Analysis of Real-time Fax over IP (FoIP) Using Simulation

By
Masood Khosroshahy

A thesis presented to
Iran University of Science and Technology (IUST)
in fulfillment of the thesis requirement
for the degree of
Bachelor of Science
in
Electrical Engineering
(Communications field)

Thesis Supervisor:
Dr. B.Abolhassani

www.m-kh.info

Abstract

It's been some time now that expressions like "Voice over IP", "Fax over IP" and the likes are heard extensively in the telecommunications industry. The idea is utilizing data networks to deliver telecommunications services which are currently provided by the PSTN. The incentive is pretty straightforward: cutting costs and yet being able to provide the previous services, not to mention the added capabilities to deliver a multitude of other services, hardly imagined feasible with the PSTN.

When considering the implementation of the aforementioned objective, one faces a lot of difficulties. Simply put, the current data networks, e.g. the Internet, have not been designed with telecommunications services in mind. They have been optimized to carry data which is bursty in nature. This design is in obvious contradiction to the requirements of the telecommunications services, one of which is fax. In this thesis, Fax over Internet protocol (FoIP) is being considered which has two possible approaches to be accomplished: Real-time and Store-and-Forward. Real-time approach is the ultimate goal since it is the real-time faxing which makes the transition from the PSTN to the Internet-based architecture smooth.

Signaling comprises initiation, management and tear-down of sessions examples of which are fax, voice, video and the like. Currently there are two protocols that can provide an end-to-end solution: H.323 and Session Initiation Protocol (SIP). SIP is the protocol of choice among other standards in the voice and fax transmission domains due to its numerous advantages.

In this thesis, we intend to closely examine some aspects of the new architecture and its implementation feasibility. Different components of the real-time Fax over IP architecture are analyzed and we pay a particular attention to the signaling part. Utilization of SIP and SDP, a companion protocol to SIP for capabilities exchange, in fax transmission is studied. What we intend to do is exploring whether fax parameters details can be negotiated using SIP/SDP. Session establishment, starting a sample file transfer, which can act on behalf of real-time fax transfer, and the subsequent session tear-down, after file transfer is complete, are demonstrated. This simulation scenario and its results exhibit the potential success of the proposed SIP/SDP combination for real-time fax session establishment, management and tear-down.

Another important analysis carried out in this thesis is the utilization of SIP contact header for reducing the load on proxy servers which is a highly desirable feature.

Acknowledgment

I would like to express my deepest gratitude to Dr Abolhassani, my thesis supervisor, for trusting me and giving me the opportunity to explore the subject in its entirety and as I wished. His satisfaction with the outcome virtually boosted my self-confidence in pursuing my research interests.

Dedication

To all who value love more than anything else,

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

"Voice over IP", "Fax over IP" and the likes are gradually becoming the next big cutting-edge technologies in the telecommunications industry [1], [2]. These are meant to replace the traditional method of delivery of telecommunications services by Public Switched Telephone Network (PSTN) through utilization of data networks e.g. the Internet. By doing so, both the telecommunication service providers and the users can save fortunes, not to mention the newly-presented capabilities to deliver a multitude of other services. Venturing into the actual implementation has proved to be a hard-to-overcome challenge and a plethora of standards are still being considered to make the new architecture a reality.

1.2 Existing Problem

When considering the implementation of the aforementioned objective, one faces a lot of difficulties. Simply put, the current data networks, e.g. the Internet, have not been designed with telecommunications services in mind. They have been optimized to carry data which is bursty in nature. With

bursty, we mean a discrete series of packets of data which travel through the net from a source to a destination with frequent idle times in transmission. It is not strictly continuous and generally the users don't mind the jitter and extended delays of the data packets. These are in obvious contradiction to the requirements of the telecommunications services. Specifically, they stipulate the existence of a network infrastructure which is either connection-oriented in nature or at least can resemble its behaviors and therefore is able to guarantee a stream of data free of any kind of interruption.

One of these telecommunication services is Fax. Fax over Internet protocol (FoIP) is being considered in this thesis but needless to say that most of the technologies are shared with Voice over IP (VoIP). There are two possible approaches to accomplish faxing over the internet protocol: Real-time and Store-and-Forward. Store-and-Forward or non-real-time usually uses E-mail capabilities to transfer fax between the end-points. In real-time approach, as the name suggests, fax is transferred in real-time manner and without delay; like the way we currently send fax using the PSTN. Real-time approach is the ultimate goal since it is the real-time faxing which makes the transition from the PSTN to the Internet-based architecture smooth.

Signaling as one can imagine is the most important part of any session initiation, management and tear-down whether it is fax, voice, video or the like. Currently there are two protocols that can provide an end-to-end solution: H.323 and Session Initiation Protocol (SIP). H.323 is a binary protocol which consists of a complex suite of protocols that reuse many older services and methods borrowed from Integrated Services Digital Network (ISDN). Being binary, among other short-comings of H.323 in comparison with SIP which is text-based, makes H.323 a platform dependant protocol. SIP, in addition to other advantages which are further discussed in the coming chapters, is capable of supporting user mobility by proxying and redirecting requests to the user's current location. Hence, SIP is the protocol of choice among other standards in the voice and fax transmission domains. But SIP has yet to prove its capabilities to be chosen as the ultimate solution for providing fax services in the Internet.

1.3 Thesis Objectives

In this thesis, we intend to explore the issues related to Fax over IP in their entirety and closely examine some aspects of the new architecture and its implementation feasibility. Different components of the real-time Fax over IP architecture are analyzed and we pay a particular attention to the signaling

part. Utilizing SIP in fax transmission is studied and feasibility of implementing Fax over IP architectures using this protocol is discussed. Based on the reasons briefly mentioned earlier, SIP is better suited to the task in comparison with H.323. More comparisons between the two protocols will be presented in the coming chapters. Computer simulations are utilized for the analysis; specifically, a simplified version of the SIP protocol and network components are developed to study whether fax parameters details can be negotiated between the end-points. These simulation scenarios and their results exhibit the potential success of the proposed implementation approach.

1.4 Thesis Organization

This thesis comprises 8 chapters. Chapters 2 and 3 are devoted to introducing the fundamental concepts behind every network. Readers with sufficient background in network technology can readily bypass these chapters and move on to Chapter 4. Specifically, in Chapter 2 we start the whole discussion by introducing the switching modes and networking modes which arguably is the main distinction between the PSTN and the Internet. We move on to discuss the PSTN and its components in the last section of Chapter 2. Data networks, because of their prime importance in this thesis, are treated in a chapter of their own: Chapter 3. In Chapter 3, discussion starts by introducing data communications basics. Two types of data networks, i.e. local area network (LAN) and wide area network (WAN), are briefly discussed next. The chapter is concluded with a thorough treatment of the Internet protocol itself and its transport layer protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

In Chapter 4 we start treating the issues directly involved in this thesis: issues related to V/FoIP. In this chapter, first some IP telephony fundamentals are presented and then issues related to call signaling are treated and the many available protocols and how they lend themselves to the job are explained. In the next section, Fax over IP is treated by first introducing the conventional Group 3 T.30 fax transmission, then T.37, ITU-T transfer of facsimile data via store-and-forward on the Internet recommendation, and T.38, ITU-T real-time Group 3 fax communication over IP networks recommendation, are explained. Various approaches to provide QoS in IP networks are discussed next. At the end of the chapter, a discussion regarding the IP telephony trends and economics is presented.

Chapter 5 is devoted entirely to presenting and exploring the Session Initiation Protocol (SIP): The protocol of choice in this thesis. In Chapter 5,

after an introduction, SIP clients and servers, request and response messages and the protocol headers are introduced. Session Description Protocol (SDP), a companion protocol to SIP, is introduced in Section 5. A thorough treatment of SIP programming is presented afterwards in Section 6. We conclude the chapter with a discussion covering T.38 and SIP utilization for FoIP.

In Chapters 6 and 7, a detailed account of computer simulations is presented. In Chapter 6, J-Sim simulator, a powerful Java-based network simulation tool is introduced. The source code of this simulator is in the public domain, so we take a close look at inner workings of different components of it, how it operates and how new modules can be developed and added to its set of supported protocols. We also examine the simulation scenario creation method in the simulator.

In Chapter 7, as a major implementation part of this thesis, the development and testing of the SIP protocol is explained. SIP network architecture components i.e. SIP user agents and proxy servers, among other components, have been developed and added to the simulator. The SIP components are put together to build a network simulation scenario to further analyze the behavior of them. As is explained in Chapter 6, building simulation scenario, node initialization and configuration, addition of measurement tools to track packets in a real-time manner and performing measurements of parameters such as throughput, packet sequence numbers and congestion are made possible with the tool command language (TCL) scripting language. A detailed analysis of the specific chosen simulation scenario is given and measurement results are discussed. Furthermore, the complete source codes of developed Java-based SIP components are provided in the appendix.

In Chapter 8, we wrap up the whole discussion and summarize the main points from concept-presenting chapters, i.e. Chapter 2 to 5 and chapters devoted to computer simulations, i.e. Chapters 6 and 7. Some important analysis results and future work directions are also presented.

Chapter 2

Fundamentals

2.1 Introduction

In this chapter, we introduce and discuss fundamental concepts related to this thesis. We first start by inspecting different switching and networking modes which paves the way for better understanding the differences between the Public Switched Telephone Network (PSTN) and the Internet and then move forward to study PSTN more closely in the third section. Then the chapter ends with introducing materials for further studying. Data networks will be examined thoroughly in a chapter of their own, Chapter 3, due their prime importance in this thesis. Having built the necessary knowledge base in Chapters 2 and 3, issues specific to V/FoIP networks will be introduced and examined in Chapter 4.

2.1.1 The PSTN versus the Internet

In one very important fashion, the PSTN and the public Internet are the same thing: they both exist on the same physical infrastructure. There would be no Internet without the PSTN. The communications links, or backbones, that ISPs run on are delivered over the PSTN, and the access lines for entry into the Internet are all subscriber lines that are part of the PSTN. But what

differentiates the PSTN and the Internet is the equipment that's attached to each network, its use, and how it formats the information it carries.

PSTN Characteristics

The PSTN basically includes telephones, fax machines, and circuit switches that set up continuous but temporary connections. In a PSTN environment, the circuit is established between two subscribers and kept open for the duration of the call, including periods of silence. This provides guaranteed QoS and minimal latencies, and it means that the PSTN is optimized for voice and other real-time applications. It also means that the PSTN uses bandwidth inefficiently, making services more expensive. But we are constantly finding ourselves able to release more bandwidth and derive more channels over that bandwidth.

Internet Characteristics

The Internet basically includes clients, which are the user interface and the input/output device for information; the servers, which are the centralized repositories of knowledge that you are seeking; and the packet switches, which route and relay the packets of information between the clients and servers. Whereas the PSTN connects together two subscribers, the Internet connects together networks. As on the PSTN, messages on the Internet are routed to specific end devices. These messages take various forms, such as e-mail, instant messaging, and real-time audio/video communications. Unlike the PSTN, however, the Internet breaks down the messages into packets of data, which contain the routing information, which will then lead them to their destination. Individual packets may take different routes, but they'll be reassembled in the proper order at the destination. This system is optimal for the most efficient use of transmission facilities, particularly when you're supporting bursty traffic that involves long periods of silence. In turn, this results in less expensive services. However, the tradeoff is that you get only best-effort QoS. Significant progress is being made on introducing QoS to the Internet though, and this will change a great deal in the coming years.

Converging Networks: The Next Generation

The decreasing cost of bandwidth, combined with the availability of low-cost and powerful chip technology, favorably highlights the economies of statistical multiplexing and packet switching, as long as latencies and loss can be controlled. From that standpoint, next-generation networks embody two fundamental concepts. First, a next-generation network is a high-speed packet- or cell-based network that's capable of transporting and routing a

multitude of services, including voice, data, video, and multimedia while supporting QoS. Second, a next-generation network is a common platform for applications and services that the customer can access across the entire network as well as outside the network.

Networks are evolving so that they can address the growing demand for QoS. The two different infrastructures—circuit switching and packet switching—are not trying to replace each other. Instead, they are converging. This convergence is required between the existing legacy environment (the circuit-switched network) and the new and unique IP marketplace (the packet-switched network). To address this convergence, a number of devices have emerged that have a number of names, including Voice over IP gateways, media gateways, next-generation switches, and softswitches. These new devices in essence allow interoperability to exist seamlessly between the PSTN and packet-switched networks, whether IP or ATM or MPLS. These issues will be explored more in the coming sections.

2.2 Switching Modes and Networking Modes

This section discusses the key definitions and characteristics that are associated with the processes involved in establishing communications channels. It covers networking modes and switching modes. Details of circuit switching and its particular applications are discussed as well. This section also looks at packet switching, what its potential prospects are, and what challenges it faces. The section ends with a quick comparison between the public switched telephone network (PSTN) and the Internet.

2.2.1 Establishing Connections: Switching Modes and Networking Modes

For messages to travel across a network, a transmission path must be established to either switch or route the messages to their final destinations. Therefore, network providers need a mechanism that allows them to deliver the proper connections when and where a customer requests them. The networking techniques that evolved over time to handle the when and where came about because traditionally, relatively few high-capacity backbone cables existed. Those few backbone cables had to be manipulated to meet the needs of many individual customers, all of whom had varied bandwidth needs. Two networking techniques arose: connection oriented and connectionless.

Switching modes-There are also two switching modes: circuit switching and packet switching. Both of these switching modes offer forms of

bandwidth on demand. The following sections describe networking modes and switching modes in detail.

Table 2.1 Relationship between different types of switching [3].

<table>
<tr><td colspan="2">Connection-Orientated</td><td>Connectionless</td></tr>
<tr><td rowspan="2">Circuit Switched</td><td colspan="2">Packet Switched
Cell Switching Message Switching</td></tr>
<tr><td>Virtual Circuit</td><td>Datagram Switching</td></tr>
</table>

2.2.2 Networking Modes

When a network is being evaluated, concentration is on circuit switching versus packet switching. But it's also very important to consider the networking mode, which can be either connection oriented or connectionless.

Connection-Oriented Networking

As time-sensitive applications become more important, connection-oriented networks are becoming increasingly desirable. In a connection-oriented network, the connection setup is performed before information transfer occurs. Information about the connections in the networks helps to provide service guarantees and makes it possible to most efficiently use network bandwidth by switching transmissions to appropriate connections as the connections are set up. Putting it in another way; after the path is determined, all the subsequent information follows the same path to the destination. In a connection-oriented network, there can be some initial delay while the connection is being set up; but because the path is predetermined, there is no delay at intermediate nodes in this type of network after the connection is set up. Connection-oriented networks can actually operate in either switching mode: They can be either circuit switched or packet switched. Connection-oriented circuit-switched networks include the PSTN, SDH/SONET, and DWDM networks. Connection-oriented packet-switched networks include X.25, Frame Relay, and ATM networks.

Connectionless Networking

In a connectionless network, no explicit connection setup is performed before data is transmitted. Instead, each data packet is routed to its destination based on information contained in the header. In other words, each packet is individually addressed and individually routed. In a connectionless network, the delay in the overall transit time is increased because each packet has to be individually routed at each intermediate

node. Applications that are time sensitive would suffer on a connectionless network because the path is not guaranteed, and therefore it is impossible to calculate the potential delays or latencies that might be encountered.

Connectionless networks imply the use of packet switches, so only packet-switched networks are connectionless. An example of a connectionless packet-switched network is the public Internet. It's a virtual network that consists of more than 150,000 separate subnetworks and some 10,000 Internet service providers (ISPs), so being able to guarantee performance is nearly impossible at this time. One solution is to use private internets which achieve cost-efficiencies but, because they are private, provide the ability to control their performance and thereby serve business-class services.

2.2.3 Switching Modes

Switched technologies came about because of the high cost and inflexibility characteristic of early dedicated communications architectures. Switching is the process of physically moving bits through a network node, from an input port to an output port. Switching elements are specialized computers that are used to connect two or more transmission lines. The switching process is based on information that's gathered through a routing process. A switching element might consult a table to determine, based on number dialed, the most cost-effective trunk over which to forward a call. This switching process is relatively straightforward compared to the type of path determination that IP routers in the Internet might use, which can be very complex.

Circuit Switching

Circuit switching has been the basis of voice networks worldwide for many years. One of the key attributes of a circuit-switched connection is that it is a reserved network resource for the full duration of a conversation. But when that conversation is over, the connection is released. When a customer wants to place a call, he picks up the handset, which notifies the switch of his desire to do so, and the switch responds by sending a dial tone to the phone. The caller then sends the destination address to the switch (the telephone number), which proceeds to establish the call. As soon as the other end goes off-hook, the call is established. A circuit-switched environment requires that an end-to-end circuit be set up before a call can begin. A fixed share of network resources is reserved for the call, and no other call can use those resources until the original connection is closed. As

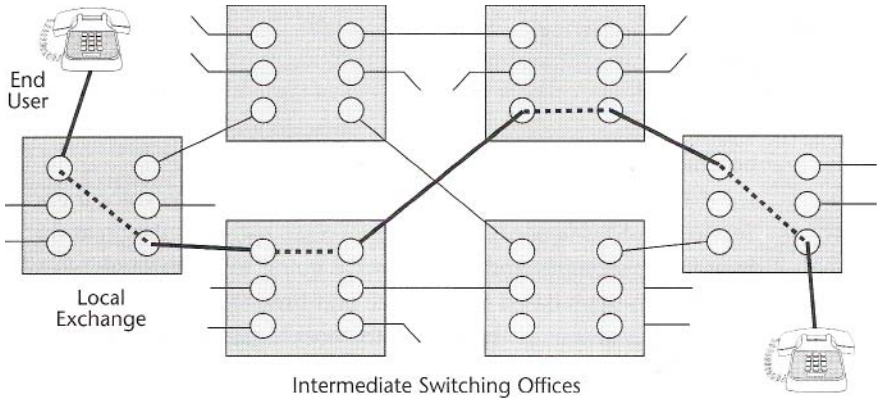

Figure 2.1 A Circuit-switched call [4].

Trunk that stays in place ----------

Connection set up when call is placed . . . . . .

Figure 2.1 illustrates, one can trace the path from one end of the call to the other end; that path and the capacity provisioned on it would not vary for the full duration of the call. Circuit switching offers the benefits of low latency and minimal delays because the routing calculation on the path is made only once, at the beginning of the call, and there are no more delays incurred subsequently in calculating the next hop that should be taken. Traditionally, this was sometimes seen as a disadvantage because it meant that the circuits might not be used as efficiently as possible. Around half of most voice calls is silence. Most people breathe and occasionally pause in their speech. So, when voice communications are conducted over a circuit that's being continuously held, and half the time nothing is being transmitted, the circuit is not being used very efficiently. But this is an issue that is important when bandwidth is constrained which does not apply to today's networks. Hence, the low latencies or delays that circuit switching guarantees are more important than its potential drawbacks in bandwidth efficiency. Circuit switching has been optimized for real-time voice traffic for which Quality of Service (QoS) is needed. Circuit switching has experienced a major problem in recent years as Internet access with its Web surfing has grown popular. Because call resources are dedicated, the use of circuit-switched facilities for modem access to the Web has resulted in the switches becoming severely overtaxed by the long call-hold times that characterize most Web sessions.

Packet Switching

Whereas circuit switching was invented to facilitate voice telephony, packet switching has its origin in data communications. Packet switching was

developed specifically as a solution for a traffic stream that's described as being bursty in nature, and bursty implies that you have long connect times but low data volumes. Therefore, circuit-switched links are not used efficiently: The connection would be established and held for a long period of time, with only little data passed. Packet switching was developed to increase the efficiencies associated with bursty transmission. Packet switching involves the multiplexing of multiple packets over a kind of circuit (an end-to-end logical connection that creates a complete path across the network from source to destination node). Packet networks combine the packet streams from multiple users onto a single facility, thus using a higher percentage of the available bandwidth. Control protocols manage the independent data streams to ensure that each user data component maintains its integrity. In some cases, this technique is called virtual circuit service; the name stems from the fact that the service is so good in these networks that users feel as if they have their own dedicated transmission channel, in spite of the fact that the channel is shared among a collection of users. There are two special types of packet switched systems. The first is message switching where the entire message forms a packet. Message switched systems are often referred to as store and forward systems. Message switching avoids splitting the information to be transmitted into smaller packets, and the consequent reassembly at the receiver. The second special type of packet switching is cell switching. In a cell switching system, all the packets, known as cells, have a fixed length. This common format reduces the amount of work the network nodes have to perform on the packet, keeps complexity low and speeds high. ATM uses cell switching with a 48 byte information field and 5 bytes of header. To keep the processing and header overheads small, cell switching systems use virtual circuits so that the path through each router is set up in advance. Information is divided into packets that contain two very important messages: the destination address and the sequence number. The original forms of packet switching (developed in the late 1960s and early 1970s) were connectionless infrastructures. In a connectionless environment, each packet is routed individually, and the packets might not all take the same path to the destination point, and hence they may arrive out of sequence.

Therefore, the sequence number is very important; the terminating point needs it to be able to reassemble the message in its proper order. Figure 2.2 illustrates a packet-switched network that uses virtual circuits. You can see that packets are queued up at the various nodes, based on availability of the virtual circuits, and that this queuing can impose delays.

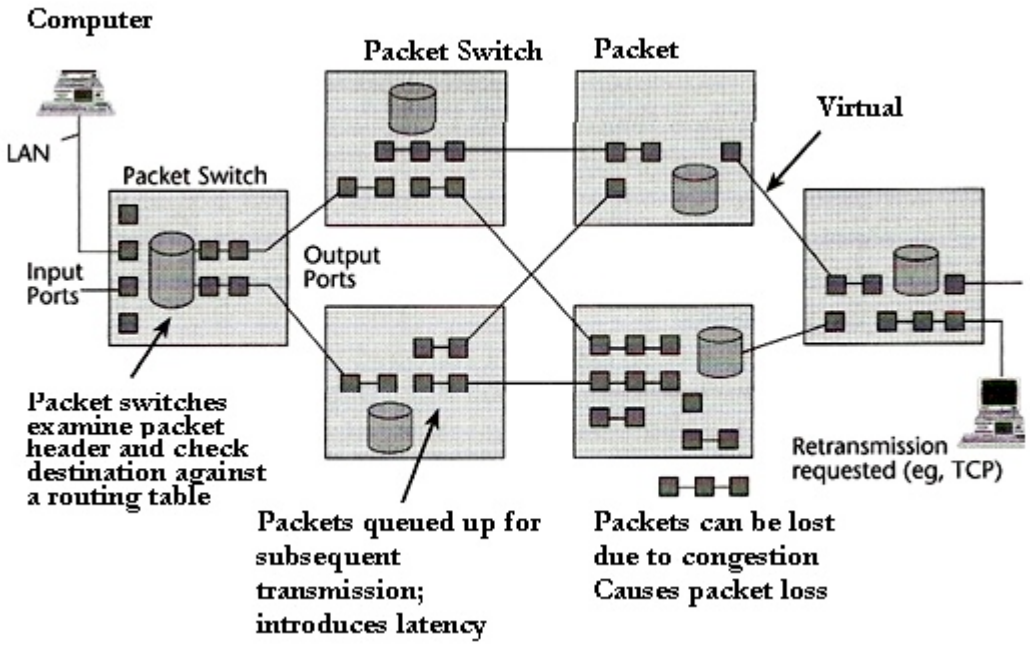

Figure 2.2 A Packet-switched network [4].

The first generation of packet-switched networks could support only data; it could not support voice or video at all because there was so much delay associated with those networks. As packet-switched environments are evolving, some techniques are being developed to be able to separate and prioritize those traffic types.

2.2.4 Connectionless Versus Connection-Oriented Packet-Switched Networks

Packet networks provide two forms of service: connectionless service, where the switches do not perceive a relationship between packets that derive from the same source, and connection-oriented service, where they do.

Connectionless Packet-Switched Networks

Connectionless networks make no guarantees of delivery or arrival sequence. They are often called best effort or spray-and-pray networks because although they make every effort to deliver the packets to the destination, they do not guarantee the delivery. That is the responsibility of a higher-layer protocol. For example, IP depends on TCP for guaranteed delivery of transmitted packets if such guarantees are required. A connectionless environment worries about getting a packet one step closer to the destination. It doesn't worry about having an end-to-end view of the path over which the message will flow; this is the fundamental difference between connection-oriented and connectionless environments, and, hence, between

infrastructures such as the Internet and the PSTN. Examples of connectionless packet-switched networks include the public Internet, private IP backbones or networks, Internet-based VPNs, and LANs. Again, each packet (referred to as a datagram transmission) is an independent unit that contains the source and destination address, which increases the overhead. That's one of the issues with connectionless packet-switched networks: If we have to address each packet, then the overall percentage of control information relevant to the actual data being transported rises.

Each router performs a path calculation function independently, and each relies on its own type of routing protocols (for example, Open Shortest Path First [OSPF], Intermediate System to Intermediate System [IS-IS], or Border Gateway Protocol [BGP]). Each router calculates the appropriate next hop for each destination, which is generally based on the smallest number of hops. Packets are forwarded, then, on a hop-by-hop basis rather than as part of an end-to-end connection. Each packet must be individually routed, which increases delays, and the more hops, the greater the delay. Therefore, connectionless environments provide less control over ensuring QoS because of unknown latencies, unknown retransmissions, and unknown sequences in which the packets will arrive.

Connection-oriented packet-switched Networks

The connection-oriented packet-switched environment is something like a telephone network, in which a call setup is performed end-to-end. X.25, Frame Relay, ATM, and Multiprotocol Label Switching (MPLS) are all connection-oriented techniques. In connection-oriented networks, there is a multistage process that must be adhered to before information is transmitted across the network. In stage one, a call setup packet is sent into the network. This initial packet contains the full address of the intended recipient of the message. Upon arrival at the first switch, the packet is examined, the address is read, and a route (outgoing port) is selected based upon known information about the network and the intended destination. The switch then makes an entry into its routing table indicating which port the packet arrived on and which one it went out through. The packet continues across the network, causing table entries to be written at each switch, and thus places a trail across the network. Once the trail has been laid down, the remaining packets can make their way across the network, following the table entries left on switches that they encounter along the way. As a result, the packets that follow the setup packet do not require a full address. All they need is a short virtual circuit identifier; the switches do the rest (see Figure 2.3). With connection-oriented networks, one does not need to route each individual packet. Instead, each packet is marked as

belonging to some specific flow that identifies which virtual circuit it belongs to. No repeated per-packet computation is required; consequently, connection-oriented networks reduce latencies, or delays. In the connection-oriented environment, the entry node contains the routing table, where the path is calculated and determined, and all packets follow that same path on to the destination node, thereby offering a better guarantee of service.

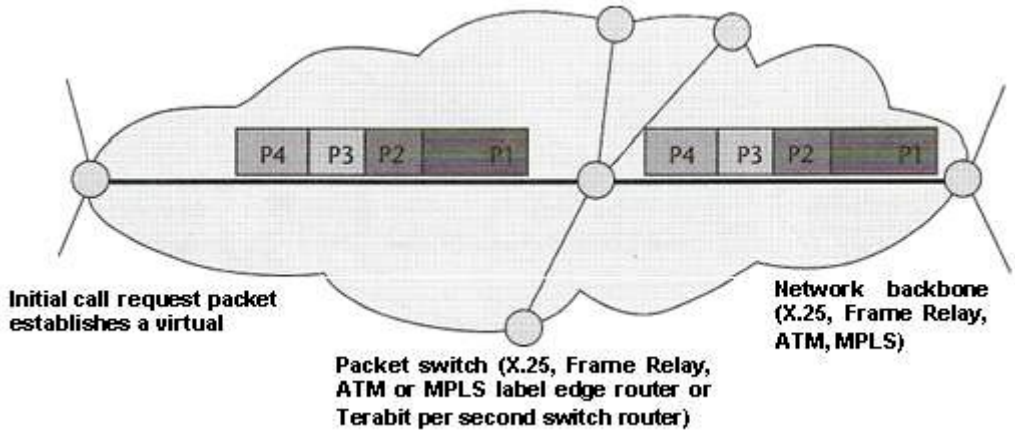

Figure 2.3 A connection-oriented network [4].

2.2.5 Comparing Circuit Switching and Packet Switching

Circuit switching is superior to packet switching in terms of eliminating queuing delays, which results in completely predictable latency and jitter in the backbone. Given the trend toward real-time visual and sensory communication streams, this seems to be the most important characteristic for us to strive toward.

Table 2.2 Circuit Switching Versus Packet Switching.

Characteristics	Circuit Switching	Packet Switching
Origin	Voice Telephony	Data Networking
Connectionless or Connection oriented	Connection oriented	Both
Key Applications	Real-time Voice, Streaming Media, videoconferencing, Video-on-demand, and other delay- and loss- sensitive applications	Bursty data traffic that has long connect times but low data volumes; applications that are delay and loss tolerant
Latency/Delay/Jitter	Low latency and minimal delays	Subject to latency, delay, and jitter because of its store-and-forward nature
Network intelligence	Centralized	Decentralized
Bandwidth efficiency	Low	High
Packet loss	Low	High

With the large capacities that are afforded with the new DWDM systems and other optical network elements, minimizing latency becomes more important than optimizing bandwidth via statistical multiplexing. Table 2.2 is a brief comparison of circuits switching and packet switching.

Here we drop the discussion of networking and switching modes and move forward toward inspecting PSTN more closely due to its importance.

2.3 The PSTN

This section talks about the public switched telephone network (PSTN). It talks about what comprises the PSTN, what sorts of technologies have been used to complete the connections, and how the signaling systems operate.

2.3.1 The PSTN Infrastructure and Architecture

Our views about what a network should be designed to support and what the infrastructure should be comprised of have changed quite a bit over the years, as applications and technology have changed. This section takes a look at how the PSTN infrastructure evolved and where it is today.

The traditional PSTN infrastructure was specifically designed to support only voice communications. At the time this infrastructure was being designed, we had no notion of data communications. Initially the traffic type the PSTN was designed to support was continuous real-time voice. Another variable that's important to the design of the PSTN has to do with the length of calls. Most voice calls are quite short, so the circuit switches in the PSTN are engineered for call durations of three minutes or less. The average Internet session, on the other hand, lasts around an hour. This means that increased Internet access through the PSTN has, in some locales, put a strain on the local exchanges. If a circuit switch is blocked because it is carrying a long Internet session, people may not be able to get a dial tone. There are several solutions to this problem. For example, we can apply intelligence in front of some exchanges so that calls destined for ISPs can be diverted over a packet-switched network to the ISP rather than being completed on a circuit-switched basis through the local exchange.

Yet another variable that's important to the design of the PSTN has to do with what it was designed to support. The capacities of the channels in the PSTN are of the narrowband generation—they are based on 64Kbps channels. The worldwide infrastructure to accommodate voice communications evolved to include a series of circuit switches. Different switches are used based on the locations to which they're connecting. The switches have a high degree of

intelligence built into them, both for establishing the communications channels and for delivering the service logic to activate a growing array of features. In the traditional framework, the monolithic switches in the network had all the smarts. The switch manufacturer and the carrier worked together very closely, and the carrier was not able to introduce new features and services into a particular area until a software release was available for the switch platform through which the neighborhood was being serviced. Thus, carriers were often unable to roll out new services and features because they hadn't yet received the new software releases from the switch manufacturers. Over time, we have separated the functions of switching and connection establishment from the functions involved in the intelligence that enables various services and features to be activated.

The traditional PSTN is associated with highly developed, although not necessarily integrated, operational support systems (such as billing systems, provisioning systems, network management systems, customer contact systems, and security systems). These systems have very well-developed business processes and techniques for managing their environments. But the various systems' databases cannot yet all speak to one another to give one comprehensive view.

PSTN Architecture

The PSTN includes a number of transmission links and nodes. There are basically four types of nodes: CPE nodes, switching nodes, transmission nodes, and service nodes.

CPE Nodes

CPE nodes generally refer to the equipment that's located at the customer site. The main function of CPE nodes is to transmit and receive user information. The other key function is to exchange control information with the network. In the traditional realm, this equipment includes PBXs, key telephone systems, and single-line telephones.

Switching Nodes

Switching nodes interconnect transmission facilities at various locations and route traffic through a network. They set up the circuit connections for a signal path, based on the number dialed. To facilitate this type of switching, the ITU standardized a worldwide numbering plan (based on ITU E.164) that essentially acts as the routing instructions for how to complete a call through the PSTN. The switching nodes include the local exchanges, tandem exchanges (for routing calls between local exchanges within a city), toll offices

(for routing calls to or from other cities), and international gateways (for routing calls to or from other countries). Primary network intelligence is contained in the Class 4 switches (that is, toll offices switches) and Class 5 switches (that is, local exchange switches). The Class 4 toll switches provide long-distance switching and network features, and the Class 5 switches provide the local switching and telephony features that subscribers subscribe to. Figure 2.4 shows where the types of telephone exchanges are located.

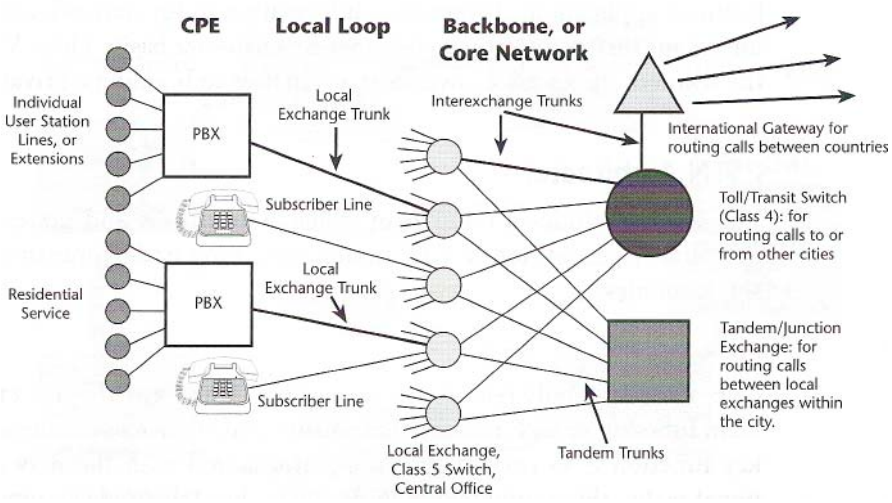

Figure 2.4 Types of telephone exchanges [4].

Transmission Nodes

Transmission nodes are part of the transport infrastructure, and they provide communication paths that carry user traffic and network control information between the nodes in a network. The transmission nodes include the transmission media as well as transport equipment, including amplifiers and/or repeaters, multiplexers, digital cross-connects, and digital loop carriers.

Service Nodes

Service nodes handle signaling, which is the transmission of information to control the setup, holding, charging, and releasing of connections, as well as the transmission of information to control network operations and billing. A very important area related to service nodes is the ITU standard specification Signaling System 7 (SS7), which is covered later in this section.

The Transport Network Infrastructure

The transport network includes two main infrastructures. The first is the PDH, also known as T-carrier, E-carrier, and J-carrier wideband transmission

standards. This infrastructure was first introduced in the early 1960s. The second infrastructure of the transport network is the Synchronous Digital Hierarchy (SDH; ITU terminology), also known as Synchronous Optical Network (SONET; ANSI terminology), which was first formalized and standardized in 1988. SDH/SONET is the second generation of digital hierarchy, and it is based on a physical infrastructure of optical fibers.

PDH and SDH/SONET are voice-centric circuit-switched network models that switch millions of 64Kbps circuits between various switching points. Each circuit is multiplexed numerous times for aggregation onto transmission facilities. Aggregation occurs at many points in the network: in the access network, within the local exchange, and throughout the interexchanges. Hence, a significant portion of the cost of a network goes to the equipment that performs this aggregation—the multiplexers and cross-connects in both the PDH and SDH/SONET environments.

2.3.2 Signaling Systems

This section discusses the nervous system of the network: the signaling system. A great deal of information needs to be passed back and forth between the network elements in the completion of a call and also in the servicing of specialized features. Four main types of signals handle this passing of information:

Supervisory signals—Supervisory signals handle the on-hook/off-hook condition. For instance, when you lift a telephone handset (that is, go off-hook), a signal tells the local exchange that you want a dial tone, and if you exist in the database as an authenticated user, you are then delivered that service; when you hang up (that is, go back on-hook), you send a notice that says you want to remove the service. A network is always monitoring for these supervisory signals to determine when someone needs to activate or deactivate service.

Address signals—Address signals have to do with the number dialed, which essentially consists of country codes, city codes, area codes, prefixes, and the subscriber number. This string of digits, which we refer to as the telephone number, is, in effect, a routing instruction to the network hierarchy.

Information signals—Information signals are associated with activating and delivering various enhanced features. For instance, a call-waiting tone is an information signal, and pressing *72 on your phone might send an information signal that tells your local exchange to forward your calls.

Alerting signals—Alerting signals are the ringing tones, the busy tones, and any specific busy alerts that are used to indicate network congestion or unavailability.

Signaling takes place in two key parts of the network: in the access network, where it's called loop signaling, and in the core, where it's called interoffice signaling (see Figure 2.5)

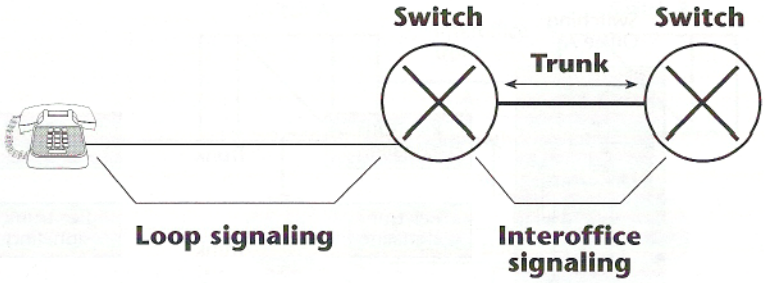

Figure 2.5 Customer loop and interoffice signaling [4].

