how many logical channels the connection corresponds to, and what their priorities are. Many
networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be put back together properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. Some of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation. Others limit the sender to an agreed upon transmission rate.

Another problem mat must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is/what to do when processes insist upon transmitting data in units that are so sin all that sending each one separately is inefficient. Rare the solution is to gather together several small messages heading toward a common destination into a single large message and dismember the large message at the other side.

When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and demultiplexing is done transparently, any layer can use it. Multiplexing is needed in the physical layer, for example. where all the traffic for all connections has to be sent over at most a few physical circuits

When there are multiple paths between source and destination, a route must be chosen. Sometimes this decision must be split over two or more layers. For example, to send data from London to Rome, a high-level decision might have to be made to go via France or Germany based on their respective privacy laws, and a low-level decision might have to be made to choose one of the many available circuits based on the current traffic load.

1.1.5 Interfaces and Services

The function of each layer is to provide services to me layer above it. In this section we will look at precisely what a service is in more detail, but first we will give some terminology.

The active elements in each layer are often called entities. An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). Entities in the same layer on different machines are called peer entities. The entities in layer n implement a service used by layer n +1. In this case layer n is called the service provider and layer n +1 is called me service user. Layer n may use the services of layer n - 1 in order to provide its service. It may offer several classes of service, for example, fast, expensive communication and slow, cheap communication.

Services are available at SAPs (Service Access Points). The layer n SAPs are the places where layer n + 1 can access the services offered. Each SAP has an address that uniquely identifies it. To make this point clearer, the SAPs in the telephone system are the sockets into which modular telephones can be plugged, and die SAP addresses are the telephone numbers of these sockets. To call someone, we must know the cal lee's SAP address. Similarly, in the postal system, the SAP addresses are street addresses and post office box numbers. To send a letter, you must know the addressee's SAP address.

In order for two layers to exchange information, there has to be an agreed upon set of rules about the interface. At a typical interface, the layer n + 1 entity passes an IDU (Interface Data Unit) to the layer n entity through the SAP. The IDU consists of an SDU (Service Data Unit) and some control information. The SDU is me information passed across the network to the peer entity and then up to layer n + 1.

The control information is needed to help the lower layer do its job (e.g., the number of bytes in the SDU) but is not part of the data itself.

In order to transfer the SDU, the layer n entity may have to fragment it into several pieces each of which is given a header and sent as a separate PDU (Protocol Data Unit) such as a packet. The PDU headers are used by the peer entities to carry out their peer protocol. They identify which PDUs contain data and which contain control information, provide sequence numbers and counts, and so on.

1.1.6 Connection-Oriented and Connectionless Services

Layers can offer two different types connection-oriented and connectionless. In this section we will Took at these two types and examine the differences between them.

Connection-oriented service is modeled after the telephone system. To talk to someone; you pick up the phone, dial the number, talk, and then hang up. Similarly, to use a connection-oriented network service, the service user first establishes a connection, uses the connection,and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out in the same order at the other end.

In contrast, connectionless service is modeled after the postal system. Each message (letter)carries the full destination address, and each one is routed through the system independent of all the others. Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrived. However, it is possible that the first one sent can be delayed so that the second one arrives first. With a connection-oriented service this is impossible.

Each service can be characterized by a quality of service. Some services are reliable in the sense that they never lose data. Usually, a reliable service is implemented by having the receiver acknowledge me receipt of each message, so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

A typical situation in which a reliable connection-oriented service is appropriate is file transfer. The owner of the file wants to be sure that all the bits arrive correctly and in the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits, even if it is much faster.

Reliable connection-oriented service has two minor variations: message sequences and byte streams, m the former, the message boundaries are preserved. When two 1-KB messages are sent, they arrive as two distinct 1-KB messages, never as one 2-KB message. (Note: KB means kilobytes; kb means kilobits.) In the latter, the connection is simply a stream of bytes with no message boundaries. When 2K bytes arrive at the receiver, there is no way to tell if they were sent as one 2-KB message, two 1-KB messages, or 2048 1-byte messages. If the pages of a book are sent over a network to a phototypesetter as separate messages, it might important to preserve the message boundaries. On the other hand, with a terminal logging
into a remote timesharing system, a byte stream from the terminal to the computer is all that
is needed.

1.1.7 Reference Models

Now that we have discussed layered networks in me abstract, it is rime to look at some examples. In the next two sections we will discuss two important network architectures, the OSI reference model and the TCP/IP reference model.

The OSI Reference Model

This model is based on a proposal developed by me International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers (Day and Zimmermann, 1983). The model is called the ISO OSI (Open Systems Interconnection) Reference Model because it deals with connecting open systems-that is,systems that are open for communication with other systems. We will usually just call it the OSI model for short.

The OSI model has seven layers. The principles that were applied to arrive at the seven layers are as follows:

1. A layer should be created where a different level of abstraction is needed.
2. Each layer should perform a well-defined function.
3. The function of each layer should be chosen with an eye toward defining standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.

5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

Below we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard.

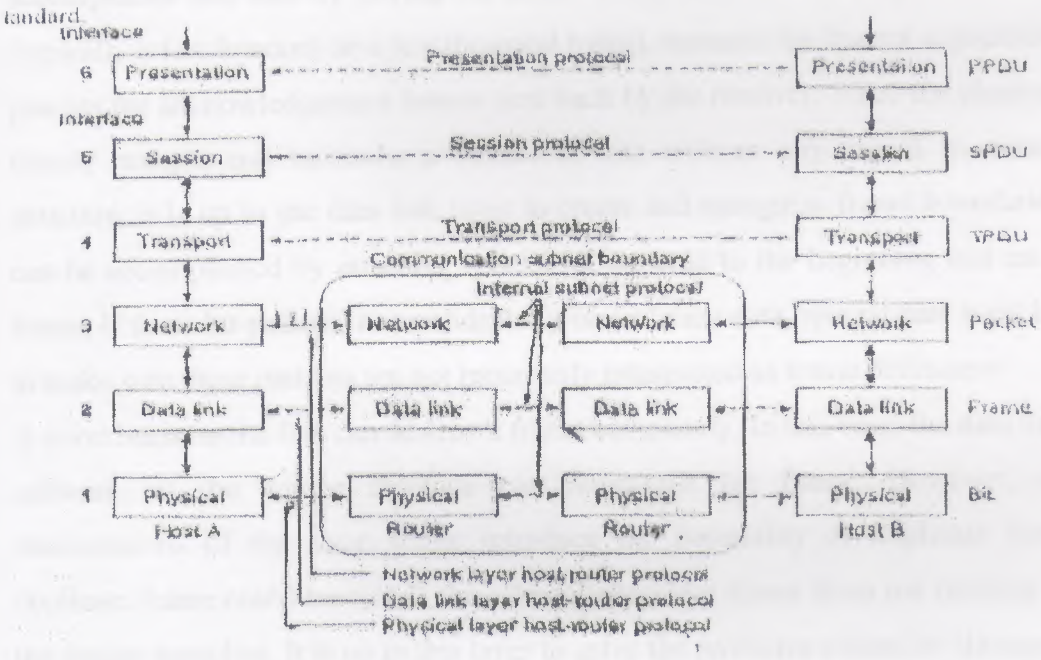


Figure 5 The OSI reference model

1.1.8 The Physical Layer

The physical layer is connected with transmitting raw bits over a communication channel.The design issues have to do with making sure that when one side sends a 1 bit, it is received by the other side as a 1 bit, not as a 0 bit. Typical questions here are how many volts should be used to represent a 1 and how many for a 0, how many

microsecond's a bit lasts- whether transmission may proceed simultaneously in both directions, how the initial connection is established and how it is torn down when both sides are finished, and how many pins the network connector has and what each pin is used for. The design issues here largely deal with
mechanical, electrical, and procedural interfaces, and the physical transmission medium which lies below the physical layer.

1.1.9 The Data Link Layer

The main task of the data link layer is to take a raw transmission facility and transform it into a line that appears free of undetected transmission errors to the network layer. It accomplishes this task by having the sender break the input data up into data frames (typically a few hundred or a few thousand bytes), transmit the frames sequentially, and process the acknowledgement frames sent back by the receiver. Since die physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in me data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters.
A noise burst on the line can destroy a frame completely. In this case, the data link layer software on the source machine can retransmit the frame. However, multiple transmissions of the same frame introduce the possibility of duplicate frames. A duplicate frame could be sent if the acknowledgement frame from me receiver back to the sender were lost. It is up to this layer to solve the problems caused by damaged, lost, and duplicate frames. The data link layer may offer several different service classes to the network layer, each of a different quality and with a different price.
Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism must be employed to let the transmitter know how much buffer space the receiver has at the moment. Frequently, miss flow regulation and the error handling are integrated.
If me line can be used to transmit data in both directions, this introduces a new complication that the data link layer software must deal with. The problem is that me acknowledgement frames for A to B traffic compete for the use of the line with data

frames for the B to A traffic. A clever solution (piggybacking) has been devised; we will discuss it in detail later.

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel. A special sub layer of the data link layer, the medium access sub layer deals with this problem.

1.1.10 The Network Layer

The network layer is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed. They can also be determined at the start of each conversation, for example a terminal session. Finally, they can be highly dynamic, being determined anew for each packet, to reflect the current network load.

If too many packets are present in the subnet at the same time, they will get in each other's way, forming bottlenecks. The control of such congestion also belongs to the network layer.

Since the operators of the subnet may well expect remuneration for their efforts, there is often some accounting function built into the network layer. At the very least, the software must count how many packets or characters or bits are sent by each customer, to produce billing information. When a packet crosses a national border, with different rates on each side, the accounting can become complicated.

When a packet has to travel from one network to another to get to its destination, many problems can arise. The addressing used by the second network may be different from the first one. The second one may not accept the packet at all because it is too large. The protocols may differ, and so on. It is up to the network layer to overcome all these problems to allow heterogeneous networks to be interconnected.

In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent.

1.1.11 The Transport Layer (IPv6 works on this layer)

The basic function of the transport layer is to accept data from the session layer, split it up into smaller units if need be, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end. Furthermore, all this must be done efficiently, and in a way that isolates the upper layers from the inevitable changes in die hardware technology.

Under normal conditions, die transport layer creates a distinct network connection for each transport connection required by the session layer. If die transport connection requires a high throughput, however, the transport layer might create multiple network connections, dividing the data among the network connections to improve throughput. On the other hand, if creating or maintaining a network connection is expensive, the transport layer might multiplex several transport connections onto the same network connection to reduce the cost.

In all cases, the transport layer is required to make the multiplexing transparent to the session layer

The transport layer also determines what type of service to provide the session layer, and ultimately, the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages or bytes in the order in which they were sent. However, other possible kinds of transport service are transport of isolated messages with no guarantee about the order of delivery, and broadcasting of messages to multiple destinations. The type of service is determined when the connection is established.

The transport layer is a true end-to-end layer, from source to destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, the protocols are between each machine and its immediate neighbors, and not by the ultimate source and destination machines, which may be separated by many routers. The difference between layers 1 through 3 which are chained and layers 4 through 7.

Many hosts are multiprogrammed.which implies that multiple connections will be entering and leaving each host. Their needs to be some way to tell which message belong to winch

connection.

In addition to multiplexing several message streams onto one channel, fife transport layer must take care of establishing and deleting connections across the network. This requires some kind of naming mechanism, so that a process on one machine has a way of describing with whom it wishes to converse. There must also be a mechanism to regulate the flow of information, so that a fast host cannot overrun a slow one- Such a mechanism is called flow control and plays a key role in the transport layer (also in other layers). Flow control between hosts is distinct from flow control between routers, although we will later see that similar principles apply to both.

1.1.12 The Session Layer

The session layer allows users on different machines to establish sessions between them A session allows ordinary data transport, as does the transport layer, but it also provides enhanced services useful in some applications. A session might be used to allow a user to log into a remote time sharing system or to transfer a file between two machines.
One of the services of the session layer is to manage dialogue control. Sessions can allow traffic to go in both directions at the same time, or in only one direction at a time. If traffic can only go one way at a time (analogous to a single railroad track), the session layer can help keep track of whose turn it is.
A related session service is token management. For some protocols, it is essential that both sides do not attempt the same operation at the same time. To manage these activities, the session layer provides tokens that can be exchanged. Only the side holding the token may perform the critical operation.
Another session service is synchronization. Consider the problems that might occur when trying to do a 2-hour file transfer between two machines with a 1-hour mean time between crashes. After each transfer was aborted, the whole transfer would have to start over again and would probably fail again the next time as well. To eliminate this problem, the session layer provides a way to insert checkpoints into the data stream, so that after a crash, only the
data transferred after me last checkpoint have to be repeated.

1.1.13 The Presentation Layer

The presentation layer performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems In particular, unlike all me lower layers, which are just interested in moving bits reliably from here to there, the presentation layer is concerned with the syntax and semantics of the information transmitted.

A typical example of a presentation service is encoding data in a standard agreed upon way.

Most user programs do not exchange random binary bit strings. They exchange things such as people's names, dates, amounts of money, and invoices. These items are represented as character strings, integers, floating-point numbers, and data structures composed of several simpler items. Different computers have different codes for representing character strings (e.g., ASCII and Unicode), integers (e.g., one 5 complement and two's complement), and so on. In order to make it possible for computers with different representations to communicate the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and converts from the representation used inside me computer to the network standard representation and back.

1.1.14 The Application Layer

The application layer contains a variety of protocols that are commonly needed. For example, there are hundreds of incompatible terminal types in the world. Consider the plight of a full screen editor that is supposed to work over a network with many different terminal types, each with different screen layouts, escape sequences for inserting and deleting text moving the cursor, etc

One way to solve this problem is to define an abstract network virtual terminal that editors and other programs can be written to deal with. To handle each terminal type, a piece of software must be written to map the functions of the network virtual terminal onto the real terminal. For example, when the editor moves the virtual terminal's cursor to the upper left-hand corner of the screen, this software must issue the proper command sequence to the real terminal to get its cursor there too. All the virtual terminal software is in the application layer.

Another application layer function is file transfer. Different file systems have different file naming conventions, different ways of representing text lines, and so on.

Transferring a file between two different systems requires handling these and other incompatibilities. This work, too, belongs to the application layer, as do electronic mail remote job entry. different lookup, and various other general-purpose and special-purpose facilities.

1.1.15 Data Transmission in the OSI Model

Data can be transmitted using the OSI model. The sending process has some data it wants to send to the receiving process. It gives the data to the application layer, which then attaches the application header, AH to the front of it and gives the resulting item to the presentation layer.

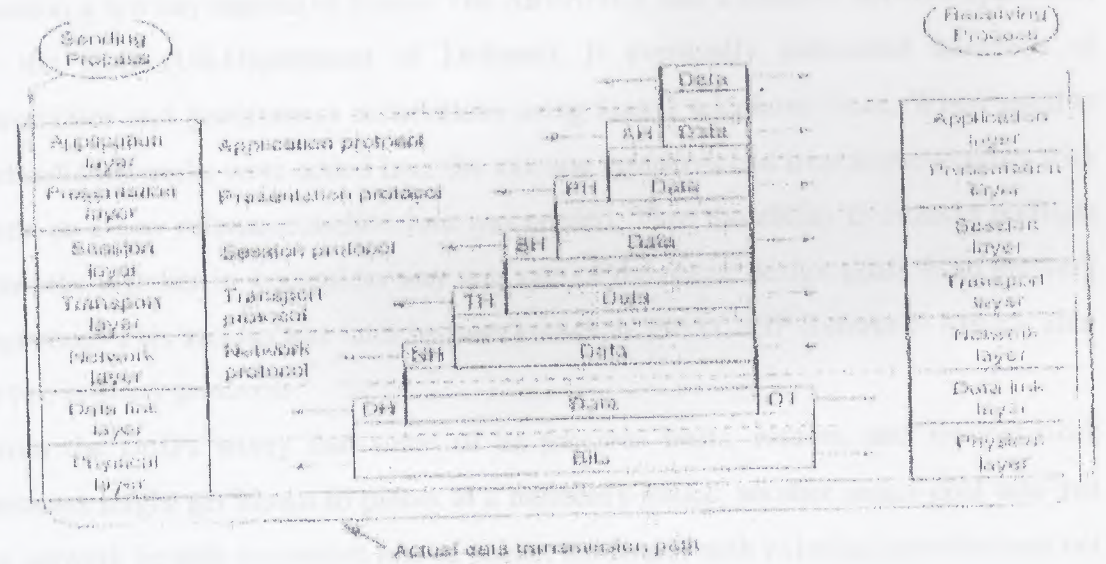


Figure 6 An example of how the OSI model is used

The presentation layer may transform this item in various ways and possibly add a header to the front, giving the result to the session layer. It is important to realize that the presentation layer is not aware of which portion of the data given to it by the application layer is AH, if any, and which is true user data.

This process is repeated until the data reach the physical layer, where they are actually transmitted to die receiving machine. On that machine the various headers are stripped off one by one as the message propagates up the layers until it finally arrives at the receiving process.

