1.1 WHAT IS VOICE OVER IP (VoIP)?

Before we start to discuss Voice over IP (VoIP) related topics, it is probably best to give a brief explanation of what it is. This way, the essence of what is discussed here will be clear throughout the document and the details can be worked out at the appropriate time.

Voice over IP is an extensive subject, but at the core it comes down to trying to transport speech signals in an acceptable way from sender to destination over an IP network. An Internet Protocol (IP) network is a computer network which uses the IP protocol to transmit information. We will give a more detailed explanation of this protocol in the next chapter, but for now it might be helpful to know that this is the basic protocol used on the Internet.

The definition of `acceptable' depends on the particular situation we are dealing with. If, for example, voice is being transported as part of a real-time communication between two persons, it will mean that the real-time aspects of this conversation must be respected: the overall delay between sending and receiving should be low to avoid irritably long gaps of silence. If, however, voice is being transmitted as part of a one-way process - e.g. an on-line radio show or a lecture - the delay constraints are less strict since the interactive aspect is no longer present.

1.2 PROJECT SUBJECT

Here, we will give the exact formulation of my project subject. This way there will be some clarity about what you may or may not expect to find in this document. The subject I have chosen is this one:

A conventional way to communicate with each other using IP-networks is through the use of textual chat facilities. The purpose of this thesis proposal is to take this one step further by using voice communication instead of these textual facilities. The goal of this proposal is to perform research and development in order to let persons which are in the same virtual environment talk to each other as they would do in reality. Their positions and orientations can be used to vary the intensity of the words: persons close to each other will hear each other clearly; persons who are moving away from each other will understand each other less and less as their distance

increases. The proposal encloses technical components (like grabbing, compression, buffering, transmission, decompression and regeneration of the signal) and also a study of what is happening in the Voice over IP world today. Also, a number of experiments will have to be conducted to justify the chosen techniques."

1.3 USES OF VOICE OVER IP

Currently, when you look at what literature can found about VoIP, you will find that most of it is about VoIP as a telephone alternative.

1.3.1 Telephone alternative

The first kind of use is the `telephone alternative'. This means that you would use some kind of VoIP system to make a voice call to another person. This can be done in several ways.

First of all, if a PC that can be connected to some kind of network is available, it can be used to make a call to somebody else who is also connected to that network. This PC would then be equipped with speakers and a microphone and some VoIP application would be used to make the call. The PC could have a direct connection to a computer network, like in figure 1.1, but a connection through a dial-up link is also possible.

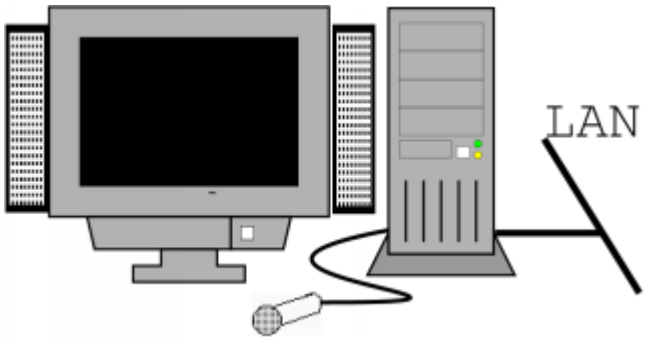

Figure 1.1: PC to LAN configuration

The second case is a slight variation of the first one. In this case, a telephone is connected to the PC and used in a similar way as you would when making a normal call. The PC does all the necessary work to set up the call and to transmit the speech signals. This also means that the PC has to be switched on before the call can be made. This type of configuration might be easier to use for people who do not work with computers often. As with the previous case, the connection to the network can be either direct, like in figure 1.2, or through a dial-up link.

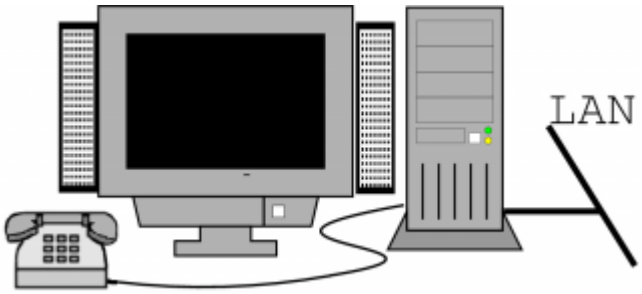

Figure 1.2: Telephone to PC to LAN configuration

Finally, the use of a PC and the requirement of a network could be omitted by the use of a VoIP gateway. This is a special device that connects the public telephone network with a computer network and performs the necessary actions and conversations to make the call possible. To make a call to somebody, you would call the gateway and specify the destination for the call. The call will then be set up and if the other end is available, the conversation can start. This configuration would be best for persons who do not have a PC. It is probably also the easiest to use, since most people are familiar with using a telephone and there does not have to be a PC around. This configuration is illustrated in figure 1.3.

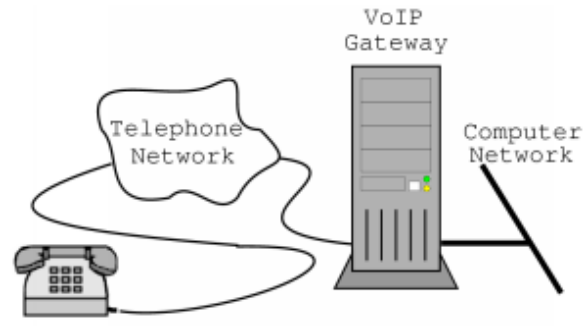

Figure 1.3: Telephone to gateway configuration

There are probably a lot of variations to these configurations, but we believe that these three give a good idea of the possibilities. Combinations of these cases can also be worked out.

Now, you may ask yourself: why use VoIP as a telephone alternative while the telephone itself is quite handy? Well, there are several arguments that can be made in favor of VoIP.

Suppose that somewhere - in a company or university for example - a computer network is needed. In that case, there are certain benefits by using Voice over IP instead of installing extra facilities to use telephones. The only requirement is that the IP protocol must be used, but nowadays this is almost always the case.

First of all, there is less cabling and equipment required. All the internal calls can be made using VoIP utilities. For outgoing and incoming calls, however, there still has to be some connection to the telephone network. This can be solved by installing a gateway that is connected to the computer network and the telephone network. This gateway will then perform the necessary signaling and conversations to make these calls possible.

Second, the capacity of the computer network will be better utilized. The available bandwidth of a network within an organization is usually quite large and rarely fully used. By using VoIP, more of the network's capacity will be used.

At home, there is also an advantage in favor of VoIP. If Voice over IP could be used over a large distance, it would be much cheaper than making that same long distance call using the telephone network. For example, you could try to make the call by using the Internet.

When using VoIP over a Local Area Network (LAN), there is usually plenty of bandwidth available and the delay between sending and receiving is usually very low. Here, VoIP can often be used without problems. But when a Wide Area Network (WAN) is used - the Internet for example - problems can arise. One problem is the delay: while the delay on a LAN is usually very low, on a WAN this is not necessarily true. If the delay gets too large, the conversation will not be very pleasant. Another problem is the quality of the speech signals. When certain routes get too heavily loaded, packets on the WAN will be lost. These lost packets cause interruptions in the speech signal. In turn, these interruptions, when large enough, can also disturb the

conversation. To alleviate the load, a lot of VoIP programs use compression techniques. However, compression often causes a certain degradation of the signal. This may or may not be disturbing to the listener, but with heavy compression, telephone quality will rarely be achieved.

Using VoIP this way is a rather new concept. This also means that currently, there is very little specific literature about it. However, it is obvious that a lot of the things that we have said in the previous section, also apply to VoIP in virtual environments.

1.3.2 Other

VoIP techniques can be used for a wide variety of other applications which require voice or sound in general to be transmitted over a computer network and where timing and synchronization are important issues. The same techniques also work when it is not sound, but video information which has to be transmitted.

Several other applications can be thought of. One is the use of VoIP techniques to create an on-line radio station, or perhaps even an on-line jukebox, where you can select the song you want to hear, which is then played almost immediately. If enough bandwidth is available, it would even be possible to add video data to all this. This way, television broadcasts and video on demand over IP networks could be made possible. In a similar way, we could extend a VoIP telephone conversation with video information about the persons involved in the call, creating a videophone application.

Another kind of application would be fax over IP. This is a bit different since we are no longer transmitting speech data, but a digitized image. Like with VoIP, this service could be made possible by connecting a computer network to the telephone network using a gateway. For fax over IP, this gateway would perform similar functions as with voice over IP.

Note that the list of applications presented here is certainly not complete. A wide range of applications using VoIP related techniques are conceivable, but many of them will resemble the ones discussed above.

2.1 OVER VIEW OF VOIP

This chapter introduces the Internet Protocol (IP), voice over IP (VoIP), packetized voice, and internet telephony. The chapter includes several sections. The first section explains why VoIP is of such keen interest to the industry .The next section explains the prevalent configurations for VoIP. The third section provides a brief introduction to the basic terms and concepts associated with IP-based packet networks, such as the Internet and internets. Following this overview, several key factors for the support of packetized voice in an internet are evaluated.

2.2 INTERNET TELEPHONY AND PACKETIZED VOICE

Voice over IP (VoIP) means the transmission of voice traffic in packets. Several terms are used to describe this process. Unless otherwise noted, I will use these terms synonymously: Internet telephony, IP telephony, packet-voice, packetized voice, and voice over IP (VoIP).

2.3 WHY INTERNET TELEPHONY?

IP telephony is viewed by some people to be an effective technology and by others as nothing more than an irritant. The irritating aspect stems from those people who have used the public Internet to make telephone calls. In most cases, they are not happy with the quality of the speech and the overall ability of the Internet to support voice traffic.

Why then is VoIP of such keen interest to the communications industry, in view of its relatively poor performance in the support of voice traffic?
There are four major reasons for this interest, and for the deployment of IP telephony. The next part of this chapter discusses these reasons in this order:

1. The business case
    (a) Integration of voice and data
    (b) Bandwidth consolidation
    (c) Tariff arbitrage
2. Universal presence of lP
3. Maturation of technologies
4. The shift to data networks

2.3.1 The Business Case

The first reason is a compelling business case for the deployment of the IP protocol suite and associated equipment to support telephony services. This case can be summarized with three suppositions.

2.3.1.1 Integration of Voice and Data.

First, clearly the *integration* of voice and data traffic will be demanded by multi application software resulting in the inevitable evolution to Web servers capable of interacting with the customer with data, voice, and video images. Text-only images with still life photos will be a thing-of-the-past.

2.3.1.2 Bandwidth Consolidation.

The next two suppositions stem from the first. The second supposition is that the integration of voice and data allows for *bandwidth consolidation*, which effectively fills up the data communications channels more efficiently. The telephony legacy of channelized voice slots, with the expensive associated equipment (channel banks, and data service units (DSUs) are inefficient tools for the support of data applications.

The commonsense idea is to migrate away from the rigid telephony- based time division multiplexing. (TDM) scheme wherein a telephony user is given bandwidth continuously, even when the user is not talking. Since voice conversations entail a lot of silence (pauses in thinking out an idea, taking turns talking during the conversation, etc.), using the data communications scheme of statistical TDM (STDM) yields a much more efficacious use of precious bandwidth. STDM simply uses the bandwidth when it needs it; otherwise, the bandwidth is made available to other talkers who need it at that instant.

To give you an idea of how wasteful the telephony TDM approach is, consider that about 50 percent of a normal speech pattern is silence (at least in most conversations). Voice networks that are built on TDM use bandwidth to carry those silent periods. Data networks do not. Furthermore, another 20 percent of speech consists of repetitive patterns that can be eliminated through compression algorithms. The conventional TDM operations do not exploit this situation.

Moreover, by using modern analog-to-digital operations, a high- quality speech channel can operate at about 4.8 to 8 kbit/s, in contrast to current TDM telephony channels that operate at 64 kbit/s. In the future, it is expected that the packet voice rate will be reduced further. Let's assume a 6

kbitls rate for purposes of comparison. The bandwidth consumption ratio is 8: 1 in favor of the packet-based method.

2.3.1.3 Tariff Arbitrage and Beyond.

The third supposition regarding the business case is based on the concept called "tariff arbitrage." This term means bypassing the public switched telephone networks' toll services and utilizing an internet backbone. This approach avoids the costly long distance charges incurred in the tariffed telephone network in contrast to lower costs of the untarrifed Internet.

2.3.2 Universal Presence of IP

The second major reason for IP telephony is the universal presence of IP and associated protocols in user and network equipment. Of key importance is the fact that IP resides in the end-user workstation (in contrast to potentially competitive technologies such as ATM and Frame Relay that operate as user network interfaces (UNI). Figure 2.1 shows where these technologies are placed.

Make no mistake; the existence of IP in user personal computers and workstations gives IP a decided advantage over other existing technologies that are not resident in the user

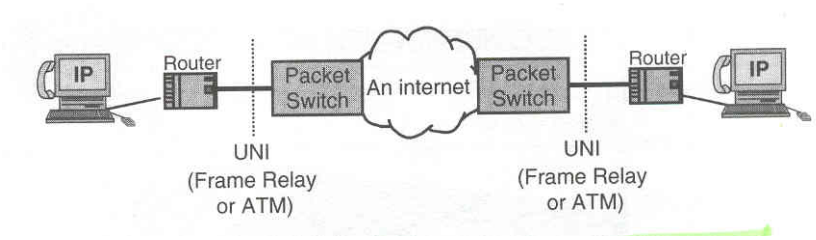

Figure 2.1

appliance. This "location" of IP makes it a very convenient platform from which to launch voice traffic.