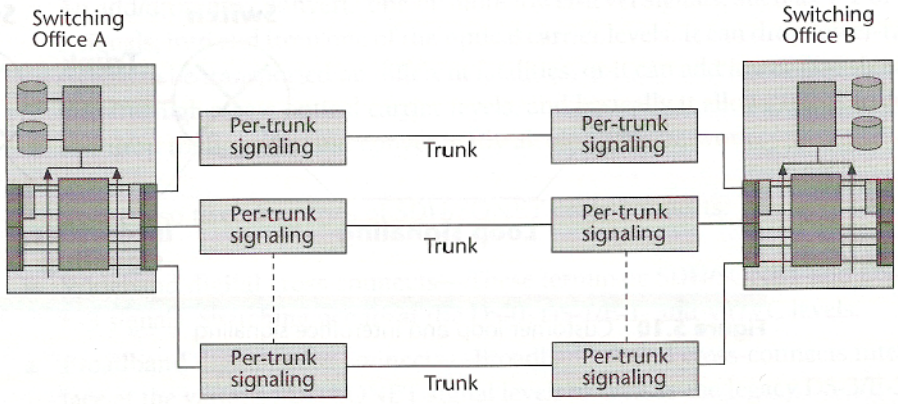

Figure 2.6 Per-trunk signaling [4].

Interoffice signaling has been through several generations of signaling approaches. In the first generation, called per-trunk signaling, the complete path— all the way to the destination point—is set up in order to just carry the signaling information in the first place (see Figure 2.6). This method uses trunks very inefficiently; trunks may be put into place to carry 20 or 30 ringing tones, but if nobody is on the other end to take that call, the network trunk is being used but not generating any revenue. Also, when a call is initiated and begins to progress, you can no longer send any other signaling information over that trunk; being able to pass a call-waiting tone, for instance, would not be feasible.

We have moved away from the per-trunk signaling environment to what we use today—common-channel signaling. You can think of common-channel signaling as being a separate subnetwork over which the signaling message flows between intelligent networking components that assist in the call completion and assist in the delivery of the service logic needed to deliver the requested feature. Today, we predominantly use the ITU-T standard for common-channel signaling: SS7.

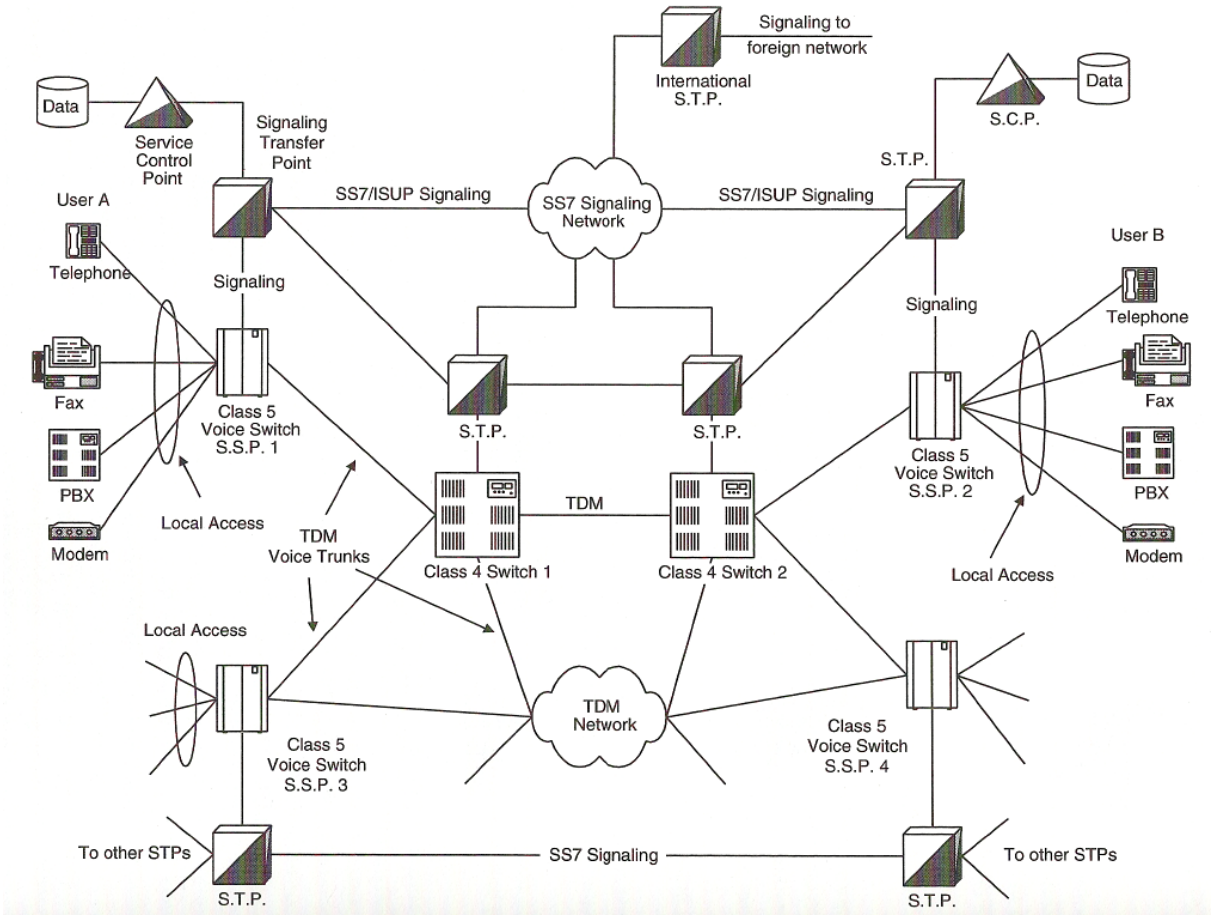

Figure 2.7 A Simplified view of SS7 component topology [5].

Common Channel Signaling System No. 7

(SS7 or C7) is a global standard for telecommunications defined by the International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T). The standard defines the procedures and protocol by which network elements in the public switched telephone network (PSTN) exchange information over a digital signaling network to accomplish wireless and wireline call setup, routing and control. The ITU definition of SS7 allows for national variants such as North America's American National Standards Institute (ANSI) and Bell Communications Research (Telcordia Technologies) standards and Europe's European Telecommunications Standards Institute (ETSI) standard. A Simplified view of SS7 component topology is depicted in Figure 2.7.

The SS7 network and protocol are used for:

-Basic call setup, management and tear down

-Wireless services such as personal communications services (PCS), wireless roaming and mobile subscriber authentication

-Local number portability (LNP)

-Toll-free (800/888) and toll (900) wireline services

-Enhanced call features such as call forwarding, calling party name/number display and three-way calling
-Efficient and secure worldwide telecommunications

Signaling Links

SS7 messages are exchanged between network elements over 56 or 64 kilobit per second (kbps) bidirectional channels called signaling links. Signaling occurs out-of-band on dedicated channels rather than in-band on voice channels. Compared to in-band signaling, out-of-band signaling provides:
-Faster call setup times (compared to in-band signaling using multi-frequency (MF) signaling tones)
-More efficient use of voice circuits
-Support for Intelligent Network (IN) services, which require signaling to network elements without voice trunks (e.g., database systems)
-Improved control over fraudulent network usage

Signaling Points

Each signaling point in the SS7 network is uniquely identified by a numeric point code. Point codes are carried in signaling messages exchanged between signaling points to identify the source and destination of each message. Each signaling point uses a routing table to select the appropriate signaling path for each message.
There are three kinds of signaling points in the SS7 network (Figure 2.8):
-SSP (Service Switching Point)
-STP (Signal Transfer Point)
-SCP (Service Control Point)

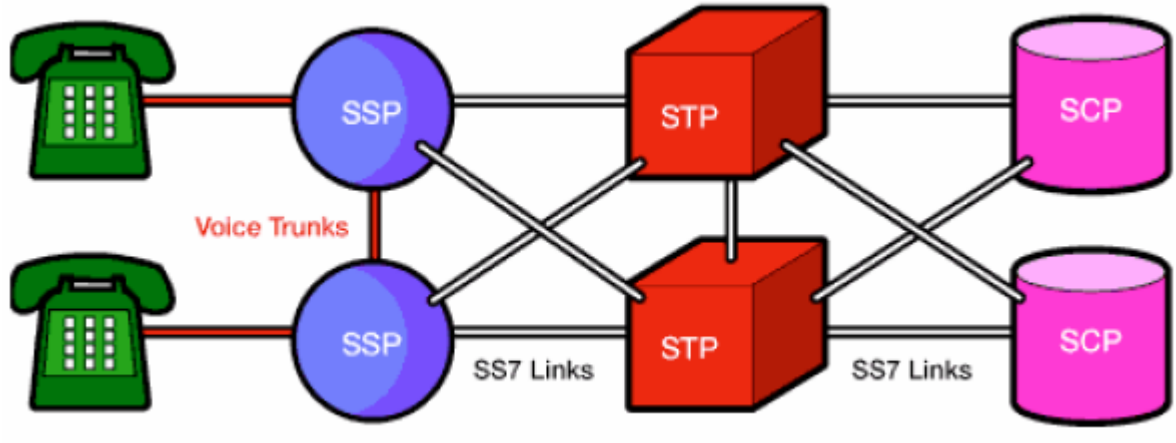

Figure 2.8 SS7 Signaling Points [6].

SSPs are switches that originate, terminate or tandem calls. An SSP sends signaling messages to other SSPs to setup, manage and release voice circuits required to complete a call. An SSP may also send a query message to a centralized database (an SCP) to determine how to route a call (e.g., a toll-free 1-800/888 call in North America). An SCP sends a response to the originating SSP containing the routing number(s) associated with the dialed number. An alternate routing number may be used by the SSP if the primary number is busy or the call is unanswered within a specified time. Actual call features vary from network to network and from service to service. Network traffic between signaling points may be routed via a packet switch called an STP. An STP routes each incoming message to an outgoing signaling link based on routing information contained in the SS7 message. Because it acts as a network hub, an STP provides improved utilization of the SS7 network by eliminating the need for direct links between signaling points. An STP may perform global title translation, a procedure by which the destination signaling point is determined from digits present in the signaling message (e.g., the dialed 800 number, calling card number or mobile subscriber identification number). An STP can also act as a "firewall" to screen SS7 messages exchanged with other networks. Because the SS7 network is critical to call processing, SCPs and STPs are usually deployed in mated pair configurations in separate physical locations to ensure network-wide service in the event of an isolated failure. Links between signaling points are also provisioned in pairs. Traffic is shared across all links in the linkset. If one of the links fails, the signaling traffic is rerouted over another link in the linkset. The SS7 protocol provides both error correction and retransmission capabilities to allow continued service in the event of signaling point or link failures.

SS7 Signaling Link Types

Signaling links are logically organized by link type ("A" through "F") according to their use in the SS7 signaling network.

A Link: An "A" (access) link connects a signaling end point (e.g., an SCP or SSP) to an STP. Only messages originating from or destined to the signaling end point are transmitted on an "A" link.

B Link: A "B" (bridge) link connects one STP to another. Typically, a quad of "B" links interconnect peer (or primary) STPs (e.g., the STPs from one network to the STPs of another network).

C Link: A "C" (cross) link connects STPs performing identical functions into a mated pair. A "C" link is used only when an STP has no other route available to a destination signaling point due to link failure(s). Note that SCPs may also be deployed in pairs to improve reliability; unlike STPs however, mated

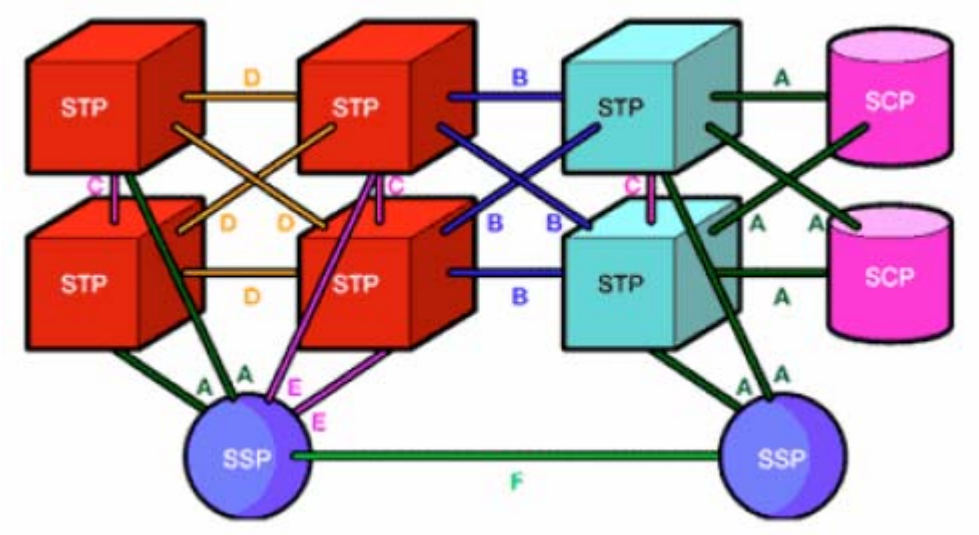

Figure 2.9 SS7 Signaling Link Types [6].

SCPs are not interconnected by signaling links.

D Link: A "D" (diagonal) link connects a secondary (e.g., local or regional) STP pair to a primary (e.g., inter-network gateway) STP pair in a quad-link configuration. Secondary STPs within the same network are connected via a quad of "D" links. The distinction between a "B" link and a "D" link is rather arbitrary. For this reason, such links may be referred to as "B/D" links.

E Link: An "E" (extended) link connects an SSP to an alternate STP. "E" links provide an alternate signaling path if an SSP's "home" STP cannot be reached via an "A" link. "E" links are not usually provisioned unless the benefit of a marginally higher degree of reliability justifies the added expense.

F Link: An "F" (fully associated) link connects two signaling end points (i.e., SSPs and SCPs). "F" links are not usually used in networks with STPs. In networks without STPs, "F" links directly connect signaling points.

SS7 Protocol Stack

The hardware and software functions of the SS7 protocol are divided into functional abstractions called "levels". These levels map loosely to the Open Systems Interconnect (OSI) 7-layer model defined by the International Standards Organization (ISO) as depicted in Figure 2.10.

Message Transfer Part

The Message Transfer Part (MTP) is divided into three levels.

The lowest level, **MTP Level 1**, is equivalent to the OSI Physical Layer. MTP Level 1 defines the physical, electrical and functional characteristics of the digital signaling link. Physical interfaces defined include E-1 (2048 kb/s; 32

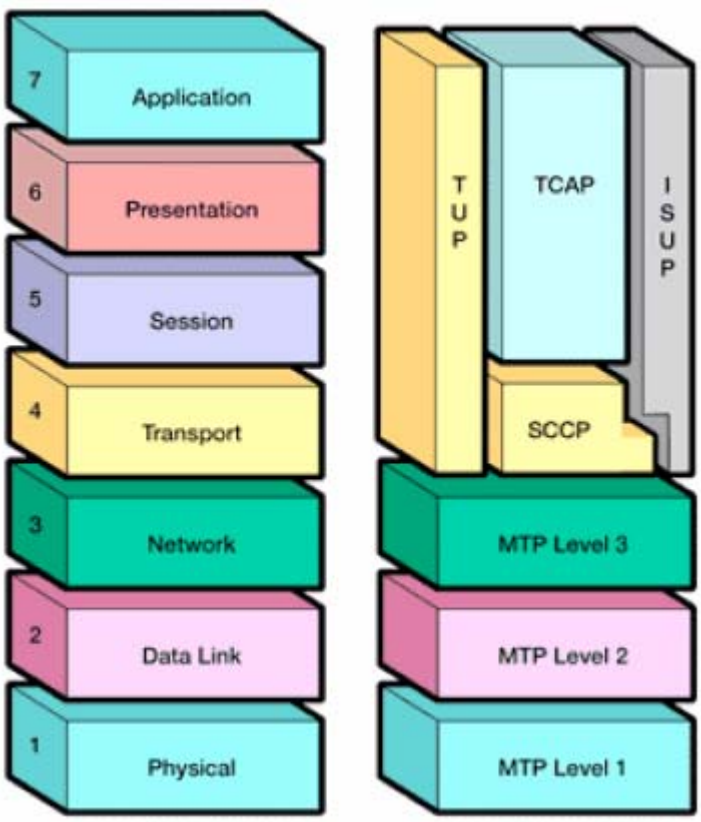

Figure 2.10 The OSI Reference Model and the SS7 Protocol Stack [6].

64 kb/s channels), DS-1 (1544 kb/s; 24 64kb/s channels), V.35 (64 kb/s), DS-0 (64 kb/s) and DS-0A (56 kb/s).

MTP Level 2 ensures accurate end-to-end transmission of a message across a signaling link. Level 2 implements flow control, message sequence validation and error checking. When an error occurs on a signaling link, the message (or set of messages) is retransmitted. MTP Level 2 is equivalent to the OSI Data Link Layer.

MTP Level 3 provides message routing between signaling points in the SS7 network. MTP Level 3 reroutes traffic away from failed links and signaling points and controls traffic when congestion occurs. MTP Level 3 is equivalent to the OSI Network Layer.

ISDN User Part (ISUP)

The ISDN User Part (ISUP) defines the protocol used to set-up, manage and release trunk circuits that carry voice and data between terminating line exchanges (e.g., between a calling party and a called party). ISUP is used for both ISDN and non-ISDN calls. However, calls that originate and terminate at the same switch do not use ISUP signaling.

Telephone User Part (TUP)

In some parts of the world (i.e., China and Brazil), the Telephone User Part (TUP) is used to support basic call setup and tear-down. TUP handles analog circuits only. In many countries, ISUP has replaced TUP for call management.

Signaling Connection Control Part (SCCP)

SCCP provides connectionless and connection-oriented network services and global title translation (GTT) capabilities above MTP Level 3. A global title is an address (e.g., a dialed 800 number, calling card number or mobile subscriber identification number) that is translated by SCCP into a destination point code and subsystem number. A subsystem number uniquely identifies an application at the destination signaling point. SCCP is used as the transport layer for TCAP-based services.

Transaction Capabilities Applications Part (TCAP)

TCAP supports the exchange of non-circuit related data between applications across the SS7 network using the SCCP connectionless service. Queries and responses sent between SSPs and SCPs are carried in TCAP messages. For example, an SSP sends a TCAP query to determine the routing number associated with a dialed 800/888 number and to check the personal identification number (PIN) of a calling card user. In mobile networks (IS-41 and GSM), TCAP carries Mobile Application Part (MAP) messages sent between mobile switches and databases to support user authentication, equipment identification and roaming.

Operations, Maintenance and Administration Part (OMAP) and ASE

OMAP and ASE are areas for future definition. Presently, OMAP services may be used to verify network routing databases and to diagnose link problems.
Here we wrap up the discussion of PSTN and its widely used signaling system and proceed with introduction of data networks and their characteristics in the next chapter.

2.4 For Further Study

For further information regarding the concepts discussed in this chapter please refer to following resources:

2.4.1 Switching Modes and Networking Modes

[4] PP: 95-111
[7] PP: 162-170
[3] PP: 65-70

2.4.2 The PSTN

[4] PP: 113-144
[8]
[5] PP: 157-195
[6]

Chapter 3

Data Networks

3.1 Introduction

In this chapter, data networks will be thoroughly investigated. Mastering the related concepts is a prerequisite for understanding the materials introduced in the next chapters. We start by discussing some basic issues in the first section and then move forward to talk about local/wide area networks very briefly in the following section. In the fourth section, Internet Protocol will be examined fully. In Section 5, transport layer protocols are discussed. And finally, chapter ends with introducing materials for further studying.

3.2 Data Communications Basics

3.2.1 The Evolution of Data Communications

Data communication is the exchange of digital information between computers and other digital devices via telecommunications nodes and wired or wireless links. To understand the evolution of networking services, it is important to understand the general computing architectures and traffic types, both of which have changed over time.

3.2.2 Data Communication Architectures

In the rather brief history of data networking, a variety of architectures have arisen, and each has had unique impacts on network variables. Table 3.1 shows a basic time line of the architectures that have prevailed during different periods. Each architecture has slightly different traffic characteristics, has slightly different requirements in terms of security and access control, and has presented a different volume and consistency of traffic to the network. With each new computing architecture, there has been a demand for new generations of network services.

Table 3.1 Time Line of Data Networking Architectures [4].

Time	Architecture
1970s	Stand-alone mainframes
Early 1980s	Networked mainframes
Early 1980s	Stand-alone workstations
Early to late 1980s	Local area networking
Mid-1980s to mid-1990s	LAN internetworking
Mid-1990s	Internet commercialization
Mid- to late 1990s	Application-driven networks
Late 1990s	Remote-access workers
Early 2000s	Home area networking

3.2.3 Data Communication Traffic

As the architecture of data networks has changed, so have the applications people use, and as applications have changed, so has the traffic on the network. This section talks about some of the most commonly used applications today and how much bandwidth they need, how sensitive they are to delay, where the error control needs to be performed, and how well they can tolerate loss.

The most pervasive, frequently used application is e-mail. Today, it's possible to append an entire family vacation photo album to an e-mail message, and this massive file would require a lot of bandwidth. But e-mail in its generic text-based form is a low-bandwidth application that

is delay insensitive. If an e-mail message gets trapped somewhere in the Net for several seconds, its understandability will not be affected because by the time you view it, it will have all been put on the server where your e-mail resides, waiting for you to pick it up. Another issue you have to consider with e-mail is error control. Networks today rarely perform error control because it slows down the traffic too much, so error control and recovery need to be handled at the endpoints. Instead, internetworking protocols deployed at the end node, such as Transmission Control Protocol (TCP), detect errors and request retransmissions to fix them.

Another prevalent data networking application is transaction processing. Examples of transaction processing include a store getting approval for a credit card purchase and a police officer checking a database for your driver's license number to see whether you have any outstanding tickets. Transaction processing is characterized by many short inputs and short outputs, which means it is generally a fairly low-bandwidth application, assuming that it involves text-based messages. Remember that if you add images or video, the bandwidth requirements grow substantially. Thus, if a police officer downloads a photo from your license, the bandwidth required will rise. Transaction processing is very delay sensitive because with transactions you generally have a person waiting for something to be completed (for example, for a reservation to be made, for a sales transaction to be approved, for a seat to be assigned by an airline). Users want subsecond response time, so with transaction processing, delays are very important, and increased traffic contributes to delay. With transaction processing, you have to be aware of delay, and error control is the responsibility of the endpoints. Transaction processing is fairly tolerant of losses because the applications ensure that all the elements and records associated with a particular transaction have been properly sent and received before committing the transaction to the underlying database.

Another type of application is file transfer, which involves getting a large bulk of data moved from one computer to another. File transfer is generally a high-bandwidth application because it deals with a bulk of data. File transfer is machine-to-machine communication, and the machines can work around delay factors, as long as they're not trying to perform a real-time function based on the information being delivered. So file transfer is a passive activity and it can tolerate delay. File transfer can also tolerate losses. With file transfer, error control can be performed at the endpoints.

Two other important applications are interactive computing and information retrieval. Here bandwidth is dependent on the objects that you are retrieving: If it's text, it's low bandwidth; if it's pictures, it’s probably not.

Interactive computing and information retrieval are delay sensitive when it comes to downloads, so higher speeds are preferred. Real-time voice is a low-bandwidth application but is extremely delay sensitive. Real-time audio and video require medium, high, and even very high bandwidth, and are extremely delay sensitive. Multimedia traffic and interactive services require very high bandwidth, and they are extremely delay sensitive and extremely loss sensitive.

Anything that is text-based—such as e-mail, transaction processing, file transfer, and even the ability to access a database for text-based information—is fairly tolerant of losses. But in real-time traffic, such as voice, audio, or video, losses cause severe degradation in the application.

3.2.4 The OSI Reference Model and Protocols

Before two computers or network devices can exchange information, they must establish communication, and this is where protocols come in. A network protocol enables two devices to communicate by using one set of rules. The OSI model and protocol standards help to ensure that networking devices are capable of working together over a network.

The OSI Reference Model

In the early 1970s, a problem was starting to develop. There were many different computer manufacturers, and there were many incompatibilities among them. Furthermore, each manufacturer created different product lines, and even within one company there were often incompatibilities between their product lines. So the International Standards Organization got involved and created the Open Systems Interconnection (OSI) reference model, which is a reference blueprint for device manufacturers and software developers to use when creating products.

The OSI model, shown in Figure 3.1, has seven layers that describe the tasks that must be performed to transfer information on a network. When data is being transferred over a network, it must pass through each layer of the OSI model. As the data passes through each layer, information is added to that data. At the destination, the additional information is removed. Layers 4 through 7 occur at the end node, and Layers 1 through 3 are the most important to telecommunications networks. It's important to understand that this model is exactly that—a model. It's a conceptual framework that is useful for describing the necessary functions required of a network device or member. No actual networking product implements the model precisely as described.

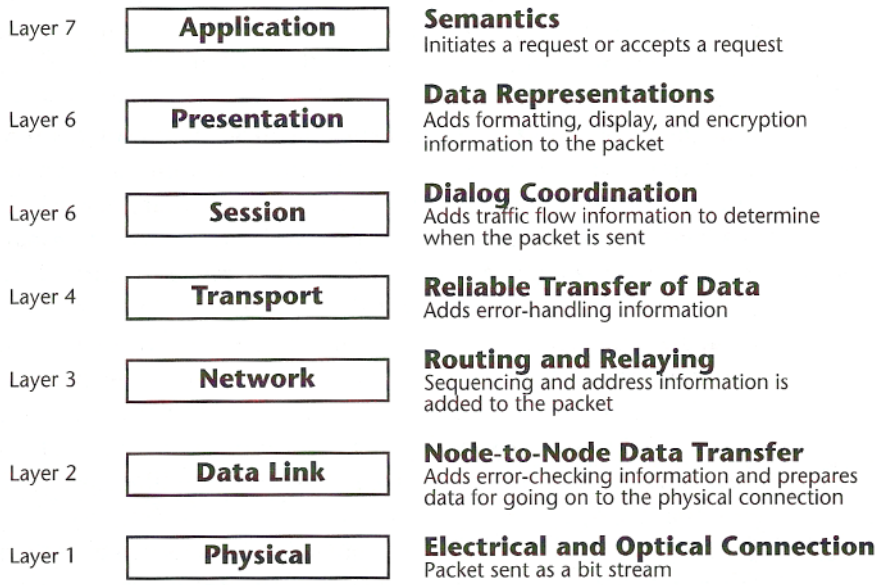

Figure 3.1 The OSI reference model [4].

Layer 7, the application layer, is responsible for exchanging information between the programs that are running on a computer and other services on a network. This layer supports application and end-user processes. It acts as a window for applications to access network services. It handles general network access, flow control, error recovery, and file transfer. Examples of application layer protocols include File Transfer Protocol (FTP), Telnet, Simple Mail Transfer Protocol (SMTP), and Hyper-text Transfer Protocol (HTTP).

Layer 6, the presentation layer, formats information so that a software application can read it. It performs transformations on the data to provide a standardized application interface and common communication services. It offers services such as encryption, compression, and reformatting. The presentation layer adds a field in each packet that tells how the information within the packet is encoded. It indicates whether any compression has been performed and, if it has, it indicates what type of compression, so that the receiver can decompress it properly. It also indicates whether there has been any encryption, and if there has, it indicates what type, so that the receiver can properly decrypt it. The presentation layer ensures that the transmitter and receiver are seeing the information in the same format.

Layer 5, the session layer, supports connections between sessions and handles administrative tasks and security. It establishes and monitors connections between computers, and it provides the control structure for communication between applications. Examples of session layer protocols include NetBIOS, DSN, and Lightweight Directory Access Protocol (LDAP).

Layer 4, the transport layer, corrects transmission errors and ensures that the information is delivered reliably. It provides an end-to-end error recovery and flow control capability. It deals with packet handling, repackaging of messages, division of messages into smaller packets, and error handling. Examples of transport layer protocols include Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Sequenced Packet Exchange (SPX).

Layer 3, the networking layer, identifies computers on a network and determines how to direct information transfer over that network. In other words, it is a routing and relaying layer. It defines how to move information between networks as well as between devices. The key responsibility of this layer is to add the addressing information and the control functions needed to move the data through the network and its intermediate nodes. It is involved in establishing, maintaining, and terminating connections, including packet switching, routing, data congestion, reassembly of data, and translation of logical addresses to physical addresses. Examples of networking layer protocols are X.25, Internet Protocol (IP), Internetwork Packet Exchange (IPX), and Message Transform Program 3 (MTP3).

Layer 2, the data link layer, groups data into containers to prepare that data for transfer over a network. It puts the ones and zeros into a container that allows the movement of information between two devices on the same network. The protocols at this layer specify the rules that must be followed in transmitting a single frame between one device and another over a single data link. Bits are packaged into frames of data, and they include the necessary synchronization, error control, and flow control information. Examples of data link layer protocols in a LAN environment include Ethernet, Token Ring, and Fiber Distributed Data Interface (FDDI). Examples of data link layer protocols in a WAN environment include Frame Relay and ATM. Example of data link layer protocol within the PSTN is Signaling System 7's MTP2.

Layer 1, the physical layer, defines how a transmission medium connects to a computer, as well as how electrical or optical information is transferred on the transmission medium. The physical layer defines the types of cables or wireless interfaces that are allowed, the voltage levels used to represent the bits or the optical levels, the types of connectors that are allowed, and the types of transmission rates that can be supported. Every network service and every network device has definitions at the physical layer in terms of what it can physically interface with.

Protocols and Protocol Stacks

Protocols are the hardware or software components that carry out the OSI model guidelines for transferring information on a network. A protocol may be

one component or a collection of components that carry out a task. A protocol stack, or protocol suite, is made up of multiple protocols that are used to exchange information between computers. One protocol in the stack might be used to specify how network interface cards (NICs) communicate, and another might specify how a computer reads information from the NIC. Figure 3.2 shows the TCP protocol stack and how it relates to the OSI model.

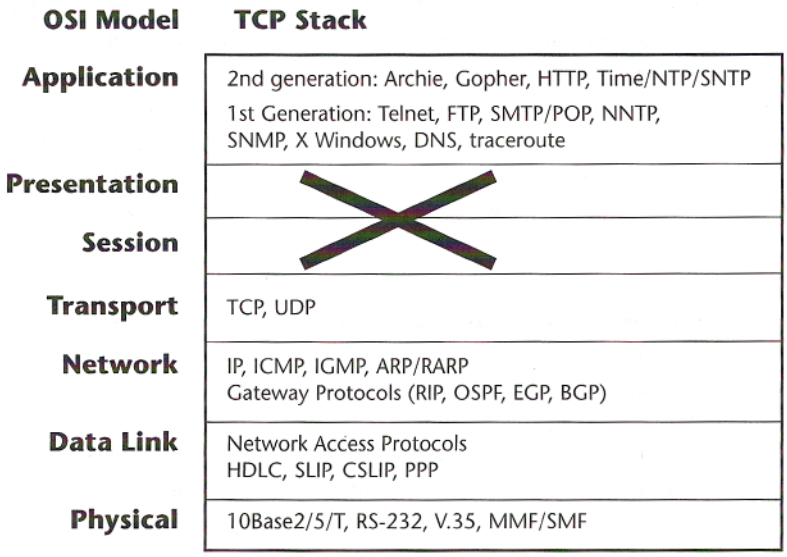

Figure 3.2 The OSI model versus the TCP/IP stack [4].

A layer is a section of a protocol stack that is responsible for performing one particular aspect of information transfer. Because some protocols are capable of performing one function, one layer in a protocol stack may not necessarily correspond to one layer in the OSI model.

3.3 Local/Wide Area Networking

Local Area Networks are, essentially, relatively high-speed data networks which interconnect computing devices within a small geographical or local area. They are generally owned by a single organization, and are independent of public networks. LANs are private networks in the sense that they serve a single organization within a limited environment. The costs associated with a LAN are the capital and installation costs which are met by the users (owners). The local environment usually refers to a site with a maximum span of a few kilometers, housing a relatively small number of buildings.

A wide area network (WAN) is a group of computer networks that are connected over long distances by telecommunications links, which can be

either wireline or wireless. A number of WAN links can be used, each of which was developed to address specific requirements in data communications. To meet specific network and application needs, a number of WAN techniques (whether deployed over public or private networks) have been developed and become popular over the years. Leased lines offer the greatest network management control. With leased lines, a known amount of bandwidth is provisioned, and no one else has access to it; also, you know who the users are. The disadvantage is that leased lines are very costly, you pay a premium for the comfort of having control over your own destiny.

To reduce the costs associated with leased lines, many customers migrate to Frame Relay services. Frame Relay was introduced in the early 1990s and was largely designed as an application for LAN-to-LAN interconnection. Because numerous subscribers share its virtual circuits, Frame Relay offers great cost-efficiency as compared to leased lines. Another WAN alternative to leased lines is Asynchronous Transfer Mode (ATM), which is perhaps the best solution, especially in environments that have intensive multimedia applications or other high-bandwidth applications. Virtual private networks (VPNs) are increasingly being used in WANs as well.

Despite the fact that there are numerous WAN options, all of which can offer various cost-efficiencies or performance improvements, the many separate networks in use translate into high costs associated with the overall infrastructure—for both the end user and the operator. One goal of WANs today is to integrate voice, data, and video traffic so that it runs through a common platform (in terms of access nodes) and through a common core network (in terms of the transport infrastructure). For example, the goal of ATM is to provide an integrated broad-band infrastructure that minimizes the range of required equipment that needs to be maintained on an on-going basis.

3.4 IP

IP is the central pillar of the Internet. The Internet Protocol was designed primarily for internetworking as being a simple protocol almost any network could carry. IP provides a “best effort” service over the network layer in the form of a datagram service. Data from the transport layer (TCP or UDP) is converted into IP datagrams and carried over the network. An IP network is a network of nodes connected using IP, but since those connections may themselves be formed over other networks, the IP network may be considered to be a network overlaid on networks of other protocols used for the actual data transport.

So as to allow wide application to different network types, IP places very few requirements on the underlying network. It considers the network to be both dumb and unreliable, so it is left to the end points or applications to confirm that application data has been delivered correctly, i.e., uncorrupted and meeting any timing requirements. If it is not, it is up to them to take any necessary action.

This simplicity extends to the intermediate nodes, or routers, within the IP network. These have two main functions to perform: to route and forward IP packets towards their destination, and to fragment larger blocks into smaller IP blocks should this be required for the underlying transport network technology.

3.4.1 IP Packet Format

The current version of IP is version 4. An IPv4 packet consists of a 13-field variable length header plus the data field itself. The maximum length of an IP datagram, header and data, is 64 Kbytes. The header fields subdivide roughly into blocks of 4 bytes, and the IP packet is represented as rows of 32 bits, as shown in Figure 3.3.

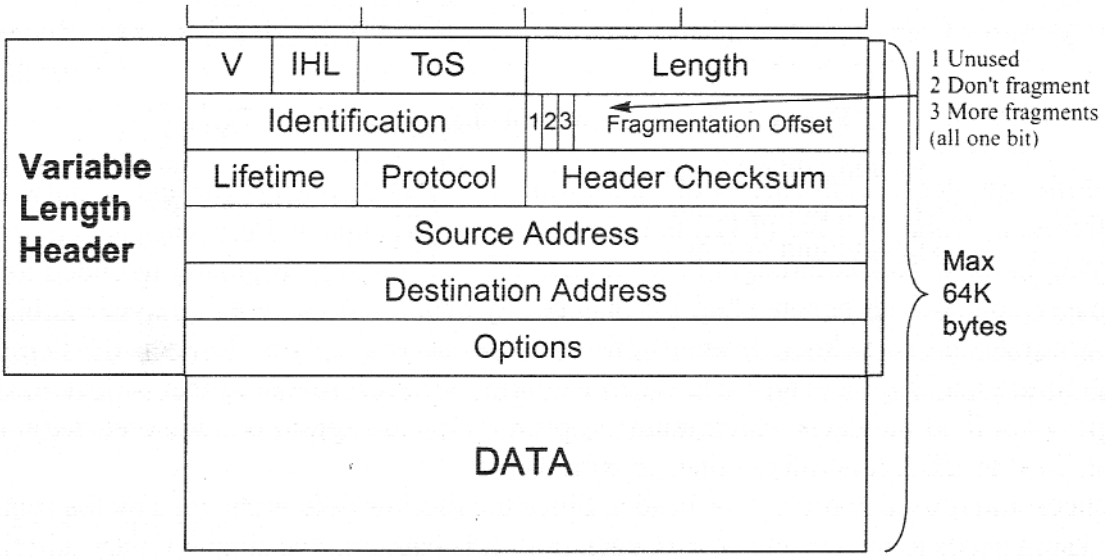

Figure 3.3 IP packet [3].

The first row corresponds to four fields associated with general formatting of the IP packet. The Version (V) field (4 bits) identifies the version of the protocol being used. The Internet Header Length (IHL) field, also 4 bits, indicates the length of the IP header in 32 bit blocks and thus whether any optional header features are invoked after the address fields. The third field is the 8 bit long Type of Service (ToS) field. In theory, this allows the host to specify priorities (0-7), to indicate whether delay, throughput or

reliability are of prime importance, but in practice this tends to be ignored by most routers. The total length field is fairly straightforward. A 16 bit field gives a maximum datagram length of 2^{16} = 65, 536 bytes, but typically datagrams are less than 1500 bytes, and are often limited to 576 bytes, the minimum size all IP transport mechanisms are guaranteed to carry.

The second row within the header is used to manage the process of fragmentation, where larger packets are broken down into smaller packets for transmission. Fragmentation can take place at intermediate routers in order to match the capabilities of the underlying network, but once fragmented, a packet is not reassembled until it reaches the receiver. The 16 bit identification field allows the host to determine which datagram a fragment belongs to. All parts of a datagram will have the same identifier field. The “Don't Fragment (DF)”, “More Fragments (MF)” and “Fragment Offset” relate to the management of datagram fragmentation. When set, DF indicates that fragmentation of the packet is not allowed. MF indicates that more fragments are following, and all fragments but the last one have this field set. The fragment offset indicates where in the current datagram the current fragment fits. The field is 13 bits long, 3 bits shorter than the 16 bit IP packet length for the DF and MF bits as well as one which is not currently used, so fragmentation takes place into IP datagrams of integer number of 8 byte blocks.