The key idea throughout is that although actual data transmission is vertical, each layer is programmed as though it were horizontal. When the sending transport layer, for example, gets a message from the session layer, it attaches a transport header and sends it to the receiving transport layer. From its point of view, the fact mat it must actually hand the message to the network layer on its own machine is an unimportant technicality.

The TCP/IP Reference Model

Let us now turn from the OSI reference model to the reference model used in the grandparent of all computer networks, the ARPANET, and its successor, the worldwide Internet Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (US.Department of Defense). It eventually connected hundreds of universities and government installations using leased telephone lines. When satellite and radio networks were added later the existing protocols had trouble networking with them, so a new reference architecture was needed. Thus the ability to connect multiple networks together in a seamless way was one of the major design goals from the very beginning. This architecture later became known as the TCP/IP Reference Model, after its two primary protocols.

Given the DoD's worry that some of its precious hosts, routers, and internet-work gateways might get blown to pieces at a moment's notice, another major goal was that the network be able to survive loss of subnet hardware, with existing conversations not being broken off. In other words, DoD wanted connections to remain intact as long as the source and destination machines were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, a flexible architecture was needed, since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission

1.1.16 The Internet Layer

All these requirements led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the Internet layer, is the linchpin that holds the whole architecture together. Its job is to permit hosts to inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a different order than they were sent, in which case it is the job of higher layers to rearrange them, if in-order delivery is desired.

Note that "internet" is used here in a generic sense, even though this layer is present in the Internet.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mail box in one country, and with a little luck, most of them will be delivered to the correct address in the destination country. Probably the letters will travel through one or more international mail gateways along the way, but this is transparent to the users. Furthermore, that each country (i.e., each network) has its own stamps, preferred envelope sizes, and delivery rules is hidden from the users.

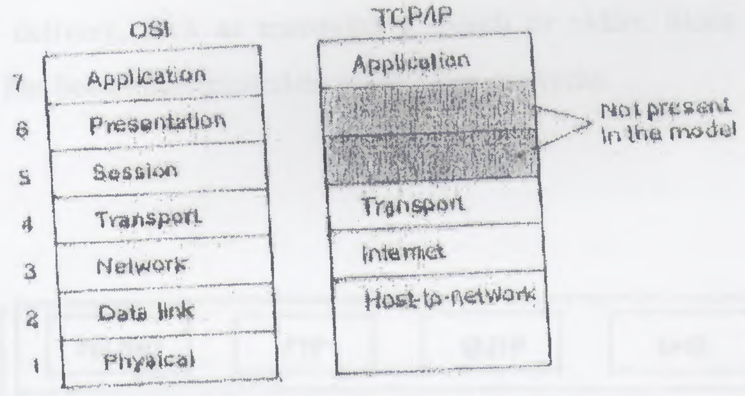


Figure 7 The TCP/IP reference model

The internet layer defines an official packet format and protocol called IP (Internet Protocol). The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, as is avoiding congestion. For these reasons, it is reasonable to say that the TCP/IP internet layer is very similar in functionality to the OSI network layer.

1.1.17 The Transport Layer

The layer above the internet layer in the TCP/IP model is now usually called the transport layer. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, the same as in the OSI transport layer. Two end-to-end protocols have been defined here. The first one, TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It fragments the incoming byte stream into discrete messages and passes each one onto the internet layer. At the destination, the receiving TCP process reassembles the received messages into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more messages than it can handle.

The second protocol in this layer, **UDP** (User **Data gram Protocol**), is an unreliable connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own. It is also widely used for one-shot, client-server type request-reply queries and applications in which prompt delivery is more important than accurate delivery, such as transmitting speech or video. Since the model was developed, IP has been implemented on many other networks.

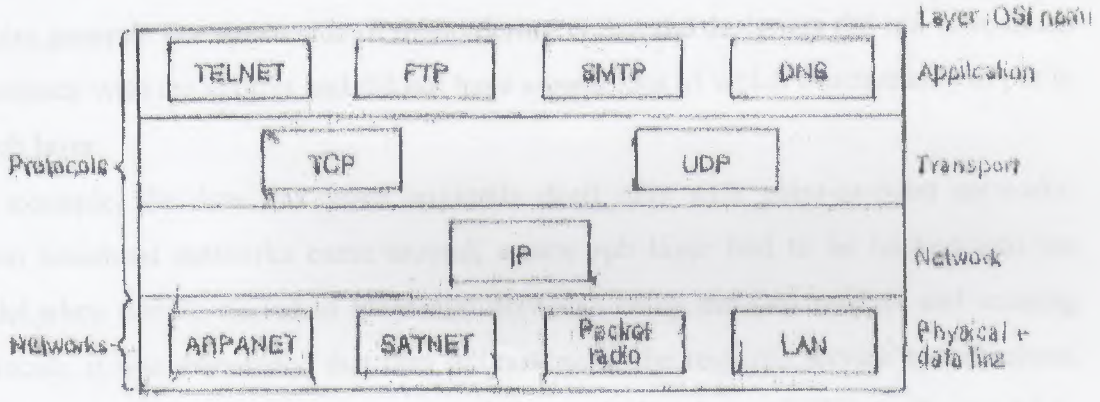


Figure 8 Protocols and networks in the TCP/IP model initially

1.1.18 The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived, so they were not included. Experience with the OSI model has proven this view correct: they are of little use to most applications.

On top of the transport layer is the application layer. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP). The virtual terminal protocol allows a user on one machine to log into a distant machine and work there. The file transfer protocol provides a way to move data efficiently from one machine to another. Electronic mail was originally just a kind of file transfer, but later a specialized protocol was developed for it. Many other protocols have been added to these over the years, such as the Domain Name Service (DNS) for mapping

As a consequence, the protocols in the OSI model are better hidden than in the TCP/IP model and can be replaced relatively easily as the technology changes. Being able to make such changes is one of the main purposes of having layered protocols in the first place.

1.120. A Comparison of the OSI and TCP reference Models

The OSI reference model was devised before the protocols were invented. This ordering means that the model was not biased toward one particular set of protocols, which made it quite general. The down side of this ordering is that the designers did not have much experience with the subject and did not have a good idea of which functionality to put in which layer.

For example, die data link layer originally dealt only with point-to-point networks. When broadcast networks came around, a new sub layer had to be hacked into the model.when people started to build real networks using the OSI models and existing protocols, it was discovered that they did not match the required service specifications (wonder of wonders), so convergence sub layers had to be grafted onto the model to provide a place for papering over the differences. Finally, the committee originally expected that each country would have one network, run by the government and using the OSI protocols, so no thought was given to internetworking. To make a long story short, things did not turn out that way.

With the TCP/IP the reverse was true: the protocols came first, and the model was really just a description of the existing protocols- There was no problem with the protocols fitting the model. They fit perfectly. The only trouble was that the model did not fit any other protocol stacks. Consequently, it was not especially useful for describing other non-TCP/IP networks

Turning from philosophical matters to more specific ones, an obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP has four layers. Both have (inter) network, transport, and application layers, but the other layers are different.

Another difference is in the area of connectionless versus connection-oriented communication. The OSI model supports both connectionless and connection oriented communication in the network layer, but only connection-oriented communication in the transport layer, where it counts (because the transport service is visible to the users). The TCP/IP model has only one mode in the network layer (connectionless) but supports both modes in me transport layer, giving the users a choice. This choice is especially important for simple request response protocols.

1.2.1 General Ethernet Information

Ethernet devices names are 'ethO', 'ethl1', 'eth2' etc. The first card detected by the kernel is assigned 'ethO', and the rest are assigned sequentially in the order in which they are detected.

Once we have our kernel properly built to support our Ethernet card, configuration of the card is easy.

Typically we would use something like (which most distributions already do for us, if we configured them to support our Ethernet):

```
Root# ifconfig ethO 192,168.0.1 netmask 255.255.255.0 up
Root# route add -net 192.168.0.0 netmask 255.255.255.0 ethO
```

1.2.2 Physical Connection

The physical connection determines how many bits (1's or 0's) can be transmitted in a single instance of time. If only 1 bit of information can be transmitted over the data transmission medium at a time then it is considered a serial communication.

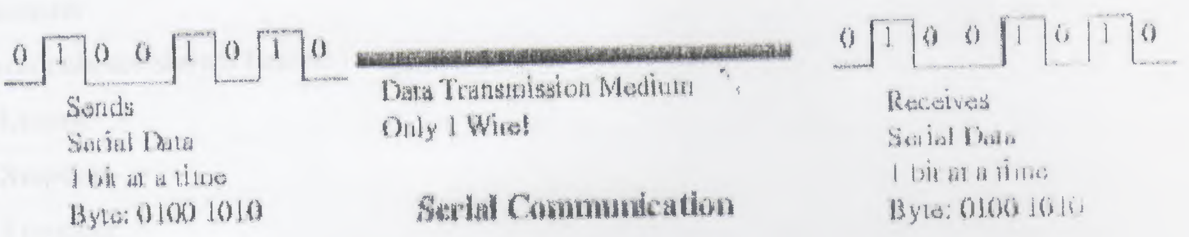


Figure Serial Communication

If more than 1 bit of information is transmitted over the data transmission medium at a time then it is considered a parallel communication.

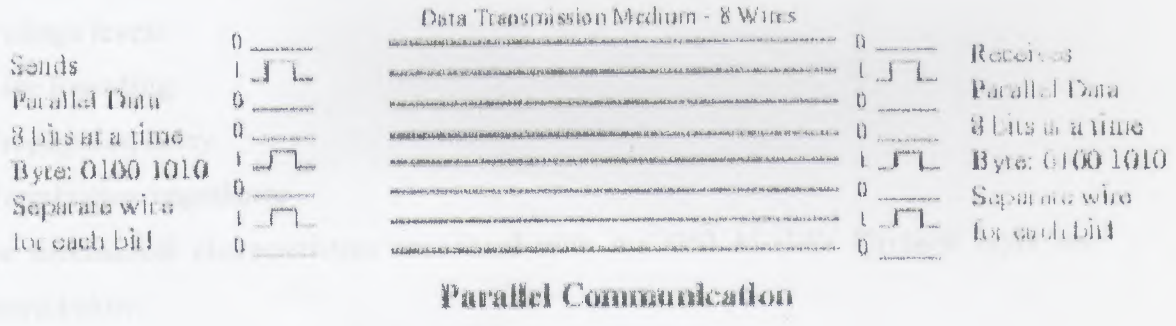


Figure Parallel Communications

1.2.3 Physical Layer

The OSI Model Physical Layer concerns itself with the transmission of bits through the communication medium. The order of the bits—and importance-is determined by the Protocol's packet

Asynchronous & Synchronous Communication in Asynchronous Communications, the OSI Physical layer concerned itself with the RS-232D standard (and the Voice

Channel). The RS-232D standard stated the electrical and mechanical characteristics of the cable: these characteristics were for the transmission of the digital signal between the DTE (PC) and DCE (modem). The Voice Channel stated the electrical and mechanical characteristics of the connection between DCE to DCE (modem to modem) through the phone lines.

The order of the bits was determined by me following: ASCII characters, the parity (Odd/Even/None), the number of Stop Bits, and the Transfer Protocol. Examples of Transfer
Protocols are shown below:

• Kermit
• Xmodem
• Ymodem
• Zmodem

Similarly, in Synchronous Communications, the electrical and mechanical characteristics of
the cable (for the transmission of the signal) are defined by the protocol that's used between Network Interface Cards.

The electrical characteristics associated with the OSI Model's Physical layer are as follows:

• Transmission rate (bits/sec)
• Voltage levels
• Line Encoding
• Propagation delay
• Termination impedance

The mechanical characteristics associated with me OSI Model's Physical layer are shown below:

• Connector type
• Cable type & size
• Cable Length
• Topology
• Shielding

In summary, the OSI Physical Layer is concerned with the transmission of bits on the network; the order of bits, bit level error checking, and the electrical / mechanical characteristics.

1.3 PROTOCOLS, SPEED, BANDWIDTH

1.3.1 IP Addresses: an Explanation.

Internet Protocol Addresses are composed of four bytes. The convention is to write addresses in what is called 'dotted decimal notation'. In this form, each byte is converted to a decimal number, (0-255). It drops any leading zeros (unless the number is zero) and written with each byte separated by a * character. By convention, each interface of a host or router has an IP address. It is legal for the same IP address to be used on each interface of a single machine but usually each interface will have its own address Internet Protocol Networks are contiguous sequences of IP addresses. All addresses within a network have a number of digits within the address in common. The portion of the address that is common amongst all addresses within the network is called the 'network portion' of the address. The remaining digits are called the 'host portion'. The number of bits that are shared by all addresses within a network is called the network. It is the role of the network to determine which addresses belong to the network it is applied to and which don't belong. For example, we consider the following:

Host Address	192.168.110.23
Network Mask	255.255.255.0
Network Portion	192.168.110.23
Network Address	192.168.110.0
Broadcast Address	192.168.110.255

Any address that is 'bitwise anded' with its netmask will reveal the address of the network that it belongs to. The network address is therefore always the lowest numbered address within the range of addresses on the network, and it always has the host portion of the address coded in all zeroes.

The broadcast address is a special address that every host on the network listens to (in addition to its own unique address). This address is tile one that datagrams are sent to if every host on the network is meant to receive it. Certain types of data, like routing information and warning messages, are transmitted to the broadcast address so that every host on the network can receive it simultaneously. There are two commonly used standards for the broadcast address. The most widely accepted one is to use the highest possible address on the network as the broadcast address. In the above example, this would be 192.168.110.255. For some reason other sites have adopted the convention of using the network address as the broadcast address. In practice it doesn't matter very much which you use, but you must make sure that every host on the network is configured with the same broadcast address.For administrative reasons (some time early in the development of the IP protocol), some arbitrary groups of addresses were formed into networks. These networks were grouped into what are called classes. Classes provide a number of standard size networks that could be allocated. The ranges allocated

are:

Network Class	Netmask	Network Addresses
A	255.0.0.0	0.0.0.0 -127.255.255.255
B	255.255.0.0	128.0.0.0 -191.255.255.255
C	255.255.255.0	192.0.0.0 -223.255.255.255
Multicast	24.0.0.0	244.0.0.0 -239.255.255.255

What addresses you should use depends on exactly what it is that you are doing. You may have to use a combination of the following activities to get all me addresses you need:

1.3.2 Linking two networks using PPP

There is basically no difference between linking a single Linux PC to a PPP server and linking two LANs using PPP (on a machine) on each LAN. Remember, PPP is a peer-to-peer protocol.

In order to link two LANs, we must be using different IP network numbers (or subnets of the same network number). We'll also need to use static IP numbers (or use IP masquerade).

Setting up the IP numbers

We arrange with the network administrator (of the other LAN) the IP numbers that
will be used for each end of the PPP interface. If we are using static IP numbers, this will probably require us to dial into a specific telephone number Now we need to edit the appropriate /etc/ppp/options(.ttyxx] file – it's a good idea to have a specific modem and port at your end for this connection. This may well require us to change our /etc/ppp/options file - and create appropriate options.ttyXX files for any other connections! We will specify the IP numbers for our end of the PPP link in link
appropriate options file.

Routing issues on a LAN

If we are connected to a LAN (but still want to use PPP on our personal Linux machine), then we need to address some route packet issues. Specifically, we're discussing here the packets that need to go from our machine to our LAN (through our Ethernet interface): we are also talking about the ones that go to the remote PPP server (and beyond).

In this section we do NOT attempt to discuss whole routing - we discuss only a simple special case of (static) routing!

The basic rule of static routing: me DEFAULT route should be the one mat points to the MOST number of network addresses. We enter specific routes to the routing table for other networks.

The ONLY situation we are going to cover here is as follows. Suppose that our Linux box is on a LAN that is not connected to the Internet, and we want to dial out to the Internet for personal use (while still connected to the LAN).We have to make sure that our Ethernet route is set up to the specific network addresses available across our LAN (NOT set to the default route) !We can check this by issuing a route command. We should see something like
below:

```
[root&hwin / root] # route –n
Kernel routing table
```

```
Destination Gateway  Genmask  Flags Mss Window Use Iface
Loopback   *    255.255.255.0 U 1936 0      50 lo
10.0.0.0   *    255.255.255.0 U 1436 0     565 eth0
```

If our Ethernet interface (ethO) is pointing at the default route, (the first column will show "default" in the ethO line) then we need to change our Ethernet initialization scripts. We need to make it point to the specific network numbers: not to point it toward the default route. This will allow pppd to set up your default route as follows:

```
[root@hwin / root] # route –n
Kernel routing table
Destination   Gateway       Genmask          Flags Mss  Window Use Iface
10.144.153.51 *             255.255.255.255  VH    488  0       0 ppp0
127.0.0.0     *             255.255.255.0    V     1936 0      50 lO
10.1.0.0      *             255.255.255.0    V     1436 0     569 eth0
default       10.144.153.51 *                VG    488  0       3 ppp0
```

As we can sec, we have a host route to the PPP server (10.144.153.51) via ppp0. We also have a default network route; it is using the PPP server as its gateway.