Many people already use the PC to assist them in making telephone calls. Before long, computer-based telephony will be common, and will be a natural extension to the telephony system. Moreover, IP operates in both wide area and local area networks (LANs), whereas Frame Relay operates only in wide area networks.

2.3.3 Maturation of Technologies

The third major reason for the deployment of internet telephony is the maturation of technologies that now make IP telephony feasible.

2.3.4 The Shift to Data Networks

Finally, the fourth major reason for the assured success of VoIP and other data networks is the fact that the world is experiencing a shift away from circuit-based networks to packet-based networks (data net- works). Some market forecasts place the ratio of data networks-to-circuit networks at 80 to 20 percent by 2005.

2.4 CONFIGURATION OPTIONS

We have discussed the issues surrounding VoIP .Let us now look at some VoIP configurations and topologies. Several configuration options are available to support VoIP operations.

2.4.1 Telephone connection with N-1 gateway

In Figure 2.2, conventional telephones are employed as well as the telephone network (you may have noticed that the term telco is used in this documentation as a shorthand notation for the telephone network). The VoIP gateway provides the translation functions for the voice/data conversions. On the transmit side, the gateway uses a low-bit rate voice coder and other special hardware and software to code, compress, and encapsulate the voice traffic into data packets (IP datagrams). It accepts conventional telco traffic (usually encoded by the telco central office into digital 64 kbit/s telco signals), and uses the voice coder to convert these signals into highly compressed samples of the telco signal, usually about 6-8 kbit/s.

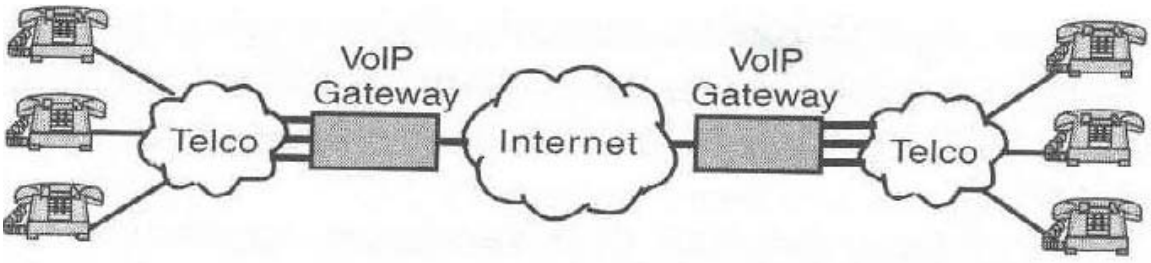

Figure 2.2

At the receiving VoIP gateway, the process is reversed. The gateway converts the low-bit rate speech back to the telco signals. These signals are converted to conventional analog signals before they are passed to the user's telephone.
This gateway is an n:1 machine, because it accepts n telephone connections and multiplexes them into IP datagrams onto one link to the Internet or an intranet.

2.4.2 PC Connection with Router

Figure 2.3

Figure 2.3 shows the use of personal computers (PC) and the employment of a router. With this operation, the encoding, compression, and encapsulation operations are performed at the personal computers. The router's job is to examine the destination IP address in the datagram and route the traffic accordingly. The router treats the traffic just like any other datagram, and is not aware that the bits in the datagram are voice traffic.

2.4.3 Telephone to PC Connection

The VolP layout depicted in Figure 2.4 eliminates background noise problems found in Figure 2.3 by using a telephone instead of an open microphone.

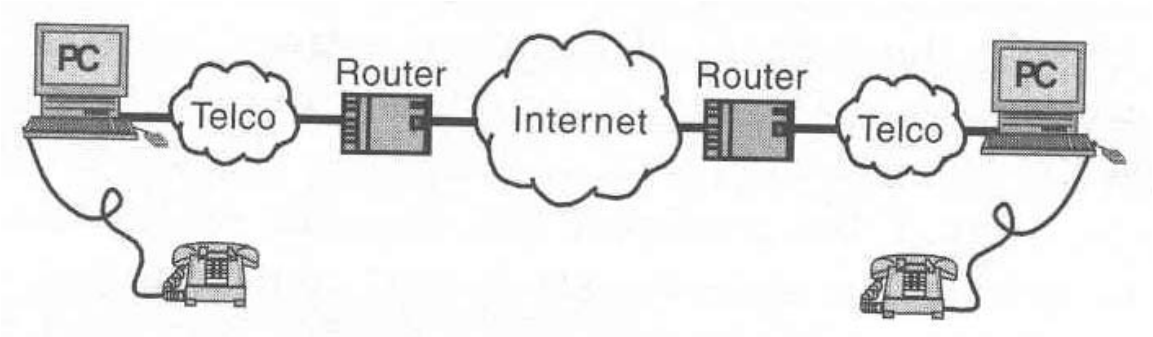

Figure 2.4

2.4.4 PC to phone Calls

This configuration is one that is gaining considerable attention in the industry, because the local LANs (such as Ethernet) can be used for both voice and data traffic. Also, for simple telephone calls, there is no expensive key system or private branch exchange (PBX) in the system.

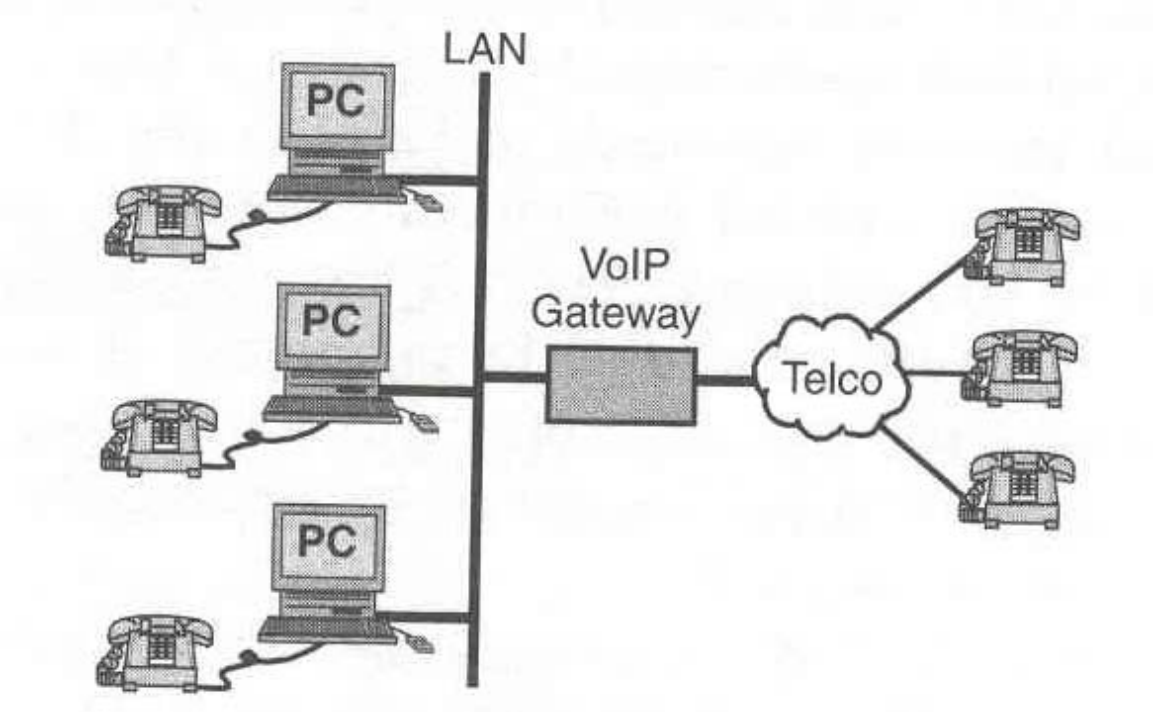

Figure 2.5

2.4.5 Problems with the Configurations

The configurations shown in above figures represent low-function systems. These are bare-bones operations when compared to the services taken for granted by most telco users. The configurations shown in above figures do not include the equipment to support call forwarding, call holding, caller id, or other telco services voice users expect. These services are provided by machines (such as key sets, PBX, centrex, etc.) absent from the above figures configurations.
Additionally, configurations In Figures utilize the public Internet, which is not set up to deliver toll-quality voice traffic.

Over view of IP

Before we can really talk about Voice over IP, it is necessary to explain what IP is? The abbreviation IP stands for Internet Protocol. Version four is currently most in use and it is common to use the term `IPv4' to indicate this version of the protocol. When no version number is mentioned, usually the discussion is about version four.

The Internet Protocol is covered in this chapter. It begins with a discussion about network software architecture, followed by a description of the workings of IP. We will also see some characteristics of IP networks and I will describe the most used protocols which run on top of IP. Afterwards, some reasons will be given for the use of IP for voice communication. Finally, the chapter contains an overview of IPv6, the new version of the Internet Protocol.

3.1 NETWORK SOFTWARE ARCHITECTURE

Nowadays, network software is usually very structured. This section is about the way this software is organized. It also contains a discussion about the OSI reference model, which is a good example of this structured design, and about the TCP/IP reference model, in which as the name suggests IP plays a very important role.

3.1.1 Layered design

To facilitate the design of network software, usually the approach of a `layered design' is used. In this approach, each layer provides a certain functionality, which can be used by the layer directly above. There are several advantages to this approach.

First of all, the software is much easier to design. Trying to implement the desired functionality all at once will be very difficult and will probably result in many flaws in the program. Furthermore, these flaws will be difficult to track. By dividing the software in layers, you only have to worry about implementing some functionality for each layer. This does not mean

that is will be an easy task, but by using a structured approach you will be able to tackle it more efficiently.

Another advantage is the adaptability. If you want to make some changes to the software, for example to correct a flaw or to improve an algorithm, you will only have to change the relevant layers if the interface with the layer above stays the same.

Closely related to this is portability. If the layers are well designed, only a few of them will have to be changed to be able to use the software with other networking hardware or on another operating system.

Finally, since many layers will probably be implemented as part of the operating system itself, the end-user applications do not have to contain those layers. This way, the size of those applications can be reduced.

To make communication between two hosts possible, they have to be connected to some kind of physical medium. All data will be sent over this medium, but only the lowest layer will have direct access to it. Conceptually, however, two layers on different machines but at the same level can be thought to communicate directly. The rules and conventions that are used in this communication are contained in the protocol for that level. The whole set of protocols is often referred to as the protocol stack. Figure 3.1 illustrates all this.

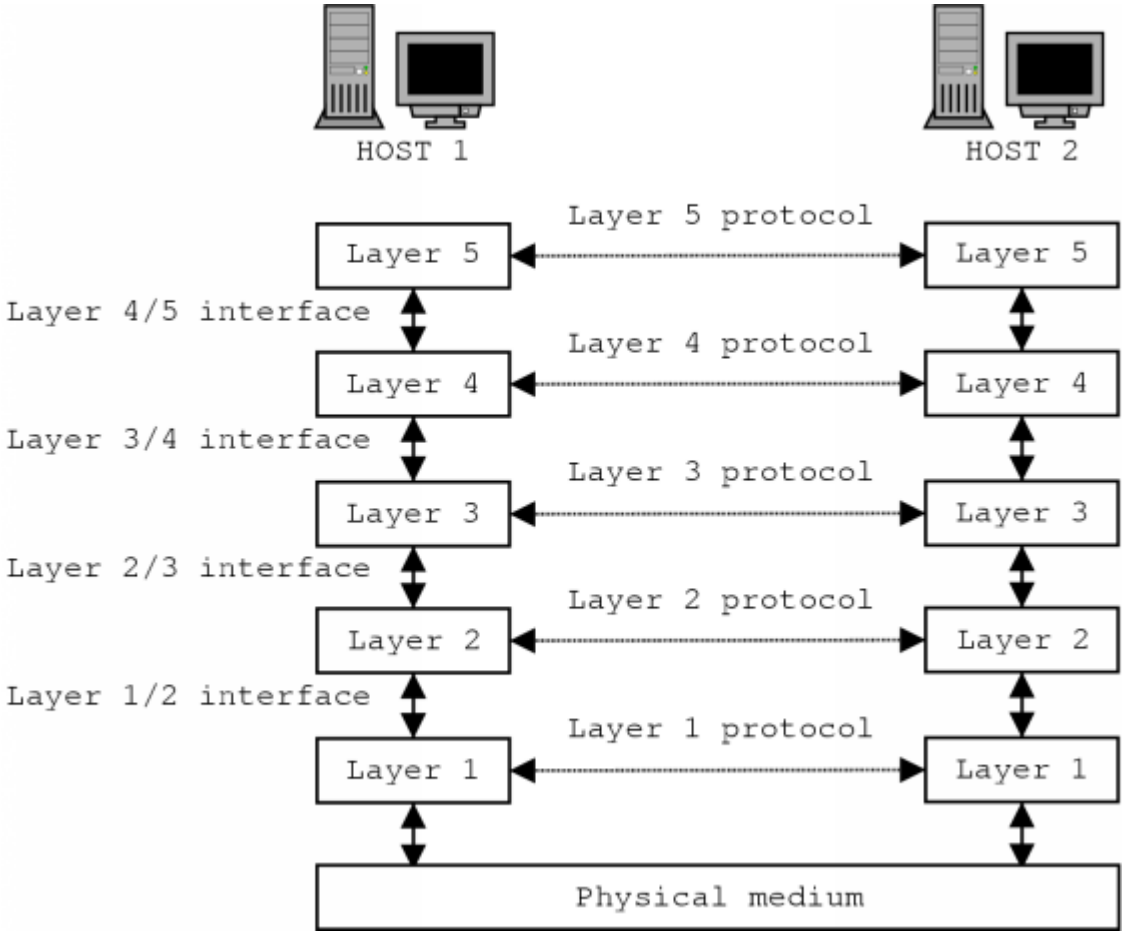

Figure3.1: Example of layered design

When a layer wants to transmit some data to its corresponding layer at another host, it uses the functionality of the layer below to do this. That layer adds some control information, usually in the form of a header, to the data and uses the layer below to transmit the data. The whole process keeps repeating itself until the data is finally sent over the physical medium. When the data reaches the receiver, the first layer processes the control information and passes the data to the layer above. At each layer, this process then repeats itself.