The third row deals with aspects of monitoring a packet's progress through a network. The Lifetime or Time To Live (TTL) field is used to count hops and prevent packets from overstaying in the network, through, for example, circular routing. Originally intended to be a time-based count, it has become hop-based count by default as most routers were unable to give an accurate and coordinated version of time. The router simply decrements the field by one each time a packet passes through. When the field reaches zero, the packet is discarded. The protocol field indicates which transport protocol the datagram is associated with, for example, TCP or UDP. It is fully defined in RFC 1700. The checksum is used to protect the header. Since the lifetime field is changed by the router, the checksum needs to be recalculated for each hop. The checksum is very basic and provides a rudimentary level of protection, although depending on their position, as little as two single bit errors can go unnoticed. Packets with corrupted headers are discarded.

3.5 Internet Transport Service Classes

The Internet model basically assumes that the network can only provide an unreliable connectionless service, and only provides two transport classes, TCP, which equates to the ISO Class 4 service, and UDP, which is connectionless. Both these protocols operate over Internet Protocol.

3.5.1 User Datagram Protocol

The User Datagram Protocol (UDP) provides a simple connectionless mechanism for applications to exchange messages. While the fact that no connection is established means that the protocol has very low signaling overheads, it also means that there is no error or flow control. For some real-time services with very low delay requirements like voice transmission, a lack of flow control is an advantage, since any lost data would not be repeated anyway. UDP is also used for broadcast messages since a connection-orientated approach is not then appropriate, and for periodic messages like routing table updates where if the data is lost, it does not matter since the existing data can be retained until the next update. Some services, like DNS, which could use TCP, usually use UDP for efficiency. Rather than wasting the time for setting up a connection, as well as adding to the load of the host, a connectionless UDP request is made. If the request or its response is lost, another DNS server will be tried after a timeout.

The UDP PDU has four 16 bit fields, shown in Figure 3.4, with the source and destination ports referring to application processes on local and remote hosts. The source port is optional; it is set to zero if it is not used. The length field refers to the total number of octets in datagram including the header.

Note that the UDP segment does not include the address of the recipient, only the port number. This is because UDP is designed for transport over IP, and the IP header holds that information. There is still the problem that since the UDP header does not contain that information directly, a UDP datagram could be delivered to the wrong host and the transport layer would be unaware of this fact. To avoid this, the source address, destination address, protocol and length of the IP packet header are considered to form a pseudo-header which is added to the UDP datagram for the purposes of calculating the frame check sequence. If the datagram is delivered to the wrong host, the checksum will fail. The checksum is optional and is set to zero if not required, but if it is used it can be checked by intermediate routers which can drop corrupted packets to save on network

load. Note that this means that UDP is dependent on IP and cannot be used as a transport protocol for other network protocols. That dependency works only one way, however. IP is not restricted to carrying UDP.

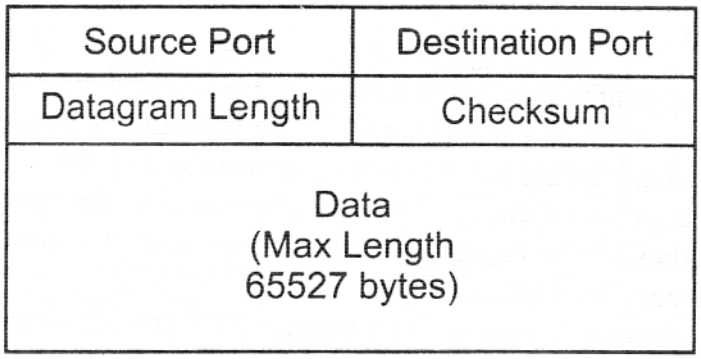

Figure 3.4 UDP packet [3].

3.5.2 Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) provides a connection-orientated communications protocol designed to work over IP (see Figure 3.5). Like UDP, TCP allows communication between specific processes on each host. So there can be many different connections between two hosts simultaneously. However, since the identification of a connection is done on the basis of ports on each host, there can only be one connection between a given source port and a given destination port on a pair of hosts.

Since IP only provides an unreliable transfer mechanism, and TCP is connection-orientated, TCP must provide mechanisms to ensure that any lost or corrupted data is replaced before delivery to the upper layers. This is done using ARQ mechanisms. TCP has three phases of operation: connection establishment, data transfer and connection termination. A three-way handshake is used in the connection phase because of the unreliable network service. TCP is a full duplex stream-oriented protocol. Data is passed to TCP from an upper layer protocol in a continuous fashion. It is then blocked arbitrarily into segments. Being full duplex, the protocol allows data to be sent in either direction between processes.

TCP Segment

The structure of a TCP segment is shown in Figure 3.6. Each segment serves a dual purpose. It sends data to its peer process at the other end of the transmitting link, and because the protocol is full duplex, acknowledges the receipt of data sent by its peer. The source and destination port fields are 16 bit numbers local to that host. These fields are combined with the IP

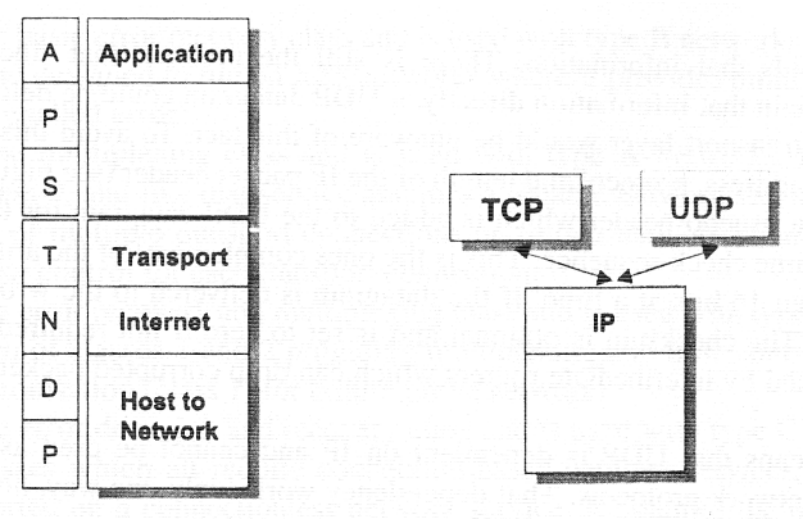

Figure 3.5 TCP [3].

address of the host to give the "socket" number, a unique address called the Transport Service Access Point (TSAP).

In order to know that data has been received correctly, TCP provides sequence numbering. Since the segments can vary in size, this numbering is done on the basis of individual bytes in the transmitted data stream rather than by segment. Each segment carries the sequence number in the second field and the acknowledgement number in the third field. The sequence number is the number of the first byte in the segment. The acknowledgement number is the number of the next byte which that host expects to receive from its peer, and therefore acknowledges to the peer that all data up to that point has been received correctly.

Each segment does not need to be acknowledged individually. The recommendation is to wait for half a second after receiving a packet before sending an acknowledgement in case further data is received, reducing the load of acknowledgement-only segments. If a second segment is received within this time, both are immediately acknowledged. If the half second passes without a further segment, an acknowledgement is sent for the first segment on its own. Lost segments could result either in data being lost or an acknowledgement being lost. The first case can be identified by the destination acknowledging a lower sequence number than expected. The data then has to be resent. A lost acknowledgement will cause the transmitter to resend the data after a time-out. If the receiver receives data it has already received correctly, it sends an additional acknowledgement and discards the duplicate data. In fact, an acknowledgement for a subsequent segment may be received within the time-out. Acknowledgements specify that all data up to the given byte has been received.

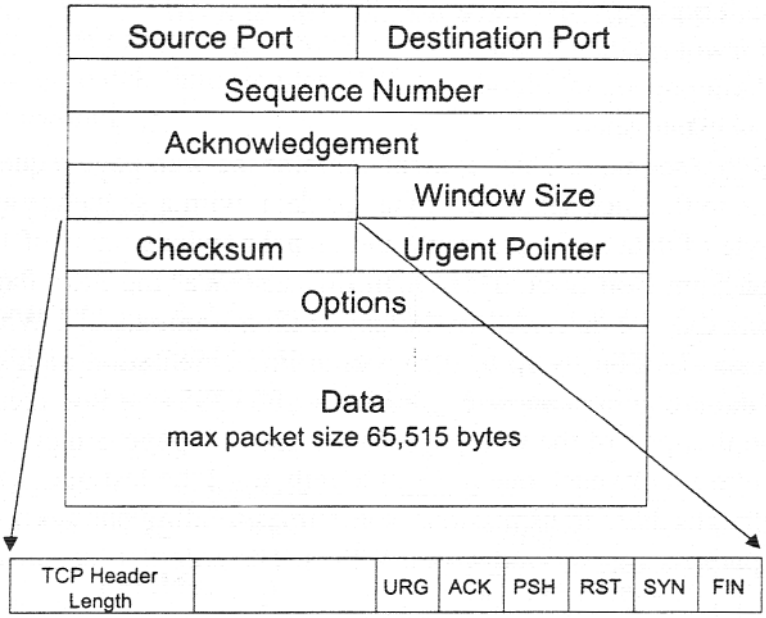

Figure 3.6 TCP segment [3].

TCP allows various optional elements in the header to allow for negotiating session parameters. A 4-bit header length field indicates the length of the header in 32 bit words. Usually the options field is not used, resulting in a header length of 20 bytes.
The flag field contains six flags as follows:

-URG: Indicates that the Urgent Pointer field is to be used. The latter contains the number of the byte in the data field where the urgent data ends. Although this is defined, many implementations do not make use of this feature.
-ACK: indicates that an acknowledgement is being carried in this PDU and thus the Acknowledgement Number field is carrying a valid number.
-PSH: Indicates that data the peer holds awaiting transmission should be sent on directly and not stored in a buffer (i.e. pushed).
-RST: Used to indicate a connection should be reset.
-SYN: Used to establish a connection.
-FIN: Used to terminate a connection.

The window size field indicates the number of bytes that the receiver would be able to accept. TCP has the same issue of a lack of receiver address as UDP, which has avoided it in the same way by constructing a pseudo-header with the source address, destination address, protocol and length of the IP packet header which is used with the other 16 bit words in the segment to construct the frame check sequence. However, the frame check sequence for TCP is not optional as it was in UDP. Like UDP, this restricts

TCP to working over IP rather than any network protocol, but does reduce the overall overhead of TCP and IP to 40 header bytes.

Having discussed the fundamental concepts related to this thesis, we then move on to concentrate more closely on issues directly affecting V/FoIP networks in the next chapter.

3.6 For Further Study

For further information regarding the concepts discussed in this chapter please refer to following resources:
[4]
[7]
[3]
[9]

Chapter 4

Voice/Fax over IP

4.1 Introduction

In this chapter, we introduce and discuss the concepts directly related to this thesis: Voice/Fax over IP architectures and standards. In Section 2, after this introduction, some fundamental issues related to IP telephony are explored. In the next section we will visit call signaling protocols and examine the current standards. Fax over IP with its 2 flavors, i.e. real-time and non-real-time, are examined thoroughly in Section 4. IP telephony quality of service is discussed in Section 5. Finally in Section 6, some statistics are presented to justify the action of venturing into producing yet another set of protocols and architectures for F/VoIP. As always, chapter ends with introducing materials for further studying.

4.2 IP Telephony Fundamentals

4.2.1 Introduction

Internet telephony is the real-time delivery of voice and other multimedia data types between two or more parties, across networks using the Internet protocols, and the exchange of information required to control this delivery.

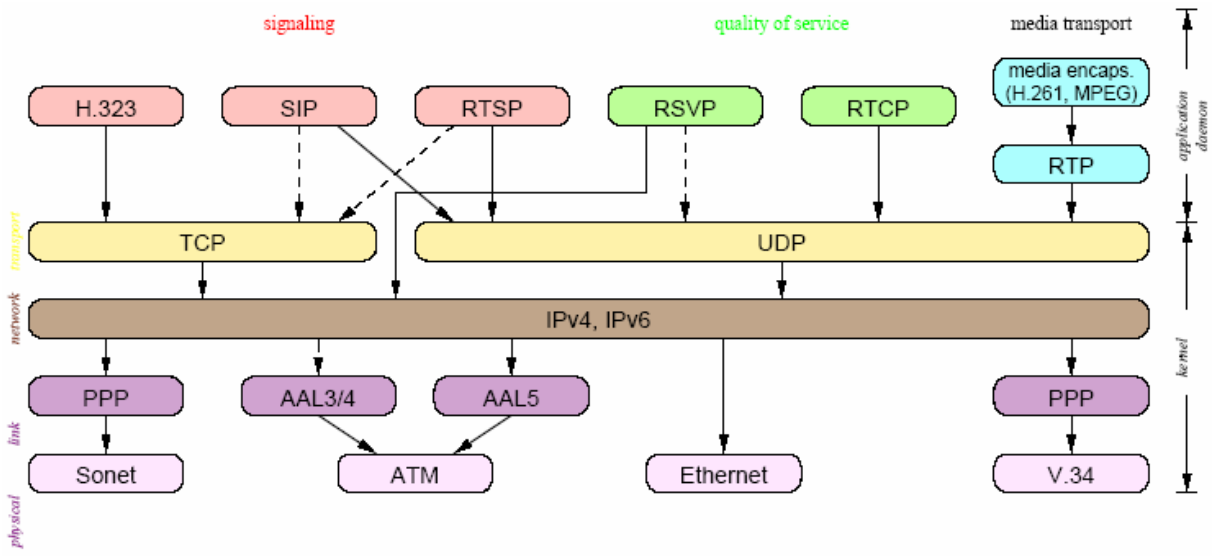

Figure 4.1 Internet telephony protocol stack [10].

Internet telephony offers the opportunity to design a global multimedia communications system that may eventually replace the existing telephony infrastructure.

Unlike in traditional telecommunications networks where distribution applications (radio, TV) and communications applications (telephone, fax) are quite distinct in terms of technology, user interface, communications devices and even regulatory environments, this is not the case for the Internet. The delivery of stored (or streaming) media and telephone-style applications can share almost the entire underlying protocol infrastructure.

Internet telephony differs from Internet multimedia streaming primarily in the control and establishment of sessions, the “signaling”. While we generally assume that a stored media resource is available at a given location, identified at different levels of abstraction by a URL (Uniform Resource Locator), participants in phone calls are not so easily located. Personal mobility, call delegation, availability, and desire to communicate make the process of signaling more complex. In the Internet, the Real Time Streaming Protocol (RTSP) is the standard protocol for controlling multimedia streams. The Session Initiation Protocol (SIP) is used for signaling Internet telephony services.

Both RTSP and SIP are part of a protocol stack that has emerged from the Internet Engineering Task Force (IETF) - the Internet Multimedia Conferencing Architecture. The protocols encompass both IPtel services and stored media services in an integrated fashion. Figure 4.1 depicts the stack, along with other protocols likely to be used for both Internet streaming media and Internet telephony. Unlike circuit-switched telephony, Internet telephony services are built on a range of packet switched protocols. For example, the functionality of the SS7 ISUP and TCAP telephony signaling

protocols encompass routing, resource reservation, call admission, address translation, call establishment, call management and billing. In an Internet environment, routing is handled by protocols such as the Border Gateway Protocol (BGP), resource reservation by the Resource Reservation Protocol (RSVP) or other resource reservation protocols. SIP translates application-layer addresses, establishes and manages calls. There is currently no Internet telephony billing protocol in the Internet, although RADIUS or DIAMETER, in combination with SIP authentication, may initially serve that purpose for calls through Internet telephony gateways. Having a number of different protocols, each serving a particular function, allows for modularity, flexibility, simplicity, and extensibility. End systems or network servers that only provide a specific service need only implement that particular protocol, without interoperability problems. Furthermore, protocol components can be reused in other applications, avoiding re-invention of specific functions in each application.

Even though the term Internet telephony is often associated with point-to-point voice service, none of the protocols described here are restricted to a single media type or unicast delivery. Indeed, one of the largest advantages of Internet telephony compared to the Plain Old Telephone System (POTS) is the transparency of the network to the media carried, so that adding a new media type requires no changes to the network infrastructure. Also, at least for signaling, the support of multiparty calls differs only marginally from two-party calls.

The protocols mentioned, and the rich services they provide, are just one part of the picture for IP telephony. As it was designed for data transport, the Internet currently offers only best effort service. The result is that voice packets suffer heavy losses and significant delays when there is network congestion, making the speech quality poor. A number of efforts in the area of Quality of Service (QoS) management are underway in order to address this issue.

Typical Internet applications use TCP/IP, whereas VoIP uses RTP/UDP/IP. Although IP is a connectionless best effort network communications protocol, TCP is a reliable transport protocol that uses acknowledgments and retransmission to ensure packet receipt. Used together, TCP/IP is a reliable connection-oriented network communications protocol suite. TCP has a rate adjustment feature that increases the transmission rate when the network is uncongested, but quickly reduces the transmission rate when the originating host does not receive positive acknowledgments from the destination host. TCP/IP is not suitable for real-time communications, such as speech transmission, because the acknowledgment/retransmission feature would lead to excessive delays.

UDP provides unreliable connectionless delivery service using IP to transport messages between end points in an internet. RTP (Real-Time Transport Protocol), used in conjunction with UDP, provides end-to-end network transport functions for applications transmitting real-time data, such as audio and video, over unicast and multicast network services. RTP does not reserve resources and does not guarantee quality of service. A companion protocol RTCP (Real-time control protocol) does allow monitoring of a link, but most VoIP applications offer a continuous stream of RTP/UDP/IP packets without regard to packet loss or delay in reaching the receiver.

4.2.2 Differences between Internet Telephony and the PSTN

Internet telephony differs in a number of aspects from the PSTN, both in terms of architecture and protocols. These differences affect the design of telephony services.

Fundamentally, IP telephony relies on the “end-to-end” paradigm for delivery of services. Signaling protocols are between the end systems involved in the call; network routers treat these signaling packets like any other data, ignoring any semantics implied by them. Note, however, that IP telephony can make use of “signaling routers” (which are effectively proxies) to assist in functions such as user location. In this case, these proxies can be used for routing only of initial signaling messages. Subsequent messages can be exchanged end-to-end. As a consequence of the end-to-end signaling paradigm, call state is as well end to end, as are instantiation of many telephony features.

The Internet itself is both multi-service and service-independent. It provides packet-level transport, end-to-end, for whatever services are deployed at the end systems through higher layer protocols and software. This leads to tremendous flexibility and extensibility. New application level services, such as the web, email, and now IP telephony, can be created and deployed by anyone with access to the Internet.

IP Telephony separates call setup from reserving resources. In the Internet, protocols such as RSVP are used to reserve resources. These protocols are application independent, and reservations may take place before or after actual flow of data begins. When used after the flow of data begins, the data will be treated as best effort. As a result, IP telephony can be used without per-call resource reservation in networks with sufficient capacity.

Due to the limited signaling abilities of PSTN end systems, PSTN addresses (phone numbers) are overloaded with at least four functions: end point identification, service indication, indication of who pays for the call and

carrier selection. The PSTN also ties call origination with payment, except as modified by the address (800 numbers) or specific manual features. IPtel addresses, which are formulated as URL's, are used solely for endpoint identification and basic service indication. The other functions, such as payment and carrier selection, are more readily handled by the protocols, such as RSVP and RTSP, which carry the addresses.

The open, multi-service, end-to-end nature of the Internet also means that various components of telephony services can be provided by completely different service vendors (of course, agreement on protocols is necessary for interoperability). For example, one vendor can provide a name to IP address mapping service, another can provide voice-mail, another can provide mobility services, while yet another can provide conferencing services. Furthermore, the end-to-end nature of the Internet means that anyone with an Internet connection can run and operate such a service. This leads to an easy-entry, highly competitive marketplace for all Internet services, such as IP telephony.

This separation of functionality also simplifies the number portability problem. As an organization may provide just a name mapping service, a user can change other service providers (such as their voicemail provider or ISP) without a change in name. Changing name providers may require a change in name. However, automated white-pages services allow another layer of indirection which further alleviates the number portability problem.

4.2.3 Features of Internet Telephony

The architectural differences described in the previous section lead to a number of advantages from both a user perspective and a carrier perspective:

Adjustable quality: While IPtel currently has the reputation for low quality (due to low bitrate codecs, in part) there is no reason (except lack of bandwidth) why the same technology cannot supply high quality music. Because the Internet is not a service-specific network, the media exchanged is chosen entirely by end systems. As such, end systems can control the amount of compression based on network bandwidth or the content to be transmitted. For example, music-on-hold (as it is not speech) is not suitable for very low bit-rate speech codecs.

Security: The Internet has the reputation as being insecure. SIP can encrypt and authenticate signaling messages; RTP supports encryption of media. Together, these provide secure communications.

User identification: Standard POTS and ISDN telephone service offers caller ID indicating the number (or, occasionally, name) of the caller, but during a bridged multi-party conference, there is no indication of who is talking. The real-time transport protocol (RTP) used for Internet telephony easily supports talker indication in both multicast and bridged configurations and can convey more detailed information if the caller desires.

User interface: Most POTS and ISDN telephones have a rather limited user interface, with at best a two line liquid crystal display or, in the public network, cryptic commands like "*69" for call-back. Advanced PSTN features such as call-forwarding are rarely used or customized, since the sequence of steps is typically not intuitive. This is due in part to the limited signaling capabilities of end systems, and the general notion of "network intelligence" as compared to "end system intelligence". As IP telephony end systems have much richer signaling capabilities, the graphical user interface offered by Internet telephony can be more readily customized and offer richer indications of features, process and progress.

Computer-telephony integration: Because of the complete separation of data and control paths and the separation of phone equipment from the PC's controlling them, computer-telephony integration (CTI) is very complex. Much of the call handling functionality can be easily accomplished once the data and control path pass through intelligent, network-connected end systems.

Feature Ubiquity: The current phone system offers very different features depending on whether the parties are connected by the same Private Branch Exchange (PBX), reside within the same local calling area or are connected by a long-distance carrier. Even trivial features such as caller ID only work for a small fraction of international calls, for example. A PBX may not allow a call to be forwarded outside that PBX, or causes the forwarded call to still go through the PBX. Internet telephony does not suffer from this problem. This is because the Internet protocols are internationally used, and because services are defined largely by the end systems.

Multimedia: Adding additional media such as video, shared whiteboards or shared applications is much easier in the Internet environment compared to the POTS and ISDN, as multiplexing is natural for packet networks. This makes signaling protocols simpler as well, as issues such as B-channel allocation and synchronization are non-existent in the Internet.

Carriers benefit as well:

Silence suppression and compression: Sending audio as packets makes it easy to suppress silence periods, further reducing bandwidth consumption, particularly in a multi-party conference or for voice announcement systems. Unlike the PSTN, which generally does such silence suppression across trans-continental links, IP telephony performs silence suppression at the endpoints. Furthermore, as packet networks are much more suitable to multiplexing, no network support is required to take advantage of end system silence suppression. This leads to a reduction in cost to perform the silence suppression (as it's distributed to end systems where it can be done cheaply), and an improvement in the scope of its benefits. Furthermore, compression can be used at end systems to reduce bandwidth consumption across the entire network. Unfortunately, compression is at odds with enhanced voice quality services. However, there exist codecs which can compress wideband speech to 16 kb/s, which can give both excellent voice quality and reduced bandwidth compared to the PSTN. Note that silence suppression and compression compensate for the decreased efficiency of packet switching.

Shared facilities: The largest operational savings can probably come from provisioning and managing a single, integrated network, rather than separate voice, data and signaling networks.

Advanced services: From first experiences and protocols, it appears to be far simpler to develop and deploy advanced telephony services in a packet-switched environment than in the PSTN. Internet protocols, such as SIP that support standard CLASS (Custom Local Area Signaling Services) features (such as Call Forward No Answer) take only a few tens of pages to specify. They can perform the functions of both the user-to-network signaling protocols such as Q.931 as well as the network signaling (ISUP, Signaling System 7).

Separation of voice and control flow: In telephony, the signaling flow, even though carried on a separate network, has to "touch" all the intermediate switches to set up the circuit. Since packet forwarding in the Internet requires no setup, Internet call control can concentrate on the call (rather than connection) functionality.

4.3 Call Signaling

Without any established standards, most early implementations were based on proprietary technology. As these packet telephony networks grew and interconnection dependencies emerged, it became clear that the industry

needed standard protocols. Several groups took up the challenge, resulting in independent standards, each with its own unique characteristics. In particular, network equipment suppliers and their customers were left to sort out the similarities and differences between four different signaling and call-control protocols:

- H.323
- Media Gateway Control Protocol (MGCP)
- Session Initiation Protocol (SIP)
- H.248/Megaco

In the process of implementing workable solutions, network engineers had to determine how each of these protocols worked and which ones were best for particular networks and applications. This section provides some guidance and understanding of these protocols.

4.3.1 IP Telephony Standards

VoIP comprises many standards and protocols. Basic terminology must be understood in order to understand the applications and usage of VoIP. The following definitions serve as a useful starting point (the protocols are listed in alphabetical order):

- H.248 is an ITU Recommendation that defines "Gateway Control Protocol." H.248 is the result of a joint collaboration between the ITU and the Internet Engineering Task Force (IETF). It is also referred to as IETF RFC 2885 (Megaco), which defines a centralized architecture for creating multimedia applications, including VoIP. In many ways, H.248 builds on and extends MGCP.
- H.323 is an ITU Recommendation that defines "packet-based multimedia communications systems." In other words, H.323 defines a distributed architecture for creating multimedia applications, including VoIP.
- IETF refers to the Internet Engineering Task Force (http://www.ietf.org/), a community of engineers that seeks to determine how the Internet and Internet protocols work, as well as to define the prominent standards.
- ITU is the International Telecommunication Union, an international organization within the United Nations System (http://www.unsystem.org/) where governments and the private sector coordinate global telecom networks and services.
- Megaco, also known as IETF RFC 2885 and ITU Recommendation H.248, defines a centralized architecture for creating multimedia applications, including VoIP.

• Media Gateway Control Protocol (MGCP), also known as IETF RFC 2705, defines a centralized architecture for creating multimedia applications, including VoIP.
• Real-Time Transport Protocol (RTP), also known as IETF RFC 1889, defines a transport protocol for real-time applications. Specifically, RTP provides the transport to carry the audio/media portion of VoIP communication. RTP is used by all the VoIP signaling protocols.
• Session Initiation Protocol (SIP), also known as IETF RFC 3261, defines a distributed architecture for creating multimedia applications, including VoIP.

4.3.2 Understanding Centralized and Distributed Architectures

In the past, all voice networks were built using a centralized architecture in which dumb endpoints (telephones) were controlled by centralized switches. Although this model worked well for basic telephony services, it mandated a trade-off between simplified management and endpoint and service innovation.

One of the benefits of VoIP technology is that it allows networks to be built using either a centralized or a distributed architecture. This flexibility allows companies to build networks characterized by both simplified management and endpoint innovation, depending on the protocol used.

In general, centralized architectures are associated with MGCP and H.248/Megaco protocols. These protocols were designed for a centralized device—called a media gateway controller or call agent—that handles switching logic and call control. The centralized device talks to media gateways, which route and transmit the audio/media portion of the calls (the actual voice information).

In centralized architectures, the network intelligence is centralized and endpoints are relatively dumb (with limited or no native features). Although most centralized VoIP architectures use MGCP or H.248/Megaco protocols, it is also possible to build SIP or H.323 networks in a centralized fashion using back-to-back user agents (B2BUAs) or gatekeeper routed call signaling (GKRCS), respectively.

Advocates of centralized VoIP architectures favor this model because it centralizes management, provisioning, and call control. It simplifies call flows for replicating legacy voice features. And it is easy for legacy voice engineers to understand. Critics of centralized architectures claim that it stifles innovation of endpoint features and that it will become a hindrance when building VoIP services that move beyond legacy voice features.

Distributed architectures are associated with H.323 and SIP protocols. These protocols allow network intelligence to be distributed between

endpoints and call-control devices. Intelligence in this instance refers to call state, calling features, call routing, provisioning, billing, or any other aspect of call handling. The endpoints can be VoIP gateways, IP phones, media servers, or any device that can initiate and terminate a VoIP call. The call-control devices are called gatekeepers in an H.323 network, and proxy or redirect servers in an SIP network.

Advocates of distributed architectures favor this model because of its flexibility. It allows VoIP applications to be treated like any other distributed IP application, and it allows the flexibility to add intelligence to either endpoints or call-control devices, depending on the business and technology requirements of the network. Distributed architectures are usually well understood by engineers who run IP data networks. Critics of distributed architectures commonly point to the existing PSTN infrastructure as the only reference model that should be used when trying to replicate legacy voice services. They also note that distributed networks tend to be more complex.

4.3.3 H.323

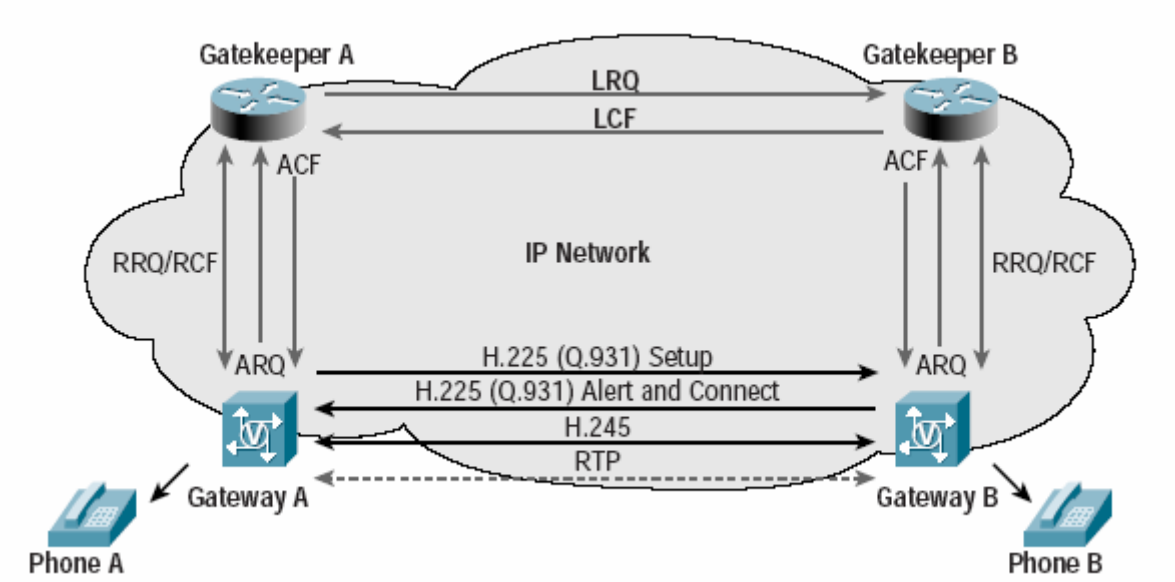

Figure 4.2 H.323 Networks [11].

H.323 was originally created to provide a mechanism for transporting multimedia applications over local-area networks. Although H.323 is still used by numerous vendors for videoconferencing applications, it has rapidly evolved to address the growing needs of VoIP networks.

H.323 is considered an “umbrella protocol” because it defines all aspects of call transmission, from call establishment to capabilities exchange to network resource availability. H.323 defines Registration, Admission, and

Status Protocol (RAS) protocols for call routing, H.225 protocols for call setup, and H.245 protocols for capabilities exchange.

H.323 is based on the Integrated Services Digital Network (ISDN) Q.931 protocol, which allows it to easily interoperate with legacy voice networks such as the PSTN or Signaling System 7 (SS7).

As a protocol used in a distributed architecture, H.323 allows companies to build large-scale networks that are scalable and resilient. It provides mechanisms for interconnecting with other VoIP networks, and supports network intelligence on either the endpoints or the gatekeepers.

4.3.4 MGCP/H.248/Megaco

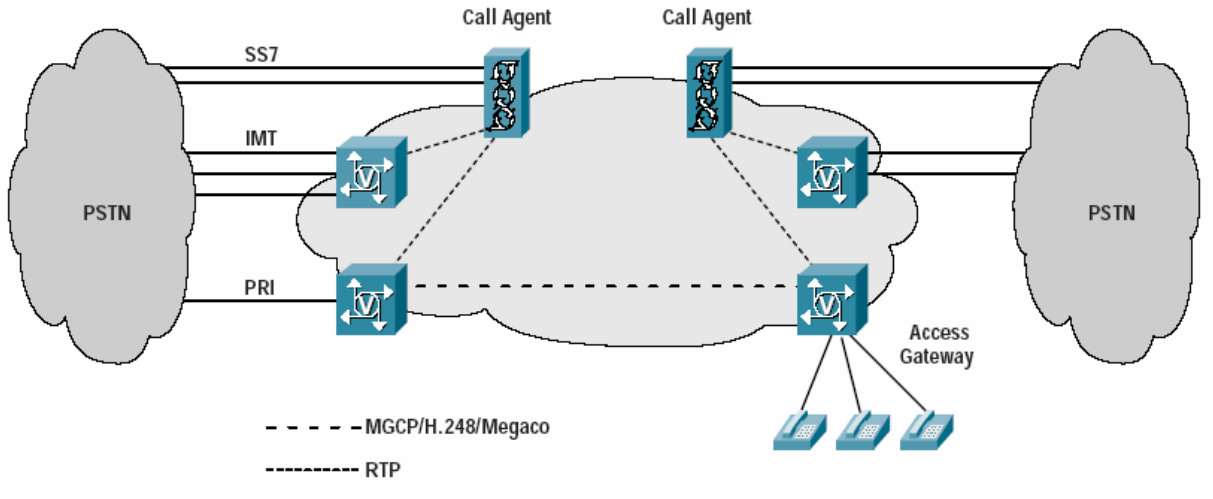

Figure 4.3 MGCP/H.248/Megaco Networks [11].

MGCP and H.248/Megaco were designed to provide an architecture where call control and services could be centrally added to a VoIP network. In that sense, an architecture using these protocols closely resembles the existing PSTN architecture and services.

MGCP and H.248/Megaco define most aspects of signaling using a model called packages. These packages define commonly used functionality, such as PSTN signaling, line-side device connectivity, and features such as transfer and hold. In addition, Session Definition Protocol (SDP) is used to convey capabilities exchange.

In a centralized architecture, MGCP and H.248/Megaco allow companies to build large-scale networks that are scalable and resilient. It provides mechanisms for interconnecting with other VoIP networks and for adding intelligence and features to the call agent.

4.3.5 SIP

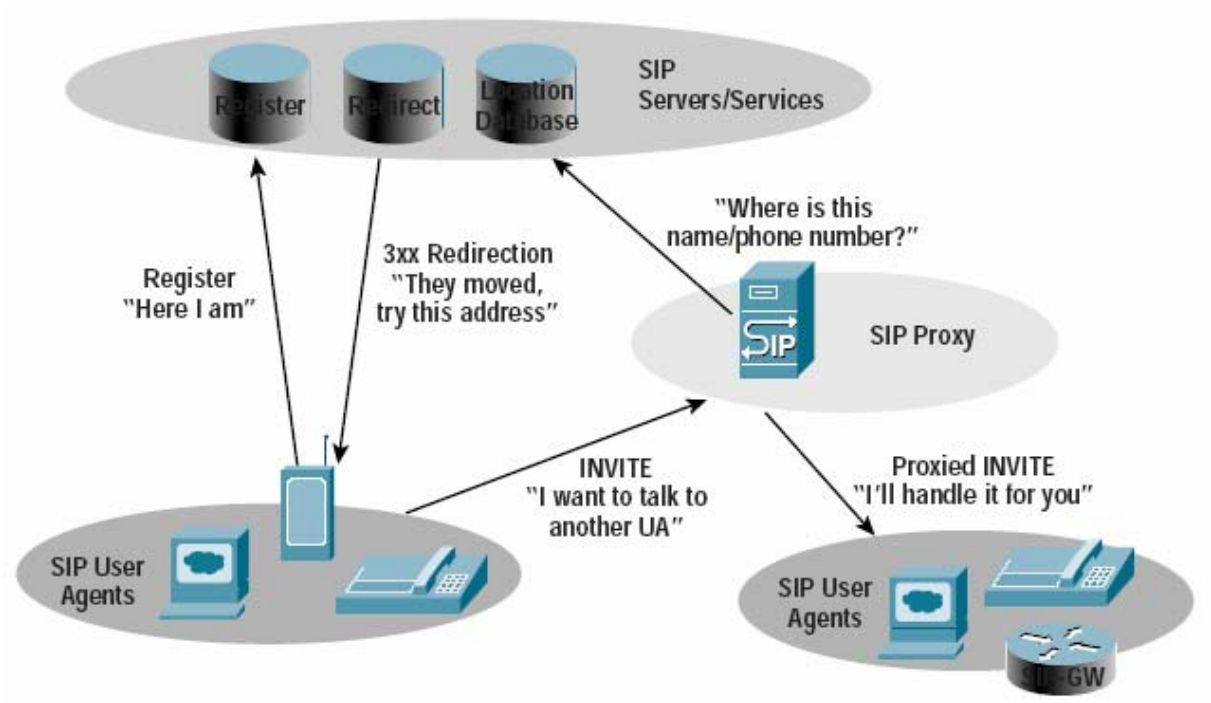

Figure 4.4 SIP Networks [11].

SIP was designed as a multimedia protocol that could take advantage of the architecture and messages found in popular Internet applications. By using a distributed architecture—with URLs for naming and text-based messaging—SIP attempts to take advantage of the Internet model for building VoIP networks and applications. In addition to VoIP, SIP is used for videoconferencing, instant messaging, fax and other session types.

As a protocol, SIP only defines how sessions are to be set up and torn down. It utilizes other IETF protocols to define other aspects of VoIP and multimedia sessions, such as SDP for capabilities exchange, URLs for addressing, Domain Name System (DNS) for service location, and Telephony Routing over IP (TRIP) for call routing.

Although the IETF has made great progress defining extensions that allow SIP to work with legacy voice networks, the primary motivation behind the protocol is to create an environment that supports next-generation communication models that utilize the Internet and Internet applications.

As a protocol used in a distributed architecture, SIP allows companies to build large-scale networks that are scalable and resilient. It provides mechanisms for interconnecting with other VoIP networks and for adding intelligence and new features on either the endpoints or the SIP proxy or redirect servers.