If our LAN already has routers on it, then we will already have gateways established to the wider networks that are available at our site- We should STILL point our default route at the
PPP interface. We make the other routes specific to the networks that they are serving,

1.3.3 Used terms Network related

Link

A link is a layer 2 network packet transport medium, examples are Ethernet, Token Ring, PPP, SLIP,
ATM, ISDN, Frame Relay

Node

A node is a host or a router.

Host

Generally a single homed host on a link. Normally it has only one active network interface e.g Ethernet or (not and) PPP.

Dual homed host

A dual homed host is a node with two network (physical or virtual) interfaces on two different links but does not forward any packets between the interfaces.

Router

A router is a node with two or more network (physical or virtual) interfaces, capable of forwarding packets between the interfaces.

Tunnel

A tunnel is topically a point-to-point connection over which packets are exchanged which carry the data of another protocol, e.g. an IPv6-in-IPv4 tunnel.

NIC

Network Interface Card

1.3.4.1 Long code line wrapping signal char

The special character ""-" is used for signaling that this code line is wrapped for better viewing in PDF and PS files-

1.3.4.2 Placeholders

In generic examples you will sometimes find the following:

My Ip address

For real use on your system command line or in scripts this has to be replaced with relevant content(removing the < and > of course), the result would be e.g

1.2.3.4

1.3.4.3 Commands in the shell

Commands executable as non-root user begin with $, e.g,

```
$ whoami
```

Commands executable as root user begin with #,e.g.

```
# whoami
```

1.4 Requirements for using

1.4.1. Personal prerequisites

We should be familiar with the major Unix tools e.g,grep,awk,find..., and know about their most commonly used command-line options,

1.4.1.2. Experience with networking theory

You should know about layers, protocols, addresses, cables, plugs, etc. We already studied about networking in the bigining chapter of this booklet.

14.1.3. Experience with IPv4 configuration

You should definitely have some experience in IPv4 configuration, otherwise it will be hard for you to understand what is really going on ,etc for example its same like Ip configuration while u r creating a network.

1.4.1.4. Experience with the Domain Name System (DNS)

Also you should understand what the Domain Name System (DNS) is, what it provides and how to use it,

1.8-1.5. Experience with network debugging strategies

You should at least understand how to use tcpdump and what it can show you. Otherwise, network debugging will very difficult for you.

Chapter 2.

Basics

2.1. What is IPv6?

IPv6 is a new layer 3 transport protocol which will supersede IPv4 (also known as IP). IPv4 was designed long time ago (RFC 760 from January 1980) and since its inception, there have been many requests for more addresses and enhanced capabilities. Major changes in IPv6 are the design of the header, including the increase of address size from 32 bits to 128 bits Because layer 3 is responsible for end-to-end packet transport using packet routing based on addresses, it must include the new IPv6 addresses (source and destination), like IPv4.For more information about the IPv6 history take a look at older IPv6 related RFCs listed e.g. at Switch.

2.2. History of IPv6

To-do: better lime-line, more conten etc,

2.2.1. Beginning

The first IPv6 related network code was added to the Linux kernel 2.1.8 in November 1996 by Pedro Roque.

It was based on the BSD API;

```
diff -u --recursive –new-file v2. 1. 7/linux/include/linux/in6.h
linux/include/linux/in6.h
---v2.1.7/linux/include/liux/in6.h Thu Han 1 02:00:00:1970
+++ linux/include/linux/in6.h sun Nov 3 11:04:42 1999
@@ -0,0 +1,99 @@
+ /*
+ * Types and definitions for AF_INET6
+ * Linux INET6 implementation
+ * + * Authours:
```

```
+ * Pedro Roque <******>
+ *
+ * Source:
+ * Ipv6 Program Interfaces for BSD System
<draft-ietf-ipngwg-bsd-api-05.txt>
```

2.2.2. In between

Because of lack of manpower, the IPv6 implementation in the kernel was unable to follow the discussed drafts or newly released RPCs. In October 2000, a project was started in Japan, called USAGI, whose aim was to implement all missing or outdated IPv6 support in Linux. It tracks the current IPv6 implementation in Free BSD made by the KAME project. From time to lime they create snapshots against current vanilla Linux

2.2.3. Current

Unfortunately, the USAG1 patch is so big, that current networking maintainers are unable to include it in the production source of the Linux kernel 2.4-x series. Therefore the 2Ax series is missing some (many) extensions and also does not confirm to all current drafts and RFCs. This can cause some interoperability problems with other operating systems.

2.2.4. Future

USAGT is now making use of the new Linux kernel development series 2.5.x to insert all of their current extensions into this development release. Hopefully the 2.6.x kernel series will contain a true and up-to-date IPv6 implementation.

2.3. How do IPv6 addresses look like?

As previously mentioned, IPv6 addresses are 128 bits long. This number of bits generates very high decimal numbers with up to 39 digits:

2^128}-1: 340262366920938463463374607431768211455

Such numbers are not really addresses that can be memorized. Also the IPv6 address schema is bit wise orientated (just like IPv4, but that's not often recognized). Therefore a better notation of such big numbers is hexadecimal. In hexadecimal, 4 bits (also known as "nibble") are represented by a digit or character from 0-9 and a-f (10-15). This format reduces the length of the IPv6 address to 32 characters.

2^(128-1:0xffffffffffffffffffffffffffffffff

This representation is still not very convenient (possible mix-up or loss of single hexadecimal digits), so the designers of IPv6 chose a hexadecimal format with a colon as separator after each block of 16 bits in addition, the leading "Ox" (a signifier for hexadecimal values used in programming languages) is removed:

2 ^(128}-1: ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

A usable address (see address types later) is e.g.:

3ffe:ffff:0100;fl01:0210:a4ff:fee3:9566

For simplifications, leading zeros of each 16 bit block can be omitted:

3ffe:ffff:0100:flOl:0210:a4ff:fee3:9566 ->

3ffe:ffff:100:fl01:210:a4ff:fee3: 9566

One sequence of 16 bit blocks containing only zeroes can be replaced with ";:". But not more than one at a lime, otherwise it is no longer a unique representation.

3ffe:ffff:100:fl01:0:0:0:l -> 3ffe:ffff:100:fl01::l

The biggest reduction is seen by the IPv6 localhost address:

0000:0000:0000:0000:0000:0000:0000:0001 -> ::1

There is also a so-called compact (base85 coded) representation defined RFC 1924 / A Compact Representation of IPv6 Addresses here is an example:

```
# ipv6calc —-addr_to_base85 3ffe: ffff: 0100:f101: 0210: a4ff: fee3: 9566
Itu&-ZQ82s>J%s99fJXT
```

Info: ipv6calc is an IPv6 address formal calculator and converter program and can be found on: ipv6calc

2.4. FAQ (Basics)

2.4.1. Why is the name IPv6 and not IPv5 as successor for IPv4?

On any IP header, the first 4 bits are reserved for protocol version. So theoretically a protocol number between 0 and 15 is possible:

4: is already used for IPv4

5: is reserved for the Stream Protocol (STP, RFC 1819) (which never really made it to the public) The next free number was 6. Hence IPv6 was born.

2.4.2. IPv6 addresses: why such a high number of bits?

During the design of IPv4, people thought that 32 bits were enough for the world. Looking back into the past 32 bits were enough until now and will perhaps be enough for another few years. However, 32 bits are not enough to provide each network device with a global address in die future. Think about mobile phones, cars (including electronic devices on its CAN-bus), toasters, refrigerators, light switches, and so on...

So designers have chosen 128 bits, 4 limes more in length and 2^96 greater in size than in IPv4 today.The usable size is smaller than it may appear however. This is because in the currently defined address schema, 64 bits are used for interface identifiers- The other 64 bits are used for routing. Assuming the current strict levels of aggregation (/48, /35,...), it is still possible to "run out" of space, but hopefully not in the near future.

2.4.3. IPv6 addresses: why so small a number of bits on a new design?

While, there are (possibly) some people on the Internet who are thinking about IPv8 and IPvl6, their design is far away from acceptance and implementation. In the meantime 128 bits was the best choice regarding header overhead and data transport. Consider the minimum Maximum Transfer Unit (MTU) in IPv4 (576 octets) and in IPv6 (1280 octets), the header length in IPv4 is 20 octets (minimum, can increase to 60 octets with IPv4 options) and in IPv6 is 48 octets (fixed). This is 3.4 % of MTU in IPv4 and 3.8% of MTU in IPv6.

This means the header overhead is almost equal. More bits for addresses would require bigger headers and therefore more overhead. Also, consider the maximum MTU on normal links (like Ethernet today): it's 1500 octets (in special cases: 9k octets using Jumbo frames).Ultimately, it wouldn't be a proper design if 10 % or 20 % of transported data in a Layer-3 packet were used for addresses and not
for payload.

Chapter 3

Address types

Like IPv4, IPv6 addresses can be split into network and host parts using subnet masks.IPv4 has shown that sometimes it would be nice, if more than one IP address can be assigned to an interface, each for a different purpose (aliases, multi-cast). To remain extensible in the future, IPv6 is going further and allows more than one IPv6 address to be assigned to an interface. There is currently no limit defined by an RFC, only in the implementation of the IPv6 stack (to prevent DoS attacks).Using this large number of bits for addresses, IPv6 defines address types based on some leading bits, which are hopefully never going to be broken in the future (unlike IPv4 today and the history of class A, B, and C).Also the number of bits are separated into a network part (upper 64 bits) and a host part (lower 64 bits), to facilitate auto-configuration .

3.1. Addresses without a special prefix

3.1.1. Localhost address

This is a special address for the loopback interface, similiar to IPv4 with its "127.0.0.1".

With IPv6 the localhost address is:

0000:0000:0000:0000:0000:0000:0000:0001

or compressed::1

Packets with this address as source or destination should never leave the sending host.

3.1.2. Unspecified address

This is a special address like "any" or "0.0.0.0" in IPv4 . For IPv6 it's:

0000:0000:0000:0000:0000:0000:0000:0000

or:

::

These addresses are mostly used/seen in socket binding (to any IPv6 address) or routing tables.Note: the unspecified address cannot be used as destination address.

3.1.3. IPv6 address with embedded IPv4 address

There are two addresses which contain an IPv4 address.

3.1.3.1. IPv4-mapped IPv6 address

IPv4-only IPv6-comparible addresses are sometimes used/shown for sockets created by an IPv6-enabled daemon, but only binding to an IPv4 address.

These addresses are defined with a special prefix of length 96 (a.b.c.d is the IPv4 address):

0: 0: 0: 0: 0:ffff:a.b.c.d/ 96

or in compressed format

::ffff:a.b.c.d/96

For example, the IPv4 address 1.2.3.4 looks like this ::ffff:1.2.3.4

3.1.3.2. IPv4-compatible IPv6 address

Also for sockets, in this case it is for a dual purpose and looks like:

0:0:0:0:0:0:a.b.c.d/96

or in compressed format

::a.b.c.d/96

These addresses are also used by automatic tunneling, which is being replaced by 6to4 tunneling

3.2. Network part, also known as prefix

Designers defined some address types and left a lot of scope for future definitions as currentlyUnknown requirements arise. RFC 2373 [July 1998]/ IP Version 6 Addressing Architecture defines the current addressing scheme but there is already a new draft available: draf-ietf-ipngwg-addr-arch-*.txt.

Now lets take a look at the different types of prefixes (and therefore address types):

3.2.1. Link local address type

These are special addresses which will only be valid on a link of an interface. Using this address as destination the packet would never pass through a router. It's used for link communications such as:anyone else here on this link? anyone here with a special address (e-g- looking for a router)? They begin with (where "x" is any hex character, normally "O")

fe8<emphasis>x: <- currently the only one in use. </emphasis>

fe9<emphasis>x: </emphasis>

fea<emphasis>x: </emphasis>

feb<emphasis>x: </emphasis>

An address with this prefix is found on each IPv6-enabled interface after stateless auto-configuration(which is normally always the case).

Note: only fe80 is currently in use.

3.2.2. Site local address type

These are addresses similar to the RFC 1918 / Address Allocation for Private Internets in IPv4 today,with the added advantage that everyone who use this address type has the capability to use the given 16 bits for a maximum number of 65536 subnets. Comparable with the 10.0.0.0/8 in IPv4 today.Another advantage: because it's possible to assign more than one address to an interface with IPv6,you can also assign such a site local address in addition to a global one.

It begins with:

fec<emphasis>x: <- most commonly used.</emphasis>

fed<emphasis>x:</emphasis>

fee<emphasis>x:</emphasis>

fef<emphasis>x:</emphasis>

(where "x" is any hex character, normally "0")

3.2.3. Global address type "Aggregatable global unicast"

Today, there is one global address type defined (the first design, called "provider based," was thrown away some years ago RFC 1884 / IP Version 6 Addressing Architecture [obsolete].

2<emphasis>xxx</emphasis>:

3<emphasis>xxx</emphasis>:

There are some further subtypes defined, see below:

3.2.3.1. 6bone test addresses

These were the first global addresses which were defined and in use. They all start with 3tfe:

Example:

```
3ffe:ffff:100:fl02::l
```

A special 6bone test address which will be never be globally unique begins with

```
3ffe:ffff:
```

and is mostly shown in examples, because if real addresses are shown, its possible for someone to do a copy & paste to their configuration files. Thus inadvertently causing duplicates on a globally unique address. This would cause serious problems for the original host (e.g. getting answer packets for request that were never sent). You can still apply for one of these prefixes, see here Mow to join 6bone. Also some tunnel brokers still distribute 6bone test address prefixes.

3.2.3.2. 6to4 addresses

These addresses, designed for a special tunneling mechanism [RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds and RFC 2893 / Transition Mechanisms for fPv6 Hosts and Routers],encode a given IPv4 address and a possible subnet and begin with 2002:For example representing 192.168.1.1/5 : 2002:c0a8:0101:5::1

A small shell command line can help you generating such address out of a given IPv4 one:

```
ipv4="1.2.3.4"; sla="5"; printf "2002:%02x%02x:%02x%02x:%04x::1" echo $ipv4 | tr
"." " " $sla
```

See also tunneling using 6to4 and information about 6to4 relay routers.

3.2.3. Global address type "Aggregatable global unicast" 15

3.2.3.3. Assigned by provider for hierarchical routing

These addresses are delegated to Internet service providers (ISP) and begin with 2001:Prefixes to major (backbone owning) ISPs are delegated by local registries and currently they assign to them a prefix with length 35.Major ISPs normally delegate to minor ISPs a prefix with length 48.

3.2.4. Multicast addresses

Multicast addresses are used for related services.

They alway start with (xx is the scope value)

ff<emphasis>x</emphasis>y:

They are split into scopes and types:

3.2.4.1. Multicast scopes

Multicast scope is a parameter to specify the maximum distance a multicast packet can travel from the sending entity. Currently, the following regions (scopes) are defined;

ffx1: node-local, packets never leave the node.

ffx2: link-local, packets are never forwarded by routers, so they never leave the specified link.

ffx5: site-local, packets never leave the site.

ffx8: organization-local, packets never leave the organization (not so easy to implement, must be covered by routing protocol).

ffxe: global scope, others are reserved

3.2.4.2. Multicast types

There are many types already defined/reserved (see RFC 2373 / IP Version 6 Addressing Architecture for details). Some examples are:

All Nodes Address: ID = 1h, addresses all hosts on the local node (ff02:0:0:0:0:0:0:1). or the connected link (ff02:0:0:0:0:0:0:1).

All Routers Address; ID=2h, addresses all routers on the local node (ff0lH:0:0:0:0:0:0:2), on the connected link (ff02:0:0:0:0:0:0:2), or on the local site (ff05:0;0;0:0:0:0:2)

3.2.4.3. Solicited node link-local multicast address

Special multicast address used as destination address in neighborhood discovery, because unlike in IPv4, ARP no longer exists in IPv6.

An example of this address looks like

ff02::1:ff00:1234

Used prefix shows that is a link-local multicast address. The suffix is generated from the destination address. In this example, a packet should be sent to address "fe80;:1234", but the network stack doesn't know the current layer 2 MAC address. It replaces the upper 104 bits with

"ff02:0:0:0:0:1:ff00::/104" and leaves the lower 24 bits untouched. This address is now used 'on-link' to find the corresponding node which has to send a reply containing its layer 2 MAC address.

3.2.5. Anycast addresses

Anycast addresses are special addresses and are used to cover things like nearest DNS server, nearest DHCP server, or similar dynamic groups. Addresses are taken out of the unicast address space(aggregatable global or site-local at the moment). The anycast mechanism (client view) will be handled by dynamic routing protocols.

Note: Anycast addresses cannot be used as source addresses, they are only used as destination addresses.

3.2.5.1. Subnet-router anycast address

A simple example for an anycast addresses is the subnet-router anycast address. Assuming that a node has the following global assigned IPv6 address:

3ffe:ffff:100:f101:210:a4ff:fee3:9566/64 <- Node's address

The subnet-router anycast address will be created blanking the suffix (least significant 64 bits)completely:

3ffe:ffff:100:flOl::/64 <- subnet-router anycast address .

3.3. Address types (host part)

For auto-configuration and mobility issues, it was decided to use the lower 64 bits as host part of the address in most of the current address types. Therefore each single subnet can hold a large amount of addresses.