3.1.2 OSI Reference model

The Open Systems Interconnection (OSI) reference model is a model with seven layers which was developed by the International Standards Organization (ISO). The model only specifies what each layer should do, without going into any detail about, for example, the protocols that should be used.

In actual implementations it turns out that some of the layers are almost empty and others are too elaborate. However, conceptually the model is quite nice and it is a good example of layered design. This is why we will describe it briefly.

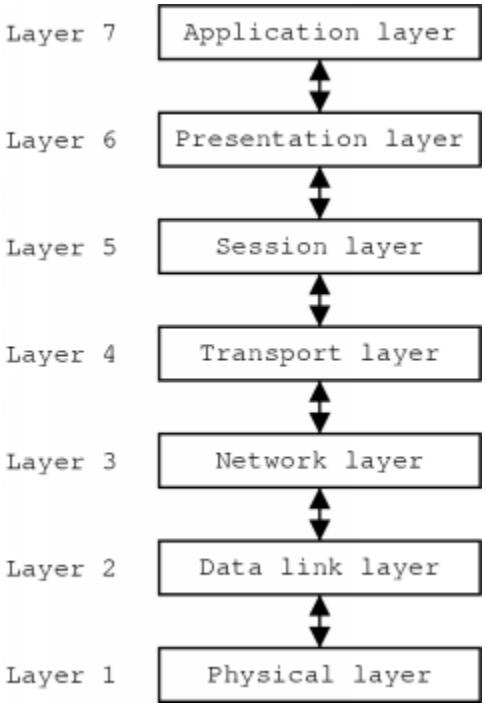

Figure 3.2: OSI seven layer model

3.1.2.1 The physical layer

The physical layer is the lowest layer in the model and this is the only one which has immediate access to the communication medium. It is responsible for the transfer of bits from the source to a destination which is connected to the same medium.

3.1.2.2 The data link layer

The data link layer uses the facilities of the physical layer to create a more reliable communication channel. This layer makes it possible to send blocks of data, called frames, reliably from one host to an adjacent one.

3.1.2.3 The network layer

So far, the layers have only been concerned with transporting information between hosts connected to the same medium. The network layer's function is to make it possible to send packets to a host that does have a connection to the sender, but is not connected to the same physical medium.

This means that between the different physical media, there have to be devices which transfer data from one medium to another. These devices are usually called routers or gateways. The use of such devices makes some extra work for the network layer necessary.

First of all, it is possible that between a certain source and destination there exist several possible routes. The network layer then has to determine which one to choose. These routes can be determined in advance but it is also possible that the network layer dynamically adjusts the routing information to achieve better performance.

Second, since the flow between adjacent networks can get very large, it is possible that a router cannot cope with all that traffic. The router then becomes a bottleneck for the data flow. The network layer tries to control such congestions.

3.1.2.4 The transport layer

The previous layer made it possible to actually send data from source to destination. In that layer communication is done by exchanging packets. The transport layer makes it possible to consider the data as a stream of bytes, and not in terms of packets. The layer itself will divide the data in smaller units and hand it over to the network layer. If some packets get lost, the layer handles this and the receiver will still receive the correct stream of bytes. To

be able to keep track of which data has already been sent and which not, the transport layer uses a connection-oriented approach.

The transport layer will also have flow control mechanisms, to prevent the flooding of a slow receiver, and congestion prevention mechanisms. Note that the network layer also has congestion control functionality. However, the best way to handle congestions is to prevent them from happening in the first place. This is what the transport layer does.

This layer is the first true end-to-end layer. The physical and data link layers were only able to communicate with an immediate neighbor. The network layer actively had to transport the packets step by step from source to destination. In this layer however, the underlying topology is transparent to its user.

3.1.2.5 The session layer

The session layer makes it possible to establish sessions between two hosts. A session extends the capabilities of the transport layer with some extra services.

An example of such an extra service is synchronization. During a transfer there would be certain synchronization points. If the data transfer would be interrupted due to an error, the transfer could be restarted from the last synchronization point rather than starting the transfer all over again.

3.1.2.6 The presentation layer

The presentation layer takes the type of information which is being transferred into consideration. This layer could, for example, make the necessary transformations if one computer is sending ASCII characters and the other one is sending Unicode characters.

3.1.2.7 The application layer

Finally, the highest layer in the model is the application layer. This is the layer in which most end-user networking applications reside. To communicate, such programs mostly use their own protocols. Examples of such applications are applications for file transfer and applications which represent a virtual terminal.

3.1.3 TCP/IP reference model

The Internet Protocol is a protocol which is used in the TCP/IP model. The TCP/IP model was originally designed for use on the ARPANET, a military network in the late 1960s. It is, in fact, this network which grew out to become the Internet as we know it today.

Because of its military background, there were two major requirements for the model. The first was robustness. The US Department of Defence (DoD) wanted to make sure that communication was still possible even if some routers or lines went down. The second requirement was interoperability. Since there were different types of hardware involved, for example copper wires and satellites, the DoD wanted a set of protocols which could not only handle these types of hardware separately, but which would also make it possible to connect them.

Compared to the OSI model there is a big difference in the way that the model came to existence. The OSI model was first carefully designed, and later protocols were designed to fit the model. This makes the OSI model a very general one. The TCP/IP model, however, originated in the opposite way. First the protocols were designed to meet the requirements of the DoD. Later, these protocols were described and it is this description which is the reference model. This means that the TCP/IP model does not really fit anything else but TCP/IP networks. Another point about TCP/IP is that the layered design is not followed very strictly. There are some violations to this principle in the model.

Despite of these arguments, the TCP/IP model has become very popular and very widely used. In contrast to the OSI model which has seven layers, the

TCP/IP model only has four, as figure 3.3 shows. Here is a description of these layers.

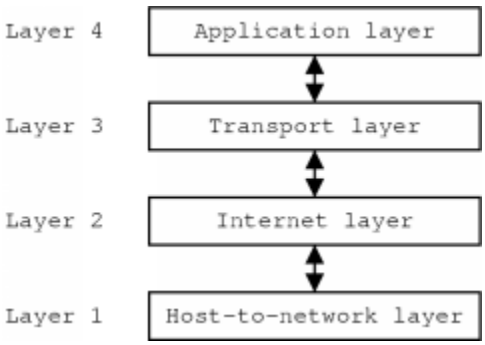

Figure 3.3: TCP/IP four layer model

3.1.3.1 The host-to-network layer

The host-to-network layer is the lowest layer of the model. Sometimes it is also called the link layer or the network interface layer. There is in fact little to be said about this layer. The only requirement which is given by the model is that this layer should be able to transmit and receive the IP datagrams of the layer above over the network. The layer has somewhat the same function as the physical and data link layers in the OSI model. This means that this layer usually is only able to send data to hosts which are connected to the same medium.

3.1.3.2 The internet layer

The internet layer corresponds to the network layer in the OSI reference model. Its job is to bring packets from source to destination, across different types of networks if necessary. There are, however, no guarantees that the packets will arrive or that their order will be preserved. The service that this layer offers is therefore called a best-effort service. There is no notion of a connection in this layer. The packets which are exchanged are called Internet Protocol datagram or IP datagrams and the protocol which is used is called the Internet Protocol or IP. The datagrams consist of a header and the actual data. The header will be described later on.

Like in the OSI network layer, intermediate devices called routers, are needed to make transmission of data across different types of networks possible. The IP datagrams can then be sent from source to destination, on a hop-by-hop basis. Again, like in the OSI network layer, this also means that routing algorithms and congestion control are important aspects of the internet layer.

3.1.3.3 The transport layer

To make sure that multiple applications can use the network facilities at once, some extra naming mechanism is needed. The internet layer does contain a naming mechanism to identify different hosts, but there still has to be some way to differentiate between the processes which are using the network. This is done in the transport layer by the use of a port number. This layer has somewhat the same functionality as the transport layer in the OSI model. Here also, the transport layer is the first real end-to-end layer.

The TCP/IP model has two major transport layer protocols. One of them is the Transmission Control Protocol (TCP). This protocol transforms the connectionless unreliable packet based service of the internet layer into a connection-oriented reliable byte stream. It is a very important protocol since it makes reliable communication possible. This is why its name is also in the name of the reference model.

The other protocol is the User Datagram Protocol (UDP). This is a protocol for applications which do not need the service offered by TCP or wants to use a protocol of their own. The User Datagram Protocol is merely a small extension to IP. It is also an unreliable packet based connectionless protocol and the only real extensions to IP itself are the presence of a port number and an optional checksum of the data.

3.1.3.4 The application layer

Like in the OSI model, the application layer contains the protocols of networking applications. Among these are virtual terminal applications (TELNET protocol), file transfer utilities (FTP protocol) and electronic mail (SMTP protocol).

3.2 HOW IP WORKS ?

Let us now take a closer look at the Internet protocol itself and how it makes communication between two hosts possible. First I will give a description of the IP packet format. Next, the addressing mechanism used by IP is discussed. We will then take a closer look at how packets are routed from source to destination. Finally, an explanation is given of multicasting, a technique which allows us to save bandwidth when the same data has to be sent to multiple destinations. This is, of course, a very interesting feature when using VoIP in virtual environments, since there will typically be many receivers for each talking participant.

3.2.1 Packet format

Any packet sent by the IP layer consists of an IP header, followed by the actual data. The format of the IP header is shown in figure 3.4. The most significant bit is the one at the left, numbered zero. The least significant bit is the one at the right, numbered thirty-one. Transmission is done in network byte order, also called big endian format. This means that in each 32-bit word the most significant byte is sent first and the least significant byte is sent last.

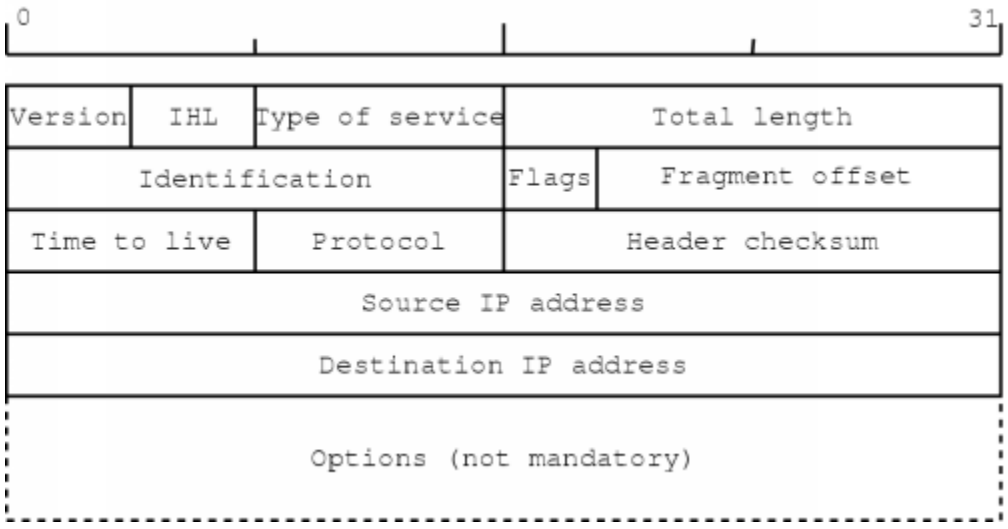

Figure 3.4: IP header format

The **version** field should contain the value `four' for the current version of the Internet Protocol. This field can be used to let different versions coexist, something which will make the transition to a new version much easier.

The **IHL** field contains the `Internet Header Length'. This specifies the length of the header in 32-bit words. Since it is a 4-bit value, the maximum length of the header will be sixty bytes. Also, since the mandatory part of the header consists of five words, the smallest legal value is five. The specification in 32-bit words also has as a consequence that the header must end on a 32-bit boundary, so it is possible that some padding is required if options are present.

The next field is the **Type of service** (TOS) field. This field was meant to supply a quality of service (QoS) mechanism, but in practice it is rarely used. However, since voice data has real-time aspects, it may be necessary to pay attention to it if we want to keep the end-to-end delay in the communication low.

An overview of the TOS field is depicted in figure 3.5. That contains a three-bit precedence field which specifies the priority of the packet. A value of zero indicates a normal priority and a value of seven indicates the highest priority. Following the precedence field, there are three bits which stand for delay, throughput and reliability. Only one of the bits can be set to one. The last two bits in the field are currently unused and should be zero.