4.3.6 Interconnecting VoIP Protocols

VoIP networks continue to be deployed at a rapid pace, and VoIP vendors and service providers continue to add new functionality. Because vendor support for each protocol differs and companies have varying business requirements, it is very likely that VoIP networks will continue to be made up of multiple protocols.

Having various protocols gives customers the flexibility they need to connect services from multiple carriers. Using standards, even multiple standards, still simplifies deployment of multivendor endpoints and increases options for network management and provisioning.

As companies expand their networks, they are faced with choices about how to interconnect segments using differing VoIP protocols. These choices often fall into one of three categories:

• Translation through time-division multiplexing (TDM)—In this model, a company uses either TDM equipment or VoIP gateways to translate from one protocol domain to another. The benefits of this model are that it can be used today. The downside is that it introduces latency into the VoIP network, and involves yet another protocol translation (VoIP no. 1 <-> TDM <-> VoIP no. 2). This model is usually considered as a short-term solution until IP-based protocol translators are available.

• Single protocol architecture—In this model, a company moves all its VoIP devices and services to a single protocol, simplifying the network as a whole. The downside to this approach is that it might not be possible to migrate existing equipment to support the new protocol, a situation that can limit the company's ability to take advantage of some existing services. In addition, it limits the potential connectivity to other networks that are using other VoIP signaling protocols.

• Protocol translation—In this model, a company uses IP-based protocol translators to interconnect two or more VoIP protocol domains. IP translators allow a company to retain the flexibility of using multiple VoIP protocols, do not introduce the delay problems that additional TDM interconnections do, and do not require a wholesale replacement or swap of existing equipment. The downside to this approach is that there is no standard for protocol translation, so not all VoIP protocol translators are exactly the same. Although the IETF has attempted to define a model for translating H.323 to SIP, it involves more than just building a protocol-translation box.

As shown in Table 4.1, although protocols are somewhat similar, they do have some differences. Vendors of protocol translators need in-depth knowledge of all the protocols being used in the VoIP network, and they

must be aware of how various VoIP components utilize different aspects of the protocol.

Table 4.1 Details of IP Telephony Protocols [11].

	H.323	SIP	MGCP/H.248/Megaco
Standards body	ITU	IETF	MGCP/Megaco—IETF; H.248—ITU
Architecture	Distributed	Distributed	Centralized
Current version	H.323v4	RFC2543-bis07	MGCP 1.0, Megaco, H.248
Call control	Gatekeeper	Proxy/Redirect Server	Call agent/media gateway controller
Endpoints	Gateway, terminal	User agent	Media gateway
Signaling transport	Transmission Control Protocol (TCP) or User Datagram Protocol (UDP)	TCP or UDP	MGCP—UDP; Megaco/H.248—both
Multimedia capable	Yes	Yes	Yes
DTMF-relay transport	H.245 (signaling) or RFC 2833 (media)	RFC 2833 (media) or INFO (signaling)	Signaling or RFC 2833 (media)
Fax-relay transport	T.38	T.38	T.38
Supplemental services	Provided by endpoints or call control	Provided by endpoints or call control	Provided by call agent

4.4 Fax over IP

4.4.1 Introduction

Facsimile transmission presents a major implementation challenge in the new packet networks. Voice support is difficult enough, with its own stringent requirements for maintaining the quality of service and user experience of the PSTN, but fax transmission escalates the problem to a higher level. Simply speaking, for the new VoIP networks to succeed, it is mandatory to support the legacy facsimile equipment, which attaches to POTS service. The installed base of fax equipment is so large that the unspoken requirement is for complete support for Group 3 fax at a minimum in any service deployment. A large percentage of homes with access to the Internet have at least simple PC-based fax capability, and as the number of telecommuters grows, this number is expected to increase.

We now explore the hurdles for supporting facsimile in a robust manner in a packet network. Let's revisit the PSTN for some of the basics. In POTS service, all calls are circuit switched and traverse digital TDM facilities between the local exchanges serving the calling and called fax terminals. These TDM facilities are extremely reliable and rarely, if ever, drop bits of digitized voiceband signals. The robustness of the PSTN network allows for reliable end-to-end message handshaking, modem

training, setting of equalizer circuits, and nearly error-free fax image transmission. If there are line quality issues affecting the analog signals, they are mostly between the subscriber's premise, and their respective COs. The user experience has been for a fax call to go through on the first try, almost all the time.

In contrast to the simple transport of voiceband data on the PSTN, packet networks come with many complexities regarding this transmission. Voiceband data is interpreted by machines, and they are far less tolerant than the human ear when things are not according to the highest quality. This is exactly the issue with fax support over packet networks. While the loss of a single packet during a human conversation may go unnoticed, the same lost packet during the handshake procedures of a fax call can result in a dropped call or lengthy recovery procedures with possible drop in speed as the outcome. All this is annoying and disturbing to users, especially if the affected call incurs long distance or international charges and must be repeated.

ITU-T Recommendation T.30 defines the procedures for the transmission of facsimile over the PSTN, which includes end-to-end capabilities negotiation between fax terminals and sending of image data encoded in a standard format. Transmission of images for Group 3 fax is specified in ITU-T Recommendation T.4. The need for an expedient solution to support facsimile in packet networks has resulted in additional specifications that preserve a local nature of Group 3 signaling and transmission. Local signaling executes between a fax terminal and the network equipment to which it connects. This type of network equipment is a media gateway that provides analog line connection to a legacy fax device or analog telephone with an RJ11 connector. The calling and called fax terminals are thus faked into believing they are negotiating fax transmission with the other terminal over the circuits and trunks of the PSTN. The packet infrastructure on the network side of the gateways is then used to transport the image data and re-create fax signaling sequences at each gateway toward their local fax terminal.

There are currently two methods that accomplish this operation and both are standards specified by the ITU. Recommendation T.37 specifies store and forward, non-real time techniques for sending facsimile with legacy equipment connected via a gateway to a packet network. Recommendation T.38 specifies a real-time operation which does not use voiceband data transport, but instead uses packetized information to carry both the handshake sequences and the digitized image data itself between the terminals. We discuss both of those techniques in this section.

Both the real-time and non-real-time methods for facsimile support over packet networks have their own benefits and shortcomings, from both a technical and an economic perspective. Fax costs can be a significant part of the POTS expenses for many corporations. Studies in this area have shown that a substantial portion of corporations' annual telecommunications budgets is allocated to long distance calls, of which about 40% represent fax calls. This is especially true in Asian countries. The availability of two methods to send faxes over packet networks has opened a new market in business segments, where previously there was no choice with POTS service. Both fax transmission techniques are expected to continue to enjoy success in the IP networks and find their way into VoIP service offerings as the technologies mature and their acceptance in the commercial sector increases.

Fax transmission over IP with a hop on the PSTN can be a difficult proposition, because the hybrid nature of the end-to-end network precludes the use of a single protocol specification. Pure IP-based facsimile in the wide area is still a few years away, and will start materializing when the success of IP networks reaches the point of carrier interoperability across packet domains. The commercial acceptance of packet-mode fax machines thus hinges on the success of VoIP networks to support facsimile transport in a compatible and robust manner, offering the same user experience as the PSTN.

The T.38 specification is an example of how interworking between a IP-based packet domain and the PSTN can be accomplished to support real-time fax. For instance, assume a client on an H.323 or SIP LAN segment wants to fax a document to a remote fax terminal of a POTS subscriber. Since the media path between the originating endpoint device (the calling fax machine) and the gateway to the TDM network is IP-based, the challenge becomes how to best map between the requirements of the T.30 and T.38 specifications at the network boundary. This must be done in a manner that preserves the timing integrity of the end-to-end signaling, or the process will not be successful and the fax service will not be robust.

Let's look now at the details of the most common fax transmission methods. The call flow diagram in Figure 4.5 shows a typical error-free fax transmission between an auto-calling and an auto-answer terminal, such as two PCs, or a PC and a fax machine, or two fax machines. We use this diagram as a basis for explaining the basic fax operation and issues.

A fax call in conventional Group 3 facsimile is completed in five phases:

1. Phase A - Call establishment

2. Phase B - Attributes, capabilities, and control signaling
3. Phase C - Single-page fax transmission
4. Phase D - End-of-page signaling and multipage notification
5. Phase E - Call termination

After dialing the remote fax number, an auto-calling fax terminal plays a CNG tone, which is an 1100 Hz tone ON for 0.5 secs and OFF for 3 secs. The called terminal answers the phone and plays the CED (Called Station ID) tone, which is a 2100 Hz tone with phase reversals, for 4 secs. The called station also sends a DIS (Digital Identification Signal) carrying the station capabilities and optionally two additional signals, NSF (Non Standard Facilities) and CSI (Called Subscriber Identification). NSF is used to identify requirements for the stations that are not explicitly covered by the ITU-T T series of recommendations. CSI provides the identity of the called subscriber in the form of a telephone number. This signaling exchange completes the call establishment—Phase A—of the process.

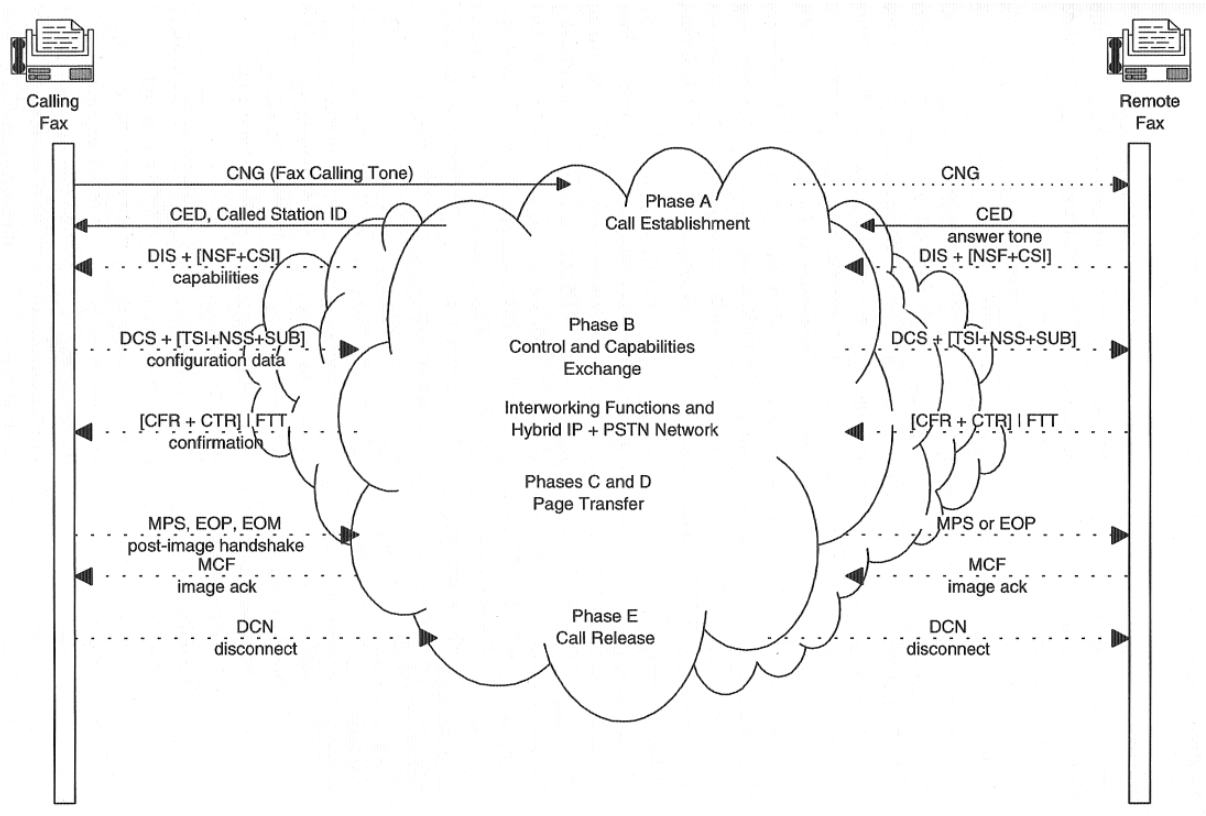

Figure 4.5 Conventional Group 3, T.30 Fax Transmission Call Flow [5].

Attributes, capabilities, and control signaling exchange is performed using the 300 bps modulation mode of V.21. Messages are preceded by a preamble consisting of one second of HDLC flags to condition the line for each turnaround. Phase B begins with the calling terminal sending DCS (Digital Command Signal) to send configuration data, complete the digital

setup, and respond to the DIS command. The TCF (Training Check Field) is sent by the sender, and the called station responds with CFR (Confirmation to Receive). At that time page transmission is ready to begin. The sender sends training flags to turn the line around and begins transmission, Phase C, in accordance to ITU-T Recommendation T.4.

At the end of each page the sender sends post-image handshakes (Phase D), which are either EOP (if that was the last page), EOM (End of Message), or MPS (MultiPage Signal) if there are multiple pages to send. MPS results in re-entering Phase C. If the sending session wishes to return to Phase B at the end of a page, it sends an EOM command. An MPS signal must be acknowledged by the receiver. This process continues until the last page has been transmitted, at which time the sending station disconnects by sending a DCN command to the receiver. This terminates the fax call.

There are some immediate considerations if we are to implement a packet network that intends to carry voiceband traffic as digitized samples of analog signals. The first issue is whether connections between the fax station endpoints use compression. If not, and PCM G.711 media encoding is specified, the main consideration is to avoid packet loss. Packet loss will affect the modulation at the far end when PCM packets are played out, and could result in page retransmissions, or worse, dropped calls.

If compression and perceptual coding are used, then all bets are off if we attempt to simply packetize the digitized and compressed analog signal to the remote station. Perceptual coding removes the characteristics of the original waveform and it cannot be reproduced. Even simple analog waveform compression can degrade the signal-to-noise ratio to the point where either the connection cannot be reliably established, or not established at all.

4.4.2 Fax over Packet Networks

The support of fax over packet networks has major business drivers. Fax is a major revenue earner for service providers and a major expenditure for corporations. More than half of the fax transmissions are long distance calls and therefore the desire to reduce costs is great. The number of installed fax machines continues to grow. The problem with the fax scenario we described in the previous section is that it is a POTS telephone call incurring toll charges, just like any voice call. The desire of the business sector to reduce the cost of fax has led to advances in packet technology to support facsimile transport.

We briefly mentioned the two ITU-T specifications for fax over packet

networks, and we visit them in this section. Recommendation T.37 defines procedures for store-and-forward fax transmission over the Internet. The functionality provided through this specification is simple facsimile transmission with non-real-time requirements. Simply stated, the fax begins and ends with a local device emulating Group 3 facsimile operation, and the actual transmission of the image to the intended final destination fax machine occurs in a second step, a short time later. Recommendation T.38 on the other hand, defines real-time procedures for fax support over IP networks, and this is the standard that has been accepted for enterprise and wide area networks. Both standards address legitimate commercial needs with slightly different business drivers. We examine the major attributes of both, and some major requirements they impose on the underlying packet networks.

4.4.3 Store-and-Forward Fax over IP Networks-T.37

Recommendation T.37 defines two modes for non-real-time fax, simple and full. Simple mode supports plain transmission of data, but capabilities negotiation between terminals may not take place and is undefined in the specification. All fax terminals must support simple mode. Image data is sent in TIFF format, specified in RFC 2301, Profile S with Modified Huffman Compression. It supports Group 3 standard and fine image resolutions.

The fundamental element for T.37 fax operation is the Internet Fax Gateway, which emulates Group 3 operation toward the attached stations, and has a direct connection to a packet network, acting as a host or router.

In Figure 4.6, the Internet Fax Gateway provides the protocol mapping between the standard Group 3 fax terminal on one side and the IP network on the other. The protocol between the gateway and the fax terminal is T.30, thus ensuring complete backward compatibility with the legacy fax machine.

On the network side, the gateway interfaces to the IP network and conforms to the requirements of RFC 3201, Profile S. It also conforms to RFC 2305 for errors in handling and delivery of the fax, information to trace the origin of the fax, ensuring MIME compliance at both ends of the IP fax gateways, sending notification to the originator of the fax regarding reception problems, and optionally using TIFF profiles for other fax types. The gateway must also implement the Simple Mail Transfer Protocol (SMTP). The Internet Fax Gateway functionality can be implemented inside the fax terminal, thus making it Internet-aware.

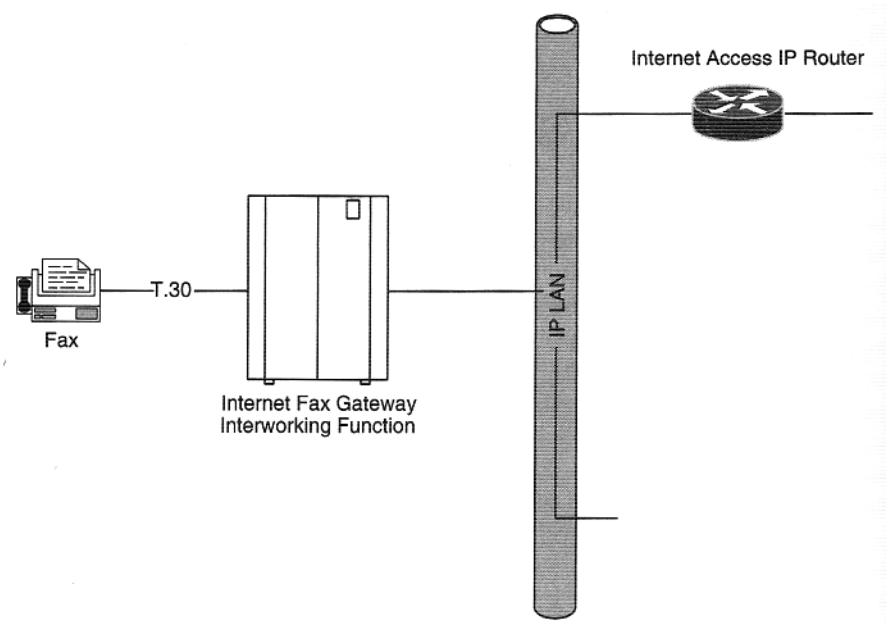

Figure 4.6 Internet Fax Gateway, Interworking Function [5].

The TIFF specification defines the method for describing, storing, and interchanging image data, such as facsimile and scanned documents. It defines a core set of fields, shown in Figure 4.7, along with the method to arrange the image data in a file which includes all the document pages in chained fashion. The exact header field definition for the image encoding is specified in RFC 2301, "File Format for Internet Fax".

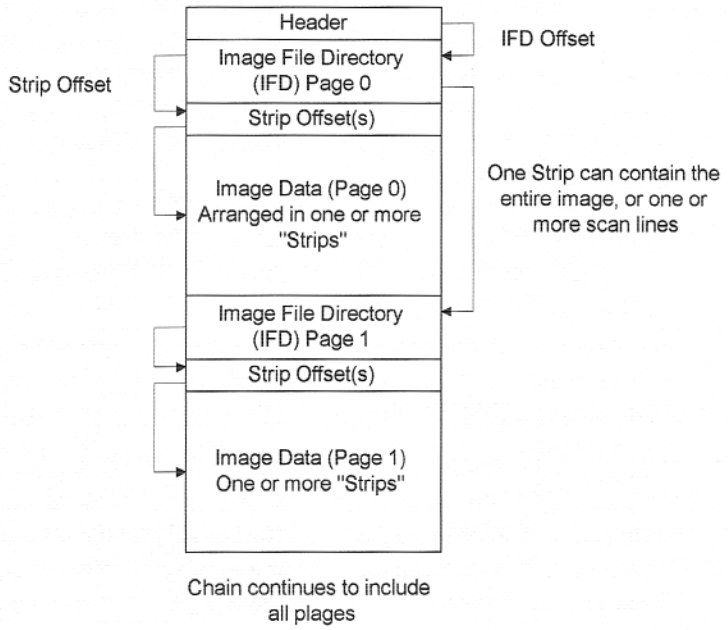

Figure 4.7 Basic TIFF image format [5].

Images in both the T.37 simple and full operating modes are sent to the remote gateway as MIME-encoded email messages containing the

image as the attachment. T.37 full mode adds the requirement to confirm that the fax was received properly, and negotiation of the capabilities of the fax terminals. Any Internet mail transport protocol can be used in full mode to carry the image data.

Delivery confirmations are returned to the sender as MIME-encoded Delivery Status Notifications (DSNs) for gateways, as described in RFC 1894. Senders and receivers require the use of Message Disposition Notifications (MDN), defined in RFC 2298.

4.4.4 Real-time Internet Fax-T.38

Store-and-forward fax is a rather primitive approach to support non-real-time facsimile, with some fairly obvious limitations in functionality. Even so, the business side of the argument points to substantial interest for this type of service and it is experiencing continuing growth in the industry. Several service providers already offer Internet fax, sometimes in package deals with other services.

The alternative to non-real-time fax over IP networks is Recommendation T.38. The T.38 protocol gives the "look and feel" of real-time facsimile by emulating the handshake activities of the T.30 protocol on the packet network side. Its basic idea is fax demodulation by a T.38 gateway at the source, packetization of all relevant handshake exchanges, sending of the IP packets across the network, and remodulation of the analog line by the receiving T.38 gateway from the information carried in the packet data. All this is accomplished with the simplicity of just two types of messages (packets), T30_INDICATOR (indicator packets) and T_DATA (data packets), which are part of the Internet Fax Protocol (IFP) recommendation of the ITU.

Indicator packets carry information to the far end about the presence of a CNG/CED tone, modem modulation training, or preamble flags each time the line is turned around. Data packets carry the Phase C data and HDLC control frames. Packets may carry one or more HDLC control frames, or a complete image. Group 3 facsimile equipments (the legacy devices) attach to T.38-compliant gateways and execute the T.30 protocol in real-time, without modifications. Adherence to the timing restrictions of T.30 is thus critical during the handshaking procedures between the terminals.

The IFP allows either TCP or UDP to be used as the transport protocol. When TCP is used, the IP payload is simply the TCP header and the concatenated indicator or data packet.

When UDP is used, the payload consists of a new layer header (UDPTL), followed by the concatenated indicator or data packet. The UDPTL header is a packet sequence number to account for packets arriving via different paths and out of order. The UDPTL payload also contains an optional Forward Error Correction (FEC) field to recover from bit errors. Also optionally, redundant messages can be included in a single UDPTL packet.

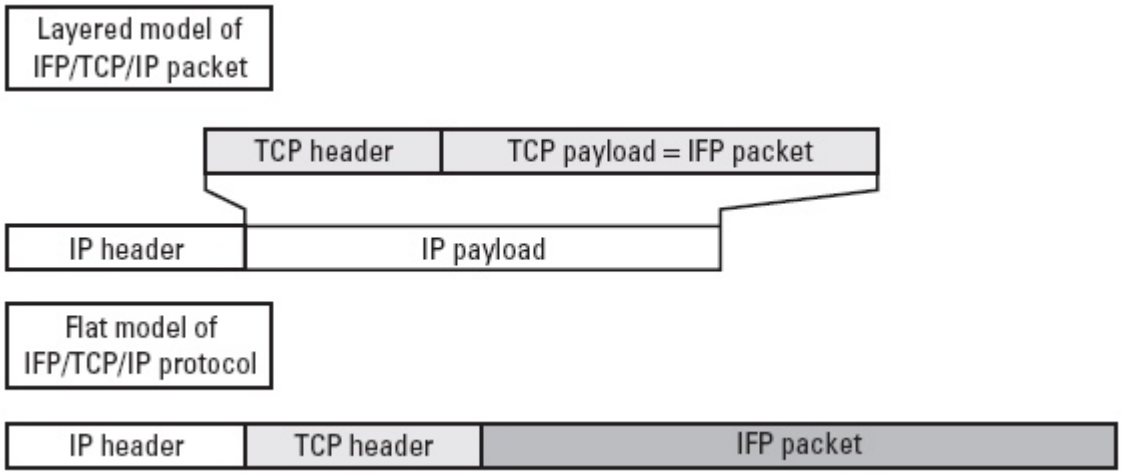

Figure 4.8 High-Level IFP/TCP Packet Structure [12].

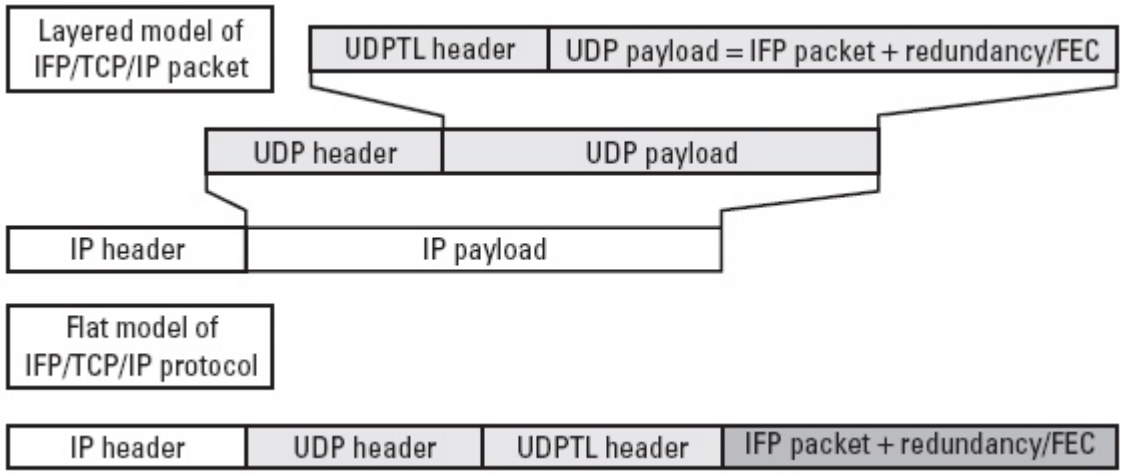

Figure 4.9 High-Level UDPTL/IP Packet Structure [12].

A simplified block diagram of the message flow under T.38 is shown in Figure 4.10. For exact details of the T.38 message types and exchanges, ITU-T Recommendation T.38 should be studied.

Flag sequences are required for every line turnaround and are transmitted as indicator (T30_INDICATOR) packets. Training is sent as an indicator packet, with the V-type modulation used by the sending terminal. This is used to adjust the speed of the terminal, for instance, to switch from

sending image data with V.17 modulation to V.21 modulation for control sequences.

The same type of training is generated by the receiving gateway at the far end toward the sending fax terminal. The modulation training sequences have timing requirements which must be carefully adhered to in an end-to-end communication, in order for the presence of the IP network between the gateways to be completely transparent to the fax application.

Finally, the Training Check Field (TCF) can be used in one of two ways in T.38-compliant networks. For connection-oriented, TCP-based implementations, the TCF is generated by the far end gateway toward the receiving fax terminal. When UDP is used, the TCF needs to be sent across the packet network. The difference is in the decision logic of the speed selection.

The call flow of Figure 4.10 shows the T.30 protocol being executed between the calling terminal and its local T.38 gateway. All signal types and their timing restrictions must be supported at that interface, regardless of what timing constraints may be challenging the gateway on the packet network side. The fact is, in the general case, communicating T.38 gateways can be located across several network segments, between one or more service providers. All the potential issues facing voice telephony, such as QoS at the packet level, can happen during a fax call.

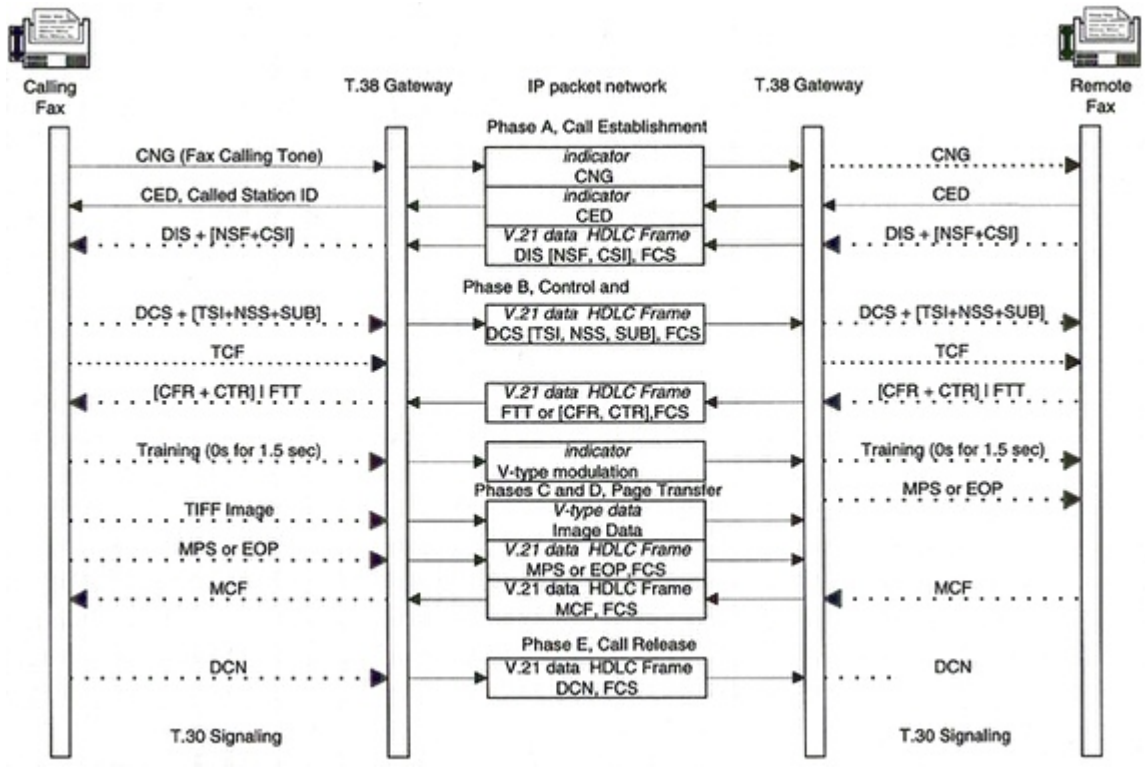

Figure 4.10 T.38 High Level Message Flow [5].

4.5 IP Telephony Quality of Service

4.5.1 Integrated Services & RSVP

There are various approaches to provide QoS in IP networks. QoS can be achieved by managing router queues and by routing traffic around congested parts of the network. Two key QoS concepts are the IntServ and DiffServ. The IntServ concept is to reserve resources for each flow through the network. RSVP was originally designed to be the reservation protocol. When an application requests a specific QoS for its data stream, RSVP can be used to deliver the request to each router along the path and to maintain router state to provide the requested service. RSVP transmits two types of Flow Specs conforming to IntServ rules. The traffic specification (Tspec) describes the flow, and the service request specification (Rspec) describes the service requested under the assumption that the flow adheres to the Tspec. Current implementations of IntServ allow a choice of Guaranteed Service or Controlled-Load Service.

There are several reasons for not using IntServ with RSVP for IP telephony. Although IntServ with RSVP would work on a private network for small amounts of traffic, the large number of voice calls that IP telephony service providers carry on their networks would stress an IntServ RSVP system. First, the bandwidth required for voice itself is small, and the RSVP control traffic would be a significant part of the overall traffic. Second, RSVP router code was not designed to support many thousands of simultaneous connections per router.

It should be noted, however, that RSVP is a signaling protocol, and it has been proposed for use in contexts other than IntServ. For example,

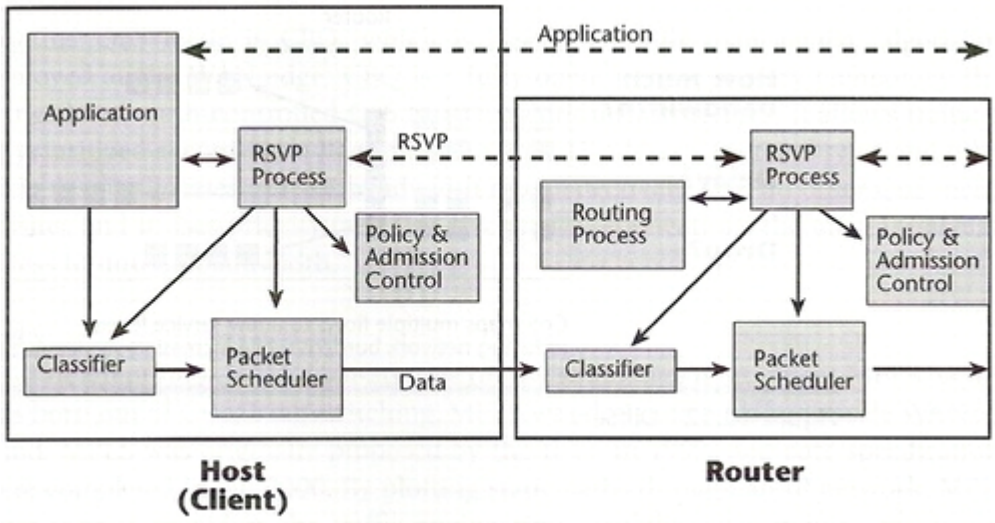

Figure 4.11 RSVP in hosts and routers [4].

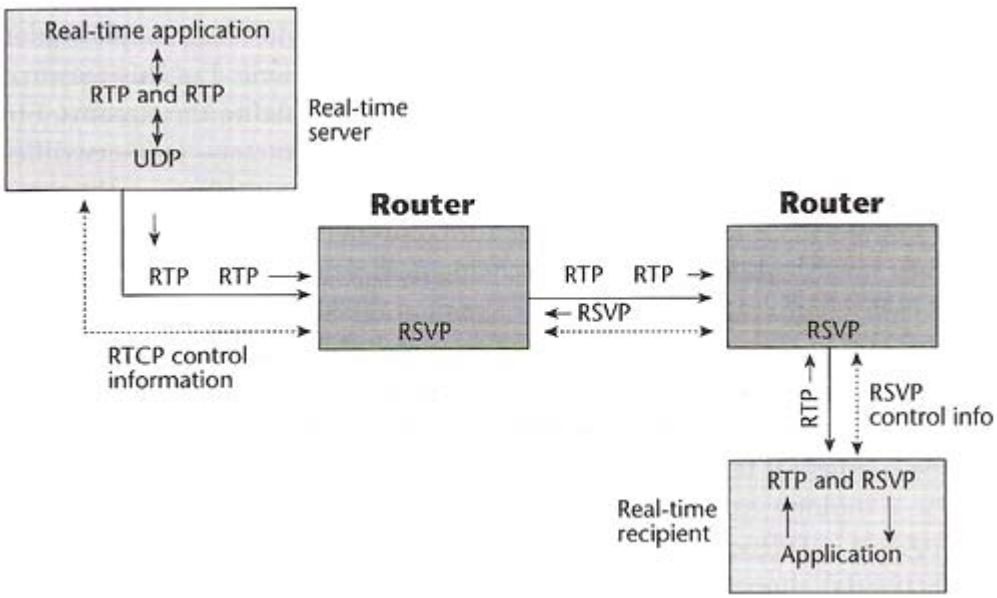

Figure 4.12 RSVP and related protocols [4].

RSVP-TE is a constraint-based routing protocol for establishing LSPs (Labeled Switched Path) with associated bandwidth and specified paths in an MPLS (Multi-Protocol Label Switching) network. RSVP has also been proposed as the call admission control mechanism for VoIP in differentiated services networks.

4.5.2 Differentiated Services

Since IntServ with RSVP does not scale well to support many thousands of simultaneous connections, the IETF has developed a simpler framework and architecture to support DiffServ. The architecture achieves scalability by aggregating traffic into classifications that are conveyed by means of IP-layer packet marking using the DS field in IPv4 or IPv6 headers. Sophisticated classification, marking, policing, and shaping operations need only be implemented at network boundaries. Service provisioning policies allocate network resources to traffic streams by marking and conditioning packets as they enter a differentiated services-capable network, in which the packets receive a particular PHB (Per Hop Behavior) based on the value of the DS field.

The primary goal of differentiated services is to allow different levels of service to be provided for traffic streams on a common network infrastructure. A variety of resource management techniques may be used to achieve this, but the end result will be that some packets will receive different (e.g., better) service than others. This will, for example, allow service providers to offer a real-time service giving priority to the use of bandwidth

and router queues, up to the configured amount of capacity allocated to real-time traffic.

Despite the term “differentiated services,” the IETF DiffServ working group undertook to define standards that have more generality than specific services. The reason is that if the IETF were to define new standard services, everyone would have to agree on what constitutes a useful service and every router would have to implement the mechanisms to support it. To deploy that new service, you would have to upgrade the entire Internet. Since a router has only a few functions, it makes more sense to standardize forwarding behavior (“send this packet first” or “drop this packet last”). So the DiffServ working group first defined PHBs, which could be combined with rules to create services.

An important requirement is scalability, since the IETF intended differentiated services to be deployed in very large networks. To achieve scalability, the DiffServ architecture prescribes treatment for aggregated traffic rather than microflows and forces much of the complexity out of the core of the network into the edge devices, which process lower volumes of traffic and lesser numbers of flows.

The DiffServ architecture is based on a simple model where packets entering a network are classified and possibly conditioned at the boundaries of the network, and then assigned to different behavior aggregates. Each behavior is identified by a single DS codepoint. Within the core of the network, packets are forwarded according to the PHB associated with the DS codepoint.

The appeal of DiffServ is that it is relatively simple (compared to IntServ), yet provides applications like VoIP some improvement in performance compared to “best-effort” IP networks. However, DiffServ relies on ample network capacity and makes use of standard routing protocols that make no attempt to use the network efficiently. DiffServ has no topology-aware admission control mechanism. The IETF DiffServWorking Group has not recommended a mechanism for rejecting additional VoIP calls if accepting them would degrade the quality of calls in progress.

4.5.3 MPLS-Based QoS

For several decades, traffic engineering and automated rerouting of telephone traffic have increased the efficiency and reliability of the PSTN. Frame relay and ATM also offer source routing capabilities that enable traffic engineering. However, IP networks have relied on destination-based routing protocols that send all the packets over the shortest path, without regard to the utilization of the links comprising that path. In some cases, links can be

congested by traffic that could be carried on other paths comprised of underutilized links. It is possible to design an IP network to run on top of a frame relay or ATM ("Layer 2") network, providing some traffic engineering features, but this approach adds cost and operational complexity.