This host part can be inspected differently:

3.3.1. Automatically computed (also known as stateless)

With auto-configuration, the host part of the address is computed by converting the MAC address of an interface (if available), with the EUI-64 method, to a unique IPv6 address. If no MAC address is available (happens e.g. on virtual devices), something else (like the IPv4 addresses or the MAC address of a physical interface) is used instead.
Consider again the first example

```
3ffe:ffff:100:f101:210:a4ff:fee3:9566
```

here,

```
210:a4ff:fee3:9566
```

is the host part and computed from the NIC's MAC address

```
00:10:A4:E3:95:66
```

using the IEEE-Tutorial EUT-64 design for EUI-48 identifiers.

3.3.1.1. Privacy problem with automatically computed and solution

Because the "automatically computed" host part is globally unique (except when a vendor of a NIC uses the same MAC address on more than one NIC), client tracking is possible on the host when not using a proxy of any kind.
This is a known problem, and a solution was defined: privacy extension, defined in RFC 3041 /Privacy Extensions for Stateless Address Autoconfiguration in IPv6 (there is also already a newel-draft available; draft-ietf-ipngwg-temp-addresses-*. txt). Using a random and a static value a new suffix is generated from time to time. Note; this is only reasonable for outgoing client connections and isn't really useful for well-known servers.

3.3.2. Manually set

For servers it's probably easier to remember simpler addresses, this can also be accommodated- It is possible to assign an additional IPv6 address to an interface, e.g.

```
3ffe:ffff:100:f101::1
```

For manual suffixes like ":: 1" shown in the above example it's required that the 6th most significant)bit is set to 0 (the universal/local bit of the automatically generated identifier). Also some other(otherwise unchosen) bit combinations are reserved for anycast addresses, too.
Linux IPv6 HOWTO

3.4. Prefix lengths for routing

In the early design phase it was planned to use a fully hierarchical routing approach to reduce the size of the routing tables maximally. The reasoning behind this approach were the number of current IPv4 routing entries in core routers (> 104 thousand in May 2001), reducing the need of memory in hardware routers (ASIC driven) to hold the routing table and increase speed (fewer entries hopefully result in faster lookups).

Todays view is that routing will be mostly hierarchically designed for networks with only one service provider. With more than one ISP connections, this is not possible, and subject to an issue named multi-homing.

3.4.1. Prefix lengths (also known as "netmasks")

Similar to IPv4 the routable network path for routing to take place. Because standard netmask notation for 128 bits doesn't look nice, designers employed the IPv4 Classless Inter Domain Routing (CIDR, R-FC 1519 / Classless Inter-Domain Routing) scheme, which specifies the number of bits of the IP address to be used for routing. It is also called the "slash" notation.

An example:

```
3ffe:ffff:100:1:2:3:4:5/48
```

This notation will be expanded Network

```
3ffe:fff:0100:0000:0000:0000:0000:0000
```

Net-mask:

```
ffff:ffff:ffff:0000:0000:0000:0000:0000
```

3.4.2. Matching a route

Under normal circumstances (no QoS) a lookup in a routing table results in the route with the most significant number of address bits means the route with the biggest prefix length matches first.

For example if a routing table shows following entries (list is not complete):

```
3ffe:ffff:100::/48 :: U 1 0 0 sit1
3ffe::/16 ::l92.88.99.1 UG 1 0 0 tun6to4
2000::/3 ::192.88.99.1 UG 1 0 0 tun6to4
```

Chapter 4

IPv6-ready system check

Before we can start using IPv6 on a network host we have to test, whether our system is IPv6-ready.We may have to do some work to enable it first.

4.1. IPv6-ready kernel

Modem Linux distributions already contain IPv6-ready kernels, the IPv6 capability is generally compiled as a module, but it's possible that this module is not loaded automatically on startup. See IPv6+Lmiix-SlaUis-Dislribntion page for most up-to-date information.

4.1.1. Check for IPv6 support in the current running kernel

To check whether your current running kernel supports IPv6, take a look into your /proc-file-system.

Following entry must exists:

```
/proc/net/if_inet6
```

A short automatical test looks like:

```
# test -f /proc/net/if_inet6 && echo "Running kernel is IPv6 ready"
```

If this fails, it is quite likely, that the IPv6 module is not loaded.

4.1.2. Try to load IPv6 module

You can try to load the IPv6 module executing

```
#modprobe  ipv6
```

If this is successful, this module should be listed, testable with following auto-magically line:

```
# 1smod |grep -w 'ipv6' && echo "IPv6 module successfully loaded"
```

4.1.2.1. Automatically loading of module

Its possible to automatically load the IPv6 module on demand. You only have to add following line in the configuration file of the kernel module loader (normally /etc/modules.confor/etc/conf.modules):

```
alias net-pf-10 ipv6 # automatically load IPv6 module on demand
```

It's also possible to disable automatically loading of the IPv6 module using following line

```
alias net-pf-10 off # disable automatically load of IPv6 module on demand
```

4.1.3. Compile kernel with IPv6 capabilities

If both above shown results were negative and your kernel has no IP6 support, than you have the following options. Update your distribution to a current one which supports IPv6 out-of-the-box(recommended for newbies).Compile a new vanilla kernel (easy, if you know which options you needed) Recompile kernel sources given by your Linux distribution (sometimes not so easy)

Compile a kernel with USAGI extensions

If you decide to compile a kernel, you should have previous experience in kernel compiling and read the.A mostly up-to-time comparison between vanilla and USAGI extended kernels is available on

4.1.4. IPv6-ready network devices

Not all existing network devices have already (or ever) the capability to transport IPv6 packets, major issue is that because of the network layer structure of kernel implementation an IPv6 packet isn't really recognized by its IP header number (6 instead of 4). It's recognized by the protocol number of the Layer 2 transport protocol. Therefore any transport protocol which doesn't use such protocol number cannot dispatch IPv6 packets. Note: the packet is still transported over the link, but on receivers side,The dispatching won't work (you can see this e.g. using tcpdump).

4.1.4.1. Currently known never "IPv6 capable links"

Serial Line IP (SLIP,RFC 1055), should be better called now to SLIPv4. device named:s1X .

Parallel Line IP (PLIP),same like SLIP,device names:plipX

ISDN with encapsulation Rawip,device names; isdnX

4.1.4.2. Currently known "not supported IPv6 capable links"

ISDN with encapsulation syncppp, device names: ipppX (design issue of the ipppd, will be merged into more general PPP layer in kernel series 2.5.x)

4.2. IPv6-ready network configuration tools

You wont get very far, if you are running an IPv6-ready kernel, but have no tools to configure IPv6.There are several packages in existence which can configure IPv6.

4.2.1. net-tools package

The net-tool package includes some tools like if config and route, which helps you to configure IPv6 on an interface. Look at the output of if config -? or route -?, if something is shown like IPv6 or inet6,then the tool is IPv6-ready.

Auto-magically check:

```
# /sbin/ifconfig -? 2>& 1|grep -qw 'inet6' && echo "utility 'ifconfig' is
 IPv6-ready"
```

Same check can be done for route:

```
# /sbin/route -? 2>& l|grep -qw 'inet6' && echo "utility 'route' is IPv6-ready"
```

4.2.2. iproute package

A newely created a tool-set which configures networks through the netlink device. Using this tool-set you have more functionality than net-tools provides, but its not very well documented and isn't for the faint of heart.

```
# /sbin/ip 2>&1 |grep -qw 'inet6' && echo "utility 'ip' is IPv6-ready"
```

If the program /sbin/ip isn't found, then I strongly recommend you install the iproute package.You can download the tar-ball and recompile it: Original FTP source and mirror (missing) You've able to look for a proper RPM package at RPMfind/iproute(sometimes rebuilding of a SRPMS package is recommended)

4.3. IPv6-ready test/debug programs

After you have prepared your system for IPv6, you now want to use IPv6 for network communications. First you should learn how to examine IPv6 packets with a sniffer program. This is strongly recommended because for debugging/troubleshooting issues this can aide in providing a diagnosis very quickly.

4.3.1. IPv6 ping

This program is normally included in package iputils. It is designed for simple transport tests sending ICMPv6 echo-request packets and wait for ICMPv6 echo-reply packets.

Usage

```
# pinq6 <hostwithipv6address>
# ping6 <ipv6address>
# ping6 [-1 <device>] <link-local-ipv6address>
```

Example

```
# ping6 -c 1 :: 1
PING ::1(::1) from ::1 : 56 data bytes
64 bytes from ::1: icmp_seq=0 hops=64 time=292 usec
--- ::1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.292/0.292/0.292/0.000 ms
```

Hint: ping6 needs raw access to socket and therefore root permissions. So if non-root users cannot use ping6 then there are two possible problems:

ping6 is not in users path (probably, because ping6 is generally stored in /usr/sbin -> add path (not really recommended)

ping6 doesn't execute properly, generally because of missing root permissions -> chmod u+s /usr/sbin/ping6

4.3.1.1. Specifying interface for IPv6 ping

Using link-local addresses for an IPv6 ping, the kernel does not know through which (physically or virtual) device it must send the packet - each device has a link-local address. A try will result in following error message:

```
# ping6 fe80::212:34ff:fel2:3456
connect: Invalid argument
```

In this case you have to specify the interface additionally like shown here:

```
# ping6 –I eth0 –c 1 fe80::2eo:18ff:fe90:9205
PING fe80::212:23ff:fel2:3456(fe80::212:23ff:fel2:3456) from
Fe80::212:34ff:fel2:3478 ethO: 56 data byte
64 bytes from fe80::212:23ff:fel2:3456:icmp_seq=0 hops=64 time-445 usec
---fe8O::2e0:18ff:fe90:9205 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss round-trip
min/avg/max/mdev = 0.445/0.445/0.445/0.000 ms
```

4.3.2 IPv6 traceroute6

This program is normally included in package ipuptils. It's a program similar to IPv4 traceroute. But unlike modern IPv4 versions, the IPv6 one still doesn't understand to traceroute using ICMP echo-request packets (which is more accepted by firewalls around than UDP packets to high ports). Below you will see an

example;

```
3: 3ffe:b00:cl8::5 asymm 2 266.145ms pmtu 1280
3: 3ffe:39OO:5::2 asymm 4 346.632ms
4: 3ffe:28ff:ffff:4::3 asymm 5 365.965ms
5: 3ffe:lcff:0:ee::2 asymm 4 534.704ms
6: 3ffe:3800::1:1 asymm 4 578.126ms !N
Resume: pmtu 1280
```

4.3.4. IPv6 tcpdump

On Linux, tcpdump is the major tool for packet capturing. Below you find some examples. IPv6 support is normally built-in in current releases of version 3.6.

tcpdump uses expressions for filtering packets to minimize the noise:

icmp6: filters native lCMPv6 traffic

ip6: filters native IPv6 traffic (including lCMPv6)

proto ipv6: filters tunneled IPv6-in-IPv4 traffic not port ssh: to suppress displaying SSH packets for running tcpdump in a remote SSH session

Also some command line options are very useful to catch and print more information in a packet,mostly interesting for digging into lCMPv6 packets:

"-s 512": increase the snap length during capturing of a packet to 512 bytes

"-vv": really verbose output

"-n": don't resolve addresses to names, useful if reverse DNS resolving isn't working proper

4.3.4.1. IPv6 ping to 3ffe:ffff:100:f101::1 native over a local link

```
# tcpdump -t -n. -i ethO -s 512 -vv ip6 or proto ipv6
tcpdump: listening on ethO
3ffe:ffff:100;flo1:2e0:18ff:fe90:9205 > 3ffe;ffff:100:flOl::1: icmp6: echo
request (len 64, hilm 64)
3ffe:ffff:100:fl01::1 > 3ffe:ffff:100:fl01:2e0:l8ff:fe90:9205: icmp6: echo
reply (len 64, hlim 64)
```

4.3.4.2. IPv6 ping to 3ffe:ffff:100::1 routed through an IPv6-in-IPv4-tunnel

1.2.3.4 and 5.6.7.8 are tunnel endpoints (all addresses are examples)

```
# tcpdump -t -n -i pppO -s 512 -vv ip6 or proto ipv6
tcpdump: listening on pppO
l.2.3.4 > 5.6.7.8: 2002:ffff:f5f8::1 > 3ffe:ffff:100::1: icinp6: echo request
(len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6,7.8 > 1.2.3.4: 3ffe:ffff:100::1 > 2002:ffff:f5f8::l: icmp6: echo reply (len
64,hlim 6l) (ttl 23, id 29887, len 124)
1.2.3.4 > 5.6.7.8; 2002:ffff:f5f8::1 > 3ffe:ffff:100::1: icmp6: echo request
(len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6.7.8 > 1.2.3.4: 3ffe:ffff:100::1 > 2002:ffff:f5f8::1: icrap6; echo reply (len
64, hlim 61) (ttl 23, id 29919, len 124)
#traceroute6
traceroute to 6bone.net (3ffe:b00:cl8:l::10) from 3ffe:ffff:0000;fl01::2, 30
hops max, 16 byte packets
1 localipv6gateway (3ffe: ffff:0000:f101::1) 1.354 ms 1.566 ms 0.407 ms
2 swi6T1-TO.ipv6.switch.ch (3ffe:2000:0:400::1) 90.431 ms 91.956 ms 92.377 ms
3 3ffe:2000:0:1::132 (3ffe:2000:0:1::132) 118.945 ms 107.982 ms 114.557 ms
4 3ffe:c00:8023:2b::2 (3ffe:c00:8023:2b::2) 968.468 ms 993.392 ms 973.441 ms
5 3ffe:2eOO:e:c::3 (3ffe:2e00:e:c::3) 507.784 ms 505.549 ms 508.928 ms
```

```
6 www.6bone.net (3ffe:b00:cl8:1::10) 1265.85 ms * 1304.74 ms
```

4.3.3. IPv6 tracepath6

This program is normally included in package iputils. It's a program like traceroute6 and traces the path to a given destination discovering the MTU along this path. Below you will see an example:

```
# tracepath6 www.6bone.net
1?: [LOCALHOST] pmtu 1480
1: 3ffe:401::2c0:33ff:fe02:14 150.705ms
2: 3ffe:b00:cl8::5 267.864ms
```

4.4. IPv6-ready programs

Current distributions already contain (he most needed IPv6 enabled client and servers.Applications whether the program is already ported to IPv6 and usable with Linux.

4.5. IPv6-ready client programs (selection)

To run the following shown tests, it's required that your system is IPv6 enabled, and some examples show addresses which only can be reached if a connection to the 6bone is available.

4.5.1. Checking DNS for resolving IPv6 addresses

Because of security updates in the last years every Domain Name System (DNS) server should run newer software which already understands the (intermediate) IPv6 address-type AAAA (the newer one named A6 isn't still common at the moment because only supported using BIND9 and newer and also the non-existent support of root domain IP6.ARPA). A simple whether the used system can resolve IPv6 addresses is

```
# host -t AAAA www.join.uni-muenster.de
```

and should show something like following:

```
www.join.unimuenstar.de. is an alias for ns.join.uni-muenster.de.
ns.join.uni-muenster.de. has AAAA address 3ffe:400:10:100:201:2ff:feb5:3806
```

4.5.2. IPv6-ready telnet clients

IPv6-ready telnet clients are available. A simple test can be done with

```
$ telnet 3ffe:400:100::1 80
Trying 3ffe:400:100::1.
Connected to 3ffe:400:100::1.
Escape character is '^]'.
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Date: Sun, 16 Dec 2003 16:07:21
GMT Server:: Apache/2.0.28 (Unix)
Last-Modified: Wed/ 01 Aug 2003 21:34:42 GMT
ETag: "3f02-a4d-blb3e080"
Accept-Ranges: bytes
content-Length: 2637
connection: close
Content-Type: text/html:charset=ISO-8859-I
Connection closed by foreign host.
```

If the telnet client don't understand the IPv6 address and says something like "cannot resolve hostname",then it's not IPv6-enabled.

4.5.3. IPv6-ready ssh clients

4.5.3.1. openssh

Current versions of openssh are IPv6-ready. Depending on configuring before compiling it has two behavior without-ipv4-default: the client tries an IPv6 connect first automatically and fall back to IPv4 if not working

with-ipv4-default: default connection is IPv4,IPv6 connection must be force like following

example shows

```
$ ssh -6 ::1
user@::1's password: ******
[user@ipv6host user]$
```

If your ssh client doesn't understand the option "-6" then it's not IPv6-enabled, like most ssh version 1 packages.

4.5.3.2. ssh.com

SSH.com's SSH client and server is also IPv6 aware now and is free for all Linux and FreeBSD machine regardless if used for personal or commercial use.

4.5.4. IPv6-ready web browsers

A current status of IPv6 enabled web browsers is available at rPv6+Linux-status-apps.hlml#HTTP.Most of them have unresolved problems at die moment If using an IPv4 only proxy in the settings, IPv6 requests will be sent to the proxy, but the proxy will fail to understand the request and the request fails. Solution: update proxy software (see later).