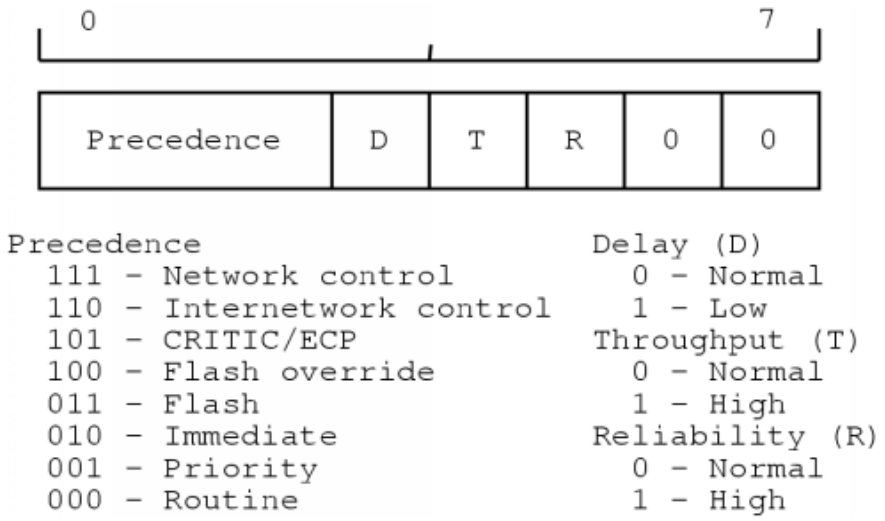

Figure 3.5: The TOS field

The size of the IP datagram is specified in the **Total length** field. It is a 16-bit field, so the maximum size is 65535 bytes. Most networks cannot handle this size so usually it is much less. All hosts are, however, required to be able to send and receive datagrams with a length of 576 bytes or less.

During the transmission of a packet it is possible that it has to traverse different kinds of networks. Each network has its own Maximum Transfer Unit (MTU) which specifies the maximum frame size it can handle, including the link layer header and trailer (if present). This means that there is always a possibility that the datagram, as it passes over the different networks, cannot be transmitted over a certain network. It then has to be fragmented and each piece has to be sent separately.

The **identification** field is an aid in reconstructing fragmented datagrams. Each datagram fragment will have the same value in this field. When sending IP datagrams, a host typically increments this field for each datagram sent.

Next, there are three **flag** bits, of which the first one is reserved and should be zero. The next one stands for `don't fragment' (DF) and the last one stands for `more fragments' (MF). If a datagram cannot be transmitted across a network because it is too large and the DF bit is set, an error will be sent back to the sender[3]. All but the last the fragment of the original datagram will have the MF bit set.

Using the **fragment offset** field, the internet layer can reassemble fragmented datagrams. This 13-bit value specifies the offset of the fragment in the original datagram. The offset is given in units of 64-bit words.

The **time to live** (TTL) field is used to limit the lifetime of a datagram. In theory the value specifies the number of seconds the datagram is allowed to exist. There is also the requirement that each router must decrement the value by at least one. If the packets stays a long time in the queue of the router, the TTL value should be decreased with the number of seconds the datagram spent in queue. When the counter is zero, the datagram must be discarded. In practice, the value is just decremented at each router, which makes the field a hop counter.

The **protocol** field is used to specify to which protocol the data in the datagram belongs. This can be a transport layer protocol, but it can also be one of the control protocols of the internet layer.

The **header checksum** is used to check the validity of the datagram. Note that the checksum is only for the header, so higher level protocols will have to use their own checksums if they want to make sure their data is valid.

Finally, the minimal header contains the **source IP address** and the **destination IP address**. These addresses must be included in each datagram since the internet layer operates in a connectionless way. Each datagram is sent separately and therefore each datagram must contain not only its destination but also its source, in case an error has to be reported. The format of the addresses is described further on.

The **options** section can be used to record the route a datagram follows, possibly with timestamps. Another option is source routing, where you can specify the route a datagram should follow.

3.2.2 Addressing

Every host on an interconnection of networks - or internet - which uses IP, should have a unique IP address. An IP address is a 32-bit value and the complete address space is divided into five classes, named class A to class E. The way these classes are represented is shown figure 3.6.

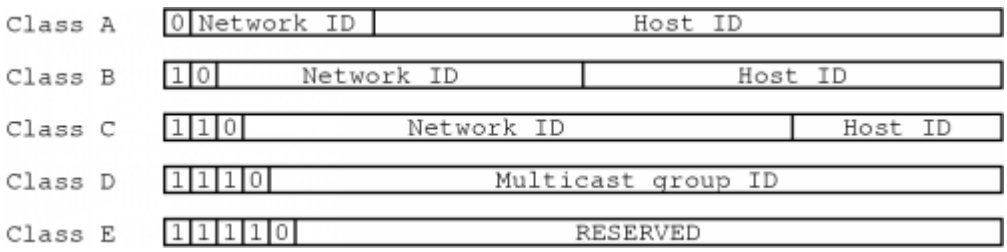

Figure 3.6: Classes of IP addresses

The way an address is usually written, is in its dotted decimal form. To obtain this the 32-bit value is split in four 8-bit values. These four values are then written in decimal form, separated by dots.

The first three classes contain the addresses which can be assigned to hosts. Not all possibilities are allowed though; there are some reserved addresses. First of all, a host ID with value zero does not specify a host, but the network on which hosts with the specified network ID are located.

If the host ID is the highest possible value for its class (all one bits in binary format), the address is a broadcast address for a certain network. This means that if you send IP datagrams to that address, they are delivered to all hosts on that network.

When the network ID of an address is zero, it specifies the local network. This type of address is only used in initializations procedures, when the local network ID is not known.

Other reserved addresses are 0.0.0.0 and 255.255.255.255. The first of these specifies the local host on the local network. It is also only used in initialization procedures. The second address is the so-called limited broadcast address. This specifies a broadcast to all hosts on the local network.

Of the remaining two classes, only class D is actually used. Class E was meant for future use. Class D specifies a multicast address. Multicasting allows data to be sent to a group of hosts. This means that when you send an IP datagram to a multicast address, the datagram is sent to all hosts in the corresponding multicast group. Multicasting is explained in more detail later.

3.2.3 Routing

The internet layer uses the link layer to actually transmit its data. The link layer, however, can only deliver this data to hosts which are connected to the same medium. To be able to send this data across several networks, routers are used. These devices connect to several networks and make sure that incoming IP datagrams are forwarded to the appropriate network. We will now take a closer look at how this process works. Note that only the basic mechanisms of routing are explained here.

When the internet layer of the sending host has to transmit a datagram to a certain destination, it first examines the destination IP address. This is

necessary because the internet layer has to tell the link layer to which machine the data has to be sent. If the destination IP address is on the same network, the machine which will receive the datagram will simply be the destination for the transmission.

If the address does not specify a host on the local network, the internet layer examines its routing table. The entries of such a routing table can be seen as pairs of a destination address and a router address. The destination address can be an address of a host or of a network.

The internet layer then starts looking for a router to send the datagram to. To do this, it compares the destination address of the datagram with the destination addresses in the routing table. If no complete match can be found, it checks if a matching network entry can be found. If not, it uses a default entry. If an entry was found, the internet layer takes the corresponding router address and tells the link layer to send the datagram to that address.

For example, consider a host with IP address 199.198.1.10 who wants to send a packet to 199.198.2.100. This destination host is not on the same network, so the internet layer of the sender will consult its routing table. Suppose that the table looks like this:

Destination	Gateway
199.198.5.10	199.198.1.251
199.198.2.0	199.198.1.252
default	199.198.1.253

The internet layer first looks in the table for a complete match for address 199.198.2.100. It finds no such match, so it will check for a matching network address. This time, it does find a matching entry: the second one describes the network on which the destination host is present. The internet layer then takes the corresponding gateway entry - address 199.198.1.252 - and sends the packet to that router (gateway).

When the datagram reaches the router, it is passed on from the link layer to the internet layer. The internet layer then follows almost the same procedure to search for a destination machine to forward the datagram to. The only

difference is that the router will usually be connected to several networks and this means that the appropriate interface to transmit the data also has to be chosen. The whole procedure is repeated until the datagram reaches its final destination.

To make sure good routes are chosen, many routers communicate with each other. They exchange their routing information and based upon this information each router updates its routing table to contain the best known route for each destination. The type of information and the way it is exchanged are determined by the routing protocol which is used. Examples of routing protocols are the Open Shortest Path First (OSPF) protocol and the Border Gateway Protocol (BGP).

3.2.4 Multicasting

Basically, there are three transmission modes that can be used when sending an IP datagram. They are called unicast, multicast and broadcast. Unicasting simply means sending a datagram from a source to one destination. The term broadcasting is used when you want to send a datagram to all hosts on a specific network. When you want to send a datagram to an arbitrary set of hosts, it is called multicasting.

A simple way to implement multicasting would be to unicast a copy of the datagram to each destination. This method obviously wastes a lot of resources. A better way would be to transmit one datagram which is copied only at points where it needs to follow different routes to reach its destinations. This is the way it is done on IP networks.

To be able to receive datagrams directed to a certain multicast address, a host must first join the multicast group associated with that address. Similarly, when it no longer wants to receive those datagrams, it leaves the multicast group. This group management is done according to the Internet Group Management Protocol (IGMP), which is formally specified in.

In general, the protocol works as follows. Each host maintains a list of multicast groups from which it wants to receive datagrams. Multicast routers periodically broadcast IGMP queries on the networks to which they are connected. The hosts then send IGMP replies, containing the groups in which they are interested.

Once these replies have been gathered using IGMP, multicast routers exchange this data with each other and use all this information to build their routing tables. When they receive a multicast datagram, they can then determine to which hosts and multicast routers the datagram should be sent.

3.3 CHARACTERISTICS OF IP NETWORKS

When datagrams have to travel across several networks, they will also need to pass through a number of routers. Each router has to examine all incoming packets and this will introduce a certain delay in the communication. Studies even show that the time it takes for a packet to reach its destination is much more affected by the number of hops the packet makes than the actual geographical distance covered.

When a router gets too heavily loaded, some packets will have to be discarded. This packet loss is usually bursty. This means that for a short period of time several consecutive packets will be lost.

Routers communicate with each other to dynamically adapt their routing tables to the current state of the network. This means that datagrams going to the same destination can sometimes follow different routes. Although it turns out that routes do not change very often during a transmission, it does happen. Such a change can cause datagrams to arrive out of order.

Besides packet loss and out-of-order arrival of packets, it can also happen that a datagram gets duplicated during its transmission. This will cause two or more identical datagrams to arrive at the destination, possibly with some delay between them.

Finally, another important feature of IP networks is the fact that when a source sends datagrams to a certain destination, the amount of time to reach the destination will differ for each datagram. This is usually called inter arrival delay, inter arrival jitter or simply jitter.

3.4 HIGHER LEVEL PROTOCOLS

The two most common transport level protocols in the TCP/IP architecture are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Each of these protocols offers a specific kind of service which applications can use to communicate across networks.

3.4.1 TCP

Currently, TCP is undoubtedly the most used protocol of the two. This protocol transforms the unreliable packet-based service of the internet layer into a reliable byte stream. The protocol is designed for communication between two hosts, so it only supports Unicasting.

To offer this kind of service, the TCP module has to do a lot of work. First of all, a connection has to be set up, and this has to be done in such a way that it is more or less safe: the module must make sure that connections cannot be established accidentally - for example because of duplicate packets.

The incoming stream of bytes then has to be split up at the side of the sender and the stream has to be reconstructed at the side of the receiver. Care must be taken to discard duplicate datagrams and to correct their arrival order if necessary. There must also be some kind of mechanism to cope with lost packets.

All this is handled quite effectively. To establish a connection the TCP module uses a handshake mechanism, called a three-way handshake. Duplicate and out-of-order datagrams are handled by using sequence numbers. Finally, lost packets are handled by an acknowledgement mechanism: all bytes of the stream have to be acknowledged by the destination. If the source did not receive an acknowledgement after a certain amount of time, it sends the necessary data again. The protocol also specifies flow control mechanisms, which prevent the swamping of a slower receiver, and congestion control mechanisms, which try to avoid congestions.

Note that the exact way in which the TCP module works is a lot more complicated than this explanation makes it seem. For a complete specification of TCP.

3.4.2 UDP

Applications which do not require the functionality that TCP provides can use UDP. To transmit data, the UDP module simply passes a UDP header followed by that data to the internet layer which then sends the datagram on its way. This means that just like IP itself, UDP is a best-effort service. No guarantees about delivery are given, datagrams can get reordered and datagrams can be duplicated. The UDP header is shown in figure 3.7. The header contains the source and destination ports, which identify the sending and receiving applications. Next, it contains the number of data bytes which must be sent and finally the header contains space for an optional checksum.

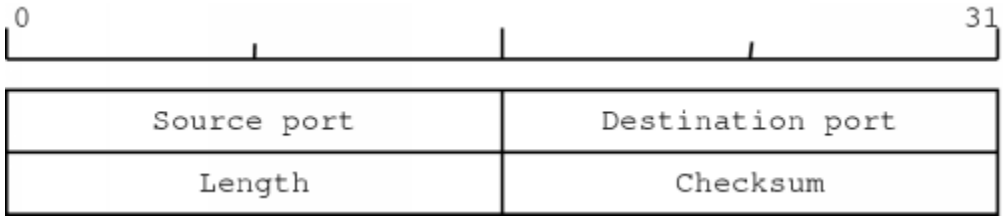

Figure 3.7: UDP header

Since the service which UDP offers is almost identical to the service of IP itself, it is possible for applications to send UDP datagrams to a multicast address and to receive UDP datagrams from a multicast group.

3.5 WHY USE IP?

Delivering speech information in packets has some advantages to the classical telephone system. When you make a `normal' telephone call, a path is set up between you and the destination of the call. You will then have a fixed amount of bandwidth you can use during the whole call.

The major advantage of that approach is that you will have some guarantees about the QoS, since you are certain to have a specific amount of bandwidth available. But this way, a lot of bandwidth is also wasted, because during a conversation there are a lot of silent intervals for each person.

Using VoIP, those silent intervals can be detected. The VoIP application can examine each packet and detect whether it contains speech information or only silence. If the latter is the case, the packet can simply be discarded.