MPLS offers IP networks the capability to provide traffic engineering as well as a differentiated services approach to voice quality. MPLS separates routing from forwarding, using label swapping as the forwarding mechanism. The physical manifestation of MPLS is the Label Switching Router (LSR). LSRs perform the routing function in advance by creating LSPs connecting edge routers. The edge router (an LSR) attaches short (four-byte) labels to packets. Each LSR along the LSP swaps the label and passes it along to the next LSR. The last LSR on the LSP removes the label and treats the packet as a normal IP packet.

Differentiated services can be combined with MPLS to map DiffServ Behavior Aggregates onto LSPs. QoS policies can be designated for particular paths. More specifically, the EXP field of the MPLS label can be set so that each label switch/router in the path knows to give the voice packets highest priority, up to the configured maximum bandwidth for voice on a particular link. When the high-priority bandwidth is not needed for voice, it can be used for lower priority classes of traffic.

DiffServ and MPLS DiffServ are implemented independently of the routing computation. MPLS-TE computes routes for aggregates across all classes and performs admission control over the entire LSP bandwidth. MPLS-TE and MPLS DiffServ can be used at the same time. Alternatively, DiffServ can be combined with traffic engineering to establish separate tunnels for different classes. DS-TE makes MPLS-TE aware of DiffServ, so that one can establish separate LSPs for different classes, taking into account the bandwidth available to each class. So, for example, a separate LSP could be established for voice, and that LSP could be given higher priority than other LSPs, but the amount of voice traffic on a link could be limited to a certain percentage of the total link bandwidth. This capability is currently being standardized by the IETF Traffic Engineering Working Group.

Voice DS-TE tunnels can be based on a delay metric or a bandwidth metric. Combining DS-TE with DiffServ over MPLS allows QoS for VoIP with the capability of fast reroute if a link or node failure occurs. DiffServ can guarantee that a specified amount of voice bandwidth is available on each link in a network. DS-TE routing and admission control can create a guaranteed bandwidth tunnel that has the required bandwidth in the highest priority queue on every link. Service conditioning at the edge can ensure that the aggregate VoIP traffic directed onto the guaranteed

bandwidth tunnel is less than the capacity of the tunnel. This allows a tight SLA with admission control without overprovisioning the network.

A VoIP network designer can choose DiffServ, MPLS-TE plus DiffServ, or DS-TE according to the economics of the situation. If VoIP is to be a small portion of the total traffic, DiffServ or MPLS-TE plus DiffServ may be sufficient. DS-TE promises more efficient use of an IP network carrying a large proportion of VoIP traffic, with perhaps more operational complexity.

4.6 IP Telephony Trends and Economics

Although VoIP calling is used for billions of billed minutes each year, it still represents a very small percentage of the market—less than 5% overall. According to Telegeography (www.telegeography.com), 40% of VoIP traffic originates in Asia and terminates in North America or Europe; 30% travels between North America and Latin America; one-third of U.S. international VoIP traffic goes to Mexico, with future volume increases predicted for calling to China, Brazil, and India, and the rest moves among the U.S., Asia Pacific, and Western European regions. It is important to closely examine who will be using this and what carriers or operators will be deploying these technologies. Probe Research (www.proberesearch.com) believes that up to 2002, 6% of all voice lines were VoIP. This is still rather minor, given the fact that some have been saying that VoIP would have replaced circuit-switched calling up to the year 2002. Piper Jaffray (www.piperjaffray.com) reports that minutes of communication services traveling over IP telephony networks will grow from an anticipated 70 billion minutes and 6% of all the PSTN traffic in the year 2003 to over a trillion minutes by the year 2006. In the United States alone, the PSTN is handling some 3.6 trillion minutes of traffic monthly.

Although VoIP has a very important place in telecommunications, it's important to realize that it is not yet taking over the traditional circuit-switched approach to accommodating voice telephony. The exciting future of VoIP lies in advanced and interesting new applications, an environment where voice is just one of the information streams comprising a rich media application. Many expect that sales of VoIP equipment will grow rapidly in the coming years. Part of the reason for this growth is that the network-specific cost for VoIP on dedicated networks is quite a bit lower than the cost of calls on circuit-switched networks—about US 1.1 cents per minute as compared with US 1.7 cents per minute. Using VoIP to carry telephony traffic greatly reduces the cost of the infrastructure for the provider, but at the expense of possibly not being able to maintain QoS. Potential savings are even greater if VoIP is implemented as an adjunct to data network.

Another factor encouraging customers to examine VoIP is the use of shared networks. Because IP emphasizes logical rather than physical connections, it's easier for multiple carriers to coexist on a single network. This encourages cooperative sharing of interconnected networks, structured as anything from sale of wholesale circuits to real-time capacity exchanges. Also, VoIP can reduce the barriers to entry in this competitive data communications world. New companies can enter the market without the huge fixed costs that are normally associated with the traditional circuit-switched network models. Furthermore, because IP telephony will enable new forms of competition, there will be pressure to better align government-controlled prices with underlying service costs. International VoIP services are already priced well below the official rates and some of VoIP's appeal is that it eliminates the access charges interexchange carriers normally have to pay to interconnect to the local exchange carrier. In the United States, these charges range from US 2 cents to US 5 cents per minute.

4.6.1 Fax over IP

Most companies are unaware of just how much time and money is lost by traditional faxing. The average Fortune 500 company spends $40 million per year on phone service, 40 percent of which goes to faxing, according to a Gallop/Pitney Bowes survey. By switching to emerging fax over IP, companies can save as much as 70 percent on their long distance phone bill, while gaining some important new features, says Maury Kauffman, managing partner of The Kauffman Group, a fax technology consulting firm in Vorhees, NJ. And that's not all. When calculating the full benefits of fax over IP, companies must factor in the cost of fax machines (which can run as high as $2,000 to $3,000 per machine); the cost of operating and maintaining those machines; and wasted labor each time an employee walks to the fax machine, waits for the fax to go through and returns to work. For larger companies using fax servers, the cost can be enormous. Companies can eliminate all of this by switching to fax over IP.

Fax over IP can also help companies cut down on the cost of other delivery methods, such as mail, overnight delivery and courier services. Fax over IP also solves the problem of mobile professionals who cannot receive faxes when they are out of the office. Users simply create a document in a program like Microsoft Word, click on file/print and then choose the installed fax-configured printer. After entering the appropriate fax number, users can send the document right from their desktops. The technologies allow users to send and receive fax documents via desktop computers. According to PSINet, users can save as much as 25 to 50 percent on long-

distance charges and reduce spending by eliminating the need for additional phone lines, modems, fax equipment and maintenance.

The fax over IP market shows no signs of slowing as more and more companies turn to the Net to transmit documents. According to research by The Gartner Group, fax over IP reached 5.6 billion pages carried in 2001, up from 44 million pages in 1997. IDC estimates that fax transmissions represented an $83 billion dollar market in 1998 and grew to $90 billion in 2000. There is a wide range of numbers describing the current size of the IP telephony market and the growth of the market over the next three to five years. While the specific projections vary, even the most conservative analysts are predicting phenomenal growth. The numbers are summarized below.

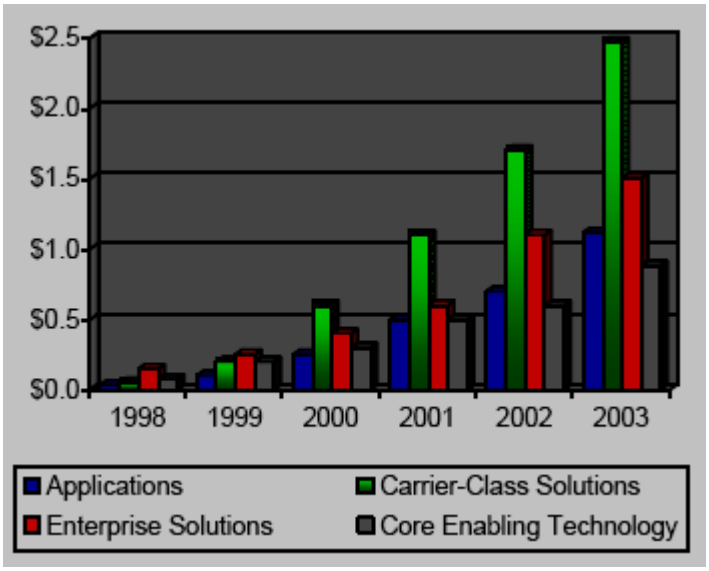

Figure 4.13 Piper-Jaffray, IP Telephony, Driving the Open Communications Revolution [13].

4.7 For Further Study

For further information regarding the concepts discussed in this chapter please refer to following resources:

4.7.1 IP Telephony Fundamentals

[10]

Media Transport in Packet Networks

[5] PP: 91-141
[14] PP: 130-135

Voice Quality

[1]
[5] PP: 223-283
[15]
[16]

4.7.2 Call Signaling

[11]

H.323

[5] PP: 36-69
[17]
[18]

SIP

[1]
[5] PP: 69-91

MGCP-Megaco

[1]
[5] PP: 18-36
[19]
[20]
[21]

4.7.3 Fax over IP

[5] PP: 283-298
[2] PP: 14-22
[22] PP: 3-13
[12]

ITU-T Recommendations

[23] ITU-T Recommendations: T.4
[24] ITU-T Recommendations: T.30
[25] ITU-T Recommendations: T.37
[26] ITU-T Recommendations: T.38 2002
[27] ITU-T Recommendations: T.38 2002-Amendment 1 (07-03)
[28] ITU-T Recommendations: T.38 2002-Amendment 3 (01-04)
[29] ITU-T Recommendations: T.38 2002-Corrigendum 1 (07-03)
[30]
T.37: [31]

Fax Gateways

[32]
[33]

4.7.4 IP Telephony Quality of Service

[1]
[3] PP: 116-120
[4] PP: 316-329
[7] PP: 294-305

RSVP

[5] PP: 141-150

4.7.5 IP Telephony Trends and Economics

[4] PP: 348-355
[34]
[13]

Chapter 5

SIP: Session Initiation Protocol

5.1 Introduction

In this chapter, the Session Initiation Protocol (SIP) is treated thoroughly. In the next section, an introduction of the protocol is given. In Section 3, SIP user agents, gateways and the 3 types of servers are discussed. A brief introduction of SIP request and response messages and headers is given in Section 4. Session Description Protocol (SDP), a companion protocol to SIP, is treated next. An almost complete introduction to SIP programming is given in Section 6. SIP and T.38 interactions are explained briefly in Section 7. And we conclude the chapter with introduction of the materials for further studying.

5.2 Introducing SIP

5.2.1 A Brief History of SIP

SIP was originally developed by the IETF Multi-Party Multimedia Session Control Working Group, known as MMUSIC. Version 1.0 was submitted as an Internet-Draft in 1997. Significant changes were made to the protocol and resulted in a second version, version 2.0, which was

submitted as an Internet-Draft in 1998. The protocol achieved Proposed Standard status in March 1999 and was published as RFC 2543 in April 1999. In September 1999, the SIP working group was established by the IETF to meet the growing interest in the protocol. An Internet-Draft containing bug fixes and clarifications to SIP was submitted in July 2000, referred to as RFC 2543 "bis". In June 2002, RFC 3261 was published which defined the latest SIP specification and made RFC 2543 obsolete. To advance from Proposed Standard to Draft Standard, a protocol must have multiple independent interworking implementations and limited operational experience. To this end, forums of interoperability tests have been organized by the SIP working group. The final level, Standard, is achieved after operational success has been demonstrated.

SIP incorporates elements of two widely used Internet protocols: HTTP (Hyper Text Transport Protocol) used for web browsing and SMTP (Simple Mail Transport Protocol) used for e-mail. From HTTP, SIP borrowed a client-server design and the use of uniform resource locators (URLs). From SMTP, SIP borrowed a text-encoding scheme and header style. For example, SIP reuses SMTP headers such as To, From, Date and Subject.

In keeping with its philosophy of "one problem, one protocol", the IETF designed SIP to be a pure signaling protocol. SIP uses other IETF protocols for transport, media transport, and media description.

5.2.2 Places Where SIP is Discussed

The main forum of SIP standardization is in the Internet Engineering Task Force (IETF), which is the primary standards body for Internet protocols. The IETF has set up the following three working groups to work on the protocol and its application:

- The SIP working group covers enhancements to the core protocol.
- The SIPPING working group covers applications of SIP.
- The SIMPLE working group covers Instant Messaging and Presence applications of SIP.

The distinction between these groups is that the SIPPING and SIMPLE working groups discuss applications of SIP and decide how SIP should be used in each of them. If they determine that the requirements of a particular application cannot be handled by the core protocol, then these requirements are passed to the SIP working group for a solution. This enables the SIP working group to maintain control over extensions to the protocol, while limiting the scope of its discussions.

Other IETF working groups whose areas touch on SIP include the following:

• IPTEL (Internet routing of telephone calls)

• MMUSIC (responsible for Session Descriptor Protocol (SDP), which SIP uses to describe its media sessions)

• MIDCOM (Middlebox communication – firewall and NAT traversal)

• SPIRITS (PSTN – Internet telephony interoperation)

• ENUM (Internet use of traditional PSTN phone numbers)

Several industry groups are also discussing how to standardize the use of SIP in their environment. These include:

• Packetcable (www.packetcable.com), who are using SIP for telephony over cable

• 3GPP (www.3gpp.org), who have adopted SIP for 3G mobile

• Multi-service Switching Forum (MSF) (www.msforum.org), which has defined SIP-T conformance levels and is now working to ensure that SIP can be deployed in large scale PSTN networks.

• ETSI TIPHON (Telecommunications and Internet Protocol Harmonization Over Networks) (www.etsi.org), who are working to ensure that SIP is suitable for deployable telephony applications.

There is a continual conflict between the requirements of the traditional telephone providers, who need to provide an end-to-end billable solution that meets their regulatory requirements, and the less controlled environment of the Internet. This is resulting in concern over the interoperability of the different flavors of SIP, including 3GPP SIP, PacketCable SIP, and IETF SIP, and discussions are ongoing to ensure that they all work together.

There is a separate initiative to standardize the programming interfaces to SIP and other telephony protocols. This work covers the following interfaces:

• JAIN (java.sun.com/products/jain) – Java APIs to SIP and other Next Generation telecom protocols.

• Parlay (www.parlay.org) – High-level, protocol independent APIs that allow the development of telecommunications applications that are independent of the underlying network.

• Call Processing Language (CPL) - XML-based language that can be used to describe and control Internet telephony services (draft-ietf-iptel-cpl-08).

• Common Gateway Interface (CGI) - HTTP CGI compatible extensions to providing SIP services on an SIP server (RFC 3050)

These standardized interfaces help the development of SIP applications that are not tied to a specific implementation of the protocol. This makes the resulting application more portable and reduces the developer's dependence

on one supplier, but they can add a processing overhead that may reduce the overall efficiency of the system. The protocol independent interfaces also limit the ability to exploit the advantages of a particular protocol.

5.2.3 Message Transport

SIP is an application layer protocol in the Internet multimedia protocol stack. It can use either TCP or UDP for transport layer, both of which use IP for the Internet layer. How an SIP message is transported using these two protocols will be described in the following sections.

UDP Transport

When using UDP, each SIP request or response message is usually carried by a single UDP datagram or packet. Most SIP messages easily fit in a single datagram. For a particularly large message body, there is a compact form of SIP that saves space in representing some headers with a single character. Figure 5.1 shows an SIP BYE request exchange during an established SIP session using UDP.

The source port is chosen from a pool of available port numbers (above 49172), or sometimes the default SIP port of 5060 is used. The lack of hand-shaking or acknowledgment in UDP transport means that a datagram could be lost and an SIP message along with it. The checksum, however, enables UDP to discard corrupt datagrams, allowing SIP to assume that a

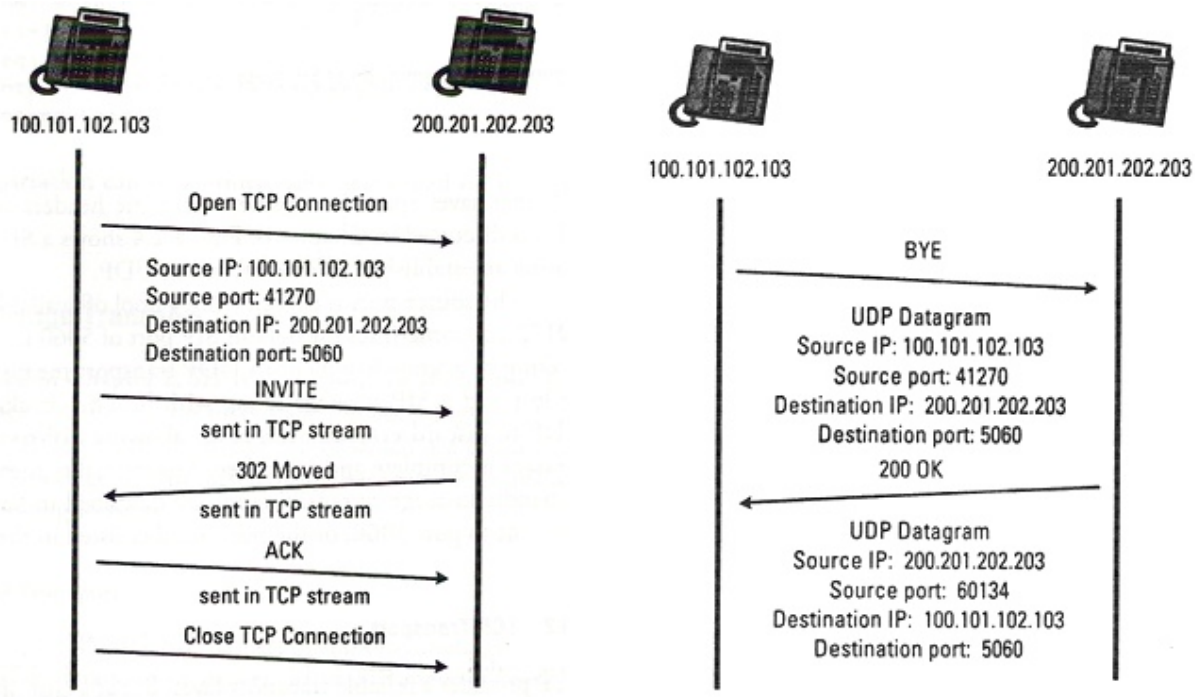

Figure 5.1 Transmission of SIP messages using TCP and UDP [14].

received message is complete and error-free. The reply is also sent to port 5060, or the port number listed in the top via header.

TCP Transport

TCP provides a reliable transport layer, but at a cost of complexity and transmission delay over the network. The use of TCP for transport in an SIP message exchange is also shown in Figure 5.1. This example shows an INVITE sent by a user agent at 100.101.102.103 to a type of SIP server called a "redirect server" at 200.201.202.203. An SIP redirect server does not forward INVITE requests like a proxy, but looks up the destination address and instead returns that address in a redirection class (3xx) response. The 302 Moved Temporarily response is acknowledged by the user agent with an ACK message. Not shown in this figure is the next step, where the INVITE would be re-sent to the address returned by the redirect server. The details of the message exchanges are not important for the moment. These will be clearer in the coming sections.

As in the UDP example, the "well-known" SIP port number of 5060 is chosen for the destination port, and the source port is chosen from an available pool of port numbers. Before the message can be sent, however, the TCP connection must be opened between the two end-points. This transport layer datagram exchange is shown in Figure 5.1 as a single arrow, but it is actually a three-way handshake between the end-points. Once the connection is established, the messages are sent in the stream. The Content-Length header is critical when TCP is used to transport SIP, since it is used to find the end of one message and the start of the next.

The 302 Moved Temporarily response is sent in the stream in the opposite direction. The acknowledgment ACK also is sent in the TCP stream. Because this concludes the SIP session, the connection is then closed. The connection must stay up until the call is established. After that, it can be safely closed without ending the media session, The TCP connection would then need to be reopened to terminate the session with a BYE request.

5.3 SIP Clients and Servers

5.3.1 SIP User Agents

An SIP-enabled end-device is called an SIP user agent (UA). The main purpose of SIP is to enable sessions to be established between user agents. As the name implies, a user agent takes direction or input from a user and acts as an agent on their behalf to set up and tear down media

sessions with other user agents. In most cases, the user will be a human, but the user could be another protocol, as in the case of a gateway described in the next section. A user agent must be capable of establishing a media session with another user agent. Since SIP may be used with any transport protocol, there is no requirement that a UA must support either TCP or UDP for message transport. The standard states, however, that a UA should support both TCP and UDP.

A UA must maintain state on calls that it initiates or participates in. A minimum call state set includes the local and remote URL, Call-ID**,** local and remote CSeq headers along with any state information necessary for the media. This information is used to store the call leg and for reliability. The remote CSeq storage is necessary to distinguish between a re-INVITE and a retransmission. A re-INVITE is used to change the session parameters of an existing or pending call. It uses the same Call-ID, but the CSeq is incremented because it is a new request. A retransmitted INVITE will contain the same Call-ID and CSeq as a previous INVITE. Even after a call has been terminated, call state must be maintained by a user agent for at least 32 seconds in case of lost messages in the call tear-down.

A minimum user agent implementation includes support of the INVITE and ACK methods. Although not required to understand every response code defined, a minimal implementation must be able to interpret any unknown response based on the class (first digit of the number) of the response.

The types of user agents defined in the standard include minimum, basic, redirection, firewall friendly, negotiation, and authentication. These are detailed in Table 5.1. A user agent server responds to an unsupported request with a 501 Not Implemented response.

Most SIP devices support much more than the minimum implementation, and often include support for authentication. An SIP user agent contains both a client application and a server application. The two parts are user agent clients (UAC) and user agent servers (UAS). The UAC initiates requests while the UAS generates responses. During a session, a user agent will usually operate as both a UAC and a UAS.

An SIP user agent must also support SDP for media description. Other types of media descriptions can be used in bodies, but SDP support is mandatory.

Table 5.1 User Agent Types [14].

User agent type	Supports
Minimum	INVITE, ACK, SDP, response classes
Basic	Minimum plus BYE
Redirection	Basic plus *Contact* header
Firewall friendly	Redirection plus *Route*, *Record-Route*, and default proxy server
Negotiation	Firewall plus OPTIONS, *Warning*, *380* response
Authentication	Negotiation plus *401* response, *WWW-Authenticate*, and *Authorization* headers

5.3.2 SIP Gateways

An SIP gateway is an application that interfaces an SIP network to a network utilizing another signaling protocol. In terms of the SIP protocol, a gateway is just a special type of user agent, where the user agent acts on behalf of another protocol rather than a human. A gateway terminates the SIP signaling path and can also terminate the media path, although this is not always the case. For example, an SIP to H.323 gateway terminates the SIP signaling path and converts the signaling to H.323, but the SIP user agent and H.323 terminal can exchange RTP media information directly with each other without going through the gateway.

A Public Switched Telephone Network (PSTN) gateway terminates both the signaling and media paths. SIP can be translated into common PSTN protocols such as Integrated Services Digital Network (ISDN), ISDN User Part (ISUP), and other Circuit Associated Signaling (CAS) protocols. A PSTN gateway also converts the RTP media stream in the IP network into a standard telephony trunk or line. The conversion of signaling and media paths allows calling to and from the PSTN using SIP.

Figure 5.2 shows an SIP network connected via gateways with the PSTN and a H.323 network. In the figure, the SIP network, PSTN network, and H.323 networks are shown as clouds, which obscure the underlying details. Shown connecting to the SIP cloud are SIP IP telephones, SIP-enabled PCs, and corporate SIP gateways with attached telephones. The clouds are connected by gateways. Shown attached to the H.323 network are H.323 terminals and H.323-enabled PCs. The PSTN cloud connects to ordinary analog telephones, digital ISDN telephones, and corporate private branch exchanges (PBXs). PBXs connect to the PSTN using shared trunks

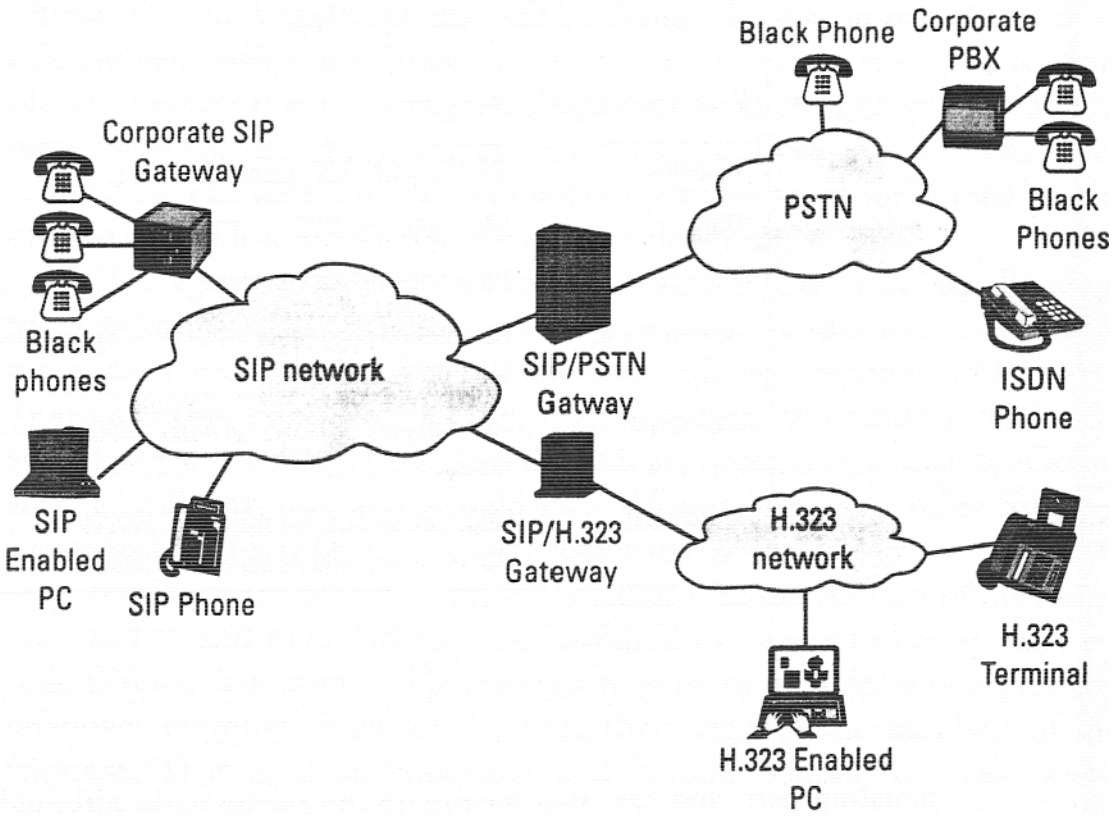

Figure 5.2 SIP Network with Gateways [14].

and provide line interfaces for either analog or digital telephones.

Gateways are sometimes decomposed into a media gateway (MG) and a media gateway controller (MGC). An MGC is sometimes called a call agent because it manages call control protocols (signaling), while the MG manages the media connection. This decomposition is transparent to SIP, and the protocols used to decompose a gateway are not described here.

Another difference between a user agent and a gateway is the number of users supported. While a user agent typically supports a single user, a gateway can support hundreds or thousands of users. A PSTN gateway could support a large corporate customer, or an entire geographic area. As a result, a gateway does not REGISTER every user it supports in the same way that a user agent might. Instead, a non-SIP protocol can be used to inform proxies about gateways and assist in routing. One protocol that has been proposed for this is the Telecommunications Routing over IP (TRIP) protocol.

5.3.3 SIP Servers

SIP servers are applications that accept SIP requests and respond to them. An SIP server should not be confused with a user agent server or the

client-server nature of the protocol, which describe operation in terms of clients (originators of requests) and servers (originators of responses to requests). An SIP server is a different type of entity. The types of SIP servers discussed in this section are logical entities. Actual SIP server implementations may contain a number of server types, or may operate as a different type of server under different conditions. Because servers provide services and features to user agents, they must support both TCP and UDP for transport. Figure 5.3 shows the interaction of user agents, servers, and a location service. Note that the protocol used between a server and the location service or database is not in general SIP and is not discussed here.

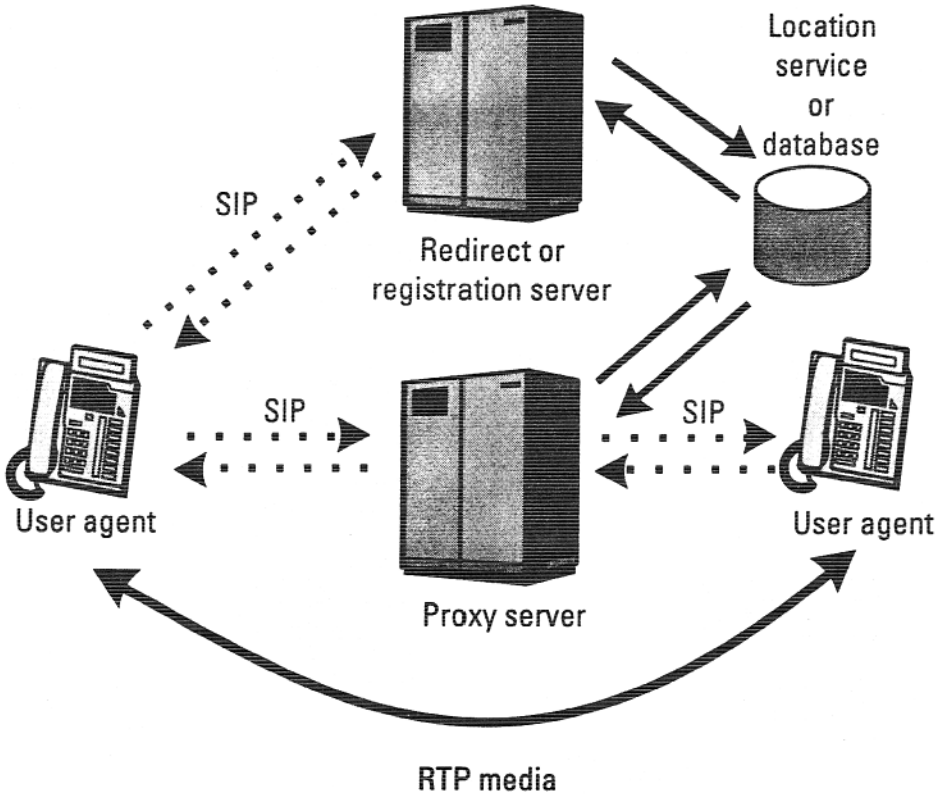

Figure 5.3 SIP user agent, server, and location service interaction [14].

Proxy Servers

An SIP proxy server that receives an SIP request from a user agent acts on behalf of the user agent in forwarding or responding to the request. A proxy server typically has access to a database or a location service to aid it in processing the request (determining the next hop). The interface between the proxy and the location service is not defined by the SIP protocol. A proxy can use any number of types of databases to aid in processing a request.

Databases could contain SIP registrations, or any other type of information about where a user is located.

A proxy server is different from a user agent or gateway in two key ways:

1. A proxy server does not issue a request; it only responds to requests from a user agent. (A CANCEL request is the only exception to this rule.)
2. A proxy server has no media capabilities.

A proxy server can be either stateless or stateful. A stateless proxy server processes each SIP request or response based solely on the message contents. Once the message has been parsed, processed, and forwarded or responded to, no information about the message is stored—no call leg information is stored. A stateless proxy never retransmits a message, and does not use any SIP timers. A stateless proxy has no memory of any requests or responses it has sent or received. A stateless proxy is still capable of detecting message looping since SIP uses a stateless method to implement loop detection using `via` headers.

A stateful proxy server keeps track of requests and responses received in the past and uses that information in processing future requests and responses. For example, a stateful proxy server starts a timer when a request is forwarded. If no response to the request is received within the timer period, the proxy will retransmit the request, relieving the user agent of this task. Also, a stateful proxy can require user agent authentication.

Redirect Servers

A redirect server is introduced as a type of SIP server that responds to, but does not forward requests. Like a proxy server, a redirect server uses a database or location service to look up a user. The location information, however, is sent back to the caller in a redirection class response, which concludes the transaction.

Registration Servers

A registration server accepts SIP REGISTER requests; all other requests receive a 501 Not Implemented response. The contact information from the request is then made available to other SIP servers within the same administrative domain, such as proxies and redirect servers. In a registration request, the To header contains the name of the resource being registered, and the Contact headers contain the alternative

addresses or aliases.

Registration servers usually require the registering user agent to be authenticated so that incoming calls can not be hijacked by an unauthorized user. This could be accomplished by an unauthorized user registering someone else's SIP URL to point to their own phone. Incoming calls to that URL would then ring the wrong phone. Depending on the headers present, a REGISTER request can be used by a user agent to retrieve a list of current registrations, clear all registrations, or add a registration URL to the list.

5.4 SIP Request and Response Messages and Headers

5.4.1 SIP Request Messages

This section explains the types of SIP requests called methods. Six are described in the SIP specification document. Two more methods are work items of the SIP working group. Other proposed methods are still in the early stages of development, or have not yet achieved working group consensus.

SIP requests or methods are considered "verbs" in the protocol, since they request a specific action to be taken by another user agent or proxy server.

The INVITE, REGISTER, BYE, ACK, CANCEL, and OPTIONS methods are the original six methods in version 2.0 of SIP. The INFO and PRACK methods are the subsequent additions.

A proxy does not need to understand a request method in order to forward the request. A proxy treats an unknown method as if it were an OPTIONS; that is, it forwards the request to the destination if it can. This allows new features and methods useful for user agents to be introduced without requiring support from proxies that may be in the middle. A user agent receiving a method it does not support replies with a 501 Not Implemented response.

5.4.2 SIP Response Messages

This section covers the types of SIP response messages. An SIP response is a message generated by a UAS or an SIP server to reply to a request generated by a UAC. A response may contain additional headers containing information needed by the UAC. Or, it may be a simple acknowledgement to prevent retransmissions of the request by the UAC. Many responses direct the UAC to take specific additional steps.

There are six classes of SIP responses. The first five classes were borrowed from HTTP; the sixth was created for SIP. The classes are shown in Table 5.2.

If a particular SIP response code is not understood by a UAC, it must be interpreted by the class of the response. For example, an unknown 599 Server Unplugged response must be interpreted by a user agent as a 500 Server Failure response. The reason phrase is for human consumption only—the SIP protocol uses only the response code in determining behavior. The reason phrases listed here are the suggested ones from the standard document. They can be used to convey more information, especially in failure class responses—the phrase is likely to be displayed to the user. Response codes in the range x00-x79 were borrowed from HTTP, perhaps with a slightly different reason phrase. New response codes created for SIP begin at x80 to avoid conflicts with future HTTP response codes.

Table 5.2 SIP Response Classes [14].

Class	Description	Action
1xx	Informational: Indicates status of call prior to completion	If first informational response, the client should switch from timer T1 to timer T2 for retransmission
2xx	Success: request has succeeded	If for an INVITE, ACK should be sent; otherwise, stop retransmissions of request
3xx	Redirection: server has returned possible locations	The client should retry request at another server
4xx	Client error: the request has failed due to an error by the client	The client may retry the request if reformulated according to response
5xx	Server failure: the request has failed due to an error by the server	The request may be retried at another server
6xx	Global failure: the request has failed	The request should not be tried again at this or other servers

5.4.3 SIP Headers

This section describes the headers present in SIP messages. There are four types of SIP headers: general, request, response, and entity. SIP headers in

most cases follow the same rules as HTTP headers. Headers are defined as `header: field` where `header` is the case-insensitive token used to represent the header, and `field` is the case-insensitive set of tokens that contain the information. Header fields can continue over multiple lines as long as the line begins with at least one space or horizontal tab character. Unrecognized headers are ignored by proxies. Many common SIP headers have a compact form, where the header name is denoted by a single lower-case character.

Headers can be either end-to-end or hop-by-hop. Hop-by-hop headers are the only ones that a proxy may insert, or with a few exceptions, modify. A proxy should never change the header order. Because SIP typically involves end-to-end control, most headers are end-to-end.

General headers: The set of general headers includes all of the required headers in an SIP message. General headers can be present in both requests and responses. These headers are created by user agents and cannot be modified by proxies, with a few exceptions. The general headers are: `Call-ID, Contact, CSeq, Date, Encryption, From, Organization, Retry-After, Subject, Supported, Timestamp, To, User Agent, Via.`

Request headers: They are added to a request by a UAC to modify or give additional information about the request. The request headers are: `Accept, Accept-Contact, Accept-Encoding, Accept-Language, Authorization, Hide, In-Reply-To, Max-Forwards, Priority, Proxy-Authorization, Proxy-Require, Record-Route, Reject-Contact, Request-Disposition, Require, Response-Key, Route, RAck, Session-Expires.`

Response headers: They are added to a response by a UAS or SIP server to give more information than just the response code and reason phrase. They are generally not added to a request. The response headers are: `Proxy-Authenticate, Server, Unsupported, Warning, WWW-Authenticate, RSeq.`

Entity headers: They are used to provide additional information about the message body or resource requested. This term comes from HTTP where it has a more specific meaning. In SIP, “entity” and “message body” are used interchangeably. The entity headers are: `Allow, Content-Encoding, Content-Disposition, Content-Length, Content-Type, Expires, MIME-Version.`

5.5 SDP: A Companion Protocol

The Session Description Protocol, defined by RFC 2327, was developed by the IETF MMUSIC working group. It is more like a description syntax than a protocol in that it does not provide a full-range media negotiation capability. The original purpose of SDP was to describe multicast sessions set up over the Internet's multicast backbone, the MBONE. The first

application of SDP was by the experimental Session Announcement Protocol (SAP) used to post and retrieve announcements of MBONE sessions.

An SAP message carries an SDP message body, and was the template for SIP's use of SDP. Even though it was designed for multicast, SDP has been applied to the more general problem of describing general multimedia sessions established using SIP.

SDP contains the following information about the media session:

- IP Address (IPv4 address or host name);
- Port number (used by UDP or TCP for transport);
- Media type (audio, video, fax, interactive whiteboard, etc.);
- Media encoding scheme (PCM A-Law, MPEG II video, etc.).

In addition, SDP contains information about the following:

- Subject of the session;
- Start and stop times;
- Contact information about the session.