1 .Automatic proxy settings (*.pac) cannot be extended to handle IPv6 requests differently (e.g. don't use proxy) because of their nature (written in Java-script and well hard coded in source like to be seen in Maxilla source code).

2.Also older versions don't understand an URL with IPv6 encoded addresses like http://[3ffe:400:100::1]/ (this given URL only works with an IPv6-enabied browser!).

A short test is to try shown URL with a given browser and using no proxy.

4.5.4.1. URLs for testing

A good starting point for browsing using IPv6 is http://www.kame.net/. If the turtle on this page is animated, the connection is via IPv6, otherwise the turtle is static.

4.6. IPv6-ready server programs

```
In this part of this HOWTO, more client specific issues are mentioned.
Therefore hints for IPv6-readservers like sshd, httpd, telnetd, etc.
are shown below in Hints for IPv6-enabled daemons.
```

4.7. FAQ (IPv6-ready system check)

4.7.1. Using tools

4.7.1.1. Q: Cannot ping6 to link-local addresses

Error message: "connect:Invalid argument"

Kernel doesn't know, which physical or virtual link you want to use to send such ICMPv6 packets.Therefore it displays this error message.

Solution: Specify interface like: "ping6 -I ethO fe80::2e0:18ff:fe90:9205", see also program ping6 usage.

4.7.1.2. Q: Cannot ping6 or traceroute6 as normal user

Error message: "icmp socket:Operation not permitted"

These utilities create special ICMPv6 packets and send them out. This is done by using raw sockets in the kernel. But raw sockets can only be used by die "root" user. Therefore normal users get such error message. Solution: If it's really needed that all users should be able to .use this utilities, you can add the "Suid" bit using "chmod u+s/path/to/program", see also program ping6 usage. If not all users should be able to, you can change the group of the program to e.g. "wheel", add this power users to this group and remove die execution bit for other users using "chmod o-rwx /path/to/program". Or configure "sudo" to enable your security policy.

Chapter 5.
Configuring interfaces

5.1. Different network devices

On a node, there exist different network devices. They can be collected in classes Physically bounded, like ethO,trO Virtually existing, like ppp0, tunO, tapO, sitO,isdnO, ipppO

5.1.1. Physically bounded

Physically bounded interfaces like Ethernet or Token-Ring are normal'ones and need no special treatment.

5.1.2. Virtually bounded

Virtually bounded interfaces always need special support

5.1.2.1. IPv6-in-IPv4 tunnel interfaces

This interfaces are normally named site. The name sit is a shortcut for Simple Internet Transition. This device has the capability to encapsulate IPv6 packets into IPv4 ones and tunnel them to a foreign end point.sitO has a special meaning and cannot be used for dedicated tunnels.

5.1.2.2. PPP interfaces

PPP interfaces get their IPv6 capability from an IPv6 enabled PPP daemon.

5.1.2.3. ISDN HDLC interfaces

IPv6 capability for HDLC with encapsulation ip is already built-in in the kernel

5.1.2.4. ISDN PPP interfaces

ISDN PPP interfaces (ippp) aren't IPv6 enabled by kernel. Also there are also no plans to do that because in kernel 2.5.+ they will be replaced by a more generic ppp interface layer.

5.1.2.5. SLIP + PIIP

Like mentioned earlier, this interfaces don't support IPv6 transport (sending is OK, but dispatching on receiving don't work).

5.1.2.6. Ether-tap device

Ether-tap devices are IPv6-enabled and also stateless configured. For use, the module "ethertap" has to be loaded before.

Currently not tested by the test teams.

5.1.2.8. ATM

Aren't currently supported by all network test programes supported by USAGI extension

5.2. Bringing interfaces up/down

Two methods can be used to bring interfaces up or down.

5.2.1. Using "ip"

Usage:

```
# ip link set  dev <interface> up
# ip link set dev <interface> down
```

Example:

```
# ip link set dev ethO up
# ip link set dev ethO down
```

5.2.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> up
# /sbin/ifconfig <interface> down
```

Example:

```
# /sbin/ifconfig ethO up
# /sbin/ifconfig ethO down
```

Chapter 6

Configuring IPv6 addresses

There are different ways to configure an IPv6 address on an interface, we can use "ifconfig" or “ip”.

6.1. Displaying existing IPv6 addresses

First you should check, whether and which IPv6 addresses are already configured (perhaps auto-magically during stateless auto-configuration).

6.1.1. Using "ip"

Usage:

```
# /sbin/ip -6 addr show dev <interface>
```

Example for a static configured host:

```
# /sbin/ip -6 addr show dev
2 :ethO hO:<BROADCAST, MULTICAST, UP&gt; mtu 1500 qdisc  pfifo_fast qlen 100
inet6 fe80::210:a4ff:fee3:9566/10 scope link
inet6 3ffe:ffff:0:fl01::l/64 scope global
net6 fecO:0:0:flOl::1/64 scope site
```

Example for a host which is auto-configured

Here you see some auto-magically configured IPv6 addresses and their lifetime.

```
# /sbin/ip -6 addr show dev ethO
3: ethO:<BROADCAST,MULTICAST,PRCMISC,UP&gt;mtu 1500 qdisc pfifo_fast qlen
 100
inet6 2002:d95O:f5f8:flOl:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft l6sec preferred_1ft 6sec
inet6 3ffe:400:100:f101:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft 2591997sec preferred_lft 604797sec inet6 fe80::2e0:18ff:fe90:9205/10
scope link
```

6.1.2.Using"ifconfig"

Usage:

```
# /sbin/ifconfig <interface>
```

Example (output filtered with grep to display only IPv6 addresses). Here you see different IPv6 addresses with different scopes.

```
# /sbin/ifconfig ethO |grep "inet6 addr:"
inet6 addr: fe80::210:a4ff:fee3:9566/10 Scope:Link
inet6 addr: 3ffe:ffff:0:fl0l::l/64 3scope:Global
inet6 addr: fec0:0:0:f101::1/64 scope:site
```

6.2. Add an IPv6 address

Adding an IPv6 address is similar to the mechanism of "IP ALIAS" addresses in Linux IPv4 addressed interfaces.

6.2.1. Using "ip"

Usage:

```
# /sbin/ip -6 addr add <ipv6address>/<prefixlength> dev <interface>
```

Example:

```
# /sbin/ip -6 addr add 3ffe:ffff:0:flOl::1/64 dev ethO
```

6.2.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> inet6 add <ipv6address>/<prefixlength>
```

Example;

```
# /sbin/ifconfig ethO inet6 add 3ffe:ffff:0:flOl::1/64
```

6.3. Removing an IPv6 address

Not so often needed be carefully with removing non existent IPv6 address, sometimes using older programs it results in a crash.

6.3.1. Using "ip"

Usage:

```
# /sbin/ip -6 addr del <ipv6address>/<prefixlength> dev <interface>
```

Example:

```
# /sbin/ip -6 addr del 3ffe:ffff:0:fl01::l/64 dev ethO
```

6.3.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> inet6 del <ipv6address>/<prefixlength>
```

Example:

```
# /sbin/ifconfig ethO inet6 del 3ffe:ffff:0:fl0l::1/64
```

Chapter 7

Configuring normal IPv6 routes

If you want to leave your link and want to send packets in the world wide IPv6-Intenet you need routing.If there is already an IPv6 enabled router on your link it's possible enough to add IPv6 routes.Also here there are different ways to configure an IPv6 address on an interface. You can use "ifconfig" or "ip"

7.1. Displaying existing IPv6 routes

First you should check whether and which IPv6 addresses are already configured (perhaps auto-magically during auto-configuration).

7.1.1. Using "IP"

Usage:

```
# /sbin/ip -6 route show [dev <device>]
```

Example:

```
# /sbin/ip -6 route show dev ethO
3ffe:ffff:0:f101::/64 proto kernel metric 256 mtu 1500 advmss 1440
fe80::/l0 proto kernel metric 256 mtu 1500 advmss 1440
ffOO::/8 proto kernel metric 256 mtu 1500 advmss 1440
default proto kernel metric 256 mtu 1500 advmss 1440
```

7.1.2. Using "route"

Usage:

```
# /sbin/route -A inet6
```

Example (output is filtered for interface ethO). Here you see different IPv6 routes for different addresses on a single interface.

```
# /sbin/route -A inet6 |grep -w "ethO"
3ffe:ffff:0:fl01 ::/64 :: UA 256 0 0 ethO <- Interface route for global
address
 fe80::/1O :: UA 256 0 0 ethO <- Interface route for link-local address
 ff0O::/8 :: UA 256 0 0 ethO <- Interface route for all multicast addresses
 ::/0 :: UDA 256 0 0 ethO <- Automatic default route
```

7.2. Add an IPv6 route through a gateway

Mostly needed to reach the outside with IPv6 using an IPv6-enabled router on your link.

7.2.1. Using "ip"

Usage:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> via <ipv6address
[dev <device>]
```

Example:

```
# /sbin/ip -6 route add 2000::/3 via 3ffe:ffff:0:fl01:: 1
```

7.2.2. Using "route"

Usage:

```
# /sbin/route -A inet6 add <ipv6network>/<prefixlength> gw
ipv6address> [dev <device>]
```

A device can be needed, too, if the IPv6 address of the gateway is a link local one.

Following shown example adds a route for all currently global addresses (2000::/3) through gateway 3ffe:ffff:o:f1o1::1

```
# /sbin/route -A inet6 add 2000::/3 gw 3ffe:ffff:0:fl01::l
```

7.3. Removing an IPv6 route through a gateway

Not so often needed manually, mostly done by network configure scripts on shutdown (full or perinterface)

7.3.1. Using "ip"

Usage:

```
# /sbin/ip -6 route del <ipv6network>/<prefixlength> via <ipv6address>
[dev <device>]
```

Example:

```
# /sbin/ip -6 route del 2000::/3 via 3ffe:ffff:0:fl01::l
```

7.3.2. Using "route"

Usage:

```
# /sbin/route -A inet6 del <network>/<prefixlength> [dev <device>]
```

Example for removing upper added route again:

```
# /sbin/route -A inet6 del 2000::/3 gw 3ffe:ffff:0:f101::!
```

7.4. Add an IPv6 route through an interface

Not often needed, sometimes in cases of dedicated point-to-point links.

7.4.1. Using "ip"

Usage:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> dev <device>
metric 1
```

Example:

```
# /sbin/ip -6 route add 2000::/3 dev ethO metric 1
```

Metric "1" is used here to be compatible with the metric used by route, because the default metric on using "ip" is "1024".

7.4.2. Using "route"

Usage:

```
# /sbin/route -A inet6 add <network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/route -A inet6 add 2000::/3 dev eth0
```

7.5. Removing an IPv6 route through an interface

Not so often needed to use by hand, configuration scripts will use such on shutdown.

7.5.1. Using "ip"

usage:

```
# /sbin/ip -6 route del <ipv6network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/ip -6 route del 2000::/3 dev ethO
```

7.5.2. Using "route"

Usage:

```
# /sbin/route -A inet6 del <network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/route -A inet6 del 2000::/3 dev ethO
```

7.6. FAQ for IPv6 routes

7.6.1. Support of an IPv6 default route

One idea of IPv6 was a hierachical routing, therefore only less routing entries are needed in routers.There are some issues in current experiments:

7.6.1.1. Clients

Client can setup a default route like prefix "O", they also learn such route on autoconfiguration e.g using radvd on the link like following example shows:

```
# ip -6 route show | grep ^default
default via fe80::212:34ff:fel2:3450 dev ethO proto kernel metric 1024 expires
29sec mtu 1500 advmss 1440
```

7.6.1.2. Routers on packet forwarding

Current mainstream Linux kernel (at least <= 2.4.17) don't support default routes. You can set them up, but the route lookup fails when a packet should be forwarded (normal intention of a router Therefore at this time "default routing" can be setup using the currently only global address prefix "2000::/3".The USAGI project already supports this in their extension with a hack.

Chapter 8.

Configuring IPv6-in-IPv4 tunnels

If you want to leave your link you have no IPv6 capable network around you, you need IPv6-in-IPv4 tunneling to reach die World Wide IPv6-Intemet.

There are some kind of tunnel mechanism and also some possibilities to setup tunnels.

8.1. Types of tunnels

There are more than one possibility to tunnel IPv6 packets over IPv4-only links.

8.1.1. Static point-to-point tunneling: 6bone

A point-to-point tunnel is a dedicated tunnel to an endpoint, which knows about your IPv6 network(for backward routing) and the IPv4 address of your tunnel endpoint and defined in RFC 2893 /Transition Mechanisms for IPv6 Hosts and Routers. Requirements:

IPv4 address of your local tunnel endpoint must be static, global unique and reachable from the foreign tunnel endpoint -A global IPv6 prefix assigned to you (see 6bone registry)

A foreign tunnel endpoint which is capable to route your IPv6 prefix to your local tunnel endpoint(mostly remote manual configuration required)

8.1.2. Automatically tunneling

Automatic tunneling occurs, when a node directly connects another node gotten die IPv4 address of the other node before.

8.1.3. 6to4-Tunneling

6to4 tunneling (RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds) uses a simple mechanism to create automatic tunnels. Each node with a global unique IPv4 address is able to be a 6to4 tunnel endpoint (if no IPv4 firewall prohibits traffic). 6to4 tunneling is mostly not a one-to-one tunnel. This case of tunneling can be divided into upstream and downstream tunneling. Also, a special IPv6 address indicates that this node will use 6to4 tunneling for connecting the world-wide IPv6 network

8.1.3.1. Generation of 6to4 prefix

The 6to4 address is defined like following (schema is taken from RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds);

```
|3+13 | 32 | 16 | 64 bits |
| FP+TLA | V4ADDR | SLA ID | Interface ID |
| 0x2002 | | | |
```

Where FP is the known prefix for global addresses, TLA is the top level aggregator. V4ADDR is the node's global unique IPv4 address (in hexadecimal notation). SLA is the subnet identifier (65536 local subnets possible). Such prefix is generated and normally using SLA "0000" and suffix "::1" assigned
to the 6to4 tunnel interface.

8.1.3.2. 6to4 upstream tunneling

The node has to know to which foreign tunnel endpoint its in IPv4 packed IPv6 packets should be send to. In "early" days of 6to4 tunneling, dedicated upstream accepting routers were defined. See NSayer's 6to4 information for a list of routers.

Nowadays, 6to4 upstream routers can be found auto-magically using the anycast address 192.88.99.1.In the background routing protocols handle this, see RFC 3068 / An Anycast Prefix for 6to4 Relay Routers for details.

8.1.3.3. 6to4 downstream tunneling

The downstream (6bone -> your 6to4 enabled node) is not really fix and can vary from foreign host which originated packets were send to. There exist two possibilities:

Foreign host uses uses 6to4 and sends packet direct- back to your node (see below)

Foreign host sends packets back to the world-wide IPv6 network and depending on the dynamic routing a relay router create a automatic tunnel back to your node.

8.1.3.4. Possible 6to4 traffic

from 6to4 to 6to4: is normally directly tunneled between the both 6to4 enabled hosts

from 6to4 to 6to4: is sent via upstream tunneling

from 6to4 to 6to4: is sent via downstream tunneling

8.2. Displaying existing tunnels

8.2.1. Using "ip"

Usage:

```
# /sbin/ip -6 tunnel show [<device>]
```

8.2.2. Using "route"

Usage:

```
# /sbin/route -A inet6
```

Example (output is filtered to display only tunnels through virtual interface sitO):

```
# /sbin/route -A inet6 | grep "\WsitO\W+$"
::/96 :: u 256 2 0 sitO
2002::/16 :: UA 256 0 0 sitO
2000::/3 ::193.113.58.75 UG 1 0 0 sit0
fe80::/10 :: UA 256 0 0 sit0
ff00::/8 :: UA 256 0 0 sit0
```

8.3. Setup of point-to-point tunnel

There are 3 possibilities to add or remove point-to-point tunnels.

8.3.1. Add point-to-point tunnels

8.3.1.1. Using "ip" and "route"

Common method at the moment for a small amount of tunnels Usage for creating a tunnel device (but it's not up afterward, also a TTL must be specified because the default value is 0).