Another advantage is the possibility of compression. With the compression methods available today, it is possible to reduce the requirement of 64 kbps[5] for uncompressed telephone-quality voice communication to amounts which are far lower. However, a high compression ratio often means that the voice signal will be of lesser quality. So packetized voice has certain advantages to the classical telephone system. But IP is not the only packet-based protocol. Why exactly should IP be used? This protocol was designed mostly for data transport, and it has only limited QoS support. The main reason IP is so important is because of its omnipresence. The TCP/IP architecture has proved to be very popular and nowadays it is very widely used. This fact gives IP a great advantage over other protocols.

Alternatives for packetized voice include Voice over Frame Relay (VoFR) and Voice over ATM (VoATM). Both allow better support for real-time traffic than an average IP network. However, these technologies are not used as widely as IP.

3.6 IPV6

With the growth of the Internet - on which IP is used - it has become clear that the current version of the Internet Protocol has some shortcomings. For this reason a new version of the protocol has been devised, now called IP version six, or just IPv6.

3.6.1 Reasons

Because of the enormous growth of the Internet, there will soon be a shortage of IP addresses. The current version uses 32-bit values, which can provide enough IP addresses in theory. However, because of the subdivision in classes and the way addresses are allocated within those classes, in practice there are far less addresses available. This lack of addresses was one of the most important reasons for the development of a new version.

Other reasons were the need for better QoS support and better support for security. Also, it turned out that some features of IPv4 were hardly ever used and bandwidth and processing time could be saved by redesigning the protocol. Finally, because the routing tables in routers kept growing, the reduction of their sizes was also an important reason for the design of an improved protocol version.

3.6.2 Description

Let us now take a closer look at this new protocol. First I will describe the format of the IPv6 header. Next, we will see what exactly changed compared to IPv4.

3.6.2.1 Header

The IPv6 header is shown in figure 3.8. In this version, the header has the fixed size of forty bytes.

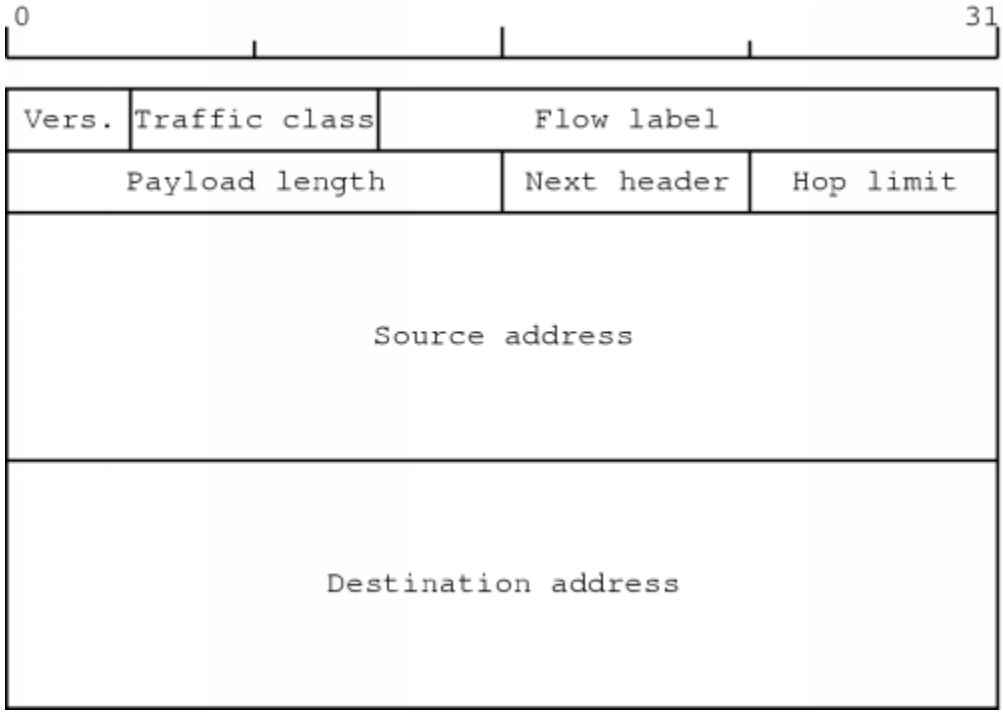

Figure 3.8: IPv6 header

The **version** field contains the value six. This way, the version of the protocol can be detected and IPv4 and IPv6 can coexist. This will make the transition to the new version easier.

The **traffic class** has somewhat the same function as the TOS field in the IPv4 header. Using this field, one could specify the type of traffic this datagram belongs to. This could then allow appropriate handling of the datagram.

A flow is defined as a sequence of datagrams which are sent from a certain host to a receiver or - in case multicasting is used - to a group of receivers, and for which the sender desires special handling by the routers along the way. The **flow label** field can then be used as an identifier for such flows.

The number of data bytes following the header is specified by the **payload length** field. This is a 16-bit wide field, so the maximum number of data bytes in a datagram is 65535. However, it is possible to create larger datagrams than this field allows. How this can be done is explained further on.

The **next header** field specifies of what type the header following the IPv6 header is. In the simplest case, this is a header from a higher level protocol. But it can also be one of the extension headers which IPv6 defines. It is because of these extension headers the IPv6 header is somewhat simpler than the header of IPv4. Some fields in the IPv4 header and the different options are now used through extension headers.

Several extension headers are defined. Fragmentation, security, authentication, source routing and many more are all made possible through these extension headers.

Earlier, I mentioned that the payload length of 65535 can be exceeded. Well, this can be done using an so-called `hop-by-hop' extension header. This header has an option called `Jumbo Payload' and allows lengths greater than 65535 to be specified. Such datagrams are often called `jumbograms'.

The **hop limit** field is a replacement for the TTL field in the IPv4 header. This field limits the lifetime of a datagram by requiring that the value in the hop limit field must be decremented by one by each node that forwards the packet.

Finally, the header contains the **source address** and the **destination address** for the datagram, which are 128-bit values.

3.6.2.2 Important changes from IPv4

First of all, there is the larger address space. The 128-bit values should be enough to continue for quite some time. On the entire planet, these addresses would allow for 7×10^{23} addresses per square meter.

Furthermore, because of the way multicast addresses are represented, the scalability of multicast routing should be improved. Also, a new type of transmission, called `anycasting', is available. This type of transmission is used to send a datagram to anyone of a group of receivers.

The header format is simpler than it was the case with IPv4. The IPv6 header has only eight fields, whereas the IPv4 header had at least twelve fields. This allows for faster processing of datagrams. The extension headers give the protocol great flexibility, certainly compared to the limited IPv4 options field.

The concept of a flow is also new to this version. This makes it possible for a certain stream of data to receive special treatment. This feature could prove to be useful for real-time services for example.

Finally, the added support for authentication and security are definitely an important improvement over version four.

4.1 PROGRAMMING LANGUAGE JAVA

The main goal of java is reducing complexity for the programmer. It goes on to wrap all the complex tasks that have become important, such as multi-threading and net work programming. It tackles some really big complexity problems, across platform programs, dynamic code changes and even security.

One of the places we see the greatest impact for this is on the web. The net work programming has always been hard, and java makes it easy (and java language designers are working on making it even easier). Net work programming is how we talk to each other more effectively and cheaper than we ever have with telephones (e-mail alone has revolutionized many businesses).

Furthermore java increases the communication bandwidth between the people.

Java is truly the tool of future communication revolution.

4.2 JAVA ENVIRONMENT

We can run Java programs on a wide variety of computers using a range of operating systems.

Java programs will run just as well on a PC running Windows 95/98/NT as it will on a Sun Solaris workstation. This Is possible because a Java program does not execute directly on your computer. It runs on a standardized hypothetical computer called the Java virtual machine which is emulated inside your computer by a program. The Java source code that you write is converted by a Java compiler to a binary program consisting of byte codes. Byte codes are machine instructions for the Java virtual machine. When we execute a Java program, a program called the Java interpreter inspects and deciphers the byte codes for it, checks it out to ensure that it h(1.s not been tampered with and is safe to execute, and then executes the actions that the byte codes specify within the Java virtual machine. A Java interpreter can run stand-alone, or it can be part of a Web browser such as Netscape Navigator or Microsoft Internet Explorer where it can be invoked automatically to run applets in a Web page.

Because a Java program consists of byte codes rather than native machine instructions, it is completely insulated from the particular hardware on which it is run. Any computer that has the Java environment implemented will work as well as any other, and because the Java interpreter sits between your program and the physical machine, it can prevent unauthorized actions in: the program from being executed.

In the past there has been a penalty for all this flexibility and protection, and that is in the speed of execution. An interpreted Java program would typically run at only one tenth of the speed of an equivalent program using native machine instructions. In programs that are not computation intensive - which is usually the case with the sort of program you would want to include in a Web page, for example, you really wouldn't notice this. .If you happen to have a Java environment which supports’ Just-In-Time' compilation of programs, we will not suffer the penalty in any event. On-the-fJy compilers convert your Java programs to native machine instructions as they are loaded. Your programs will take a little longer to load, but once loaded they execute at maximum speed.

5.1 TIME-BASED MEDIA

Any data that changes meaningfully with respect to time can be characterized as time-based media. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media. Such media data can be obtained from a variety of sources, such as local or network files, cameras, microphones, and live broadcasts.

This chapter describes the key characteristics of time-based media and describes the use of time-based media in terms of a fundamental data processing model:

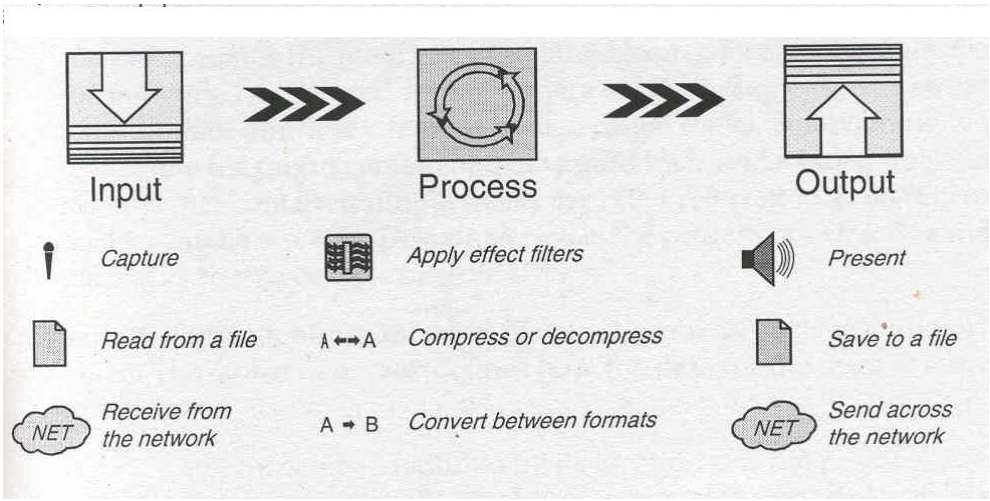

Figure 5.1

5.1.1 Streaming Media

A key characteristic of time-based media is that it requires timely delivery and processing. Once the flow of media data begins, there are strict timing deadlines that must be met, both in terms of receiving and presenting the data. For this reason, time-based media is often referred to as streaming media. It is delivered in a steady stream that must be received and processed within a particular timeframe to produce acceptable results.

For example, when a movie is played, if the media data cannot be delivered quickly enough, there might be odd pauses and delays in playback. On the

other hand, if the data cannot be received and processed quickly enough, the movie might appear jumpy as data is lost or frames are intentionally dropped in an attempt to maintain the proper playback rate.

5.1.2 Content Type

The format in which the media data is stored is referred to as its content type. QuickTime, MPEG, and WAV are all examples of content types. Content type is essentially synonymous with file type. Content type is used because media data is often acquired from sources other than local files.

5.1.3 Media Streams

A media stream is the media data obtained from a local file, acquired over the network, or captured from a camera or microphone. Media streams often contain multiple channels of data called tracks. For example, a Quicktime file might contain both an audio track and a video track. Media streams that contain multiple tracks are often referred to as multiplexed or complex media streams. Demultiplexing is the process of extracting individual tracks from a complex media stream. A track's type identifies the kind of data it contains, such as audio or video. The format of a track defines how the data for the track is structured. A media stream can be identified by its location and the protocol used to access it. For example, a URL might be used to describe the location of a QuickTime file on a local or remote system. If the file is local, it can be accessed through the FILE protocol. On the other hand, if it's on a web server, the file can be accessed through the HTTP protocol. A media locator provides a way to identify the location of a media stream when a URL can't be used. Media streams can be categorized according to how the data is delivered:

- Pull-data transfer is initiated and controlled from the client side. For example, Hypertext Transfer Protocol (HTTP) and FILE are pull protocols.
- Push-the server initiates data transfer and controls the flow of data. or example, Real-time Transport Protocol (RTP) is a push protocol used for streaming media. Similarly, the SGI MediaBase protocol is a Push protocol used for video-on-demand (VOD).

5.1.4 Common Media Formats

The following tables identify some of the characteristics of common media formats. When selecting a format, it’s important to take into account the characteristics of the format, the target environment, and the expectations of the intended audience.

The CPU Requirements column characterizes the processing power necessary for optimal presentation of the specified format. The Bandwidth Requirements column characterizes the transmission speeds necessary to send or receive data quickly enough for optimal presentation.