Like SIP, SDP uses text coding. An SDP message is composed of a series of lines, called fields, whose names are abbreviated by a single lower-case letter, and are in a required order to simplify parsing. The set of SDP fields is shown in Table 5.3.

SDP was not designed to be easily extensible, and parsing rules are strict. The only way to extend or add new capabilities to SDP is to define a new attribute type. However, unknown attribute types can be silently ignored. An SDP parser must not ignore an unknown field, a missing mandatory field, or an out-of-sequence line.

5.5.1 Use of SDP in SIP

The default message body type in SIP is `application/sdp`. The calling party lists the media capabilities that they are willing to receive in SDP in either an INVITE or in an ACK. The called party lists their media capabilities in the 200 OK response to the INVITE.

Because SDP was developed with scheduled multicast sessions in mind, many of the fields have little or no meaning in the context of dynamic sessions established using SIP. In order to maintain compatibility with the SDP protocol, however, all required fields are included. A typical SIP use of SDP includes the version, origin, subject, time, connection, and one or more media and attribute fields. The origin, subject, and time fields are not

Table 5.3 SDP Field List in Their Required Order [14].

Field	Name	Mandatory/ Optional
v=	Protocol version number	m
o=	Owner/creator and session identifier	m
s=	Session name	m
i=	Session information	o
u=	Uniform Resource Identifer	o
e=	Email address	o
p=	Phone number	o
c=	Connection information	m
b=	Bandwidth information	o
t=	Time session starts and stops	m
r=	Repeat times	o
z=	Time zone corrections	o
k=	Encryption key	o
a=	Attribute lines	o
m=	Media information	m
a=	Media attributes	o

used by SIP but are included for compatibility. In the SDP standard, the subject field is a required field and must contain at least one character, suggested to be s =- if there is no subject. The SIP standard, however, allows the subject field to be omitted for two-party sessions. The time field is usually set to t = 0 0.

SIP uses the connection, media, and attribute fields to set up sessions between user agents. Because the type of media session and codec to be used are part of the connection negotiation, SIP can use SDP to specify multiple alternative media types and to selectively accept or decline those media types. When multiple media codecs are listed, the caller and called party's media fields must be aligned—that is, there must be the same number, and they must be listed in the same order. A media stream is

declined by setting the port number to zero for the corresponding media field in the SDP response.

Table 5.4 SDP Attribute values [14].

Attribute	Name
a=rtpmap:	RTP/AVP list
a=cat:	Category of the session
a=keywds:	Keywords of session
a=tool:	Name of tool used to create SDP
a=ptime:	Length of time in milliseconds for each packet
a=recvonly	Receive only mode
a=sendrecv	Send and receive mode
a=sendonly	Send only mode
a=orient:	Orientation for whiteboard sessions
a=type:	Type of conference
a=charset:	Character set used for subject and information fields
a=sdplang:	Language for the session description
a=lang:	Default language for the session
a=framerate:	Maximum video frame rate in frames per second
a=quality:	Suggests quality of encoding
a=fmtp:	Format transport

5.6 SIP Programming Services

SIP's intimate association with all things Internet establishes telephony as part of a continuum of Internet media options. Its similarities with HTTP and SMTP and its text-based format mean that SIP is familiar to web developers. In order to develop services, programmers need APIs. There have been many advances in this area of SIP, resulting in numerous new interfaces.

5.6.1 CPL (Call Processing Language)

This was the first API developed for SIP. Strictly speaking, it is not really an API at all, but rather an XML-based scripting language for describing and controlling call services. It is designed to be implemented on either network servers or user agent servers and is meant to be simple, extensible, easily edited by graphical clients, and independent of operating system or signaling protocol.

CPL is engineered for end-user service creation: a CPL interpreter is very lightweight and a server can easily parse and validate a CPL, guarding against malicious behavior. It is suitable for running on a server where users may not be allowed to execute arbitrary programs, as it has no variables, loops, or ability to run external programs. It has primitives for making decisions and taking actions based on call properties, such as time of day, caller, called party etc.

5.6.2 SIP-CGI

In the World Wide Web, the Common Gateway Interface (CGI) has served as a popular means of programming web services. CGI scripts have been the initial mechanism to make websites interact with databases and other applications. Due to the similarities between the SIP and HTTP, CGI is a good candidate for service creation in an SIP environment.

Like HTTP CGI, an SIP CGI script resides in the server and passes message parameters through environment variables to a separate process. The process sends instructions back to the server through its standard output file descriptor. SIP CGI is almost identical to HTTP CGI and is particularly suitable for services that contain substantial web components. A CGI script can be written in Perl, Tcl, C, C++ or Java making it accessible to a large community of developers.

5.6.3 SIP and Java

Introduction

The goal of this section is to identify the specifications to the Session Initiation Protocol (SIP) defined through the Java Community ProcessSM (JCPSM). This section inspects each Java SIP specification, describing their functionalities and the supported platforms.

The Java language defines three platforms, namely Java 2 Platform, Standard Edition (J2SE™), Java 2 Platform, Enterprise Edition (J2EE™) and

Java 2 Platform, Micro Edition (J2ME™). Targeting the three Java platforms are four Java Specification Requests (JSRs) for SIP. These are:

-JAIN™ SIP: JSR 32 defines the JAIN SIP specification. Sun Microsystems is the specification lead.

-SIP Lite: JSR 125 defines the SIP Lite specification. Ubiquity is the specification lead.

-SIP Servlet: JSR 116 defines the SIP Servlet specification. Dynamicsoft is the specification lead.

-SIP for J2ME: JSR 180 defines the SIP for J2ME specification. Nokia Corporation is the specification lead.

The substantial overlap in the experts and companies defining the SIP specifications through the JCP helps ensure consistent API specifications across the various expert groups. The collection of Java SIP specifications does not redefine or modify the SIP protocol. They simply define standardized API specifications specific to the Java environment, with the goal of simplified application development and application portability across different implementations of the SIP protocol.

The Java SIP Specifications

JAIN SIP

The JAIN SIP specification was the first SIP specification standardized through the JCP. The JAIN SIP specification is a general purpose transaction based Java interface to the SIP protocol. It is rich both semantically and in definition to the SIP protocol. The motivation behind JAIN SIP is to develop a standard interface to the SIP protocol that can be used independently or by higher level programming entities and environments. JAIN SIP can be used in multiple ways:

-As a specification for the J2SE platform that enables the development of stand alone user agent, proxy and registrar applications.

-A base SIP implementation for an SIP Servlet container that enables the development of user agent, proxy and registrar applications in a Servlet-based environment.

-A base SIP implementation for an Enterprise JavaBeans™ (EJB™) container that enables the development of user agent, proxy or registrar applications in an EJB environment.

JAIN SIP provides a standardized interface that can be used by communications developers as a minimum to support SIP in their applications. The JAIN SIP reference implementation provides a fully functional SIP implementation that can be used by developers to talk SIP from the Java environment. The target developer community for JAIN SIP is

developers that are familiar with the SIP protocol and require transactional control over the SIP implementation.

SIP Lite

The SIP Lite specification is an abstracted view of the SIP protocol that provides an SIP programming environment for developers who are not SIP literate. The API specification is primarily developed for the J2SE platform, however as the specification is quite small it can also be implemented on the J2ME platform. The motivation behind SIP Lite on the J2ME platform is to provide a rich object model that may be suitable for midsize devices with more processing power and memory than mobile handsets, i.e. PDAs and SIP phones. SIP Lite is most common as a J2SE platform based user agent or a programming interface to an SIP Phone.

SIP Servlet

The SIP Servlet API specification defines an environment for execution of network based SIP applications. It is implemented on application servers that support SIP and optionally also HTTP and the J2EE platform. It builds on the HTTP Servlet API specification and like HTTP Servlet defines both an API and a file format for application packaging. SIP Servlet supports baseline SIP as defined in RFC 3261 and also supports the following SIP extensions; the Event Notification framework (RFC 3265) and the Message method (RFC 3428) for instant messaging.

At the heart of SIP Servlet lies the ability of applications to perform SIP signaling, either as an endpoint (user agent) or as a proxy. The API specification aims to allow applications complete control over SIP signaling while at the same time hiding much of the non-essential complexity of SIP, which is not relevant to application developers.

The main difference between a non-application server based SIP API specification and an application server based SIP API specification is that an application server itself creates and manages resources whereas the former generally speaking does not. SIP Servlet containers manage resources like listening points, threads, transactions and dialogs, session state, and application components.

SIP for J2ME

The SIP for J2ME API specification defines an SIP interface for small platforms and is still quite early in the definition process. SIP's acceptance as the protocol of choice by the IP Multimedia Subsystem (IMS) architecture within the Third Generation Partnership Project (3GPP™), highlights the

value of mobile devices understanding and communicating SIP. The goal of the SIP for J2ME API specification is to address this need.

Possible End to End architectures using the SIP APIs

Network architectures are often unique unto themselves and network requirements often determine architecture choices. However it is foreseen that the Java SIP specifications will be used as follows in end-to-end network architectures:

-SIP for J2ME will be implemented in mobile handsets.

-SIP Lite will be implemented in midsize devices, i.e. PDAs, SIP phones, and desktops.

-JAIN SIP will be implemented on desktops and in business tier enterprise application servers. It may also be the base SIP implementation for SIP Servlets.

-SIP Servlet will be implemented in web tier enterprise application servers, bringing the benefit of converged SIP and HTTP applications.

-SIP Servlet, JAIN SIP or JAIN SIP wrapped in a J2EE connector will co-exist in the core telecom network as the backbone for SIP network signaling and custom based SIP servers.

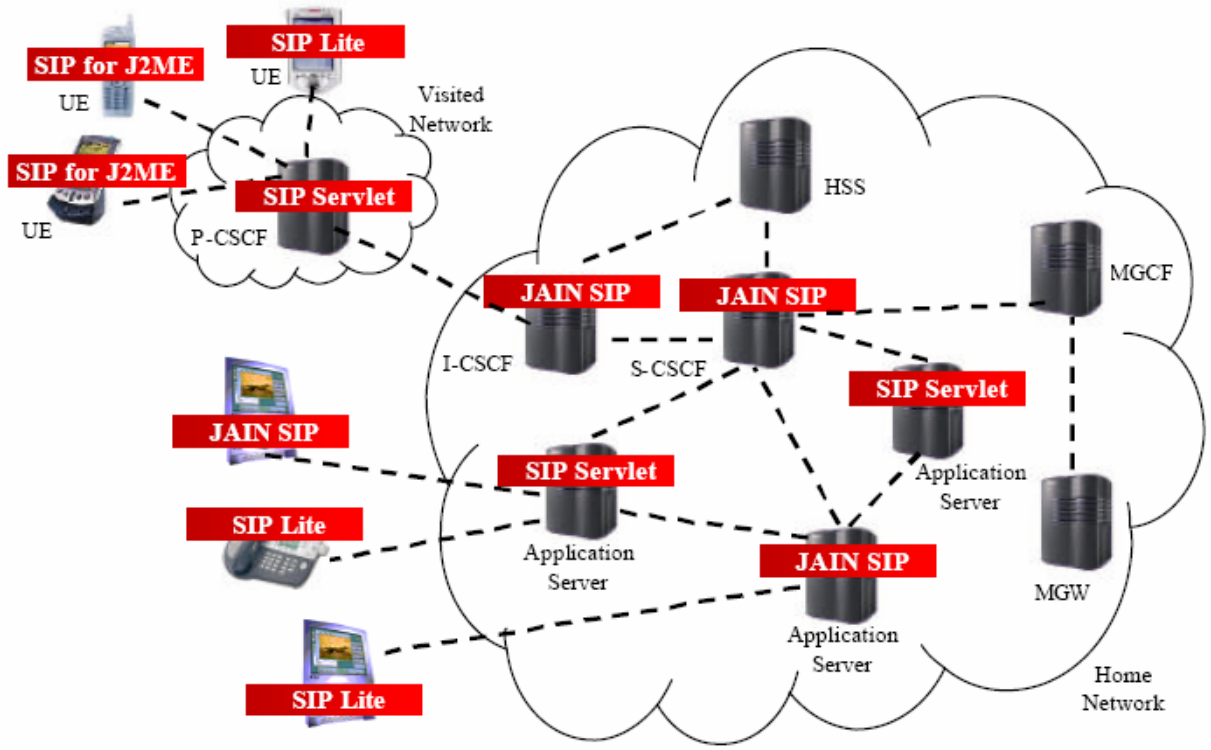

Figure 5.4 IMS architecture and the applicability of the Java SIP specifications [35].

Figure 5.4 shows how existing Java specification interfaces to SIP map onto the IP Multimedia Subsystem (IMS) architecture as defined by 3GPP.

The Future of SIP and the Java platforms

The SIP protocol was designed with Internet thinking, which compliments the enterprise thinking of many Java developers. This attribute should help lower the barrier of convergence between the telephony and Internet worlds. The communications industry is behind the success of both SIP and the Java platforms, highlighted by the numerous SIP specifications being standardized through the JCP. The acceptance of SIP as the universal signaling language for internet communications, coupled with the acceptance of the Java language as a preferred alternative to C or C++ as a development language for communication protocols demonstrates an inherent shift in thinking. The rapid growth of the Java language and the use of SIP for communications offer a great formula for communication network architectures.

The Java language has a number of features that make it very attractive for the SIP protocol. The dynamic loading features make it easy to deploy and update applications at runtime; errors can be caught and handled gracefully; the built-in security framework allows containers to restrict applications in what actions they can perform. The Java platform also provides access to a large set of useful functionality, such as JDBC for database access, JNDI for directory lookups and JMF for handling streamed media. Furthermore, integration of the SIP protocol with the J2EE platform ensures SIP applications can plug into back-end infrastructure already in networks today.

The Java SIP standards are also important for the success of the SIP protocol. Communications protocols traditionally are standardized in text document or unified modeling language (UML) format only. This leaves scope for interpretation when realizing an API specification for a specific programming environment, which has been the downfall of many good protocols in the past. The Java SIP standards unify the SIP communications development community and drives SIP into the enterprise domain by providing standardized API specifications to the SIP protocol specific to each Java platform.

When the SIP protocol gains widespread adoption, a strong case will be presented to bundle SIP functionality into the J2SE platform. Acceptance of SIP in the J2SE platform may in turn lead to SIP being adopted by both the J2EE platform and the J2ME platform (similar to HTTP) in the future. It is expected in the coming years that the J2EE platform will experience the biggest growth of SIP enabled applications; however acceptance of SIP in the J2EE platform indirectly means success for SIP in the J2SE platform. The production of SIP applications on the J2ME platform may take a little longer

due to air interface bottlenecks and limited processing power on the mobile handset.

5.7 SIP and T.38 Utilization for FoIP

The best current practices for SIP T.38 fax and SIP fax pass-through sessions are documented in this IETF Internet-Draft: “SIP Support for Real-time Fax: Call Flow Example And Best Current Practices” [36]. Here we provide a brief overview of this document.

The Session Initiation Protocol (SIP) allows the establishment of real-time Internet fax communications. Real-time facsimile communications over IP may follow 2 modes of operation: T.38 fax relay as defined by the ITU-T T.38 recommendation or fax pass-through.

This document clarifies the options available to Internet telephony gateway vendors to handle real-time fax calls using SIP. While the primary focus is to address the more reliable real-time T.38 Group 3 fax mode, fax pass-through mode to enable fallback operations and super G3 fax communications using SIP are also briefly covered. Examples of SIP call flows for real-time Internet fax gateways or SIP proxy redirect servers are given as well. Elements in these call flows include SIP User Agents, SIP Proxy Servers, and Gateways to the PSTN (Public Switch Telephone Network).

A session starts with audio capabilities, and, upon fax tone detection, T.38 fax capabilities are negotiated; upon successful negotiation, the session continues with fax capabilities and the media termination hosts exchange T.38 Internet fax packets. The T.38 fax call scenarios include various aspects of the call sequence: the detection of fax transmission, the usage of the T.38 session description attributes, the optional fallback into fax pass-through mode and the session termination. The fax pass-through call scenarios involve some specific SDP media attributes to enable proper fax transmission. Fax transmission can be detected by the receiving side, the emitting side or both.

This document only covers the case when the fax transmission is detected by the receiving side (it is the most common practice and the other cases do not represent any particular challenges and are therefore left for future discussions). Call flow diagrams and message details are shown. A list of IANA defined SDP attribute names for T.38 is summarized in Section 7 of the document.

For T.38, this document deals primarily with one transport protocol for the media: T.38 over UDP/UDPTL; T.38 fax packet transport over TCP using

SIP session establishment can easily be considered as well. These T.38 call flows were developed in the design of carrier-class SIP Telephony products supporting voice and real-time fax traffic.

Some variants of these best current practices may apply depending on the nature or the configuration of Internet telephony gateways. Two distinct cases are considered in this document:

-The Internet telephony gateway only supports T.38 real-time fax communications. In this case, the Internet fax gateway should initiate the SIP session with T.38 SDP capabilities (this is typically the case of Internet fax terminals, also called Internet-aware fax devices or the case of gateways statically configured to support T.38 fax calls only);

-The Internet telephony gateway supports voice and real-time fax communications. In this case, the Internet telephony gateway initiates the SIP session with audio capabilities, and, upon fax detection, the switchover to T.38 fax capabilities is triggered. The fax media connection may replace or be added to the audio connection depending on the target applications.

For a detailed description of SDP attribute table for T.38 sessions, refer to IANA's SDP parameters registration page.

5.8 For Further Study

For further information regarding the concepts discussed in this chapter please refer to following resources:

5.8.1 SIP

[14]
[37]
[10]
[38]
[39]
[40]
[41]
[42]
[43]
[36]

5.8.2 Call Flow Examples

[14] PP: 153-185
[44]

5.8.3 SDP

[14]
[45]
[46]

5.8.4 Programming SIP

[35]
[47]
[48]
[49]
[50]
[51]

CGI

[52]
[47]

JAIN™ SIP

[53]
[54]

Relevant Links

- The JAIN website: http://java.sun.com/products/jain
- The Java Community Process website: http://jcp.org/

Chapter 6

Simulator Implementation Details

6.1 Introduction

In this chapter, J-Sim, an open-source simulator, is introduced. This simulator is utilized throughout the implementation part of this thesis to explore the behavior of the developed protocol and components. Many details have been left out to simplify the introduction of the simulator. It should be mentioned that the simulator enjoys having a comprehensive and detailed documentation which has to be consulted for further delving into the details. All of the materials in this chapter, including the figures, have been taken from the simulator documentation and have been slightly modified and summarized. These materials, without further referencing, are therefore considered the intellectual property of the J-Sim developers team. In the next section, a brief introduction is given and the salient features of the simulator are discussed in the Section 3. A quick overview of the inner workings of the simulator and how one can develop new modules are given in Section 4. Simulation scenario creation, configuration and running are briefly introduced in the last section.

6.2 Introducing J-Sim

J-Sim is a component-based, compositional simulation environment. It has been built upon the notion of the autonomous component programming model. Similar to COM/COM+, JavaBeans, or CORBA, the basic entity in J-Sim is component, but unlike the other component-based software packages/standards, components in J-Sim are autonomous and are realization of software ICs.

The autonomous component architecture mimics the IC design architecture in the closest possible way. The behavior of J-Sim components are defined in terms of contracts (in much the same way IC chips are defined in the specification in the cookbook) and can be individually designed, implemented, tested, and incrementally deployed in a software system. A system can be composed of individual components in much the same way a hardware module is composed of IC chips. Moreover, components can be plugged into a software system, even during execution.

For the purpose of network modeling and simulation, it is defined, on top of the autonomous component architecture, a generalized packet switched network model. The model defines the generic structure of a node (either an end host or a router) and the generic network components, both of which can then be used as base classes to implement protocols across various layers. Although the model is derived by factoring out the common attributes of network entities in the current best-effort Internet, it is general enough to accommodate other network architectures, such as the IETF differentiated services architecture, the mobile wireless network architecture, and the WDM-based optical network architecture.

J-Sim has been developed entirely in Java. This, coupled with the autonomous component architecture, makes J-Sim a truly platform-neutral, extensible, and reusable environment. J-Sim also provides a script interface to allow integration with different script languages such as Perl, Tcl, or Python. In the current release, J-Sim is fully integrated with a Java implementation of the Tcl interpreter (with the Tcl/Java extension), called Jacl. So, similar to ns-2, J-Sim is a dual-language simulation environment in which classes are written in Java (for ns-2, in C++) and glued together using Tcl/Java. However, unlike ns-2, classes/methods/fields in Java need not be explicitly exported in order to be accessed in the Tcl environment. Instead, all the public classes/methods/fields in Java can be accessed (naturally) in the Tcl environment.

6.3 J-Sim features

6.3.1 Loosely coupled, autonomous component programming model

The software architecture is component-based with clear, well-defined interfaces. Similar to JavaBeans, CORBA, and COM/DCOM, the basic entity in J-Sim is component**,** and an application is simply a composition of components. Ports are part of a component and are used by the component to communicate with other components. However, unlike the above component-based software packages/standards, the components defined in J-Sim are loosely coupled and autonomous. By loosely coupled, it is meant that the behavior of a component is specified in terms of contracts. A contract is bound to a specific port or a group of ports, and defines the causality of data sent/received between the component that owns the port(s) and whichever components that are connected to the port(s). In particular, it does not specify the components that participate in the communication. Component binding is deferred until system integration time. By autonomous, it is referred to the capability of components to handle data in independent execution contexts. When data arrives at a port of a component, the component processes the data immediately in an independent execution context. The interference between different pieces of data handled by the same component at the same time is minimal.

The ability to handle data in independent execution contexts, along with the fact that components are loosely coupled and only bound to one another at system integration time, is the key reason that a component can be reused in other software systems with the same contract context, in much the same fashion as IC chips are used in hardware design.

6.3.2 Dynamic thread execution framework for real-time process-driven simulation

In J-Sim, execution contexts are implemented by Java threads, with the thread scheduler in the Java Virtual Machine (JVM) scheduling thread execution. The runtime contains three classes, WorkerThread, WorkerManager and WorkerPool:

1. The WorkerThread wraps the Java Thread class up with the runtime, execution context information.
2. The WorkerManager class implements the control mechanism that controls the number of WorkerThreads that can be simultaneously active.

3. The WorkerPool class creates new WorkerThreads, recycles WorkerThreads that finish processing data, and provides execution control (e.g., stopping or resuming execution of a thread) to the entire system.

In J-Sim, the simulation engine extends the WorkerPool class and monitors the activities of all WorkerThreads. It maintains a globally-observed, virtual system time that is proportional to the real time (e.g. 1 second in virtual time = 1000 seconds in real time). When no WorkerThread is currently active, the simulation engine adjusts the virtual system time to the nearest future so that at least one WorkerThread may become active. With the above mechanism, a simulation runs in the same manner a real system does, in the sense that event executions are carried out in real time as opposed to at fixed time points in discrete event simulation. The interactions and interferences among event executions, hence take place naturally as in real systems. When no thread is currently active, the simulation engine performs a "fast-forward" operation in time to the nearest future at which at least one execution can be activated. This preserves the behavior of real systems and hence enhances the fidelity of the simulation. The thread overhead incurred is mitigated by reusing threads.

6.3.3 Implementation of a complete suite of Internet Integrated /Differentiated /Best Effort Services protocols

For the purpose of network modeling and simulation:

-An abstract network model is defined and implemented on top of the component-based architecture. In particular, the generic structure of a node and several network components are defined. The centerpiece of the nodal structure is the core service layer that provides the fundamental core internetworking services (Figure 6.1). The core service layer is also decomposed into components and associated contracts are defined well.

-Every component in the autonomous component architecture or in the abstract network model is implemented as a class, and classes are organized into layers. Figure 6.2 depicts the five-layer class organization in J-Sim.

-An (almost) complete suite of Internet best effort, integrated services, and differentiated services protocols are implemented. Table 6.1 lists the models/protocols currently supported in J-Sim.

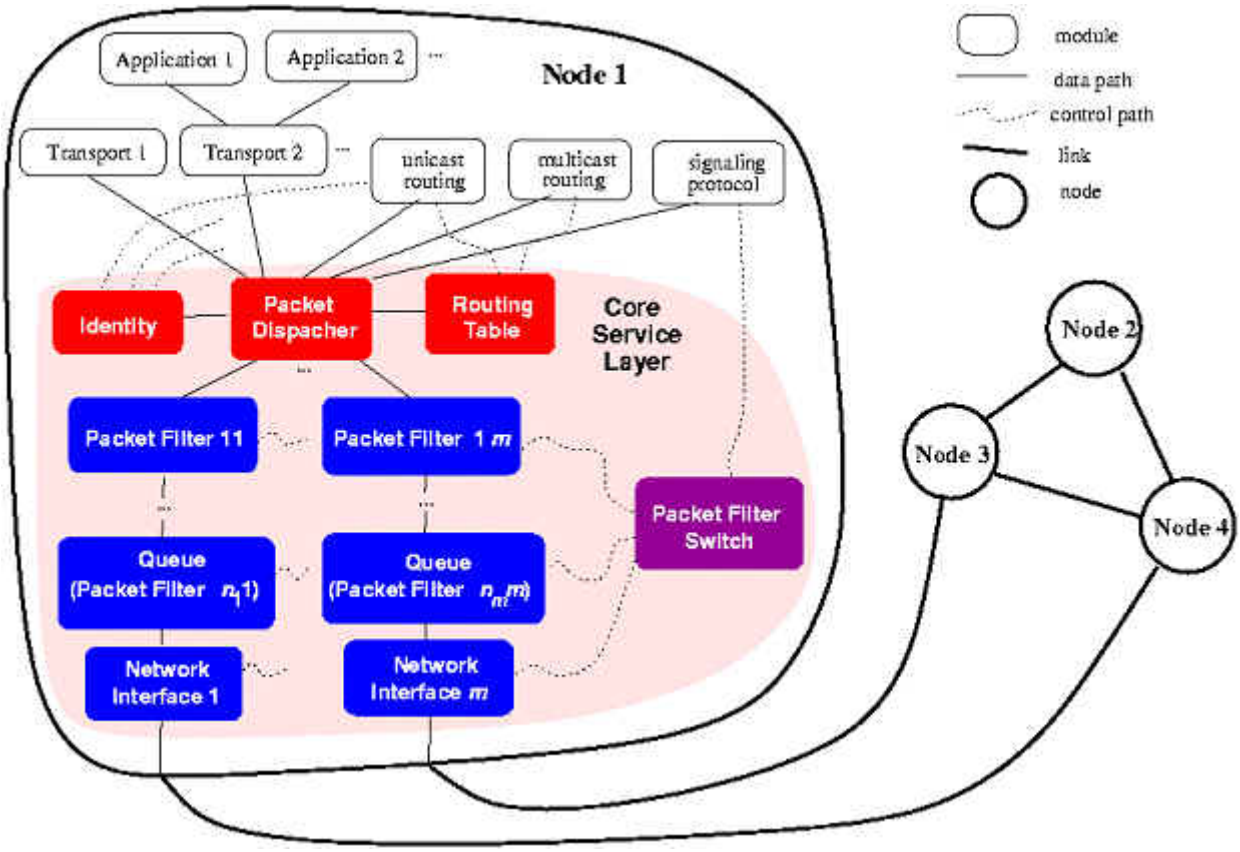

Figure 6.1 The internal structure of a node.

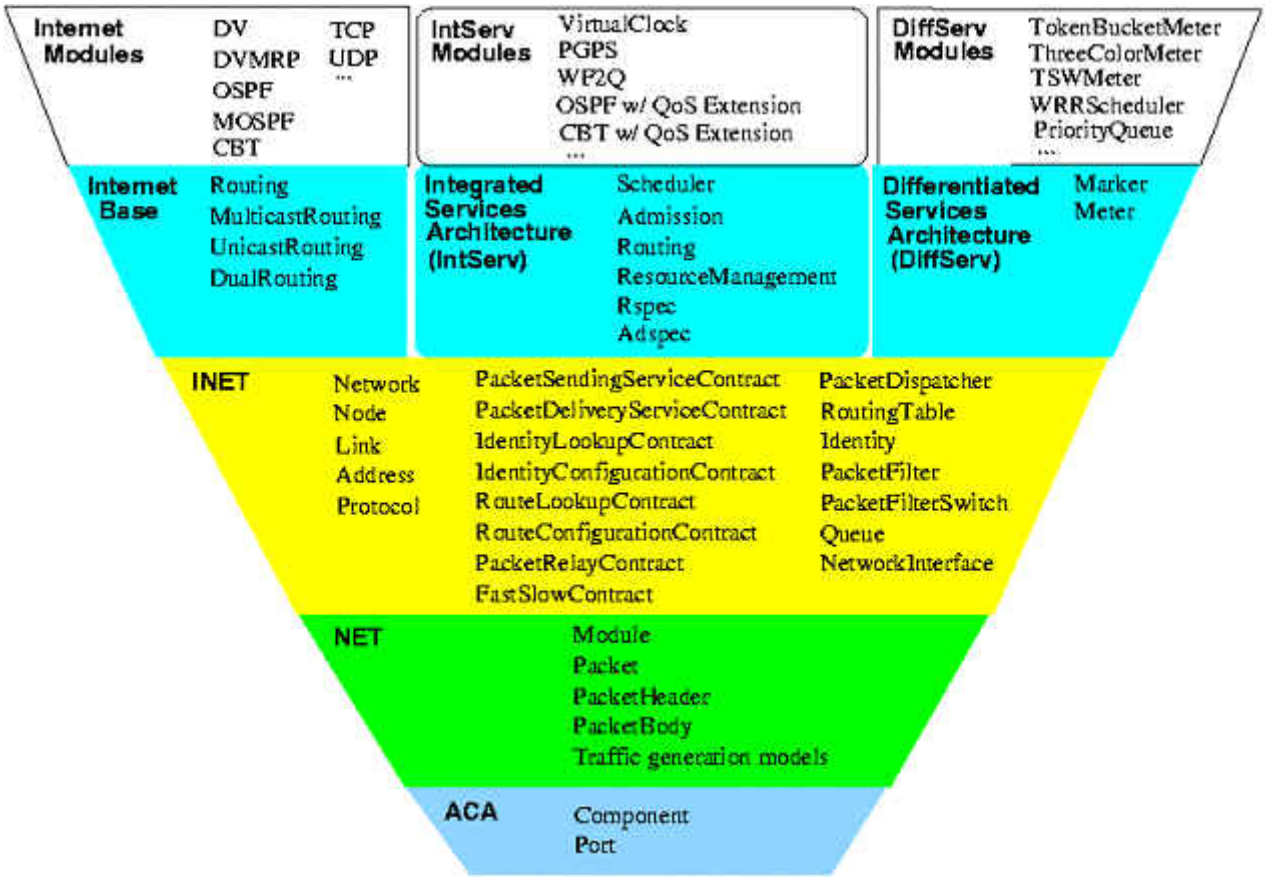

Figure 6.2 The class pyramid in J-Sim.

The implication of the implementation is three fold:

1. With all the Internet protocol classes available, one can compose the protocol stack and conduct the simulation under different network scenarios in a plug-and-play fashion.

2. With the abstract classes that capture the fundamental features of network entities and yet are flexible enough to accommodate new technology advances, one can extend J-Sim to a new network architecture, e.g., wireless LANs, optical networks with WDM technology, networks with satellite communication links, or ad hoc networks consisting primarily of mobile sensors. This is done by subclassing appropriate network modules and redefining their network attributes and methods that manipulate the attributes. For example, one needs only to modify the network interface card (NIC) component and the link component to incorporate the error characteristics and the mobility

Table 6.1 Algorithms and protocols supported in J-Sim.

Network Architecture	Application	Socket Layer	Transport	Routing	Traffic Model	Tagger Marker	Buffer Management	NI Scheduling
Best Effect Services	**FTP, FSP WWW**	**BSD 4.3**	**TCP-Reno TCP-Tahoe TCP-Vegas TCP Sack UDP**	**RIP (DV) OSPFv2 Multicast shortest path tree Multicast minimum load tree Multicast Steiner tree DVMRP MOSPF CBT**			**Drop-Tail**	**FIFO**
Differentiated Services						**Token Bucket TSW ETSW**	**RED FRED SRED BRED**	**FIFO**
Integrated Services			**RSVP**	**Unicast QoS routing QoS-enhanced OSPFv2 QoS-enhanced CBT**	**Periodic message (CBR) Peak rate model Leaky bucket model Token bucket model IETF/Intserv Flowspec (r,t)-smooth model (C,D)-smooth model**			**RM EDF Stop-and-go DCTS VirtualClock LFVC SCFQ PGPS STFQ WF^2Q Leave-in-time**

FTP: file transfer protocol
FSP: file service protocol
BSD: Berkeley socket distribution
TCP: transmission control protocol
RIP: routing information protocol
OSPF: open shortest path first
DVMRP: distance vector multicast routing protocol
MOSPF: multicast extension to OSPF
CBT: core based tree protocol
FIFO: first in first out
TSW: time sliding window
ETSW: enhanced time sliding window
RED: random early drop
FRED: fair random early drop
SRED: stable random early drop
BRED: balanced random early drop
RSVP: Resource reservation protocol
CBR: constant bit rate
RM: rate monotonic
EDF: earliest deadline first
DCTS: distance constrained task system
LFVC: leap forward virtual clock
SCFQ: self-clocked fair queueing
PGPS: packet-by-packet generalized processing sharing
STFQ: start time fair queueing
WF^2Q: worst-case fair weighted fair queueing

characteristics in order to model wireless mobile networks. Similarly, one can readily implement a new algorithm/protocol for experimentation and validation, simply by subclassing one or more appropriate protocol modules.

3. By virtue of the component hierarchy (i.e., a component can be a composite component that contains child components), one can vary the level of details to which simulation is conducted.

6.3.4 A dual-language environment that allows auto-configuration and online monitoring

A dual-language environment is provided in which Java is used to create components and a script language is used as the glue or control language to integrate components at run time and to provide high-level, dynamic control. This facilitates fast configuration of customized simulation scenarios, and online monitoring and data collection. In the current release, J-Sim is fully integrated with a Java implementation of the Tcl interpreter (with the Tcl/Java API extension), called Jacl developed by Scriptics Corporation. This enables access to, and manipulation of, Java objects, such as creating an object from a Java class and invoking a method or accessing a field variable of a Java object, in the Tcl environment.

In conjunction with the dual-language environment, information and event ports for data collection and debugging at the component level have been designed. One may connect an "instrument" component or script to the information port and the ports in the event group of a component of interest to configure/inspect the component at runtime. This closely mimics the IC debugging and testing process.

6.4 Working with J-Sim

J-Sim is an application development environment based on the component-based software architecture, Autonomous Component Architecture or ACA. This section serves these purposes: first, it gives an overview of the scripting language and the component-based architecture; then it outlines the Runtime Virtual System (RUV) and how to write components.

6.4.1 Scripting Using Tcl

Scripting is commonly used in a large software environment for users to access and manipulate components at the desirable granularity. Scripting is an essential part of J-Sim and it is used to glue all the components and defines how the system operates. Tcl/Java, made available by third parties, is an extension to the core Tcl. It makes it possible to manipulate Java objects in the Tcl environment. The pure-Java implementation of Tcl 8.0 with Tcl/Java, called Jacl, is integrated in the toolkit. Tcl/Java defines a set of new Tcl commands for manipulating Java objects, such as creating an object from a Java class, invoking a method of a Java object, or accessing a field variable of a Java object. A list of Tcl/Java commands can be found in the online manuals at the Tcl/Java website.

6.4.2 ACA Overview - Component and Port

The basic entity in the software architecture is a component. An application is envisioned as a composition of components. The notion of components is not new, and has been used in several commercial component-based software standards such as JavaBeans, CORBA and COM/DCOM/COM+. Unlike JavaBeans, CORBA, and COM/DCOM, the components in J-Sim are loosely coupled, communicate with one another by wiring their ports together, and are bound to contracts. Contracts are bound at design time and components are bound at system integration time. One immediate advantage of this separation is that different components can be independently developed (on different platforms and/or different programming languages) and integrated later.

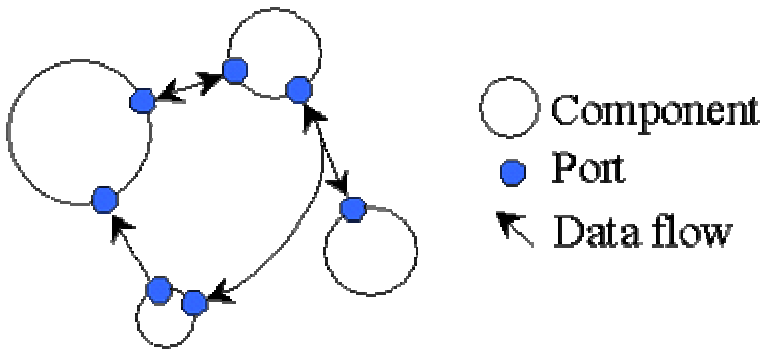

Figure 6.3 The component-based architecture.

Port: A component communicates with the rest of the world via its ports. A component may own more than one port. The programming

interface between a component and its port is well defined. Since a component only interfaces with its ports, one component can be developed without the existence of other components. Also, the actual communication mechanism a component uses to communicate with the rest of the world is completely hidden in ports.

Contract: The behavior of a component is described by the port contract and the component contract. A port contract is bound to a specific port or a group of ports, defines the communication pattern between the component that owns the port(s), and the other components that are connected to the port(s). The component contract is the same as the traditional blackbox specification and characterizes the input/output relation of a component. A component is expected to work properly if all the adopted contracts are fulfilled.

It should be clear now that when a user writes a component, they have to follow only the contracts adopted by the component, but do not need worry about the other components or the communication mechanism between them.

A good analogy of the component-based architecture is the current IC architecture, where a hardware module (a software system) is assembled by connecting a set of IC chips (components) through their pins (ports). When the signals arrive at the pins of an IC chip, the chip performs certain tasks in compliance with the specification in the cookbook (contract), and may send signals at some other pins. A component can be reused in other software systems with the same contract context, in much the same fashion as IC chips are used in hardware design. Ports in a component can be organized into different groups. A port group is uniquely identified within a component by its group ID. A port is uniquely identified within a port group by its assigned port ID. Therefore, a port is uniquely identified within a component by its port group and port IDs. A component has a default port group with empty ID ("").