```
# /sbin/ip tunnel add <device> mode sit tt1 <ttldefault> remote
 <ipv4addressofforeigntunnel> local <ipv4addresslocal>
```

Usage (generic example for three tunnels):

```
#/sbin/ip tunnel add sit1 mod sit ttl <ttldefault> remote
 <ipv4addressofforeigntunnell> local <ipv4addresslocal>
# /sbin/ifconfig sit1 up
# /sbin/route -A inet6 add <prefixtoroutel> dev sit1
# /sbin/ip tunnel add sit2 mode sit ttl <tt1default>
<ipv4addressofforeigntunnel2> local <ipv4addresslocal>
# /sbin/ifconfig sit2 up
# /sbin/route -A inet6 add <prefixtoroute2> dev sit2
# /sbin/ip tunnel add sit3 mode sit ttl <ttldefault>
<ipv4addressofforeigntunnel3> local <ipv4addresslocal>
# /sbin/ifconfig sit3 up
# /sbin/route -A inet6 add <prefixtoroute3> dev sit3
```

8.3.1.2. Using "ifconfig" and "route" (deprecated)

This not very recommended way to add a tunnel because it's a little bit strange. No problem if adding only one, but if you setup more than one, you cannot easy shutdown the first ones and leave the others running. Usage (generic example for three tunnels):

```
# /sbin/ifconfig sitO up
# /sbin/ifconfig sitO tunnel <ipv4addressofforeigntunnell>
# /sbin/ifconfig sit1 up
# /sbin/route -A inet6 add <prefixtoroutel> dev sit1
# /sbin/ifconfig sitO tunnel <ipv4addressofforeigntunnel2>
# /sbin/ifconfig sit2 up
# /sbin/route -A inet6 add <prefixtoroute2> dev sit2
# /sbin/ifconfig sitO tunnel <ipv4addressofforeigntunnel3>
# /sbin/ifconfig sit3 up
# /sbin/route  -A inet6 add <prefixtoroute3> dev sit3
```

Important: DONT USE THIS, because this setup implicit enable "automatic tunneling" from anywhere in the Internet, this is a risk, and it should not be advocated.

8.3.1.3 Using "route" only

It's also possible to setup tunnels in Non Broadcast Multiple Access (NBMA) style, it's a easy way to add many tunnels at once. But none of the tunnel can be numbered (which is a not required feature).

Usage (generic example for three tunnels):

```
# /sbin/ifconfig sitO up
# /sbin/route -A inet6 add <prefixtoroutel> gw
:: <ipv4addressofforeigntunnel1> dev sitO
# /sbin/route -A inet6 add <prefixtoroute2> gw
:: <ipv4addressofforeigntunnel2> dev sitO
# /sbin/route -A inet6 add <prefixtoroute3> gw
::<ipv4addressofforeigntunnel3> dev sitO
```

Important: DON'T USE THIS, because this setup implicit enable "automatic tunneling" from anywhere in the Internet, this is a risk, and it should not be advocated.

8.3.2. Removing point-to-point tunnels

Manually not so often needed, but used by scripts for clean shutdown or restart of IPv6 configuration.

8.3.2.1. Using "ip" and "route"

Usage for removing a tunnel device:

```
# /sbin/ip tunnel del <device>
```

Usage (generic example for three tunnels):

```
# /sbin/route -A inet6 del <prefixtoroute1> dev sit1
# /sbin/ifconfig sit1 down
# /sbin/ip tunnel del sit1
# /sbin/route -A inet6 del <prefixtoroute2> dev sit2
# /sbin/ifconfig sit2 down
# /sbin/ip tunnel del sit2
# /sbin/route -A inet6 del <prefixtoroute3> dev sit3
# /sbin/ifconfig sit3 down
# /sbin/ip tunnel del sit3
```

8.3.2.2. Using "ifconfig" and "route" (deprecated because not very funny)

Not only the creation is strange, the shutdown also...you have to remove die tunnels in backorder,means the latest created must be removed first.Usage (generic example for three tunnels);

```
# /sbin/route -A inet6 del <prefixtoroute3> dev sit3
# /sbin/ifconfiq sit3 down
# /sbin/route -A inet6 del <prefixtoroute2> dev sit2
# /sbin/ifconfig sit2 down
# /sbin/route -A inet6 add <prefixtoroutel> dev sit1
# /sbin/ifconfig sit1 down
# /sbin/ifconfiq sitO down
```

8.3.2.3. Using "route"

This is like removing normal IPv6 routes Usage (generic example for three tunnels):

```
# /sbin/route -A inet6 del <prefixtoroute1> gw
::<ipv4addressofforeigntunnel1> dev sitO
# /sbin/route -A inet6 del <prefixtoroute2> gw
::<ipv4addressofforeigntunnel2> dev sitO
# /sbin/route -A inet6 del <prerixtoroute3> gw
::<ipv4addressofforeigntunnel3> dev sitO
# /sbin/ifconfig sitO down
```

8.3.3. Numbered point-to-point tunnels

Sometimes it's needed to configure a point-to-point tunnel with IPv6 addresses like in IPv4 today.This is only possible with the first (ifconfig+route - deprecated) and third (ip+route) tunnel setup. In such cases, you can add the IPv6 address to the tunnel interface like shown on interface configuration.

8.4. Setup of 6to4 tunnels

Pay attention that the support of 6to4 tunnels currently lacks on vanilla kernel series 2.2.x (see/systemcheck/kernel for more information). Also note that mat the prefix length for a 6to4 address is 16 because of from network point of view, all other 6to4 enabled hosts are on the same layer 2.

8.4.1. Add a 6to4 tunnel

First, you have to calculate your 6to4 prefix using your local assigned global routable IPv4 address (if your host has no global routable IPv4 address, in special cases NAT on border gateways is possible)Assuming your IPv4 address is 1.2.3.4

the generated 6to4 prefix will be

```
2002:0102:0304::
```

Local 6to4 gateways should always assigned the manual suffix "::1", therefore your local 6to4 address will be

```
2002:0102:0304::1
```

Use e.g. following for automatic generation:

```
ipv4="1.2.3.4"; printf "2002:%02x%02x:%02x%02x::1" 'echo $ipv4 | tr "." " "
```

There are two ways possible to setup 6to4 tunneling now.

Bring interface up

```
# /sbin/ip link set dev tun6to4 up
```

Add local 6to4 address to interface

```
# /sbin/ip -6 addr add <local6to4address>/16 dev tun6to4
```

Add (default) route to the global IPv6 network using the all-6to4-routers IPv4 anycast address

```
# /sbin/ip -6 route add 2000::/3 via :: 192.88.99.1 dev tun6to4 metric 1
```

8.4.1.2. Using "ifconfig" and "route" and generic tunnel device "sitO" (deprecated)

This is now deprecated because using the generic tunnel device sitO doesn't let specify filtering perdevice. Bring generic tunnel interface sitO up

```
# /sbin /ifconfig sit0 up
```

Add local 6to4 address to interface

```
# /sbin/ifconfiq sitO add <local6to4address>/16
```

Add(default)route to the global IPv6 network using the all-6to4-relays IPv4 anycast address

```
# /sbin/route -A inet6 add :2000::/3 gw :: 192.88.99.1 dev sitO
```

8.4.2. Remove a 6to4 tunnel

8.4.2.1. Using "ip" and a dedicated tunnel device# /sbin/ip tunnel del tun6to4

8.4.2.2. Using "ifconfig" and "route" and generic tunnel device "sitO" (deprecated)

Remove (default) route through the 6to4 tunnel interface

```
# /sbin/route -A inet6 del 2000::/3 gw ::192.88.99.1 dev sitO
```

Remove local 6to4 address to interface

```
# /sbin/ifconfig sitO del <local6to4address>/16
```

Shut down generic tunnel device (take care about this, perhaps it's still in use...)

```
# /sbin/ifconfig sitO down
```

Remove all routes through this dedicated tunnel device

```
# /sbin/ip -6 route flush dev tun6to4
```

Shut down interface

```
# /sbin/ip link set dev tun6to4 down
```

Remove created tunnel device

```
# /sbin/ip tunnel del tun6to4
```

8.4.2.2. Using "ifconfig" and "route" and generic tunnel device "sit0" (deprecated)

Remove (default) route through the 6to4 tunnel interface

```
# /sbin/route -A inet6 del 2000::/3 gw ::192.88.99.1 dev sitO
```

Remove local 6to4 address to interface

```
# /sbin/ifconfig sitO del <local6to4address>/16
```

Shut down generic tunnel device (take care about this, perhaps it's still in use...)

```
# /sbin/ifconfig sitO down
```

Chapter 9.

Network debugging

9.1. Server socket binding

9.1.1. Using "netstat" for server socket binding check

It's always interesting which server sockets are currently active on a node. Using "netstat" is a short way to get such information; Used options: -n1ptu

Example:

```
# netstat –n1ptu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
┐ PID/Proqram name
tcp 0 0 0.0.0.0:32768 0.0.0.0:* LISTEN
┐ 1258/rpc.statd
tcp 0 0 0.0.0.0:32769 0.0.0.0:* LISTEN
┐ 1502/rpc.mountd
tcp 0 0 0.0.0.0:515 0.0.0.0:* LISTEN
┐ 22433/lpd Waiting
tcp 0 0 1.2.3.1:139 0.0.0.0:* LISTEN
┐ 1746/smbd
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN
┐ 1230/portmap
tcp 0 0 0.0.0.0:6000 0.0.0.0:* LISTEN
┐ 3551/X
tcp 0 0 1.2.3.1:8081 0.0.0.0:* LISTEN
┐ 18735/junkbuster
tcp 0 0 1.2.3.1:3128 0.0.0.0:* LISTEN
┐ 18822/(squid)
tcp 0 0 127.0.0.1:953 0.0.0,0:* LISTEN
┐ 30734/named
tcp 0 0 ::ffff:l.2.3.1:993 :::* LISTEN
┐ 6742/xinetd-ipv6
tcp 0 0 :::13 :::* LISTEN
┐ 6742/xinetd-ipv6
```

```
tcp 0 0 ::fffc:1.2.3.1:143 :::* LISTEN
┐ 6742/xinetd-ipv6
tcp 0 0 :::53 :::* LISTEN
┐ 30734/named
tcp 0 0 :::22 :::* LISTEN
┐ 1410/sshd
tcp 0 0 :::6010 :::* LISTEN
┐ 13237/sshd
udp 0 0 0.0.0.0:32768 0.0.0.0:*
┐ 1258/rpc.statd
udp  0 0 0.0.0.0:2049 O.0.O.0:*
┐ -
udp 0 0 0.0.0.0:32770 0.0.0.0:*
┐ 1502/rpc.mountd
udp O 0 0.0.0.0:32771 0.0.0.0:*
┐ -
udp 0 0 1.2.3.1:137 0.0.0.0:*
┐ 1751/nmbd
udp  0 0.0.0.0:137 0.0.0.0:*
┐ 1751/nmbd
udp 0 0 1.2.3.1:138 O.O.O.O:*
┐ 1751/nmbd
udp 0 0 0.0.0.0:138 0.0.0.0:*
┐ 1751/nmbd
udp O 0 0.0.0.0:33044 0.0.0.0:*
┐ 30734/named
udp 0 0 1.2.3.1:53 0.0.0.0:*
┐ 30734/naiiied
udp 0 0 127.0.0.1:53 0.0.0.0:*
┐ 30734/named
udp O 0 0.0.0.0:67 0.0.0.0:*
┐ 1530/dhcpd
udp 0 0 0.0.0.0:67 0-0.0.0:*
┐ 1530/dhcpd
udp 0 0 0.0.0.0:32859 0.0.0.0:*
┐ 18822/ (squid)
udp 0 0 0.0.0.0:4827 0.0.0.0:*
┐ 18822/ (squid)
udp 0 0 0.0.0.0:111 0.0.0.0:*
┐ 1230/portmap
```

```
udp 0 0 :::53 :::*
┐ 30734/named
```

9.2 Examples for tcpdump packet dumps

Here some examples of captured packets are shown, perhaps useful for your own debugging more coming next.

9.2.1. Router discovery

9.2.1.1. Router advertisement

```
1
5:43:49.484751 fe80::212:34ff:fel2:3450 > ff02::l: icmp6: router
┐ advertisement(chlim=64, router_ltime=30, reachable_time=0,
┐ retrans_time=0)(prefix info: AR valid ltime=30, preffered_ltime=20,
┐ prefix=2002:0102:0304:l::/64) (prefix info: LAR valid_ltime=2592000,
┐ preffered_ltime=604800, prefix=3ffe:ffff:0:l::/64)(src lladdr:
┐ 0:12:34:12:34:50) (len 88, hlim 255)
```

Router with link-local address "fe80::212:34ff:fel2:3450" send an advertisement to the all-node-on-link multicast address "ff02::l" containing two prefixes "2002:0102:0304:1::/64" (lifetime 30 s) and "3ffe:fflF:0:l::/64" (lifetime 2592000 s) including its own layer 2 MAC address "0:12:34:12:34:50"

9.2.1.2. Router solicitation

```
15:44:21.152646 fe80::212:34ff:fel2:3456 > ff02::2: icmp6: router solicitation
┐ (src lladdr: 0:12:34:12:34:56) (len 16, hlira 255)
```

Node with link-local address "fe80::212:34ff:fel2:3456" and layer 2 MAC address "0:12:34:12:34:56"is looking for a router on-link,therefore sending this solicitation to the all-router-on-link multicast address "ff02::2".

9.2.2. Neighbor discovery

9.2.2.1. Neighbor discovery solicitation for duplicate address detection

following packets are sent by a node with layer 2 MAC address "0:12:34:12:34:56" during autoconfiguration to check whether a potential address is already used by another node on the link sending tills to the solicited-node link-local multicast address Node wants to configure its link-local address "fe80::212:34ff:fel2:3456", checks for duplicate now

```
15:44:17.712338 :: > ff02::1:ffl2:3456: icmp6: neighbor sol: who has
┐ fe80::212.34ff:fe12:3456(src lladdr: 0:12:34:12:34:50) (len 32, hlim 255)
```

Node wants to configure its global address "3ffe:ffff:0:1:212:34ff:fel2:3456" (after receiving advertisement shown above),checks for duplicate now

```
15:44:22.304028 :: > ff02::1:ffl2:3456: icmp6: neighbor sol: who has
┐ 3ffe:ffff:0:1:212:34ff:fel2:3456(src  lladdr: 0 : 12: 34 : 12: 34 : 56) (len 32, hlim 255)
```

Node wants to configure its global address "3ffe:ffff:0:l:212:34ff:fel2:3456" (after receiving advertisement shown above), checks for duplicate now

```
15 :44:22.304028 :: > ff02::l:ffl2:3456: icmp6: neighbor sol: who has
┐ 3ffe:ffff:0:l:212:34ff:fel2:3456(src lladdr: 0:12:34:12:34:56) (len 32, hlim
255)
```

9.2.2.2. Neighbor discovery solicitation for looking for host or gateway

Note wants to send packages to "3ffe;ffff:0:1::10" but has no layer 2 MAC address to send packet, so send solicitation now

```
13:07:47.664538 2002:0102:0304:1:2e0:18ff:fe90:9205 > ff02::l:ff00:10: icmp6:
┐ neighbor sol: who has 3ffe:ffff:0:1::10(src lladdr: 0:e0:18:90:92:5) (len 32,
┐ hlim 255)
```

Node looks for "fe80::10" now •

```
13:11:20.870070 fe8O::2e0:l8ff:fe90:9205 > ff02::l:ff00:10: icmp6: neighbor
┐ sol: who has fe80:: 10 (src lladdr: 0:e0: 18: 90: 92: 5) (len 32, hlim 255)
```

Chapter 10

Support for persistent IPv6 Configuration in Linux distributions

Some Linux distribution contain already support of a persistent IPv6 configuration using existing or new configuration and script files and some hook in the IPv4 script files.

10.1. Red Hat Linux and "clones"

To enable a persistent IPv6 configuration which catch most of the wished cases like host-only, router-only, dual-homed-host, router with second stub network, normal tunnels, 6to4 tunnels and so on.Nowadays there exists a set of configuration and script files which do the job very well (never heard about real problems, but we don't know how many use the set. Because this configuration and scrips files are extended from time to time, they got their own HOWTO page: IPv6-HOWTO/scripts/current.Because we are using IPv6 experience using a Red
Hat Linux 5.0 clone, my IPv6 development systems are mostly Red Hat Linux based now it's kind a logic that the scripts are developed for this kind of distribution (so called historic issue). Also it was very easy to extend some configuration files, create new ones and create some simple hook for calling IPv6 setup during IPv4 setup.
Fortunately, in Red Hat Linux since 7.1 a snapshot of IPv6 scripts is included, this was and is still
Further Mandrake since version 8.0 also includes an IPv6-enabled initscript package, but a minor bug still prevents usage ("ifconfig" misses "inet6" before "add"),

10.1.1. Test for IPv6 support of network configuration scripts

You can test, whether your Linux distribution contain support for persistent IPv6 configuration using my set Following script library should exist:

/etc/sysconfig/network-scripts/network-functions-ipv6

Auto-magically test:

```
# test -f /etc/sysconfig/network-scripts/network-functions-ipv6 && echo "Main.
IPv6 script libary exists"
```

The version of the library is important if you miss some features. You can get it executing following(or easier look at the top of the file):

```
# source/etc/sysconfig/network-scripts/network-functions-ipv6 &&
getversion_ipv6_functions
20011124
```

In shown example, the used version is 20011124. Check this against latest information on IPv6-HOWTO/scripts/current to see what has been changed. There is also a change-log available in the distributed tar-ball.

10.1.2. Short hint for enabling IPv6 on current RHL 7.1,7.2,...

Check whether running system has already IPv6 module loaded

```
# modprobe -c | grep net-pf-10
alias net-pf-10 off
```

If result is "off",then enable IPv6 networking by editing/etc/sysconfig/network, add following new line.