Format	Content Type	Quality	CPU Requirements	Bandwidth Requirements
PCM	AVI QuickTime WAV	High	Low	High
Mu-Law	AVI QuickTime WAV RTP	Low	Low	High
ADPCM (DVI, IMA4)	AVI QuickTime WAV RTP	Medium	Medium	Medium
MPEG-1	MPEG	High	High	High
MPEG Layer3	MPEG	High	High	Medium
GSM	WAV RTP	Low	Low	Low
G.723.1	WAV RTP	Medium	Medium	Low

Table 5.1

5.1.5 Media Presentation

Most time-based media is audio or video data that can be presented through output devices such as speakers and monitors. Such devices are the most common destination for media data output. Media streams can also be sent to other destinations, for example, saved to a fie or transmitted across the network. An output destination for media data is some times referred to as a data sink.

5.1.5.1 Presentation Controls

While a media stream is being presented, VCR-style presentation controls are often provided to enable the user to control playback. For example, a control panel for a movie player might offer buttons for stopping, starting, fast-forwarding, and rewinding the movie.

5.1.5.2 Latency

In many cases, particularly when presenting a media stream that resides on the network, the presentation of the media stream cannot begin immediately. The time it takes before presentation can begin is referred to as the start latency. Users might experience this as a delay between the time that they click the start button and the time when playback actually starts.

Multimedia presentations often combine several types of time-based media into a synchronized presentation. For example, background music might be played during an image slide-show, or animated text might be synchronized with an audio or video clip. When the presentation of multiple media streams is synchronized, it is essential to take into account the start latency of each stream, otherwise the playback of the different streams might actually begin at different times.

5.1.5.3 Presentation Quality

The quality of the presentation of a media stream depends on several factors, including:

- The compression scheme used
- The processing capability of the playback system
- The bandwidth available (for media streams acquired over the network)

Traditionally, the higher the quality, the larger the file size and the greater the processing power and bandwidth required. Bandwidth is usually represented as the number of bits that are transmitted in a certain period of time, the bit rate.

To achieve high-quality video presentations, the number of frames displayed in each period of time (the frame rate) should be as high as possible. Usually movies at a frame rate of 30 frames-per-second are considered indistinguishable from regular TV broadcasts or video tapes.

5.1.6 Media Processing

In most instances, the data in a media stream is manipulated before it is presented to the user. Generally, a series of processing operations occur before presentation:

- If the stream is multiplexed, the individual tracks are extracted.
- If the individual tracks are compressed, they are decoded.
- If necessary, the tracks are converted to a different format.
- Effect filters are applied to the decoded tracks (if desired).

The tracks are then delivered to the appropriate output device. If the media stream is to be stored instead of rendered to an output device, the processing stages might differ slightly. For example, if you wanted to capture audio and video from a video camera, process the data, and save it to a file:

- The audio and video tracks would be captured.
- Effect filters would be applied to the raw tracks (if desired).

- The individual tracks would be encoded.
- The compressed tracks would be multiplexed into a single media stream.
- The multiplexed media stream would then be saved to a file.

5.1.6.1 Demultiplexers and Multiplexers

A demultiplexer extracts individual tracks of media data from a multiplexed media stream. A mutliplexer performs the opposite function, it takes individual tracks of media data and merges them into a single multiplexed media stream.

5.1.6.2 Codecs

A codec performs media-data compression and decompression. When a track is encoded, it is converted to a compressed format suitable for storage or transmission; when it is decoded it is converted to a non-compressed (raw) format suitable for presentation.

Each codec has certain input formats that it can handle and certain output formats that it can generate. In some situations, a series of codecs might be used to convert from one format to another.

5.1.6.3 Effect Filters

An effect filter modifies the track data in some way, often to create special effects such as blur or echo. Effect filters are classified as either pre-processing effects or post-processing effects, depending on whether they are applied before or after the codec processes the track. Typically, effect filters are applied to uncompressed (raw) data.

5.1.6.4 Renderers

A renderer is an abstraction of a presentation device. For audio, the presentation device is typically the computer’s hardware audio card that outputs sound to the speakers. For video, the presentation device is typically the computer monitor.

5.1.6.4.1 Compositing

Certain specialized devices support compositing. Compositing time-based media is the process of combining multiple tracks of data onto a single presentation medium. For example, overlaying text on a video presentation is one common form of compositing. Compositing can be done in either hardware or software. A device that performs compositing can be abstracted as a renderer that can receive multiple tracks of input data.

5.2 Working with Real Time Media Stream

To send or receive a live media broadcast or conduct a video conference over the Internet or Intranet, you need to be able to receive and transmit media streams in real-time. This chapter introduces streaming media concepts and describes the Real-time Transport Protocol JMF uses for receiving and transmitting media streams across the network.

5.2.1 Streaming Media

When media content is streamed to a client in real-time, the client can begin to play the stream without having to wait for the complete stream to download. In fact, the stream might not even have a predefined duration downloading the entire stream before playing it would be impossible. The term *streaming media* is often used to refer to both this technique of delivering content over the network in real-time and the real-time media content that's delivered.

Streaming media is everywhere you look on the web-live radio and television broadcasts and web cast concerts and events are being offered by a rapidly growing number of web portals, and it’s now possible to conduct audio and video conferences over the Internet. By enabling the delivery of dynamic, interactive media content across the network, streaming media is changing the way people communicate and access information.

5.2.1.1 Protocols for Streaming Media

Transmitting media data across the net in real-time requires high network throughput. It’s easier to compensate for lost data than to compensate for large delays in receiving the data. This is very different from accessing static

data such as a file, where the most important thing is that all of the data arrive at its destination. Consequently, the protocols used for static data don't work well for streaming media.

The HTTP and FTP protocols are based on the Transmission Control Protocol (TCP). TCP is a transport-layer protocol designed for reliable data communications on low-bandwidth, high-error-rate networks. When a packet is lost or corrupted, it's retransmitted. The overhead of guaranteeing reliable data transfer slows the overall transmission rate.

For this reason, underlying protocols other than TCP are typically used for streaming media. One that's commonly used is the User Datagram Protocol (UDP). UDP is an unreliable protocol; it does not guarantee that each packet will reach its destination: There's also no guarantee that the packets will arrive in the order that they were sent. The receiver has to be able to compensate for lost data, duplicate packets, and packets that arrive out of order.

Like TCP, UDP is a general transport-layer protocol-a lower-level networking protocol on top of which more application-specific protocols are built. The Internet standard for transporting real-time data such as audio and video is the Real- Time Transport Protocol (RTP).

RTP is defined in IETF RFC 1889, a product of the AVT working group of the Internet engineering Task Force (IETF).

5.2.2 Real Time Transport Protocol

RTP provides end-to-end network delivery services for the transmission of real-time data.. RTP is network and transport-protocol independent, though it is often used over UDP.

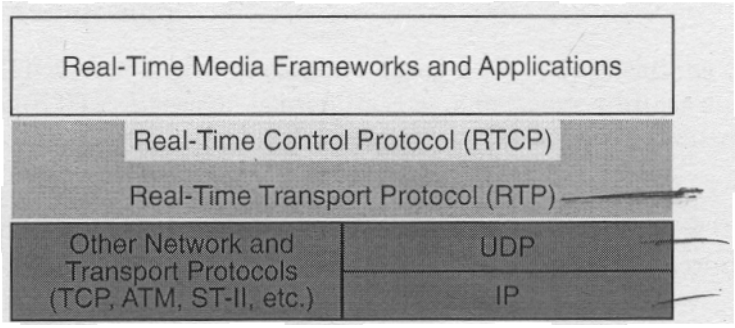

Figure 5.1: RTP architecture.

RIP can be used over both unicast and multicast network services. Over a unicast network service, separate copies of the data are sent from the source to each destination. Over a multicast network service, the data is sent from the source only once and the network is responsible for transmitting the data to multiple locations. Multicasting is more efficient for many multimedia applications, such as video conferences. The standard Internet Protocol (IP) supports multicasting.

5.2.2.1 RTP Services

RTP enables you to identify the type of data being transmitted, determine what order the packets of data should be presented in, and synchronize media streams from different sources.

RTP data packets are not guaranteed to arrive in the order that they were sent-in fact, they're not guaranteed to arrive at all. It's up to the receiver to reconstruct the sender's packet sequence and detect lost packets using the information provided in the packet header.

While RTP does not provide any mechanism to ensure timely delivery or provide other quality of service guarantees, it is augmented by a control protocol (RTCP) that enables you to monitor the quality of the data distribution. RTCP also provides control and identification mechanisms for RTP transmissions.

If quality of service is essential for a particular application, RTP can be used over a resource reservation protocol that provides connection-oriented services.

5.3 Transmitting RTP Media Streams

To transmit an RTP stream, you use a Processor to produce an RTP-encoded **DataSource** and construct either a **SessionManager** to control the transmission.

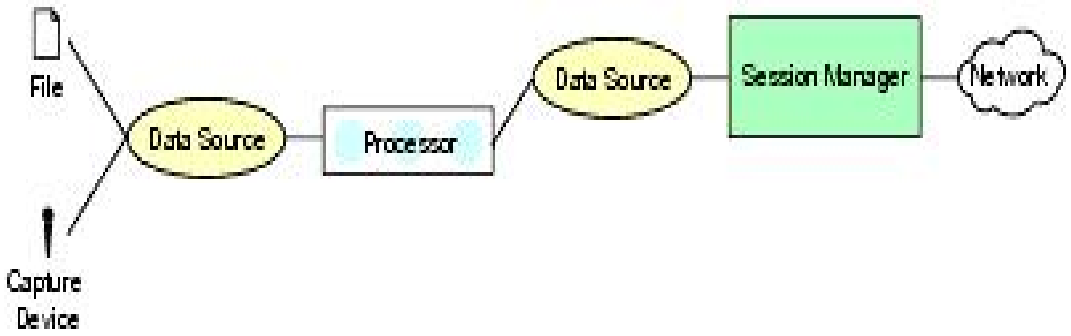

Figure 5.2 Transmitting RTP Stream

The input to the Processor can be either stored or live captured data. For stored data, you can use a **MediaLocator** to identify the file when you create the Processor. For captured data, a capture **DataSource** is used as the input to the Processor.

There are two ways to transmit RTP streams:

- Use a **MediaLocator** that has the parameters of the RTP session to con-
  struct an RTP **DataSink** by calling **Manager.createDataSink**.
- Use a session manager to create send streams for the content and control the transmission.

To transmit multiple RTP streams in a session or need to monitor session statistics, you need to use the **SessionManager** directly.
Regardless of how you choose to transmit the RTP stream, you need to:

- Create a Processor with a **DataSource** that represents the data you want
  to transmit.
- Configure the Processor to output RTP-encoded data.
- Get the output from the Processor as a **DataSource**.

5.3.1 Media Capture

Time-based media can be captured from a live source for processing and playback. For example, audio can be captured from a microphone or a video

capture card can be used to obtain video from a camera. Capturing can be thought of as the input phase of the standard media processing model.

A capture device might deliver multiple media streams. For example, a video camera might deliver both audio and video. These streams might be captured and manipulated separately or combined into a single, multiplexed stream that contains both an audio track and a video track.

5.3.1.1 Capture Devices

To capture time-based media you need specialized hardware, for example, to capture audio from a live source, you need a microphone and an appropriate audio card. Similarly, capturing a TV broadcast requires a TV tuner and an appropriate video capture card. Most systems provide a query mechanism to find out what capture devices are available.

Capture devices can be characterized as either push or pull sources. For example, a still camera is a pull source, the user controls when to capture an image. A microphone is a push source, the live source continuously provides a stream of audio. The format of a captured media stream depends on the processing performed by the capture device. Some devices do very little processing and deliver raw, uncompressed data. Other capture devices might deliver the data in a compressed format.

5.3.1.2 Capture Controls

Controls are sometimes provided to enable the user to manage the capture process. For example, a capture control panel might enable the user to specify the data rate and encoding type for the captured stream and start and stop the capture process.

5.3.2 PROCESSOR

Processors can also be used to present media data. A Processor is just a specialized type of Player that provides control over what processing is performed on the input media stream. A Processor supports all of the same presentation controls as a Player.

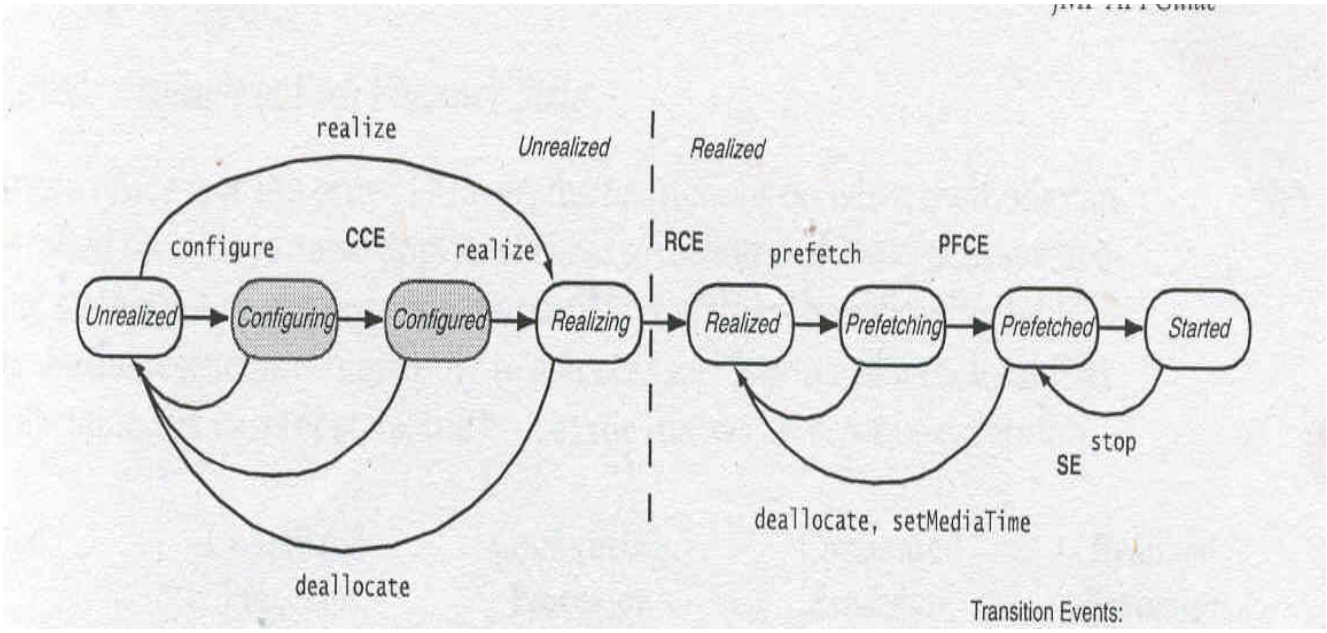

Figure 5.3: JMF processor model

In addition to rendering media data to presentation devices, a Processor can output media data through a **DataSource** so that it can be presented by another Player or Processor, further manipulated by another Processor, or delivered to some other destination, such as a file.