The ultimate goal of the component architecture is to mimic the current hardware manufacturing architecture. By selecting and connecting an appropriate set of chips, one can readily compose a hardware component with desirable functions. An important step towards the goal is to build a set of components that can be re-used in applications of similar nature.

6.4.3 More on Components - Component Hierarchy

In J-Sim, a (parent) component may include several sub-components (called child components). Figure 6.4 illustrates the concept. A component is uniquely identified within its parent component by its ID.

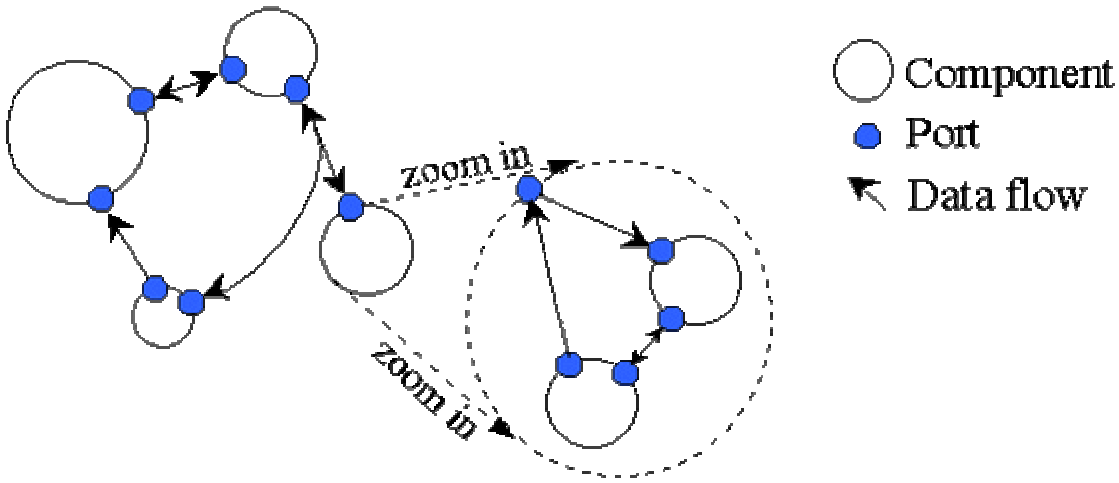

Figure 6.4 The component hierarchy.

Ports of a child component or the child component itself may be exposed to the outside world of its parent. Port exposure is realized by creating a port for the parent and then connecting it to the port of the child. The port of the parent component acts as a shadow port of that of the child component. The real communication occurs between the outside world and the child component's port.

6.4.4 The Runtime Virtual (RUV) System

In the course of developing a large software project, it may become cumbersome to use many of the Tcl/Java commands because one has to store the references of the Java objects in Tcl variables in order to access them. Naming of these Tcl variables is not at all trivial. For example, in the component hierarchy, a Tcl variable can only be used to access one component in the hierarchy and the other components/ports have to be accessed by using methods like `getParent()`, `getComponent()` and `getPort()`.

To mitigate the referencing problem, a referencing system is constructed, called RUntime Virtual system or RUV in short, on top of Tcl/Java. Because the component hierarchy resembles in essence the UNIX file system hierarchy, the same notation is employed and a component or a port is represented as a path, in the same manner a file is represented in a file system. For example, `/component1/child2` represents the component with

ID child2 within the component represented by /component1; /component3/port3@group4 represents the port with ID port3 in the port group of ID group4 within the component represented by /component3. Note that a "@" is used in a port path to separate the port ID and the port group of the port.

Moreover, several UNIX-file-system-like commands have been provided, such as ls, cd, pwd, mkdir, cp, mv, rm and cat in the context of the component hierarchy to navigate a component system and manipulate components and ports in the system. The commands are summarized in Table 6.2 (Detailed syntax of each command can be found in the provided documentation and hence is omitted here due their large size).

Table 6.2 RUV commands.

Command	Description
! <path> ?<method> ?<arg>	Convert the path to the reference to the Java object and then invoke the method specified in the argument list.
cat ?<path>?	Print the internal state (invoking Component.info(), Port.info()), and/or the connections, of a component /port.
cat <obj_ref>	Print the values of a Java object; especially handy when the object is a Java array.
cd <path>	Change the current working directory to path.
connect <path>... -to|-and <path>...	Connect components/ports. Options -and Set up a bidirectional connection. -to Set up a unidirectional connection.
cp <source path>... <dest path> **cp** <source path>... -d <dest path>...	Copy the components/ports.
disconnect <path>...	Disconnect components/ports.
exit	Close the current terminal. If the current terminal is the last terminal, executing this command also exits J-Sim.
inject <data> <port_path>...	Inject the specified data to the port(s).
ls ?<path>...?	List the child components (and ports).
mkdir <className> <path>... **mkdir** <obj_ref> <path>... **mkdir** <port_path> ?<port_path>...? **mkdir** <component_path>	Create components/ports. If the class name or the Java object is not a component/port, a wrapper component (drcl.comp.WrapperComponent) is created to encapsulate the object. The fourth form uses the default class to create component(s) at the specified path. It is equivalent to "mkdir <default_class> <component_path>".
mv <source path>... <dest path>	Move (rename) components/ports.

pwd	Print the current working path.
rm <path>...	Remove components/ports.
set_default_class ?<class_name>?	Set the default class for 'mkdir'.
term <title> ?-t <terminal_class>? ?-s <shell_class>|<shell_object>? ?<init_script>?	Create a terminal. One may specify the terminal class (e.g., Tcl/Java, Perl, Python), the shell class or object to be associated with the terminal, and/or the initial script to execute.
whats_default_class	Print the name of the default class for 'mkdir'.

RUV has powerful path expressions. In addition to the wildcard expressions using "*" and "?", one may use the range expression "<ID>-<ID>" that specifies a range of components.

6.4.5 A Template to Start Writing a Component With

The following example is provided which may serve as a template for writing a complete and correct component. Essentially one must override one method, process(), and may override six other methods, reset(), duplicate(), info() and _start()/_stop()/_resume() in a component.

Component Template:

```
 1 import drcl.comp.*;
 2 import drcl.comp.Port;
 3
 4 /**
 5  * Template for writing a component.
 6  */
 7 public class ComponentTemplate extends drcl.comp.Component
 8                                     implements drcl.comp.ActiveComponent
 9 {
10     // The fields here are simply defined to demonstrate how
11     // methods in this template are used.
12     int x;
13     Port outPort;
14
15     /**
16      * Constructor.
17      */
18     public ComponentTemplate()
19     {
20         super();
21     }
22
23     /**
24      * Constructor.
25      */
26     public ComponentTemplate(String id_)
27     {
```

```
28          super(id_);
29      }
30
31      /**
32      * Invoked when data_ arrives at this component at the inPort_
33      * port. */
34      protected void process(Object data_, Port inPort_)
35      {
36          // Put codes here for handling the incoming data.
37              outPort_.doSending(data_);
38         }
39     }
40      /**
41       * Resets this component to the initial state to use anew.
42       * Must call super.reset() in the beginning.
43       */
44      public void reset()
45      {
46          super.reset(); // Let super class reset its fields.
47          x = 0;         // Reset the fields defined in this class.
48      }
49      /**
50       * Copies the content from the source_ to this component.
51       * Must call super.duplicate() in the beginning.
52       */
53      public void duplicate(Object source_)
54      {
55          super.duplicate(source_); // Let super class copy its fields.
56          ComponentTemplate that_ = (ComponentTemplate)source_;
57          x = that_.x;              // Duplicate the fields defined in
58           // this class.
59      }
60      /**
61       * Invoked when the component is run()ed.
62       */
63      protected void _start()
64      {
65          debug(this + " is starting!");
66      }
67      /**
68       * Script interface which reveals the internal states of the
69       * component. It is for debugging and demonstration purposes. */
70      public String info()
71      {
72          return "Current count = " + x + "\n";
73      }
74      /**
75       * Script interface which increments the counter by +1.
76       */
77      public void increment()
78      {
79          x++;
80      }
81 }
```

Method Overrides

1. A component is triggered by data (`process()`, line 34). The `process()` method is the heart of the component and implements the behavior of

the component. Specifically, the method is invoked by the system whenever some data arrives at one of the component's ports. As mentioned earlier, the method is executed in a new thread context. It is possible for a component to handle multiple data simultaneously in multiple threads. It is component writer's responsibility to ensure data integrity and synchronization among multiple threads.

2. A component can be reset (`reset()`, line 44). The `reset()` method sets the component to its initial state. It allows the component to be started anew after being executed for some time. To correctly override the method, the subclass must call `super.reset()`. When recursive calling of the `super.reset()` method reaches `drcl.comp.Component`, the method resets the ports and the child components in a recursive manner.
3. A component can be duplicated (`duplicate()`, line 53) The `duplicate()` method allows the component to be cloned: `clone()`. A subclass should override this method to copy the fields defined in the subclass. When overriding the method, the subclass must call `super.duplicate()` so that the super class can copy the fields defined in it. When recursive calling of the `super.duplicate()` method reaches `drcl.comp.Component`, the method duplicates the ports and the child components in a recursive manner, and then connects the child component in the same manner as in the source component. The method originates from the `drcl.ObjectDuplicable` interface. It complements the clone mechanism that exists in the `java.lang.Object`. The RUV system command cp uses this method to do the tricks.
4. A component may be started as a data source (`_start()` line 63) In addition to being triggered by data that arrive at ports, a component can be started by the `run()` method. The `run()` method creates a new thread and then the thread calls the _start() method of the component. The `run()` method also calls the `run()` method of the child components, until all the components in the hierarchy are started: `_start()`. As not all the components in the hierarchy need to be started: _start(), the `drcl.comp.ActiveComponent` interface is provided. In the `run()` process mentioned above, only when a component implements the interface, a new thread is created which in turn calls the `_start()` method of the component.
5. One can get state information of a component during runtime (`info()` line 70). For the purpose of online debugging and monitoring, a component should override the `info()` method to provide useful information such as the internal states of the component.

Rules of Component Writing

1. Script interface: In addition to the methods discussed above, a component may provide several script interfaces to manipulate the component from the scripting environment (e.g. in a Tcl terminal). However, as viewed from a component, the script interfaces of the other components can not be accessed because by virtue of the loosely coupled component architecture, these interfaces are blocked by ports. As a component is interfaced with ports, it does not know to which components it is connected and cannot readily access the interfaces in other components.
2. Data sent is out of sender's control: When a component sends a piece of data, it does not own the data anymore. This rule is not enforced by the programming language. Programmers must make sure that the component does not operate on data that is already sent. No assumption can be made on the data (e.g. modified or recycled by the receiving component) unless it is clearly stated in the contract.
3. Using wrapped APIs to do thread synchronization: In order to gain better control on threads which process data on a component, wrapped thread synchronization APIs are provided, namely `wait(Object)/notify(Object)/notifyAll()` in `drcl.comp.Component` in replace of `wait()/notify()/notifyAll()` in `java.lang.Object`. The semantics of thread synchronization are still the same as in Java.

Using the SUDPApplication Class

To facilitate programming, several template classes have been provided in J-Sim. To implement an application layer protocol, one can extend an application layer class `SUDPApplication` that has been designed to take care of many low-level details. In particular, this class provides a set of methods to send/receive a datagram. Below is a template application layer protocol which extends `SUDPApplication`.

```
import drcl.comp.Port;
import drcl.comp.Contract;

/**
 * Example application class.
 */
 public class Application extends drcl.inet.application.SUDPApplication
 {
       aPacketType packet;
       // A packet construction mechanism should be provided

       public Application ()
       {
```

```
                super();
        }
      // This method is used to send packets(Any kind of object) to the
      // mentioned node.
        sendmsg(packet, 10/*size*/, destinationAddress, destinationPort);

        protected void dataArriveAtDownPort(Object data, Port downPort)
        {
                long peerAddress = getPeerAddress(data);
                int peerPort = getPeerPort(data);
                aPacketType packet = (aPacketType)getContent(data);
           // processing methods should be located here or called from here
        }
}
```

6.5 Network Simulation Framework and Simulation Scenario Creation

INET is a network simulation framework built upon the autonomous component architecture and specific to network simulation. Essentially features common to each network component (such as an IP layer, a network interface card, a link, etc.) have been factored out and all the network components (and their contracts) are defined and implemented in INET. Internal structure of a node (either an end host or a router) is also defined. Users may then compose a network scenario in a plug-and-play fashion, by connecting components in their desired manner. Users may also subclass an appropriate component and redefine new attributes and methods to incorporate their own protocols/algorithms. To create a network simulation scenario, following items should be taken into consideration:

-Topology creation
-Building the internal structure of nodes
-Configuring the network scenario and miscellaneous issues

Figure 6.5 gives an example network and the internal structure of its nodes. As the example shows, a network is a composite component which consists of nodes, links and smaller networks. A node is also a composite component which consists of applications, protocol modules, and a core service layer (CSL).

The core service layer is an abstract component which encapsulates the functions of the network layer and the layers beneath the network layer. It provides network services and events to protocols, in the form of inter-component contracts.

With all the components available in INET, one may readily compose a network scenario of their like. Moreover, several utility classes have been

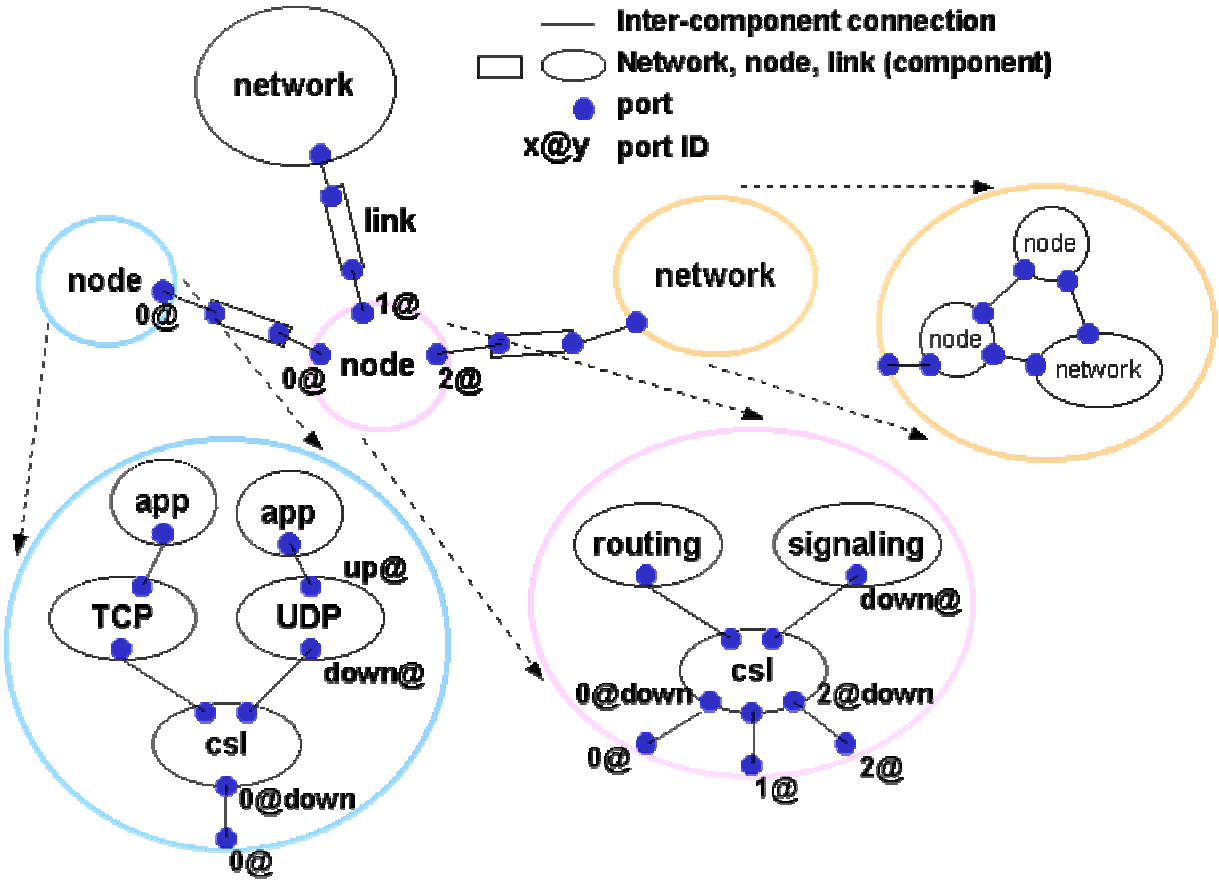

Figure 6.5 An example network scenario that can be simulated.

provided to help with scenario construction. In particular, since configuring the internal structure of nodes usually follows similar patterns and repetitively cycles through several tedious steps, some utility classes have been provided to automate the process. The basic way to compose a scenario is to build it "by hand". That is, every necessary component, from networks, nodes, links to protocols and modules inside a node is created and they are connected together afterwards. The idea is very simple but the tasks are repetitive and can get a bit tedious even for a small-sized network. Fortunately, both tasks of creating a network topology and building network nodes can be made follow certain patterns and then automate the processes. The utility class `drcl.inet.InetUtil` and the builder classes `drcl.inet.NodeBuilder` and `drcl.inet.CSLBuilder` are developed for this purpose. Building a network simulation scenario in J-Sim is outlined in the following TCL script:

```
# Create a container to hold the scenario
cd [mkdir drcl.comp.Component scene]
# Step 1: Create topology
...
# Step 2: Build nodes
...
# Step 3: Configure nodes
...
```

```
# Attach simulator runtime to "scene"
attach_simulator .
# Start all "active" components under "scene" if there is any
run .
```

In what follows, the process of creating a scenario with the utility functions in `drcl.inet.InetUtil` are introduced. Followed by that, builder classes and their usages are introduced. Finally, few other utility functions in `drcl.inet.InetUtil` are introduced (The process of building/configuring scenario by hand is not discussed here).

6.5.1 Create Topologies

There is a set of createTopology(...) methods in the `drcl.inet.InetUtil` class for automating the process of creating a topology . The simplest form of all is as follows:

```
public static void createTopology(Component network_,
                                  int[][] adjMatrix_, Link link_);
```

The network_ argument is where the nodes are to be created in. The most important argument in all the `createTopology()` methods is the adjacency matrix, `adjMatrix_`. It is a two-dimensional array. The length of the first dimension, i.e., `adjMatrix_.length`, is the number of nodes. Each element in the first dimension is a one-dimensional array, which represents the neighbors of the node. The position of a neighbor in this one-dimensional array is the ID of the port that the node uses to connect to the neighbor. Nodes are indexed as `0`, `1`,... (`adjMatrix_.length-1`). The neighbors are represented by their indices. Each node may have a different number of neighbors. `link_` is the physical link component used to connect two nodes.
The most complete form of all the `createTopolgy()` methods is the following:

```
public static void createTopology(Component network_,
                                  String routerIDPrefix_,
                                  String hostIDPrefix_,
                                  Object[] existing_,
                                  int[][] adjMatrix_,
                                  long[] ids_,
                                  Link link_,
                                  boolean assignAddress_);
```

6.5.2 Builders

After a network topology is created, the next task is to build the nodes. Note that the nodes created during the process of creating a network topology are just empty composite components. Appropriate protocols and modules need to be put in to make them functional. One way to do this, is of course building them by hand. In this section, using builder classes to automate the process is explained. The rationale behind this is very simple. Instead of building network nodes one by one by hand, nodes are first categorized. Supposedly there should be far less types of nodes in the network than the number of nodes themselves. Then a node template for each type of node is built by hand, and then the nodes of the same type are built by duplicating the structure of the template node.

Node Builder

The node builder class `drcl.inet.NodeBuilder` has a set of `build(...)` methods to build the internal structure of nodes. But before the other nodes can be built, the `NodeBuilder` itself needs to be built. Building a `NodeBuilder` is no different from building a real node (i.e., adding components and connecting them). But since the relation between the modules in the upper protocol layer (UPL) and the core service layer (CSL) is defined, the process can be further automated by having a set of port naming rules for a protocol. If a protocol follows the rules, it can be added to the `NodeBuilder` and the `NodeBuilder` will take care of the connections between the protocol and CSL. Some of `build(...)` methods of `NodeBuilder` are listed below:

```
public void build(Object c_);
public void build(Object c_, CSLBuilder cb_);
```

The second `build(...)` method, `NodeBuilder` uses CSL builder to build the internal structure of CSL. In the first form of the `build(...)` method, the default CSL builder is used. To use a different CSL implementation, one may supply the corresponding CSL builder to `NodeBuilder`. The CSL builder introduction is not provided here and information about it can be found in the simulator documentation.

6.5.3 Configuring the Network Scenario and Miscellaneous Issues

Static Routes Setup

Instead of using the node properties to manually set up static route entries along a path, `drcl.inet.InetUtil` includes a set of `setupRoutes(...)` methods

to automate the task. Given source and destination nodes, the methods compute the unicast or multicast routes with minimum hop count, and then install appropriate route entries in the nodes along the routes. The following is one of the forms of the method:

```
public static void setupRoutes(Node src_, Node dest_, String bidirect_);
```

Online Interactions

In addition to creating scenarios, running simulations often involve a lot of online activities such as debugging, tuning parameters and collecting results. A set of RUV system commands and utility components are developed to facilitate these tasks:

script

The `script` command schedules a script to be executed by the specified runtime at a specified future time instance. It is useful in constructing a complex simulation scenario by scripting. The basic form of the command is as follows:

```
script <script> -at <time> ?later? -on <simulation_runtime>
  ?-period <period>?
```

If `"later"` is present, then the script is executed at a relative future time instance. If the `period` option is present, the script schedules itself recursively in the specified period after being executed the first time at the specified time instance.

Save Results Directly to a File - drcl.comp.io.FileComponent

The `drcl.comp.io.FileComponent` component saves incoming data to a file. To use it, connect a `FileComponent` to the port at which the target component originates interested results.

xy Plot - drcl.comp.tool.Plotter

The `drcl.comp.tool.Plotter` component displays incoming data on an xy plot. The `Plotter` component is able to display multiple datasets on a plot as well as display multiple plots at the same time. Multiple plots are ordered by IDs starting from 0, so the datasets on a plot. The port ID and the port group ID of the port at which data arrives are used as the dataset ID and the plot ID respectively to draw the data on its corresponding plot. In addition to be integrated as part of the component system, `Plotter` can be used as a standalone Java Program with the following usage:

```
java drcl.comp.tool.Plotter ?-1? <file1> ?<file2>...?
```

NAM Trace - drcl.inet.tool.NamTrace

The `NamTrace` component is an instrument class that probes appropriate components to collect interested events and produce trace outputs in the NAM (VINT/UCB Network Animator) trace format. Currently, `NamTrace` supports the following NAM events/configurations: node, link, queue, color and packet. The first four events are usually used in the initial/configuration part of a trace that defines the network topology and the color index. Packet events are collected from probing appropriate components in the system. In all cases, the traces are produced at the output port of the `NamTrace` component. To save the output in a file, one must connect a file component, `drcl.comp.io.FileComponent`, to the output port of the `NamTrace` component.

With the following utility method, all the necessary configurations can be done in one line no matter how many nodes and links exist:

```
set nam [java::call drcl.inet.InetUtil setNamTraceOn [! .] \
"SimNAMTrace.nam" [_to_string_array "red blue yellow green black orange"]]
```

Here we wrap up the introduction of the relevant simulator capabilities and move on to study the developed modules and the simulation scenario in the next chapter.

Relevant links:

J-Sim Homepage: http://www.j-sim.org

Java: http://java.sun.com/

COM/COM+: http://www.microsoft.com/com/

JavaBeans: http://java.sun.com/products/javabeans/

CORBA: http://www.corba.org/

Jacl: http://tcljava.sourceforge.net/docs/website/

Perl: http://www.perl.com/

TCL: http://dev.scriptics.com/

Python: http://www.python.org/

Scriptics Corporation: http://www.scriptics.com/

TCL/Java commands: http://tcljava.sourceforge.net/docs/website/manual.html

Chapter 7

Developed Modules, Simulation Scenario & Corresponding Results

7.1 Introduction

In this chapter, as the title of the chapter suggests, we wrap up the thesis with presenting the final elements, i.e. presenting the developed modules, studying the simulation scenario and presenting the accomplished results. In Section 3, the developed SIP protocol and components are presented. Specifically, some extracts of the outputs of the Javadoc software produced from sifting through the source codes are presented. These APIs can visualize the outline of how modules really operate. In Section 4, a typical simulation scenario is analyzed and different stages of scenario construction and running are explained. In the last section, results of the aforementioned scenario are presented and discussed.

7.2 Things That Are Implemented

First, we review what is supposed to come out of this simulation and then move on to the modules details in the next section. We intend to explore whether fax parameters details can be negotiated using SIP/SDP. Specific

SDP attributes and the interaction of SIP and T.38 protocols are not of high importance in this thesis. The implemented parts are the session establishment, starting a typical file transfer, which can serve as a demonstration of T.38 fax transfer, and the subsequent session tear-down after file transfer is complete. Due to the fact that specific T.38 protocol SDP headers are not studied in this thesis, in the message content prints in the simulation results section, only few constant symbolic SDP header fields are present. As pointed out in the next chapter, studying of these fields is considered as possible future work. One other important analysis carried out in this simulation, is the utilization of SIP contact header for reducing the load on proxy servers which is a highly desirable feature.

7.3 Developed Modules

In total, six modules have been developed and they are: `SDPMessage` which is a class implementing SDP headers; `SIPMessage` which is a class implementing SIP headers and also embeds an instance of `SDPMessage` in itself if the body type indicates so; `SipPS` which is a class implementing an SIP proxy server; `SipUA` which is a class implementing an SIP user agent and finally `T38Receiver` and `T38Sender` which subclass `ftpd` and `ftp` respectively and act on behalf of real T.38 modules.

The classes implementing SDP headers and T.38 receiver and sender are some simple classes, source codes of which are provided in the appendix. Some extracts of the APIs of classes implementing SIP message, user agent and proxy server are provided here and briefly explained. The full source codes of these modules can be found in the appendix as well.

7.3.1 SIP Message Class

This class provides a mechanism for storing the SIP headers. It utilizes the `java.util.Properties` class of Java for easily setting and retrieving the SIP headers and their corresponding values. It also embeds an instance of `SDPMessage` in itself if in its constructor the type of content is set to "`application/sdp`". It also provides methods for retrieving both SIP and SDP headers as `Properties` objects to further manipulate them. Part of its API appears in Table 7.1.

Table 7.1 SIP Message Class API.

Field Summary	
`java.util.Properties`	**`headers`**

	SIP headers are held in this Properties object.
SDPMessage	**sdpMessage** An SDP message which is in the SIP message.
java.lang.String	**SIPContentType** If set to "application/sdp", an SDP message is created as an embedded object.

Constructor Summary

SIPMessage(java.lang.String contentType) Constructor.

Method Summary

java.util.Properties	**returnSDPHeaders**() This is for someone who wants to set other SDP headers as well.
java.util.Properties	**returnSIPHeaders**() This is for someone who wants to set other SIP headers as well.

7.3.2 SIP Proxy Server

This class implements the SIP proxy server. It provides some initializing methods such as: `setAddress()`, `setNodeViaField()`, `setRegisteredNode()` and `setOtherNetworkProxyServerAddress()`. These method are called with appropriate arguments during the scenario building in the TCL script. It also defines methods for sending and processing these SIP requests: ACK, INVITE and BYE. It defines a response processor which prints informational messages based on the response class and it especially handles the OK response. The method `dataArriveAtDownPort()` handles the incoming data and directs it to the appropriate processor. Two utility methods, `constructMessage()` and `printMessageContent()`, are also provided and they carry out tasks described by their names. User agent registers itself with the proxy server during the initialization process through the TCL script. The transaction ID of the first received SIP message, its `Call-ID` header, is stored in the proxy server so the server can discard messages not belonging to this transaction. The rest of the details about the proxy server can be found in the following API or its full source code in the appendix.

Table 7.2 SIP Proxy Server API.

Field Summary	
java.lang.String	**contentType** A variable used for setting the content type of the SIP message, typically set to "application/sdp".
java.lang.String	**messageType** An intermediate variable which is used for checking whether a message is a request or it is a response and directing it to the related processor.
int	**nodeAddress** Node address is set during initialization through the TCL interface.
java.lang.String	**nodeViaField** It is set during initialization through the TCL interface.
int	**otherNetworkProxyServerAddress** It is set during initialization through the TCL interface.
SIPMessage	**receivedMessage** An intermediate variable which is used for processing.
int	**registeredNode** All requests of the set node first goes to this proxy server and it is set during initialization through the TCL interface.
int	**responseMessageClass** A vaiable to store the response class from one of six possible classes.
SIPMessage	**toBeSentMessage** This is a message which is created by different methods of the class.
java.lang.String	**transactionID** Used for storing the transaction ID so that junk messages can be discarded.

Constructor Summary	
SipPS() Constructor.	

Method Summary	
SIPMessage	**constructMessage**(java.lang.String startLine, java.lang.String via, java.lang.String to, java.lang.String from, java.lang.String callID, java.lang.String contentType)
protected void	**dataArriveAtDownPort**(java.lang.Object data,

	drcl.comp.Port downPort) Arrived data first gets processed by this method.
void	**duplicate**(java.lang.Object source)
java.lang.String	**info**()
void	**printMessageContent**(SIPMessage message)
void	**processACK**(SIPMessage receivedMessage)
void	**processBYE**(SIPMessage receivedMessage)
void	**processINVITE**(SIPMessage receivedMessage)
void	**processResponse**(SIPMessage receivedMessage) This method first checks to see whether the message is a valid one then checks to see if it's a response or an unsupported request, after that if the message is a response it goes on to handle each type of response classes.
void	**reset**()
void	**sendACK**(SIPMessage receivedMessage, int nextHop)
void	**sendBYE**(SIPMessage toBeSentMessage, int nextHop)
void	**sendINVITE**(SIPMessage toBeSentMessage, int nextHop)
void	**sendOK**(SIPMessage receivedMessage, int address)
void	**setAddress**(int address)
void	**setNodeViaField**(java.lang.String s)
void	**setOtherNetworkProxyServerAddress**(int address)
void	**setRegisteredNode**(int address)

7.3.3 SIP User Agent

This class implements the SIP user agent. Like proxy server, it provides some initializing methods: `setAddress()`, `setConfiguredProxyServerAddress()`, `setNodeViaField()` and `setAlwaysUseProxyServer()`. The last method sets the user agents to always use the proxy servers and never bypass them and

contact each other directly. The default is false which means user agents, whenever they can, contact each other directly using the address found in the received SIP message Contact header. The same set of sending and processing requests and responses methods, found in proxy servers, are present here as well with some modifications. Some other methods which were in the proxy server class, as can be seen in the API, are also present here. There is also a field call `faxPort` which is used to alert the T.38 module to start sending the fax after the session establishment is complete. Again, the rest of the details about the user agent can be found in the following API or its full source code in the appendix.

Table 7.3 SIP User Agent API.

Field Summary	
boolean	**alwaysUseProxyServer** It is set during initialization through the TCL interface.
int	**configuredProxyServerAddress** All requests of the node first goes to this address which is set during initialization through the TCL interface.
java.lang.String	**contentType** A variable used for setting the content type of the SIP message, typically set to "application/sdp".
int	**destination** Used for storing the other party's address.
drcl.comp.Port	**faxPort** Used for alerting the T.38 fax module to start sending the fax.
java.lang.String	**messageType** An intermediate variable which is used for checking whether a message is a request or it is a response and directing it to the related processor.
int	**nodeAddress** Node address is set during initialization through the TCL interface.
java.lang.String	**nodeViaField** It is set during initialization through the TCL interface.
SIPMessage	**receivedMessage** An intermediate variable which is used for processing.
int	**responseMessageClass** A vaiable to store the response class from one of six possible classes.
SIPMessage	**toBeSentMessage** This is a message which is created by different methods of the class.
java.lang.String	**transactionID**

	Used for storing the transaction ID so that junk messages can be discarded.
boolean	**transactionInitiator** Used for determining if the node should respond like acknowledging an OK with ACK only if the node is indeed the initiator of the request-response.

Constructor Summary

SipUA() Constructor.

Method Summary

SIPMessage	**constructMessage**(java.lang.String startLine, java.lang.String via, java.lang.String to, java.lang.String from, java.lang.String callID, java.lang.String contentType)
protected void	**dataArriveAtDownPort**(java.lang.Object data, drcl.comp.Port downPort) Arrived data first gets processed by this method.
void	**duplicate**(java.lang.Object source)
java.lang.String	**info**()
void	**printMessageContent**(SIPMessage message)
void	**processACK**(SIPMessage receivedMessage)
void	**processBYE**(SIPMessage receivedMessage)
void	**processINVITE**(SIPMessage receivedMessage)
void	**processResponse**(SIPMessage receivedMessage) This method first checks to see whether the message is a valid one then checks to see if it's a response or an unsupported request, after that if the message is a response it goes on to handle each type of response classes.
void	**reset**()
void	**sendACK**(SIPMessage receivedMessage)
void	**sendBYE**()

void	**sendINVITE**(SIPMessage toBeSentMessage)
void	**sendOK**(SIPMessage receivedMessage)
void	**setAddress**(int address)
void	**setAlwaysUseProxyServer**(boolean x)
void	**setConfiguredProxyServerAddress**(int address)
void	**setNodeViaField**(java.lang.String s)

7.4 Simulation Scenario

As pointed out in the previous chapter, J-Sim simulator uses TCL scripts to carry out scenario building and configuration. In this section the TCL scripts used to do such tasks are analyzed line by line. In the first part, we analyze the script used for building the simulation scenario and in the next part, the script used for running the simulation is presented.

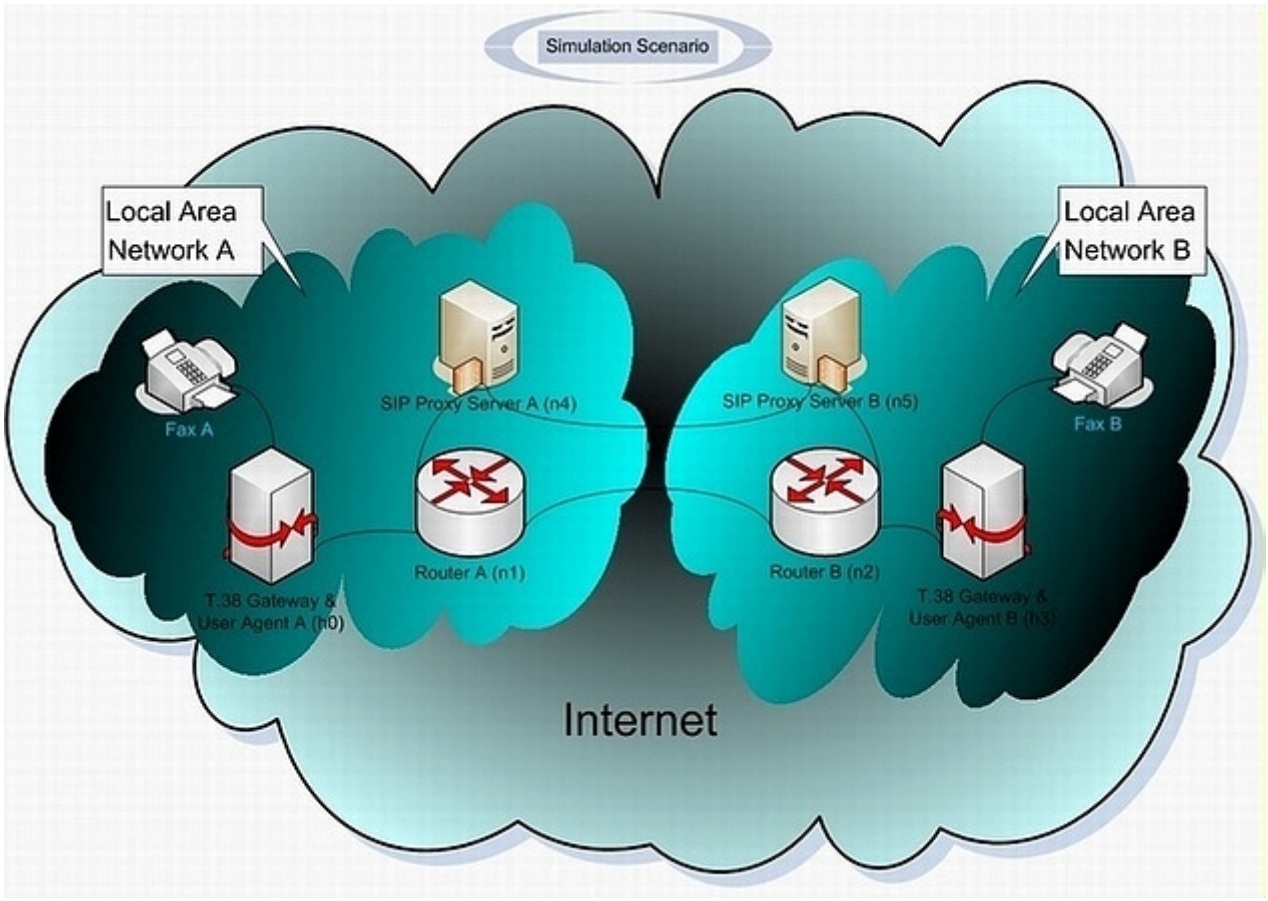

Figure 7.1 The Simulation Scenario.

7.4.1 The Network Topology

The simulated network scenario is depicted in Figure 7.1.