```
NETWORKING_IPV6=yes
```

Reboot or restart networking using

```
# service network restart
```

Now IPv6 module should be loaded

```
# modprobe -c |  grep ipv6
alias net-pf-10 ipv6
```

If your system is on a link which provides router advertisement, autoconfiguration will be done automatically. For more information which settings are supported see /usr/share/doc/initscripts-$version/sysconfig.txt.

10.2.SuSELinux

In newer versions there is a really rudimentary support available, see /etc/rc.config for details Because of the really different configuration and script file structure it is hard (or impossible) to use the set for Red Hat Linux and clones with this distribution.

Chapter 11

Firewalling and security issues

IPv6 firewalling is important, especially if using IPv6 on internal networks with global IPv6 addresses. Because unlike at IPv4 networks where in common internal hosts are protected automatically using private IPv4 addresses like RFC 1918 / Address Allocation for Private Internets or APIPA / Automatic Private IP Addressing, in IPv6 normally global addresses are used and someone with IPv6 connectivity can reach all internal IPv6 enabled nodes.

11.1.Firewalling

11.1.1.Firewalling using netfilter6

Native IPv6 firewalling is only supported in kernel versions 2.4+. In older 2.2- you can only filter IPv6 -in-IPv4 by protocol 41. Attention: no warranty that described rules or examples are really protect your system!

11.1.1.1.More information

Netfilter project

maillist archive of netfilter users

maillist archive of netfilter developers

Unofficial status informations

11.1.2.Preparation

11.1.2.1.Extract sources

Change to source directory:

```
# cd /path/to/src
```

Unpack and rename kernel sources

```
# tar z|jxf kernel-version.tar.gz|bz2
# mv linux linux-version-iptables-version+IPv6
```

Unpack iptables sources

```
# tar z|jxf iptables-version.tar.gz|bz2
```

11.1.2.2.Apply latest iptables/IPv6-related patches to kernel source

Change to iptables directory

```
# cd iptables.version
```

Apply pending patches

```
# make pending-patches KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Apply additional IPv6 related patches (still not in the vanilla kernel included)

```
# make patch-o-matic KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Say yes at following options (iptables-1.2.2)

ah-esp.patch

masq-dynaddr.patch(only needed for systems with dynamic IP assigned WAN connections like PPP or PPPoE)

```
# make print-extensions
```

Extensions found: IPv6:owner IPv6:Limit IPv6:mac IPv6:multiport

11.1.2.3.Configure, build and install new kernel

Change to kernel sources

```
# cd /path/to/src/linux-version-iptables-version/
```

Edit Makefile

```
- EXTRAVERSION =
+ EXTRAVERSION = -iptables-version+IPv6-try
```

Run configure enable IPv6 related

```
Code maturity level options
Prompt for development and/or incomplete code/drivers : yes
Networking options
Network packet filtering: yes
The IPv6 protocol: module
IPv6: Netfilter Configuration
IPV6 tables support: module
All new options like following:
limit match support: module
MAC address match support: module
Multiple port match support: module
Owner match support: module
```

netfilter MARK match support: module

Aggreigated address check: module

Packet filtering: module

REJECT target support: module

LOG target support: module

Packet mangling: module

MARK target support: module

Configure other related to your system, too

Compilation and installing: see the kernel section here and other HOWTOs

11.1.2.4.Rebuild and install binaries of iptables

Make sure, that upper kernel source tree is also available at/usr/src/linux/

Rename older directory

```
# mv/usr/src/linux/usr/src/linux.old
```

Create a new softlink

```
# ln /path/to/src/linux-version-iptables-version/usr/src/linux
```

Rebuild SRPMS

```
# rpm --rebuild /path/to/SRPMS/iptables-version-release.src.rpm
```

Install new iptables packages (iptables + iptables-ipv6)

On RH 7.1 systems, normally, already an older version is installed, therefore use "freshen"

```
# rpm -Fhv / path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

If not already installed,use "install"

```
# rpm -ihv /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

On RH 6.2 systems, normally, no kernel 2.4.x is installed therefore the requirements don't fit. Use "—nodeps" to install it

```
# rpm -ihv --nodep /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

Perhaps it's necessary to create a softlink for iptables libraries where iptables looks for them

```
# ln -s /lib/iptables/ /usr/lib/iptables
```

11.1.3.Usage

11.1.3.1. Check for support

Load module, if so compiled

```
# modprobe ip6_tables
```

Check for capability

```
# [ ! -f /proc/net/ip6 tables names ] && echo "Current kernel doesn't support
ipbtables' firewailing (IPv6)!"
```

11.1.3.2.Learn how to use ip6tables

List all IPv6 netfilter entries

Short

```
# ip6tables -L
```

Extended

```
# ip6tables -n -v --line-numbers -L
```

List specified filter

```
#Ip6tables –n –v –line-number –L INPUT
```

Insert a log rule at the input filter with options

```
# ip6tables --table FiLter --append INPUT -j LOG --log-prefix "INPUT:"
--log-Level 7
```

Insert a drop rule at the input filter

```
# Ip6tables --table filter --append INPUT -j DROP
```

Delete a rule by number

```
# ip6tables --table filter --delete INPUT 1
```

Allow ICMPv6,at the moment with unpatched kernel 2.4.5 and iptables-1.2.2 no type can be specified Accept incoming ICMPv6 through tunnels

```
# ip6tables -A INPUT -i sit+ -p icmpv6 -j ACCEPT
```

Allow outgoing ICMPv6 through tunnels

```
# ip6tables -A OUTPUT -o sit+ -p icmpv6 -j ACCEPT
```

Allow incoming SSH, here an example is shown for a ruleset which allows incoming SSH connection from a specified IPv6 address Allow incoming SSH from 3ffe:400:100::l/128

```
# ip6tables -A INPUT -i sit+ -p tcp -s 3ffe:400:100::1/128 --sport 512:65535
 --dport 22 -j ACCEPT
```

Allow response packets (at the moment IPv6 connection tracking isn't in mainstream netfilter6 implemented)

```
# iptitables -A OUTPUT -o sit+ -p tcp -d 3ffe:400:100::1/128 dport 512:65535
--sport 22 ! --syn j ACCEPT
```

Enable tunneled IPv6-in-IPv4, to accept tunneled IPv6-in-IPv4 packets, you have to insert rules in your IPv4 firewall setup relating to such packets, for example Accept incoming IPv6-in-IPv4 on interface pppO

```
# iptables -A INPUT -i pppO -p ipv6 -j ACCEPT
```

Allow outgoing IPv6-in-IPv4 to interface ppp

```
# iptables -A OUTPUT -o pppO -p Ipv6 -j ACCEPT
```

If you have only a static tunnel, you can specify the IPv4 addresses, too, like

Accept incoming IPv6-in-IPv4 on interface pppO from tunnel endpoint 1.2.3.4

```
# iptables -A INPUT -i pppO -p ipv6 -s 1.2.3.4 -j ACCEPT
```

Allow outgoing IPv6-in-IPv4 to interface pppO to tunnel endpoint 1.2.3.4

```
# iptables -A OUTPUT -o pppO -p ipv6 -d 1.2.3.4 -j ACCEPT
```

Protect against incoming TCP connection requests (VERY RECOMMENDED*-), for security issues you should really insert a rule which blocks incoming TCP connection requests. Adapt "-i" option, if other interface names are in use!

Block incoming TCP connection requests to this host

```
# ip6tables -I INPUT -i sit+ -p tcp --syn -j DROP
```

Block incoming TCP connection requests to hosts behind this router

```
# ip6tables -I FORWARD -i sit+ -p tcp --syn -j DROP
```

Perhaps the rules have to be placed below others, but that is work you have to think about it.Best way is to create a script and execute rules in a specified way. Protect against incoming UDP connection requests (ALSO RECOMMENDED!), like mentioned on my firewall information it's possible to control the ports on outgoing UDP/TCP sessions. So if all of your local IPv6 systems are use local ports e.g. from 32768 to 60999 you are able to filter UDP connections also

(until connection tracking works) like; Block incoming UDP packets which cannot be responses of outgoing requests of this host

```
# ip6tables -I INPUT -i sit+ -p udp ! --dport 32768:60999 -j DROP
```

Block incoming UDP packets which cannot be responses of forwarded requests of hosts behind this router

```
Ip6tables -I FORWARD -i. sit+ -p udp ! --dport 32768:60999 -j DROP
```

11.1.3.3.Demonstration example

Following lines show a more sophisticated setup as an example. Happy netfilter6 ruleset creation….

```
# ip6tables -n -v -L
Chain INPUT (policy DROP 0 packets,0 bytes)
pkts bytes target prot opt in out source destination
0 0 extIN all sit+ * ::/0 ::/0
4 384 intIN all ethO * ::/0 ::/0
0 0 ACCEPT all * * ::1/128 ::1/128
0 0 ACCEPT all lo * ::/0 ::/0
0 0 LOG all * * ::/0 ::/0
log flags 0 level 7 prefix INPUT-default:
0 0 DROP all * * ::/0 ::/0
chain FORWARD(policy DROP 0 packets,0 bytes)
pkts bytes target prot opt in out source destination
0 0 int2ext all ethO sit+ ::/0 ::/0
0 0 ext2int all sit+ ethO ::/0 ::/0
0 0 LOG all * * ::/0 ::/0
LOG flags 0 level 7 prefix 'FORNARD-default:'
0 0 DROP all * * : :/0 : :/0
Chain OUTPUT (policy DROP 0 packets,0 bytes)
pkts  bytes target prot opt in out source destination
O 0 extOUT all * sit+ ::/0 ::/0
4 384 intOUT all * ethO ::/0 ::/0
0 0 ACCEPT all * * :: 1/128 :: 1/128
0 0 ACCEPT all * lo ::/0 ::/0
0 0 LOG all * * ::/0 ::/0
Log flags 0 level 7 prefix 'OUTPUT-default: '
0 0 DROP all * * ::/0 ::/0
Chain ext2int (1 references)
pkt bytes target prot opt in out source destination
0 0 ACCEPT icmpv6 * * ::/0 ::/0
0 0 ACCEPT tcp * * ::/0 ::/0
tcp spts:l:65535 dpts:1024:65535 flags: !0x16/0x02
0 0 LOG all * * ::/0 ::/0
L0G flags 0 level 7 prefix 'ext2int-default: '
0 0 DROP tcp * * ::/0 ::/0
0 0  DROP udp * * ::/0 ::/0
0 0 DROP all * * ::/0 ::/0
Chain extIN (1 references)
pks bytes target prot opt in out source destination
0 0 ACCEPT tcp * * 3ffe:400:100::1/128::/0
tcp spts:512:65535 dpt:22
```

```
0 0 ACCEPT tcp * * 3ffe:400:100::/128 ::/O
tcp spbs:512:65535 dpt:22
0 0 ACCEPT icmpv6 * * ::/O ::/0
0 0 ACCEPT tcp * * ::/0 ::/0
tcp spts:1:65535 dpts:1024:65535 flags:!0x16/0x02
0 0 ACCEPT udp * * ::/0 ::/0
udp spts:l:65535 dpts:1024:65535
0 0 LOG all * ^ ::/0 ::/0
limit: avg 5/min burst 5 LOG flags 0 level 7 prefix 'extIN-default: '
0 0 DROP all * * ::/0 ::/0
chain extOUT (1 references)
pkt bytes target prot opt in out source destination
0 0 ACCEPT top * * ::/0
3ffe:400:100::l/12Stcp spt:22 dpts:512:65535 flags:!0x16/0x02
0 0 ACCEPT tcp * * ::/0
3ffe:400:100::1/128tcp spt:22 dpts:512:65535 flags:! 0x16/0x0
0 0 ACCEPT icmpv6 * * ::/0 ::/0
0 0 ACCEPT tcp * * ::/0 ::/0
tcp spts:1024:65535 dpts:1:65535
0 0 ACCEPT udp * * ::/0 ::/0
udp spts:1024:65535 dpts:l:65535
0 0 LOG all * *::/0 ::/0
LOG flags 0 level 7 prefix extOUT-default:
0 0 DROP all * * ::/0 ::/0
Chain int2ext (1 references)
pkts bytes target prot opt in out source destination
0 0 ACCEPT icmpv6 * * ::/0 ::/0
0 0 ACCEPT tcp * * ::/0 ::/0
tcp spts:1024:65535 dpts:l:65535
0 0 LOG all * * ::/O ::/O
LOG flags 0 level 7 prefix 'int2ext:'
0 0 DROP all * *  ::/0 ::/0
0 0 LOG all * * ::/O ::/O
LOG flags 0 level 7 prefix 'int2ext-dafauit'
0 0 DROP tcp * * ::/0 ::/0
0 0 DROP udp * * ::O ::/0
0 0 DROP all * * ::/0 ::/0
Chain intIN (1 references)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all * * ::/0
```

```
fe80::/ffc0::
4 384 ACCEPT all * * ::/0 ff02::/16
Chain intOUT (1 references)
pkts bytes target prot opt in out source destination
0 0  ACCEPT all * * ::/0
fe80::/ffc0::
4 384 ACCEPT all * * ::/0 ff02::/16
0 0 LOG all * * ::/0 ::/0
LOG flags 0 level 7 prefix 'intOUT-default:'
0 0 DROP all * * ::/0 ::/0
```

Chapter 12

Encryption and Authentication

Unlike in IPv4 encryption and authentication is a mandatory feature of IPv6. This features are normally implemented using IPsec (which can be also used by IPv4)But because of the independence of encryption and authentication from the key exchange protocol here exists currently some interoperability problems regarding this issue.

12.1.Support in kernel

12.1.1.Support in vanilla Linux kernel

There is an issue about keeping the Linux kernel source free ofexport/import-control-laws regarding encryption code. This is also one case why FreeS/WAN project (IPv4 only IPsec) isn't still contained in vanilla source.

12.1.2.Support in USAGI kernel

The USAGI project has taken over in July 2001 the IPv6 enabled FreeS/WAN code from the IABG /IPv6 Project and included in their kernel extensions, but still work in progress, means that not all IABG features are already working in USAGI extension.

13.1.1.2.Disable BIND named for listening on IPv6 address

To disable IPv6 for listening, following options are requested to change

```
options {
# sure other options here, too
listen-on-v6 { none; };
};
```

13.1.2.IPv6 enabled Access Control Lists (ACL)

IPv6 enabled ACLs are possible and should be used whenever it's possible. An example looks like following:

```
acl internal-net {
127.0.0.1;
1.2.3.0/24;
3ffe:ffff:lOO::/56;
::1/128 ;
::ffff:l.2.3.4/123;
};
acl ns-internal-net {
1.2.3.4;
l.2.3.5;
3ffe:ffff:100::4/128;
3ffe:ffff:100::5/128;
};
```

This ACLs can be used e.g. for queries of clients and transfer zones to secondary name-servers. This prevents also your caching name-server to be used "from outside using IPv6.

```
options {
# sure other options here, too
listen-on-v6 { none; } ;
allow-query [ internal-net; };
allow-transfer { ns-internal-net; } ;
```

It's also possible to set the allow-query and allow-transfer option for most of single zone definitions too.

13.1.3. Sending queries with dedicated IPv6 address

This option is not required but perhaps needed:

query-source-v6 address <ipv6address |*> port <port |*>;

13.1.4. Per zone defined dedicated IPv6 addresses

It's also possible to define per zone some IPv6 addresses.IPv6 enabled Access Control Lists (ACL) 6917.1.4.1. Transfer source address Transfer source address is used for outgoing zone transfers:

```
transfer-source-v6 <ipv6addr|*> [port port ] ;
```

13.1.4.2. Notify source address

Notify source address is used for outgoing notify messages:

```
notiy-source-v6 <ipv6addr |*> [port port];
```

13.1.5. Serving IPv6 related DNS data

For IPv6 new types and root zones for reverse lookups are defined:

AAAA and reverse IP6.INT: specified in RFC 18S6 / DNS Extensions to support IP version 6, usable since BIND version 4.9.6

A6, DNAME and reverse IP6.ARPA; specified in RFC 2874/DNS Extensions to Support IPv6 Address Aggregation and Renumbering usable since BIND 9, but see also an information about die current state at draft-ietf-dnsext-ipv6-addresses-00.txt

Perhaps filled later more content, for the meantime take a look at given RFCs and AAAA and reverse IP6.INT: IPv6 DNS Setup Information A6 DNAME and reverse IP6.ARPA: take a look into chapter 4 and 6 of the BIND 9 Administrator Reference Manual (ARM) distributed which die hind-package or get this here:BIND version 9 ARM (PDF) Because IP6.INT is deprecated (but still in use)a DNS server which will support IPv6 information has to serve both reverse zones.

13.1.5.1.Current best practice

Because there are some troubles around using die new formats current best practice is;

Forward lookup support:

AAAA

A6 without chaining, means prefix length value set to 0

Reverse lookup support:

Reverse nibble format for zone ip6.int

Reverse nibble format for zone ip6.arpa

Per zone defined dedicated IPv6 addresses 70

13.1.6.Checking IPv6-enabled connect

To check whether BIND is listening on an IPv6 socket and serving data see following examples.