5.3.2.1 Presentation Controls

In addition to the standard presentation controls defined by Controller, a Player or Processor might also provide a way to adjust the playback volume. If so, you can retrieve its **GainControl** by calling **getGainContro**l. A **GainControl** object posts a **GainChangeEvent** whenever the gain is

modified. By implementing the **GainChangeListener** interface, you can respond to gain changes. For example, you might want to update a custom gain control Component. Additional custom Control types might be supported by a particular Player or Processor implementation to provide other control behaviors and expose custom user interface components. You access these controls through the **getControls** method. For example, the CachingControl interface extends Control to provide a mechanism for displaying a download progress bar. If a Player can report its download progress, it implements this interface. To find out if a Player supports **CachingControl**, you can call **getControl** (CachingControl) or use **getControls** to get a list of all the supported Controls.

A Processor generally provides two standard user interface components, a visual component and a control-panel component.You can access these Components directly through the **getVisualComponent** and **getControlPanelComponent** methods.

5.3.2.2 Processing

A Processor is a Player that takes a DataSource as input, performs some user-defined processing on the media data, and then outputs the processed media data.

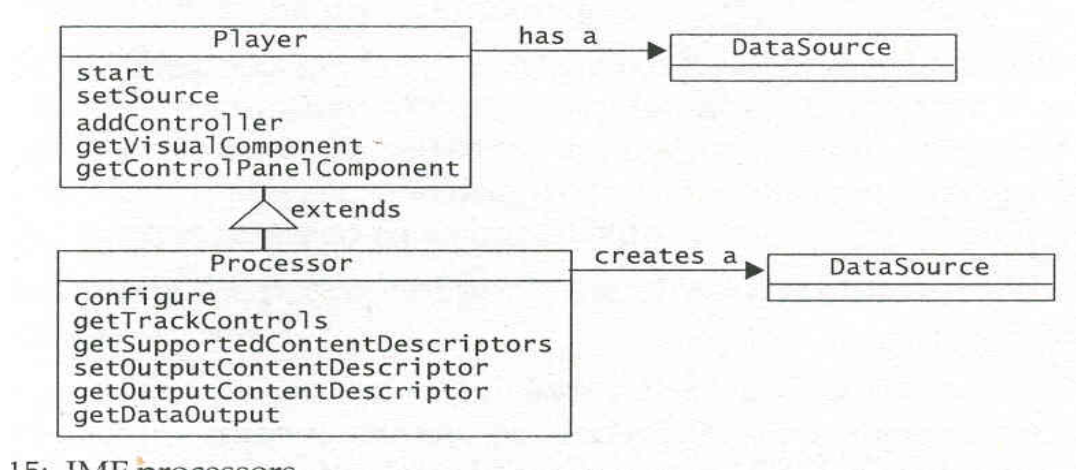

Figure 5.4 JMF processors

A Processor can send the output data to a presentation device or to a DataSource. If the data is sent to a DataSource, that DataSource can be used as the input to another Player or Processor.

While the processing performed by a Player is predefined by the implementor, a Processor allows the application developer to define the type of processing that is applied to the media data. This enables the application of effects, mixing, and compositing in real-time.

The processing of the media data is split into several stages:

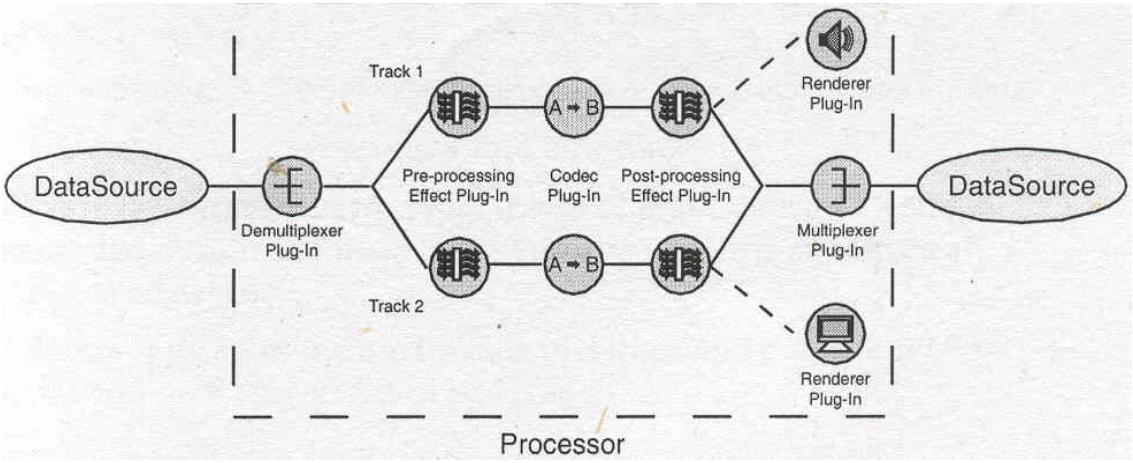

Figure 5.5 Processor stages.

- Demultiplexing is the process of parsing the input stream. If the stream contains multiple tracks, they are extracted and output separately. For example, a QuickTime file might be demultiplexed into separate audio and video tracks. Demultiplexing is performed automatically whenever the input stream contains multiplexed data.

- Pre-Processing is the process of applying effect algorithms to the tracks extracted from the input stream.

- Transcoding is the process of converting each track of media data from one input format to another. When a data stream is converted from a compressed type to an uncompressed type, it is generally referred to as decoding. Conversely, converting from an uncompressed type to a compressed type is referred to as encoding.

- Post-Processing is the process of applying effect algorithms to decoded tracks.

- Multiplexing is the process of interleaving the transcoded media tracks into a single output stream. For example, separate audio and video tracks might be multiplexed into a single MPEG-1 data stream. You can specify the data type of the output stream with the Processor **setOutputContentDescriptor** method.

- Rendering is the process of presenting the media to the user.

The processing at each stage is performed by a separate processing component. These processing components are JMF plug-ins. If the Processor supports **TrackControls**, you can select which plug-ins you want to use to process a particular track. There are types of JMF plug-ins:

- **Demultiplexer** parses media streams such as WAV, MPEG or QuickTime. If the stream is multiplexed, the separate tracks are extracted.
- Effect performs special effects processing on a track of media data.
- Codec performs data encoding and decoding.
- Multiplexer combines multiple tracks of input data into a single interleaved output stream and delivers the resulting stream as an output **dataSource**.
- **Renderer** processes the media data in a track and delivers it to a destination such as a screen or speaker.

5.3.2.3 Processor States

A Processor has two additional standby states, Configuring and Configured, which occur before the Processor enters the Realizing state.

- A Processor enters the Configuring state when configure is called. While the Processor is in the Configuring state, it connects to the DataSource, demultiplexes the input stream, and accesses information about the format of the input data.

- The Processor moves into the Configured state when it is connected to the DataSource and data format has been determined. When the Processor reaches the Configured state, a **ConfigureCompleteEvent** is posted.

- When Realize is called, the Processor is transitioned to the Realized state. Once the Processor is Realized it is fully constructed.

While a Processor is in the Configured state, **getTrackControls** can be called to get the **TrackControl** objects for the individual tracks in the media stream. These **TrackControl** objects enable you specify the media processing operations that you want the Processor to perform.

Calling realize directly on an Unrealized Processor automatically transitions it through the Configuring and Configured states to the Realized state. When you do this, you cannot configure the processing options through the **TrackControls,** the default Processor settings are used.

Calls to the **TrackControl** methods once the Processor is in the Realized state will typically fail, though some Processor implementations might support them.

Since a Processor is a type of Player, the restrictions on when methods can be called on a Player also apply to Processors. Some of the Processor-specific methods also are restricted to particular states.

5.3.2.4 Configuring the Processor

To configure the Processor to generate RTP-encoded data, you set RTP specific formats for each track and specify the output content descriptor you want. The track formats are set by getting the **TrackControl** for each track and calling **setFormat** to specify an RTP-specific format. An RTP specific format is selected by setting the encoding string of the format to an RTP specific string such as "**AudioFormat.GSM_RTP**". The Processor attempts to load a plug-in that supports this format. If no appropriate plug-in is installed, that particular RTP format cannot be supported and an **UnSupportedFormatException** is thrown.

The output format is set with the **setOutputContentDescriptor** method.If no special multiplexing is required, the output content descriptor can be set to "**ContentDescriptor.RAW**". Audio and video streams should not be interleaved. If the Processor's tracks are of different media types, each media stream is transmitted in a separate RTP session.

5.3.2.5 Retrieving the Processor Output

Once the format of a Processor's track has been set and the Processor has been realized, the output DataSource of the Processor can be retrieved. You retrieve the output of the Processor as a DataSource by calling **getDataOutput**. The returned DataSource can be either a **PushBufferDataSource** or a **PullBufferDataSource**, depending on the source of the data. The output DataSource is connected to the **SessionManager** using the **createSendStream** method. The session manager must be initialized before you can create the send stream. If the DataSource contains multiple SourceStreams, each SourceStream is sent out as a separate RTP stream, either in the same session or a different session. If the DataSource contains both audio and video streams, separate RTP sessions must be created for audio and video. You can also clone the DataSource and send the clones out as different RTP streams in either the same session or different sessions.

5.3.3 Controlling the Packet Delay

The packet delay, also known as the packetization interval, is the time represented by each RTP packet as it is transmitted over the network. The packetization interval determines the minimum end-to-end delay; longer packets introduce less header overhead but higher delay and make packet loss more noticeable. For non-interactive applications such as lectures, or for links with severe bandwidth constraints, a higher packetization delay might be appropriate. A receiver should accept packets representing between 0 and 200 ms of audio data. (For framed audio encodings, a receiver should accept packets with 200 ms divided by the frame duration, rounded up.) This restriction allows reasonable buffer sizing for the receiver. Each packetizer codec has a default packetization interval appropriate for its encoding. If the codec allows modification of this interval, it exports a corresponding **PacketSizeControl**. The packetization interval can be changed or set by through the **setPacketSize** method.

5.3.4 Transmitting RTP Data with the Session Manager

The basic process for transmitting RTP data with the session manager is:

- Create a JMF Processor and set each track format to an RTP-specific format.
- Retrieve the output DataSource from the Processor.
- Call createSendStream on a previously created and initialized **SessionManager**, passing in the DataSource and a stream index. The session
  manager creates a SendStream for the specified SourceStream.
- Start the session manager by calling SessionManager startSession.
- Control the transmission through the SendStream methods. A **SendStreamListener** can be registered to listen to events on the SendStream.

5.3.4.1 Creating a Send Stream

Before the session manager can transmit data, it needs to know where to get the data to transmit. When you construct a new SendStream, you hand the **SessionManager** the DataSource from which it will acquire the data. Since a DataSource can contain multiple streams, you also need to specify the index of the stream to be sent in this session. You can create multiple send streams by passing different DataSources to **createSendStream** or by specifying different stream indexes. The session manager queries the format of the SourceStream to determine if it has a registered payload type for this format. If the format of the data is not an RTP format or a payload type cannot be located for the RTP format, an **UnSupportedFormatException** is thrown with the appropriate message. Dynamic payloads can be associated with an RTP format using the **SessionManager addFormat** method.

5.3.4.1.1 Using Cloneable Data Sources.

Many RTP usage scenarios involve sending a stream over multiple RTP sessions or encoding a stream into multiple formats and sending them over multiple RTP sessions. When a stream encoded in a single format has to be

sent over multiple RTP sessions, you need to clone the DataSource output from the Processor from which data is being captured. This is done by creating a cloneable DataSource through the Manager and calling getClone on the cloneable DataSource. A new Processor can be created from each cloned DataSource, its tracks encoded in the desired format, and the stream sent out over an RTP session.

5.3.4.2 Controlling a Send Stream

You use the RTPStream start and stop methods to control a SendStream. Starting a SendStream begins data transfer over the network and stopping a SendStream indicates halts the data transmission. To begin an RTP transmission, each SendStream needs to be started. Starting or stopping a send stream triggers the corresponding action on its DataSource. However, if the DataSource is started independently while the SendStream is stopped, data will be dropped (**PushBufferDataSource**) or not pulled (**PullBufferDataSource**) by the session manager. During this time, no data will be transmitted over the network.

5.4 RECEIVING AND PRESENTING RTP MEDIA STREAMS

JMF Players and Processors provide the presentation, capture, and data conversion mechanisms for RTP streams.

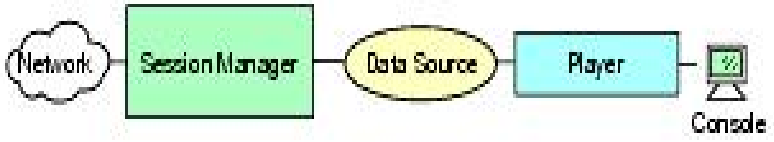

Figure 5.6 RTP reception data flow.

A separate player is used for each stream received by the session manager. You construct a Player for an RTP stream through the standard **ManagercreatePlayer** mechanism. You can either:

- Use a **MediaLocator** that has the parameters of the RTP session and construct a Player by calling **Manager.createPlayer(MediaLocator).**

- Construct a Player for a particular ReceiveStream by retrieving the DataSource from the stream and passing it to **Manager.createPlayer(DataSource**).

If you use a **MediaLocator** to construct a Player, you can only present the first RTP stream that's detected in the session. If you want to play back multiple RTP streams in a session, you need to use the **SessionManager** directly and construct a Player for each **ReceiveStream**.

5.4.1 Player

To play a media stream, you need to construct a Player for the stream, configure the Player and prepare it to run, and then start the Player to begin playback.

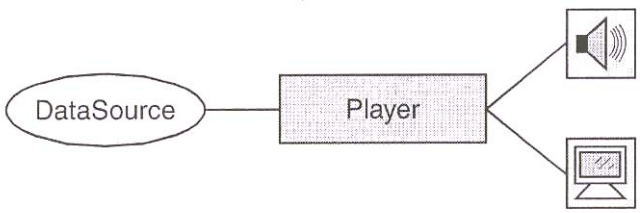

Figure 5.7 Player Model

5.4.1.1 Creating a Player

You create a Player indirectly through the media Manager. To display the Player, you get the Player object's components and add them to your applet's presentation space or application window. When you need to create

a new Player, you request it from the Manager by calling **createPlayer**. The Manager uses the media URL or **MediaLocator** that you specify to create an appropriate Player. A URL can only be successfully constructed if the appropriate corresponding URL **StreamHandler** is installed. **MediaLocator** doesn't have this restriction.

5.4.1.1.1 Blocking Until a Player is Realized

Many of the methods that can be called on a Player require the Player to be in the Realized state. One way to guarantee that a Player is Realized when you call these methods is to use the **Manager createRealizedPlayer** method to construct the Player. This method provides a convenient way to create and realize a Player in a single step. When this method is called, it blocks until the Player is Realized. Manager provides an equivalent **createRealizeProcessor** method for constructing a Realized Processor.

Note: Be aware that blocking until a Player or Processor is Realized can produce unsatisfactory results. For example, if **createRealizedPlayer** is called in an applet, **Applet.start** and **Applet.stop** will not be able to interrupt the construction process.

A Player processes an input stream of media data and renders it at a precise time. A DataSource is used to deliver the input media-stream to the Player.The rendering destination depends on the type of media being presented.

A Player does not provide any control over the processing that it performs or how it renders the media data. Player supports standardized user control and relaxes some of the operational restrictions imposed by Clock and Controller.

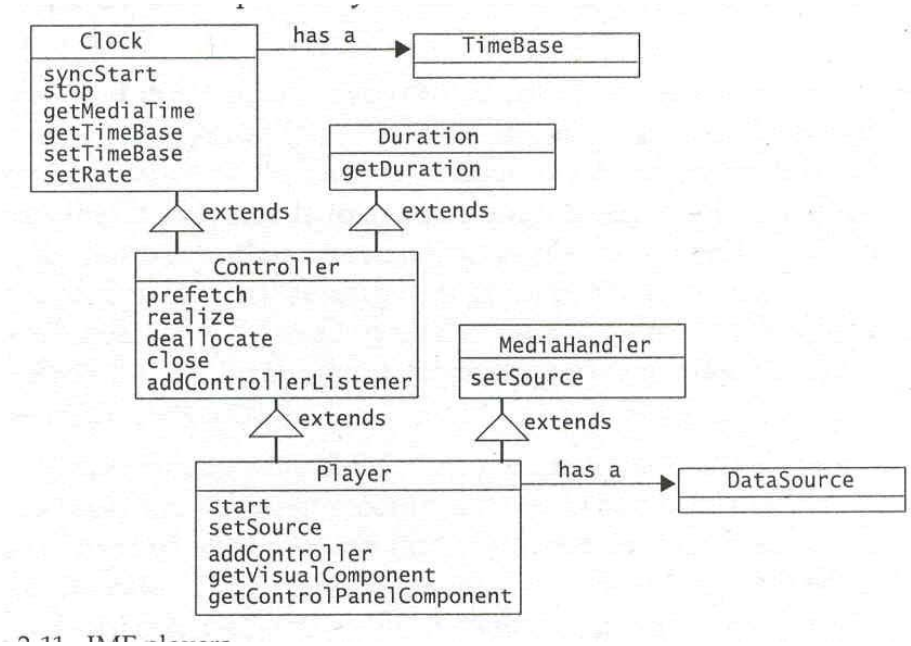

Figure 5.8 JMF players.

5.4.1.2 Player States

A Player can be in one of six states. The Clock interface defines the two primary states: Stopped and Started. To facilitate resource management, Controller breaks the Stopped state down into five standby states: Unrealized, Realizing, Realized, Prefetching, and Prefetched.

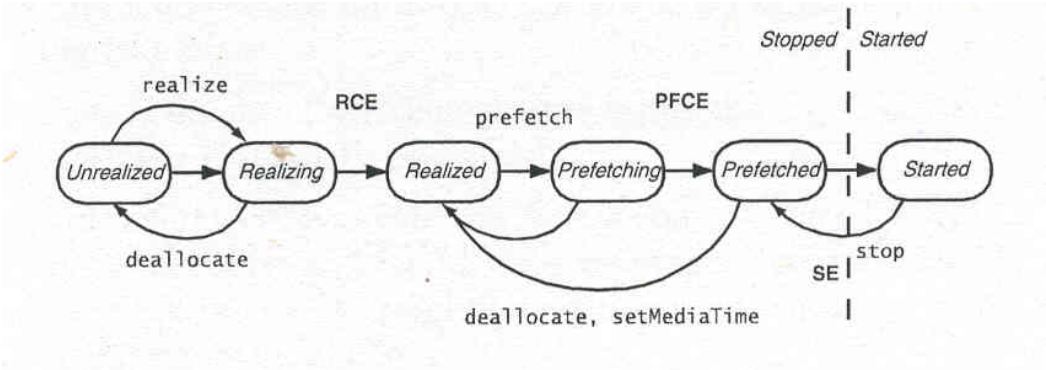

Figure 5.9 Player States

In normal operation, a Player steps through each state until it reaches the Started state:

- A Player in the Unrealized state has been instantiated, but does not yet know anything about its media. When a media Player is first created, it is Unrealized.

- When realize is called, a Player moves from the Unrealized state into the Realizing state. A Realizing Player is in the process of determining its resource requirements. During realization, a Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources. (Exclusive use resources are limited resources such as particular hardware devices that can only be used by one Player at a time; such resources are acquired during Prefetching.) A Realizing Player often downloads assets over the network.

- When a Player finishes Realizing, it moves into the Realized state. A Realized Player knows what resources it needs and information about the type of media it is to present. Because a Realized Player knows how to render its data, it can provide visual components and controls. Its connections to other objects in the system are in place, but it does not own any resources that would prevent another Player from starting.

- When prefetch is called, a Player moves from the Realized state into the Prefetching state. A Prefetching Player is preparing to present its media. During this phase, the Player preloads its media data, obtains exclusive-use resources, and does whatever else it needs to do to prepare itself to play. Prefetching might have to recur if a Player object’s media presentation is repositioned, or if a change in the Player object’s rate requires that additional buffers be acquired or alternate processing take place.

- When a Player finishes Prefetching, it moves into the Prefetched state. A Prefetched Player is ready to be started.

- Calling start puts a Player into the Started state. A Started Player object's time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

A Player posts **TransitionEvents** as it moves from one state to another. The **ControllerListener** interface provides a way for your program to determine what state a Player is in and to respond appropriately. For example, when your program calls an asynchronous method on a Player. It needs to listen for the appropriate event to determine when the operation is complete.

Using this event reporting mechanism, you can manage a Player object's start latency by controlling when it begins Realizing and Prefetching. It also enables you to determine whether or not the Player is in an appropriate state before calling methods on the Player.

5.4.2 Creating a Player for an RTP Session

When you use a **MediaLocator** to construct a Player for an RTP session, the Manager creates a Player for the first stream detected in the session. This Player posts a **RealizeCompleteEvent** once data has been detected in the session.

By listening for the RealizeCompleteEvent, you can determine whether or not any data has arrived and if the Player is capable of presenting any data. Once the Player posts this event, you can retrieve its visual and control components.

Note: Because a Player for an RTP media stream doesn't finish realizing until data is detected in the session, you shouldn't try to use **Manager.createRealizedPlayer** to construct a Player for an RTP media stream. No Player would be returned until data arrives and if no data is detected, attempting to create a Realized Player would block indefinitely.

5.4.2.1 Creating an RTP Player for Each New Receive Stream

To play all of the **ReceiveStreams** in a session, you need to create a separate Player for each stream. When a new stream is created, the session manager posts a **NewReceiveStreamEvent**. Generally, you register as a **ReceiveStreamListener** and construct a Player for each new **ReceiveStream**. To construct the Player, you retrieve the DataSource from the **ReceiveStream** and pass it to **Manager.createPlayer**.

To create a Player for each new receive stream in a session:

- Set up the RTP session
  (a)Create a **SessionManager**. For example, construct an instance of **com.sun.media.rtp.RTPSessionMgr**. (**RTPSessionMgr** is an implementation of **SessionManager** provided with the JMF reference implementation)

  (b)Call RTPSessionMgr **addReceiveStreamListener** to register as a listener.

  (c)Initialize the RTP session by calling **RTPSessionMgr initSession**.

  (d)Start the RTP session by calling **RTPSessionMgr startSession**.

- In your **ReceiveStreamListener** update method, watch for **NewReceiveStreamEvent**, which indicates that a new data stream has been detected.

- When a **NewReceiveStreamEvent** is detected, retrieve the **ReceiveStream** from the **NewReceiveStreamEvent** by calling **getReceiveStream**.

- Retrieve the RTP DataSource from the **ReceiveStream** by calling **getDataSource**. This is a **PushBufferDataSource** with an RTP specific Format. For example, the encoding for a DVI audio player will be DVI_RTP.

- Pass the DataSource to Manager.createPlayer to construct a Player. For the Player to be successfully constructed, the necessary plug-ins for decoding and depacketizing the RTP-formatted data must be available.

5.4.3 Controlling Buffering of Incoming RTP Streams

You can control the RTP receiver buffer through the **BufferControl** exported by the **SessionManager**. This control enables you to set two parameters, buffer length and threshold.

The buffer length is the size of the buffer maintained by the receiver. The threshold is the minimum amount of data that is to be buffered by the control before pushing data out or allowing data to be pulled out (jitter buffer). Data will only be available from this object when this minimum threshold has been reached. If the amount of data buffered falls below this threshold, data will again be buffered until the threshold is reached.

The buffer length and threshold values are specified in milliseconds. The number of audio packets or video frames buffered depends on the format of the incoming stream. Each receive stream maintains its own default and maximum values for both the buffer length and minimum threshold. (The default and maximum buffer lengths are implementation dependent.)

To get the **BufferControl** for a session, you call **getControl** on the **SessionManager**. You can retrieve a GUI Component for the **BufferControl** by calling **getControlComponent**.

6.1 DIFFICULTIES

The main difficulties in this project have been to familiarize oneself with the JMF API and to set up the working environment properly.

Regarding JMF the big problem has been understanding how to use it, dissect the class relationships of the framework, and realize what classes that constitute the frame of the framework. JMF consists of classes that are kind of core classes (e.g. the static Manager class) that one almost never can do without, and the relationship between these classes follow some kind of pattern. It requires hard work from each student to dissolve the JMF API and understand it to the grade that it can be used successfully. Therefore (to gain valuable time) we think that it would have been appropriate to have some introduction lecture in the course that handles JMF (just a suggestion). Such a lecture would ease much of the burden for the students.

6.2 FUTURE WORK

Future Work will be on incremental refinement of the design and the application program. The first step will be on creating a video conferencing program and there after put things together to get an working audio/video conferencing tool. The future product will be based on a (hopefully) good design and above other things support the user with an graphical user interface (GUI) by which the user can controll the program and session(s). The futureproduct will also handle packet loss in some way, and fix lacks that is mentioned elsewhere in the report.

6.3 CONCLUSION

A real-time audio conference application must be able to capture sound, digitizing it (a process known as digital signal processing), packetise it in appropriate network packets, stream the data on to the network, receive data from the network, process it, and finally present it to the receiver on an appropriate output device. Moreover the network must be able to transmitt the packets to the destination(s) which is not in the application's scope of concern. However the application should be able to detect packet loss when the network used is not relable (such as an IP network) and in some way

repair the loss. To get real-time functionality the application also must be able to do much of this in parallell. Finally the program should provide the user with a GUI by which the program can be controlled.

Many of the things just listed above is supported by our audio conferencing program. The program lacks some functionality that will be implemented in the future product (a real-time audio/video conferencing tool).

The project has been very fun and interesting. Learning JMF has been especially rewarding. As so many time before the time has however been sparse.