13.1.6.1. IPv6 connect but denied by ACL

Specifying a dedicated server for the query an IPv6 connect can be forced:

```
$ host -t aaaa www.6bone.net 3ffe:ffff:200: flOl::1
Using domain server:
Name:3ffe:ffff:200:fl01::l
Address:3ffe:ffff:200:fl01::1#53
A1iases:
Host www.6bone.net.not found: 5(REFUSED)
```

Related log entry looks like following:

```
Jan 3 12:43:32 gate named[12347]: client
3ffe:ffff:200:f101:212:34ff:fel2:3156#32770:
query denied
```

If you see such entries in die log check whether requests from this client should be allowed and perhaps review your ACL configuration.

13.1.6.2. Successful IPv6 connect

A successful IPv6 connect looks like following:

```
$ host -t aaaa www.6bone.net 3ffer:ffff:200:flOl::1
using domain server:
Name: 3ffe:ffff:200:fl01::1
Address: 3ffe:ffff:200:fl01::1#53
Aliases:
www.6bone.net. is an alias for 6bone.net.
6bone.net.has AAAA address 3ffe:b00: cl8:1::10
```

13.2.Internet super daemon (xinetd)

IPv6 is supported since version around 1.8.9. Always use newest available version. At least version 2.3.3 must be used, older versions can contain remote exploitable security holes.Some Linux distribution contain an extra package for the IPv6 enabled xinetd, some others start the IPv6-enabled xinetd if following variable is set: NETWORKING_IPV6="yes", mostly done by /etc/sysconfig/network (only valid for Red Hat like distributions).

If you enable a built-in service like e.g. daytime by modifying the configuration file in /etc/xinetd.d/daytime

like

```
# diff -u /etc/xinetd.d/daytime.orig/etc/xinetd.d/daytime
--- /etc/xinetd.d/daytiine.orig Sun Dec 16 19:00:14 2003
+++ /etc/xinetd.d/daytime Sun Dec 16 19:00:22 2003
@@ -10,5 +10,5 @@
protocol = tcp
user =root:
wait = no
-disable = yes
+disable = no
}
```

After restarting the xinetd you should get a positive result like:

```
# netstat -lnptu -A inet6 |grep "xinetd*"
tcp 0 0 ::ffff:192.168.1.1:993 :::* LISTEN 12345/xinetd-ipv6
tcp 0 0 :::13 :::* LISTEN 12345/xinetd-ipv6 <- service
daytime/tcp
tcp 0 0 ::ffff:192.168.1.1:143 :::* LISTEN 123-35/xinetd-ipvt6
```

Shown example also displays an IMAP and IMAP-SSL IPv4-only listening xinetd.

Note: An IPv4-only xinetd won't start on an IPv6-enabled node and also the IPv6-enabled won't start on an IPv4-only node (will be hopefully fixed in the future).

13.3.Listening on IPv6 addresses

Note: virtual hosts on IPv6 addresses are broken in versions until 2.0.28 (a patch is available for 2.0.28).

13.3.1.Virtual host listen on an IPv6 address only

```
Listen [3ffe:ffff:100::1]:80
<VirtualHost [3ffe:ffff:100::1]:80>
serverName ipv6only.yourdomain.yourtopleveldomain
# ...sure more config lines
<VirtualHost>
```

13.3.1.1.Virtual host listen on an IPv6 and on an IPv4 address

```
Listen [3ffe:ffff:100::2]:80
Listen 1.2.3.4:80
<virtualHost [3ffe:ffff:100::2] :8O 1.2.3.4:80>
serverName ipv6andipv4.yourdomain.yourtopleveldomain
# ...sure more config lines
</virtualHost>
```

This should result after restart in e.g.

```
# netstat -1nptu |grep "httpd2\W*$"
tcp 0 0 1.2.3.4:80 0.0.0.0:* LISTEN 12345/httpd2
tcp 0 0 3ffe:ffff:100::1:80 :::* LISTEN 12345/httpd2
tcp 0 0 3ffe:ffff:100::2:8O :::* LISTEN 12345/httpd2
```

For simple tests use the telnet example already shown.

13.4.Router Advertisement Daemon (radvd)

The router advertisement daemon is very useful on a LAN, if clients should be auto-configured. The daemon itself should run a Linux router (not necessary the default IPv4 gateway).You can specify some information and flags which should be contained in the advertisement.

Common used are Prefix (needed) Lifetime of the prefix

Frequency of sending advertisements (optional)

After a proper configuration the daemon sends advertisements through specified interfaces and clients are hopefully receive them and auto-magically configure addresses with received prefix and the default route.

13.4.1.Configuring radvd

13.4.1.1. Simple configuration

Radvd's config file is normally /etc/radvd.conf. An simple example looks like following:

```
interface ethO {
AdvsendAdvert on ;
MinRtrAdvInterval 3;
MaxRtrAdvInterval 10;
prefix 3ffe:ffff:nl00:fl01::/64 {
AdvOnLink on;
AdvAutonomous on;
AdvRouterAddr on;
};
};
```

This results on client side in

```
# ip -6 addr show eth0
3:eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
inet6 3ffe:ffff:100:fl01:2e0:12ff:fe34:1234/64 scope global dynamic
valid_lft 2591992sec preferred_lft 604792sec
inet6 fe80::2e0:12ff:fe34:1234/10 scope link
```

Because no lifetime was defined a very high value was used.

13.4.1.2.Special 6to4 configuration

Version since 0.6.2pl3 support the automatic (re)-generation of the prefix depending on an IPv4 address of a specified interface. This can be used to distribute advertisements in a LAN after the 6to4 tunneling has changed. Mostly used behind a dynamic dial-on-demand Linux router. Because of the sure shorter lifetime of such prefix (after each dial-up, another prefix is valid), the lifetime configured to minimal values:

```
interface eth0 [
AdvsendAdvert on ;
MinrtrAdv Interval 3;
MaxrtrAdvInterval 10;
prefix 0:0:0:flOl::/64 {
advonlink off;
advautonomous on ;
advrouterAddr on;
base6to4Interface pppO;
Advpreferredlifetime 20 ;
AdvValidLifetime 30;
};
};
```

This results on client side in (assuming, pppO has currently 1.2.3.4 as local IPv4 address):

```
# ip -6 addr show ethO
3: ethO: OROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
inet6 2002:0102:0304:fl01:2e0:12ff:fe34:1234/64 scope global dynamic
valid_lft 22sec preferred_lft 12 sec
```

```
inet6 fe80::2e:l2e:fe4:1234/10 scope Link
```

Because a small lifetime was defined, such prefix will be thrown away quickly, if no related advertisement was received.

13.4.2.Debugging

A program called "radvdump" can help you looking into sent or received advertisements- Simple to use:

```
# radvdump
Router advertisement from fe80::280:c8ff:feb9:cef9 (hoplimit 255)
AdvCurHopLimit: 64
AdvManaqedFlaq: off
AdvOtherConfigFlag: off
AdvHomeAgent Flag: off
AdvReachableTime: 0
AdvRetransTimer: 0
Prefix 2002:0102:0304:fl01::/64
AdvValidLifetime: 30
AdvPreferredLifetime: 20
AdvonLink: off
AdvAutonomous: on
AdvRouterAddr: on
Prefix 3ffe:ffff:100:fl01::/64
AdvValidLifetime: 2592000
AdvPreferredLifetime: 604300
AdvonLink: on
AdvAatonomous: on
AdvRouterAddr: on
AdvSourceLLAddress: 00 80 12 34 56 78
```

Output shows you each advertisement package in readable format. You should see your configured values here again, if not, perhaps it's not your radvd which sends the advertisement...look for another router on the link (and take die LLAddress, which is the MAC address for tracing).

13.5.tcp_wrapper

tcp_mapper is a library which can help you to protect service against misuse.

13.5.1.Filtering capabilities

you can use tcp_wrapper for Filtering against source addresses (IPv4 or IPv6)

Filtering against users (requires a running ident daemon on the client)

13.5.2.Which program uses tcp_wrapper

Following are known-Each service which is called by xinetd(if xinetd is compiled using tcp_wrapper library) sshd (if compiled using tcp_wrapper)

13.5.3.Usage

Tcp_wrapper is controlled by two files name /etc/hosts, allow and /etc/hosts-deny. For more information see

```
$ man hosts.all
```

13.5.3.1.Example for/etc/hosts.allow

In this file each service which should be positive filtered (means connects are accepted) need a line.

```
Sshd:1.2.3. [3ffe:ffff:100:200::]/64
daytime-stream: 1.2.3. [3ffe:ffff:100:200::]/64
```

13.5.3.2.Example for /etc/hosts.deny

This file contains all negative filter entries and should normally deny the rest using

```
ALL: ALL
```

If this node is a more sensible one you can replace the standard line above with this one, but this can cause a DoS attack (load of mailer and spool directory), if too many connects were made in short time.

Perhaps a logwatch is better for such issues.

```
ALL: ALL: spawn (echo "Attempt from %h %a to %d at 'date'"
| tee -a /var/log/tcp.deny.log | mail root@localhost)
```

13.5.4.Logging

Depending on the entry in the syslog daemon configuration file etc/syslog.conf the tcp_wrapper logs normally into /var/log/secure.

13.5.4.1.Refused connection

A refused connection via IPv4 to an xinetd covered daytime service produces a line like following

example

```
Jan 2 20:40:44 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
from=::ffff:1.2.3.4
Jan 2 20:32:O6 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
from=3ffe:ffff:100:200::212:34ff:fel2:3456
```

A refused connection via IPv4 to an dual-listen sshd produces a line like following

example

```
Jan 2 20:24:17 gate sshd[12345]: refused connect from (::ffff:1.2.3.4)
jan 2 20:39:33 qate sshd[123451: refused connect
form 3ffe:ffff:l00:200::212:34ff:fel2:3456
(3ffe:ffff:100:200::212:34ff:fel2:3456)
```

13.5.4.2.Permitted connection

A permitted connection via IPv4 to an xinetd covered daytime service produces a line like following

example

```
Jan 2 20:37:50 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
from=::ffff:1.2.3.4
jan 2 20:37:56 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
from=3ffe:ffff:100:200::212:34ff:fel2:3456
```

A permitted connection via IPv4 to an dual-listen sshd produces a line like following

example

```
Jan 2 20:43:10 gate sshd[21975]: Accepted password for user from ::ffff:1.2.3.4
port 33391 ssh2
Jan 2 20:42:19 gate sshd[12345]: Accepted password for user
from 3ffe:ffff:100:200::212:34ff:fel2:3456 port 33380 ssh2.
```

CONCLUSION

The field of networking is as old as Computer, and to do networking we use various types of protocols to configure IP addresses . Presently what we use to configure IP addresses is IPv4 Addressing type which is in use from the begging of the computers network age. As IPv4 supports 32 bit addressing scheme, which is getting old and out of space because of the expenditure of internet users. To allow more addressing space to the internet the new and better version of IPv4 is the IPv6(Internet protocol version6) which is supporting 128 bits addressing schemes. So as IPv6 is a new version of IPv4 so a was to field of new techno1logies are supporting IPv6 .

REFERENCES

Articles, Books, Online Reviews (mixed)

Getting Connected with 6to4 by Huber Feyrer,06/01/2001

how Long the Aversion to IP Version 6 - Review of META Group, Inc., full access needs (free)registration at META Group, Inc.

O'reilly Network search for keyword IPv6 results in 29 hits (28. January 2002)

Wireless boosting IPv6 by Carolyn Duffy Marsan, 10/23/2000

IPv6, theorie et pratique (french) 2e edition, mars 1999, O'Reilly (??? no newer one available ???)

ISBN: 2-84177-085-0

Intenetworking IPv6 with Cisco Routers by Silvano Gai, McGrawHill Italia, 1997 13 chapters and appendix A-D are downloadable as PDF-documents.

Secure and Dynamic Tunnel Broker by Vegar Skaerven Wang, Master of Engineering Thesis in Computer Science, 2.June 2000, Faculty of Science, Dep.of Computer Science, University of Tromso, Norway.

Aufbruch in die neue Welt - IPv6 in IPv4 Netzen von Dipl.Ing. RalfDoring, TU Illmenau, 1999

Migration and Co-existence of IPv4 and IPv6 in Residential Networks by Pekka Savola,CSC/FUNET, 2002

Book IPv6 Essentials written by Silvia Hagen, release planned for April 2002

Others

See following URL for more: SWITCH IPv6 Pilot / References

WWW.IPV6.COM

WWW.FAQIPv6.COM

Ipv6-net.org,

WWW.CISCO.COM

WWW.LINUX.COM

Major regional registries

America: AR1N,ARIN / registration page, ARFN / IPv6 guidelines

EMEA: Ripe NCC, Ripe NCC / registration page. Ripe NCC / IPv6 registration

Asia/Pacific: APN1C, APNIC / IPv6 informai.ion

Latin America and Caribbea: LACNIC

Affria: *AfriNIC*

Also a list of major (prefix length 35) allocations per local registry is available here:

Ripe NCC /

IPv6

allocations.

Tunnel brokers

Freenet6,Canada

Hurricane Electric, US backbone

Centro Studi e Laboratory Telecomunicazioni, Italy

Wanadoo, Belgium

CERTNET-Nokia, China

Tunnelbroker Leipzig, Germany –Dialup Users with dynamic IP's can get a fix IPv6 IP...

Internet Initiative Japan, Japan - with IPv6 native line service and IPv6 tunneling Service

XS26 - "Access to Six", Netherland - with POPs in Slovak Republic, Czech Republic, Netherlands,

Germany and Hungary.

IPng Netherland, Netherland - Intouch. SurfNet, AMS-IX, UUNet,Cistron, RIPE NCC andAT&T are connected at the AMS-IX. It is possible (there are requirements...) to get an static tunnel.

UNINETT, Norway - Pilot IPv6 Service (for Customers): tunnelbroker & address allocation NTT Europe, NTT Europe, United Kingdom - IPv6 Trial. IPv4 Tunnel and native IPv6 leased Line connections. POPs are located in London, UK Dusseldorf, Germany New Jersey, USA (East Coast) Cupertino, USA (West Coast) Tokyo, Japan

ESnet, USA - Energy Sciences Network; Tunnel Registry & Address Delegation for directly connected ESnet sites and ESnet collaborators.

6REN, USA - The 6ren initiative is being coordinated by the Energy Sciences Network (ESnet),the network for the Energy Research program of the US Dept. of Energy, located at the University of California's Lawrence Berkeley National Laboratory

See also here for more information and URLs: ipv6-net.org.

NSayer's 6to4 information

RFC 3068 / An Anycast Prefix for 6to4 Relay Routers

Protocol references

Current IEFT drafts of IP Version 6 Working Group (ipv6)

Network Sorcery / IPv6, Internet Protocol version 6, IPv6 protocol header

SWITCH IPv6 Pilot / References, big list of IPv6 references maintained by Simon Leinen

Linux related

Ipv6-HowTo for LUILIX by Peter Bieringer - Germany, and his Bieringer / IPv6 - software archive

Linux+IPv6 status by Peter Bieringer - Germany

USAGI project - Japan, and their USAGI project - software archive

General

IPv6.org

6bone

UK IPv6 Resource Centre - UK

JOIN:IPv6 information - Germany,by the JOIN project team maintaining also Links to external

WWW pages comprising IPv6/IPng

TIPSTER6 project - Hungary, "Testing Experimental IPv6 Technology and Services in Hungary"

WIDE project - Japan

SWITCH IPv6 Pilot - Switzerland

IPv6 Corner of Hubert Feyrer - Germany

Vermicelli Project -Norway

IPv6 Forum - a world-wide consortium of leading Internet vendors. Research & Education Networks...

Playground.sun.com /IPv6 Info Page - maintained by Robert Hinden, Nokia

NASA Ames Research Center (old content)

6INIT -IPv6 Internet Initiative -an EU Fifth Framework Project under the 1ST Programme

Something missing? Suggestions are welcome!

FCCN (National Foundation for the Scientific Computation)

Grupo de Pesquisa em IPv6 do Brasil

University of Algarve, Portugal

IPv6-MFA

BY Countries

Australia

Carl's Australian IPv6 Pages (old content)

Belgium

TELNET -the Belgian Research Network

Euronet-one of the biggest ISP's of Belgium...

Germany

Completel IPv6 infomiation, German ISP

IPv6-net-org, German IPv6 forum

France

Renater -Renater IPv6 Project Page

Italy

Edisontel-IPv6 Portal of Edisontel

Japan

Yamaha IPv6 (sorry, all in Japanese native ...)

Korea

IPv6 Forum Korea - Korean IPv6 Deployment Project

Mexico

IPv6 Mexico (spain & english version) - IPv6 Project Homepage of The National Autonomous

University of Mexico (UNAM)

Netherland

SURFnet-SURFnet IPv6 Backbone
STACK, STACK (IPv6) - Students* computer association of the Eindhoven University of Technology, Netherland.

IPng.nl,collaboration between WiseGuys and Intouch.

By operating systems

Cisco IOS

Cisco IOS IPv6 Entry Page

Compaq

IPv6 at Compaq - Presentations, White Papers, Documentation...

Microsoft

Microsoft Windows 2000 IPv6
MSRIPv6 - Microsoft Research Network - IPv6 Homepage
Getting Started with the Microsoft IPv6 Technology Preview for Windows 2000