7.4.2 Scenario Building

```
1 ##########################################################
2 # Topology:
3 #
4 #            -------------------------
5 #           /                         \
6 # h0-------n1-------n4-------n5-------n2-------h3
7 #
8 ##########################################################
9 cd [mkdir drcl.comp.Component /SimScenario]
10
11 puts "Creating topology..."
12    set link [java::new drcl.inet.Link]
13    $link setPropDelay 0.3; # 300ms
14    set adjMatrix [java::new {int[][]} 6 {{1} {0 2 4} {5 1 3} {2} {1 5}
{4 2}}]
15    java::call drcl.inet.InetUtil createTopology [! .] $adjMatrix $link
16
17 puts "Creating builders..."
18 # router builder:
19    set rb [mkdir drcl.inet.NodeBuilder .routerBuilder]
20    $rb setBandwidth 1.0e6; #1Mbps
21 # Host builder:
22    set hb [cp $rb .hostBuilder]
23 #Setting up transport protocols
24    set TCPModule [mkdir drcl.inet.transport.TCPb $hb/tcp]
25    set UDPModule [mkdir drcl.inet.transport.UDP $hb/udp]
26
27 # Adding a Data Counter to each host:
28    mkdir drcl.comp.tool.DataCounter $hb/counter
29
30 #Setting up T.38 Fax Senders/Receivers:
31
32    $TCPModule addPort "up" "t38fax"
33    set T38FaxReceiver [mkdir mkh.sip.T38Receiver $hb/t38FaxReceiver]
34    connect -c $hb/t38FaxReceiver/down@ -and $hb/tcp/up@
35
36    set T38FaxSender [mkdir mkh.sip.T38Sender $hb/t38FaxSender]
37    connect -c $hb/t38FaxSender/down@ -and $hb/tcp/t38fax@up
38
39 puts "Building Nodes ..."
40    $rb build [! n?]
41    $hb build [! h?]
42
43 #Setting up UAs:
44    set SipUA1 [mkdir mkh.sip.SipUA h0/ua]
45    set SipUA2 [mkdir mkh.sip.SipUA h3/ua]
46    $SipUA1     setName "SIP User Agent 1"
47    $SipUA2     setName "SIP User Agent 2"
48    $SipUA1     setAddress 0
49    $SipUA2     setAddress 3
50    $SipUA1     setNodeViaField "SIP/2.0/UDP here.com:5060"
51    $SipUA2     setNodeViaField "SIP/2.0/UDP there.com:5060"
52
```

```
53 #  $SipUA1 setAlwaysUseProxyServer true
54 #  $SipUA2 setAlwaysUseProxyServer true
55
56    connect -c h0/ua/down@ -and h0/udp/5060@up
57    connect -c h3/ua/down@ -and h3/udp/5060@up
58
59 # Fax port configuration:
60
61    connect -c h3/ua/faxPort@down -to h0/t38FaxSender/faxPort@down
62    connect -c h3/ua/faxPort@down -to h3/t38FaxReceiver/faxPort@down
63
64 # For sending a BYE request after fax transfer completes:
65    connect -c h3/t38FaxReceiver/notify@ -to h3/ua/down@
66
67 #Setting up proxy servers:
68    set SipPS1 [mkdir mkh.sip.SipPS n4/ps]
69    set SipPS2 [mkdir mkh.sip.SipPS n5/ps]
70    $SipPS1     setName "SIP Proxy Server 1"
71    $SipPS2     setName "SIP Proxy Server 2"
72    $SipPS1     setAddress 4
73    $SipPS2     setAddress 5
74    mkdir drcl.inet.transport.UDP n4/udp
75    mkdir drcl.inet.transport.UDP n5/udp
76    connect -c n4/ps/down@ -and n4/udp/5060@up
77    connect -c n5/ps/down@ -and n5/udp/5060@up
78
79    $SipUA1 setConfiguredProxyServerAddress 4
80    $SipUA2 setConfiguredProxyServerAddress 5
81    $SipPS1 setOtherNetworkProxyServerAddress 5
82    $SipPS2 setOtherNetworkProxyServerAddress 4
83    $SipPS1 setRegisteredNode 0
84    $SipPS2 setRegisteredNode 3
85    $SipPS1     setNodeViaField "SIP/2.0/UDP SIP Proxy Server 1:5060"
86    $SipPS2     setNodeViaField "SIP/2.0/UDP SIP Proxy Server 2:5060"
87
88 # Configure the bottleneck bandwidth and buffer size
89    ! n1 setBandwidth 1 1.0e5; # 100Kbps at interface 1
90    ! n1 setBufferSize 1 6000; # ~10 packets at interface 1
91
92 puts "Setting up static routes..."
93    java::call drcl.inet.InetUtil setupRoutes [! h0] [! h3]
"bidirection"
94    java::call drcl.inet.InetUtil setupRoutes [! h0] [! n4]
"bidirection"
95    java::call drcl.inet.InetUtil setupRoutes [! n4] [! n5]
"bidirection"
96    java::call drcl.inet.InetUtil setupRoutes [! n5] [! h3]
"bidirection"
97    ! h0/tcp setPeer 3
98    ! h3/tcp setPeer 0
99 #  Realistically, these should be set from "destination" fields of UAs
100 puts "Creating The Plotters..."
101   set plot1 [mkdir drcl.comp.tool.Plotter .plot1]
102   set file1 [mkdir drcl.comp.io.FileComponent .file1]
103   $file1 open "SimPlot.plot"
104   connect -c $plot1/.output@ -to $file1/in@
105
106   attach -c $plot1/0@0 -to h0/tcp/cwnd@
107   attach -c $plot1/3@0 -to h3/tcp/cwnd@
108
109   set tm1 [mkdir drcl.net.tool.TrafficMonitor .tm1]
```

```
110   connect -c h3/csl/6@up -to $tm1/in@
111   connect -c $tm1/bytecount@ -to $plot1/3@1
112
113   attach -c $plot1/3@2 -to h3/tcp/rcv/seqno@
114   attach -c $plot1/0@2 -to h0/tcp/rcv/seqno@
115
116   attach -c h0/counter/in@ -to h0/csl/6@up
117   attach -c h3/counter/in@ -to h3/csl/6@up
118
119   puts "Setting up the NAM trace..."
120   set nam [java::call drcl.inet.InetUtil setNamTraceOn [! .] \
"SimNAMTrace.nam" [_to_string_array "red blue black orange"]]
121
122 puts "Building the scenario is finished now, you can proceed with
running the simulation ..."
```

Analyses:

-Line 9: Creating the network element itself to hold the network components.
-Lines 12-13: Creating and configuring the link element.
-Line 14: Creating the adjacency matrix; the matrix which defines the topology by determining the neighboring nodes of each node.
-Line 15: Creating the network by calling `createTopology()` and passing the network component, adjacency matrix and link component as arguments.
-Lines 19-20: Creating a typical node builder and naming it `rb`.
-Line 22: Duplicating the `rb` object and naming it `hb` for building the hosts after further customizing the host builder (`hb`).
-Lines 24-25: Creating the transport protocol modules (The UDP module is used for signaling and the TCP module is used for the fax data transfer).
-Line 28: Putting a data packet counter in each of the hosts.
-Lines 32-37: Creating T.38 sender and receiver modules and putting them in the host builder and connecting the modules to the TCP module.
-Lines 40-41: Building the hosts and routers using the previously configured host builder and node builder.
-Lines 44-51: Creating the SIP user agent modules in the hosts and configuring their names, addresses and `nodeViaField` parameters.
-Lines 53-54: Setting whether the user agents always have to use the proxy servers for signaling or they are allowed to bypass them if they can.
-Lines 56-57: Configuring the SIP modules to use UDP as transport.
-Lines 61-65: Setting a mechanism for auto-starting the fax transfer immediately after signaling is complete and also auto-starting the signaling (the last `BYE-OK`) after the fax transfer is complete.
-Lines 68-77: Creating SIP proxy server modules in the nodes and configuring their names and addresses. The UDP modules are also setup in the nodes and proxy server modules are set to use them for transport.

-Lines 79-86: Further customizing the SIP modules: Setting user agents' configured proxy server addresses, making each proxy server know the other network's proxy server, setting each proxy server's registered node and setting the `nodeViaField` of each proxy server.
-Lines 89-90: A sample bottleneck is incorporated in the node 1 to examine its effect on fax data transfer parameters.
-Lines 93-98: Setting up the nodes routing table entries and TCP peers.
-Lines 101-117: Creating the plotters and data counter instruments. Plotters monitor fax data throughput, congestion window and received packets sequence numbers. The parameters are shown in real-time manner during the simulation and since the plots are saved into file, they can be seen and examined later as well.
-Line 120: The nodes are configured to output packets traces in the format understandable by Network Animator.

7.4.3 Scenario Running

```
1 puts ""
2 puts "Starting the Simulation..."
3     set sim [attach_simulator .]
4
5 puts "Fax initiation..."
6 puts "Negotiating the fax parameters with the other party..."
7
8 ##########################################
9 #   Fax Signaling and sending
10 ##########################################
11
12 set message [$SipUA1 constructMessage "INVITE" "SIP/2.0/UDP
here.com:5060" "sip: user@there.com" "0" 1234@here.com "application/sdp"]
13
14 $SipUA1 sendINVITE $message
15
16
17
18
19 ##########################################
20 ## SIP message constructing method syntax:
21 ##
22 ## SIPMessage constructMessage(String startLine, String via, String
to, String from, String ##callID, String contentType)
23 ##########################################
24
25 ##########################################
26 ## Further SIP message headers customization can be done using this
syntax:
27 ##
28 ## set hd [$message returnSIPHeaders]
29 ## set SDPhd [$message returnSDPHeaders]
30 ## puts "SIP header (Call-ID):"
31 ## $hd getProperty "Call-ID"
32 ## puts "SDP header (v):"
33 ## $SDPhd getProperty "v"
```

```
34 ## $hd setProperty "CSeq" "1"
35 ########################################
36
37 ########################################
38 ## SIP message sending method syntax:
39 ##
40 ## sendINVITE(SIPMessage toBeSentMessage)
41 ########################################
42
43
44 ### The command for viewing the plots:
45 ### java drcl.comp.tool.Plotter SimPlot.plot
```

Analyses:

Line 3: Attaching simulation run-time to the network.
Line 12: Creating an SIP message in user agent 1 using its `constructMessage()` method according to the syntax given in line 22.
Line 14: The constructed message is then sent using the `sendINVITE()` method of user agent 1 (The syntax is given in line 40).
Lines 19-45: These are provided to serve as reference and because each line is prefixed with a #, it doesn't get executed.

7.5 Simulation Results

7.5.1 The Simulated SIP Call Flow

Please note that the generation and processing of informational class responses have not been simulated. The simulated call flow is depicted in Figure 7.2.

7.5.2 Terminal Output

As shown in the terminal output following Figure 7.2, after building and configuring the network, the simulation starts by user agent 1 sending an INVITE request. From there, one can track the call flow from the terminal output and also from Figure 7.2. As can be seen, user agents contact each other directly after knowing each other's address from the SIP contact header.

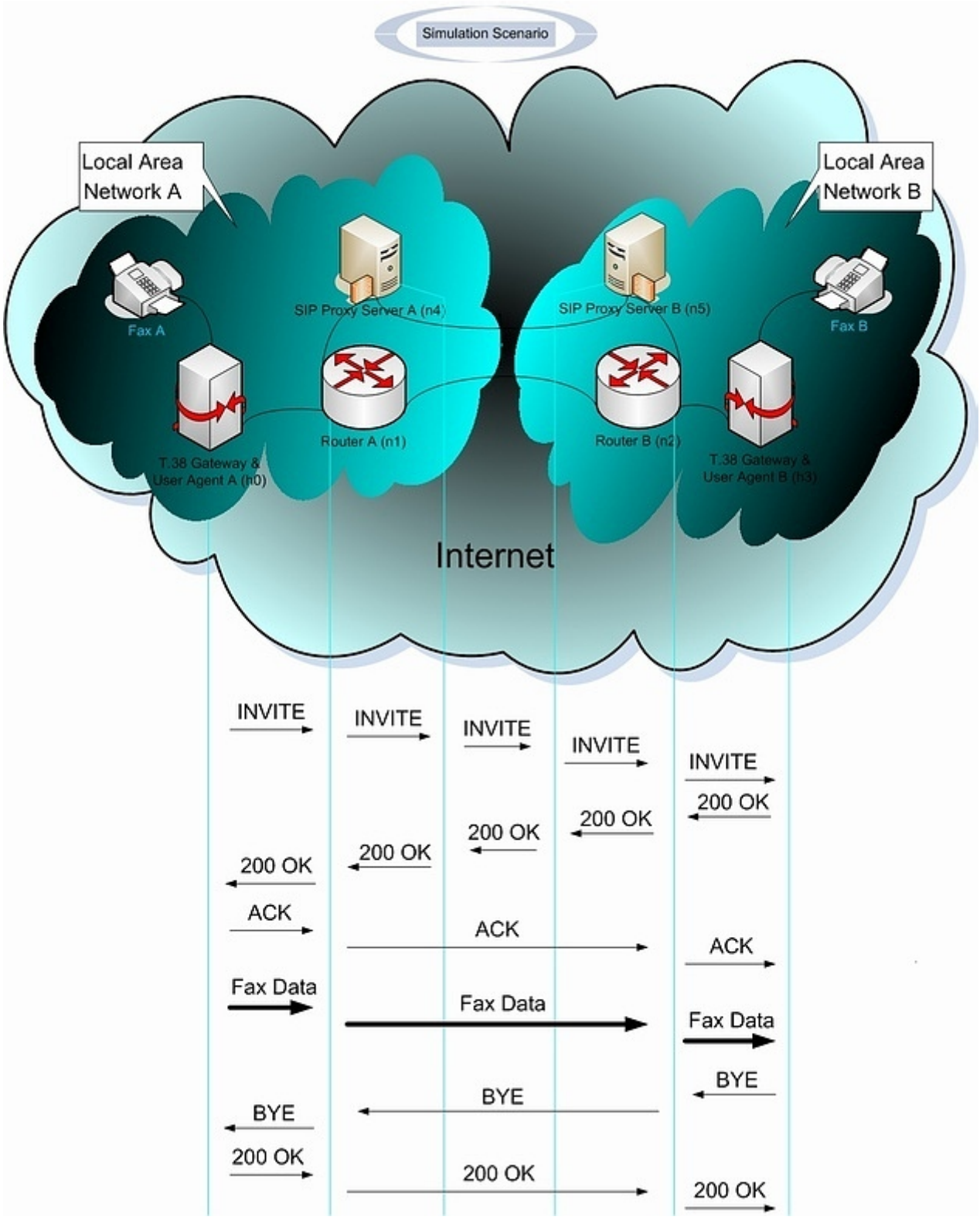

Figure 7.2 The Simulated SIP Call Flow.

```
Creating topology...
Creating builders...
Building Nodes ...
Setting up static routes...
Creating The Plotters...
Setting up the NAM trace...
Building the scenario is finished now, you can proceed with running the
simulation ...

Starting the Simulation...
Fax initiation...
Negotiating the fax parameters with the other party...

An SIP message is being created ...
An SDP body is being created ...
The SIP message is being sent now by SIP User Agent 1

An SIP message has been received and is being processed by SIP Proxy Server 1
The following INVITE request has been received by the request processor...

The message content:
Start-Line: INVITE
Request-URI: sip: user@there.com
Via: SIP/2.0/UDP here.com:5060
Via 1: Not Set
Via 2: Not Set
To: sip: user@there.com
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 0
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

The message is being forwarded to the other network's proxy server now...
The SIP message is being sent now by SIP Proxy Server 1

An SIP message has been received and is being processed by SIP Proxy Server 2
The following INVITE request has been received by the request processor...

The message content:
Start-Line: INVITE
Request-URI: sip: user@there.com
Via: SIP/2.0/UDP here.com:5060
Via 1: SIP/2.0/UDP SIP Proxy Server 1:5060
Via 2: Not Set
To: sip: user@there.com
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 0
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

The message is being forwarded to the intended node now...
The SIP message is being sent now by SIP Proxy Server 2

An SIP message has been received and is being processed by SIP User Agent 2
The following INVITE request has been received by the request processor...

The message content:
Start-Line: INVITE
Request-URI: sip: user@there.com
Via: SIP/2.0/UDP here.com:5060
Via 1: SIP/2.0/UDP SIP Proxy Server 1:5060
```

```
Via 2: SIP/2.0/UDP SIP Proxy Server 2:5060
To: sip: user@there.com
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 0
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

The response of that is being issued...
An OK message is being sent now by SIP User Agent 2

An SIP message has been received and is being processed by SIP Proxy Server 2
This is a success class response.

The message content:
Start-Line: 200
Request-URI:
Via: SIP/2.0/UDP here.com:5060
Via 1: SIP/2.0/UDP SIP Proxy Server 1:5060
Via 2: SIP/2.0/UDP SIP Proxy Server 2:5060
To: sip: user@there.com
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 3
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

The message is being forwarded to the other network's proxy server now...
An OK message is being sent now by SIP Proxy Server 2

An SIP message has been received and is being processed by SIP Proxy Server 1
This is a success class response.

The message content:
Start-Line: 200
Request-URI:
Via: SIP/2.0/UDP here.com:5060
Via 1: SIP/2.0/UDP SIP Proxy Server 1:5060
Via 2: Removed
To: sip: user@there.com
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 3
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

The message is being forwarded to the intended node now...
An OK message is being sent now by SIP Proxy Server 1

An SIP message has been received and is being processed by SIP User Agent 1
This is a success class response.

The message content:
Start-Line: 200
Request-URI:
Via: SIP/2.0/UDP here.com:5060
Via 1: Removed
Via 2: Removed
To: sip: user@there.com
From: 0
```

```
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 3
Content-Type: application/sdp
SDP fields:
SDP Protocol Version Number: RFC 2327
SDP Protocol Session Information: T.38 fax description

An ACK message is being sent now by SIP User Agent 1

An SIP message has been received and is being processed by SIP User Agent 2
An ACK message has been received...

The message content:
Start-Line: ACK
Request-URI: 3
Via: SIP/2.0/UDP here.com:5060
Via 1: Not Set
Via 2: Not Set
To: 3
From: 0
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 0
Content-Type: No Body

Starting the T.38 fax transmission...


Fax reception is complete now...

An SIP message is being created ...
A BYE request is being sent now by SIP User Agent 2

An SIP message has been received and is being processed by SIP User Agent 1
The following BYE request has been received by the request processor...

The message content:
Start-Line: BYE
Request-URI: 0
Via: SIP/2.0/UDP there.com:5060
Via 1: Not Set
Via 2: Not Set
To: 0
From: 3
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 3
Content-Type: No SDP Body

The response of that is being issued...
An OK message is being sent now by SIP User Agent 1

An SIP message has been received and is being processed by SIP User Agent 2
This is a success class response.

The message content:
Start-Line: 200
Request-URI:
Via: SIP/2.0/UDP there.com:5060
Via 1: Not Set
Via 2: Not Set
To: 0
From: 3
Call-ID: 1234@here.com
Subject: A Fax session
Contact: 0
Content-Type: No SDP Body
```

7.5.3 Packets Traces Analyzed With Network Animator

The simulation's packets traces output file is opened using NAM. A time instance when fax data is being sent directly from user agent 1 to user agent 2, bypassing proxy servers, is shown in Figure 7.3.

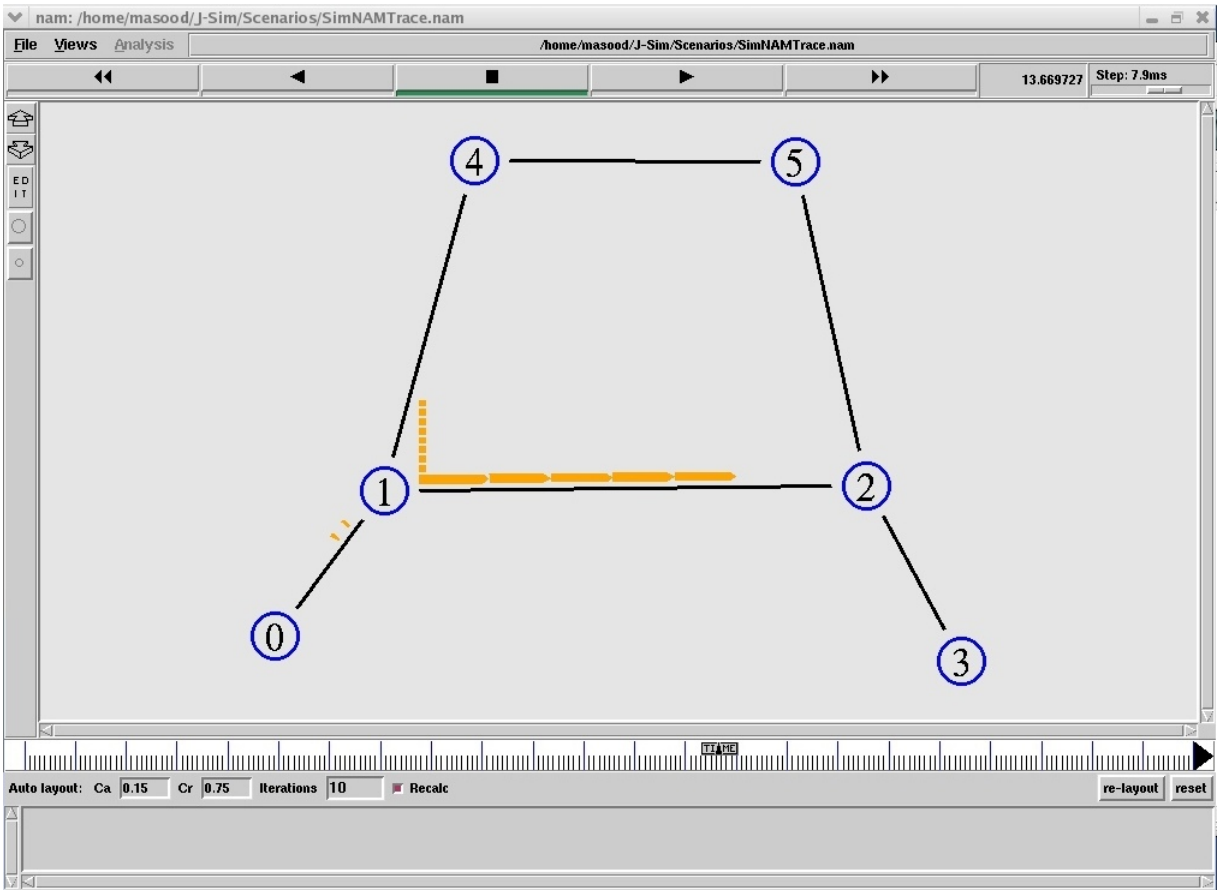

Figure 7.3 Packets Traces Analyzed With Network Animator.

7.5.4 Possible Fax Data Transfer Analyses

Fax data analysis is not a concern of this thesis and the carried out simulation. But for depicting the types of possible analyses which can be done using J-Sim, as pointed out in the scenario building TCL script lines 89-90, a bottleneck is incorporated in node 1 and its effect in fax data transfer is shown in the following figures:

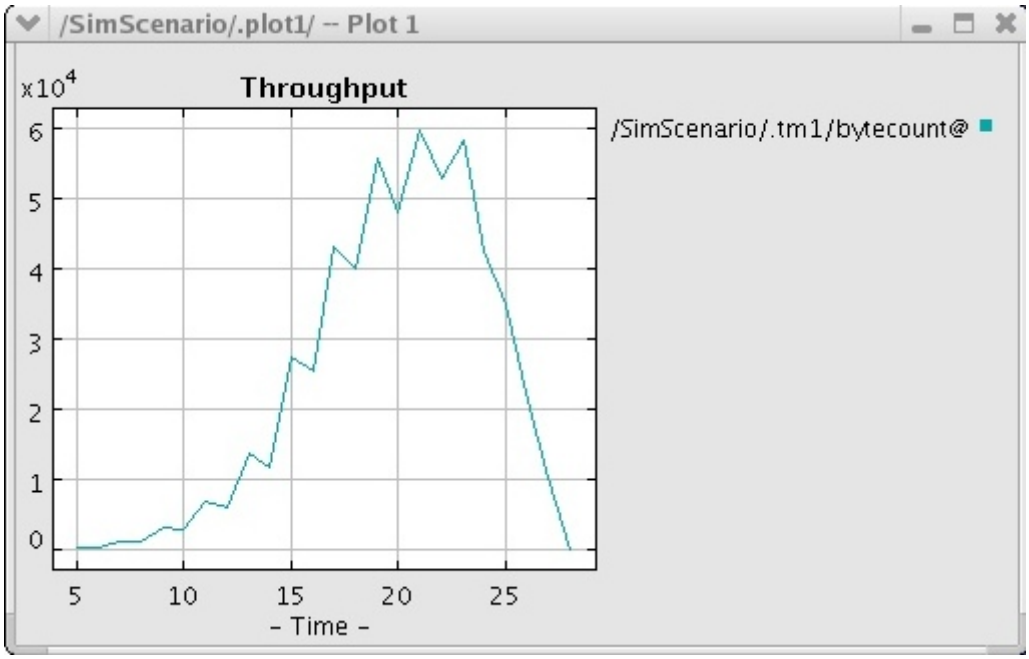

Figure 7.4 Throughput.

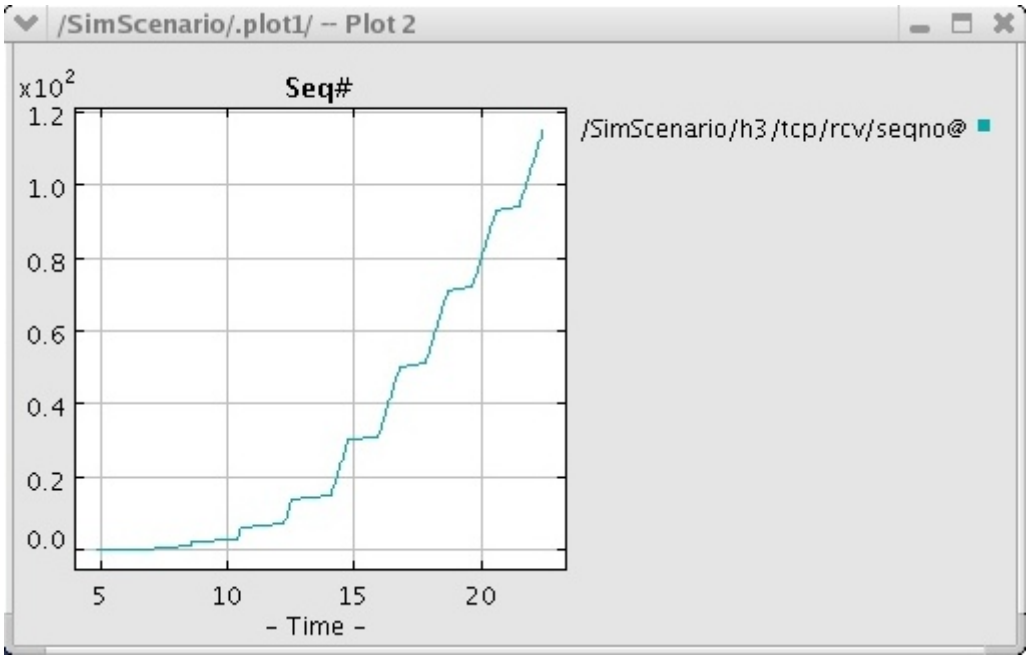

Figure 7.5 Received Data Packets Sequence Number.

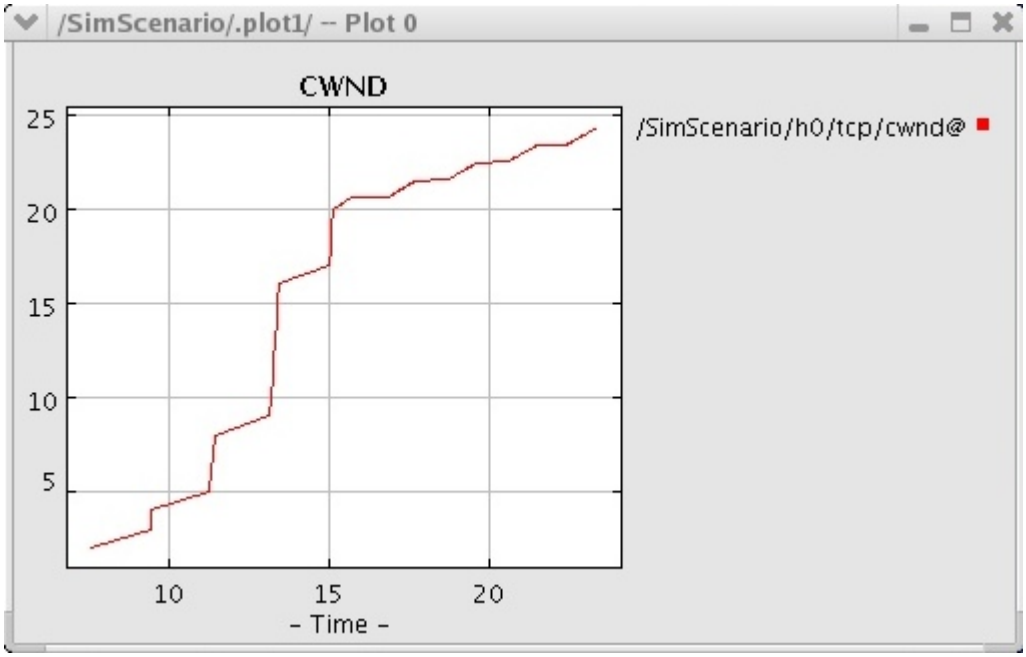

Figure 7.6 Congestion Window.

Here the discussion regarding simulation scenario building and configuration and corresponding analyses comes to an end. In the next chapter a summary of all the important points of this thesis including this chapter is given.

Chapter 8

Summary & Concluding Remarks

8.1 Conclusions

What we intended to do was investigating whether fax parameters details could be negotiated using SIP/SDP. In this direction, session establishment, starting a typical file transfer, which served as a demonstration of T.38 fax transfer, and the subsequent session tear-down, after file transfer was complete, were demonstrated in this thesis, through computer simulation. Call flow could be tracked and analyzed using the terminal output and also the simulation schematic diagram; the former was in complete agreement with the claimed results regarding the SIP capabilities.

Based on the investigated simulation scenario, we showed that SIP nicely lends itself to the task. Although only few constant symbolic SDP fields were used and hence present in the terminal output, but as mentioned before, the simulation results provided a good starting point for the full definition and usage of SDP attribute fields to thoroughly specify the T.38 fax parameters. This simulation scenario and its results exhibited the potential success of the proposed SIP/SDP combination for real-time fax session establishment, management and tear-down.

Another important analysis carried out in this simulation was utilization of SIP contact header for reducing the load on proxy servers which is a highly desirable feature. As shown in the terminal output of the simulator, after building and configuring the network, the simulation started by user agent 1 sending an INVITE request. From there, the call flow could be tracked from the terminal output and as could be seen, user agents contacted each other directly after knowing each other's address from the SIP contact header.

8.2 Summary

In Chapter 2, we introduced and discussed fundamental concepts related to this thesis. We first started by inspecting different switching and networking modes which paved the way for better understanding the differences between the PSTN and the Internet and then moved forward to study PSTN more closely.

In Chapter 3, data networks were investigated. We started by discussing some basic issues and then moved forward to talk briefly about local/wide area networks. Internet Protocol and transport layer protocols were also discussed.

In Chapter 4, we introduced and discussed the concepts directly related to this thesis: Voice/Fax over IP architectures and standards. At first, some fundamental issues related to IP telephony were explored. Then, call signaling protocols and both real-time and non-real-time fax over IP were examined. IP telephony's QoS were also discussed. Finally, some statistics related to current IP telephony market were presented.

In Chapter 5, the Session Initiation Protocol was treated thoroughly. SIP user agents, gateways and the 3 types of servers were discussed. A brief introduction of SIP request and response messages and headers was then given. Session Description Protocol (SDP), a companion protocol to SIP, was treated as well. A comprehensive introduction to SIP programming was also given. At the end, SIP and T.38 interactions were explained briefly.

In Chapter 6, J-Sim, an open-source simulator, was introduced. This simulator has been utilized throughout the implementation part of this thesis to explore the behavior of the developed protocol and components. To start familiarizing the reader with the simulator, salient features of it were discussed at first. A quick overview of the inner workings of the simulator and how one could develop new modules were given as well. Simulation scenario creation, configuration and running were briefly introduced in the last section of the chapter.

In Chapter 7, we presented the developed modules, studied the simulation scenario and presented the accomplished results. First, the developed SIP protocol and components were presented. Specifically, some extracts of the outputs of the Javadoc software produced from sifting through the source codes were presented. A typical simulation scenario was analyzed and different stages of scenario construction and running were explained. In the last section of the chapter, results of the simulation scenario were presented and discussed.

8.3 Possible Future Works

Specific T.38 protocol SDP attributes have not been studied in this thesis. As a natural next step, if the thorough investigation of real-time fax is contemplated, one can refer to this IETF Internet-Draft: “SIP Support for Real-time Fax: Call Flow Example And Best Current Practices” [36], which can serve as a very good starting point.

Addition of few more capabilities to the proxy server can be a good proposition as well. Additional capabilities can be: handling multiple transactions at the same time, stateful operation of the proxy server, having location server and multiple registered nodes, implemented either in the proxy server code itself or in separate entities, to name a few.

Appendix: Source Codes

SIP Message Class

```
package mkh.sip;

import java.util.Properties;

/**
 * This class is a Session Initiation Protocol (SIP) Message.
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class SIPMessage
{
      /** SIP headers are held in this Properties object. */
      public Properties headers;
      /** An SDP message that is contained in the SIP message. */
      public SDPMessage sdpMessage;
      /** If set to "application/sdp" an SDP message is created as an
embedded object. */
      public String SIPContentType;

      /** Constructor. */
      public SIPMessage(String contentType) {

            SIPContentType=contentType;
            System.out.println("\n"+"An SIP message is being created ...");

            headers= new Properties();
            if (contentType.equals("application/sdp"))
            {
                  headers.setProperty("Content-Type", "application/sdp");

                  sdpMessage = new SDPMessage();
            }
            else headers.setProperty("Content-Type", "No SDP Body");
```

```
            headers.setProperty("Start-Line", "Not Set");
            headers.setProperty("Request-URI", "Not Set");

// General Headers
            headers.setProperty("Call-ID", "Not Set");
            headers.setProperty("Contact", "Not Set");
            headers.setProperty("CSeq", "Not Set");
            headers.setProperty("Date", "Not Set");
            headers.setProperty("Encryption", "Not Set");
            headers.setProperty("From", "Not Set");
            headers.setProperty("Organization", "Not Set");
            headers.setProperty("Retry-After", "Not Set");
            headers.setProperty("Subject", "Not Set");
            headers.setProperty("Supported", "Not Set");
            headers.setProperty("Timestamp", "Not Set");
            headers.setProperty("To", "Not Set");
            headers.setProperty("User Agent", "Not Set");
            headers.setProperty("Via", "Not Set");
            headers.setProperty("Via 1", "Not Set");
            headers.setProperty("Via 2", "Not Set");
// Request Headers
            headers.setProperty("Accept", "Not Set");
            headers.setProperty("Accept-Contact", "Not Set");
            headers.setProperty("Accept-Encoding", "Not Set");
            headers.setProperty("Accept-Language", "Not Set");
            headers.setProperty("Authorization", "Not Set");
            headers.setProperty("Hide", "Not Set");
            headers.setProperty("In-Reply-To", "Not Set");
            headers.setProperty("Max-Forwards", "Not Set");
            headers.setProperty("Priority", "Not Set");
            headers.setProperty("Proxy-Authorization", "Not Set");
            headers.setProperty("Proxy-Require", "Not Set");
            headers.setProperty("Record-Route", "Not Set");
            headers.setProperty("Reject-Contact", "Not Set");
            headers.setProperty("Request-Disposition", "Not Set");
            headers.setProperty("Require", "Not Set");
            headers.setProperty("Response-Key", "Not Set");
            headers.setProperty("Route", "Not Set");
            headers.setProperty("RAck", "Not Set");
            headers.setProperty("Session-Expires", "Not Set");
// Reponse Headers
            headers.setProperty("Proxy-Authenticate", "Not Set");
            headers.setProperty("Server", "Not Set");
            headers.setProperty("Unsupported", "Not Set");
            headers.setProperty("Warning", "Not Set");
            headers.setProperty("WWW-Authenticate", "Not Set");
            headers.setProperty("RSeq", "Not Set");
// Entity Headers
            headers.setProperty("Allow", "Not Set");
            headers.setProperty("Content-Encoding", "Not Set");
            headers.setProperty("Content-Disposition", "Not Set");
            headers.setProperty("Content-Length", "Not Set");
            headers.setProperty("Expires", "Not Set");
            headers.setProperty("MIME-Version", "Not Set");
      }

      /** This is for someone who wants to set other SIP headers as well. (
printMessageContent() function in User Agent and Proxy Server should be
amended accordingly) */
      public Properties returnSIPHeaders()
      {
            return headers;
      }

      /** This is for someone who wants to set other SDP headers as well. (
printMessageContent() function in User Agent and Proxy Server should be
amended accordingly) */
```

```
	public Properties returnSDPHeaders()
	{
		if(SIPContentType.equals("application/sdp"))
		{
			return sdpMessage.sdpFields;
		}else return null;
	}

}
```

SDP Message Class

```
package mkh.sip;

import java.util.Properties;

/**
 * This class is a Session Description Protocol (SDP) Message.
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class SDPMessage
{

	/** SDP headers are held in this Properties object. */
	protected Properties sdpFields;

	/** Constructor. */
	public SDPMessage() {

		System.out.println("An SDP body is being created ...");

		sdpFields = new Properties();

		// Protocol Version Number
		sdpFields.setProperty("v", "RFC 2327");
		// Owner/Creator and session identifier
		sdpFields.setProperty("o", "Not Set");
		// Session name
		sdpFields.setProperty("s", "Not Set");
		// Session information
		sdpFields.setProperty("i", "T.38 fax description");
		// Uniform resource indentifier
		sdpFields.setProperty("u", "Not Set");
		// Email address
		sdpFields.setProperty("e", "Not Set");
		// Phone number
		sdpFields.setProperty("p", "Not Set");
		// Connection information
		sdpFields.setProperty("c", "Not Set");
		// Bandwidth information
		sdpFields.setProperty("b", "Not Set");
		// Time session starts and stops
		sdpFields.setProperty("t", "Not Set");
		// Repeat times
		sdpFields.setProperty("r", "Not Set");
		// Time zone corrections
		sdpFields.setProperty("z", "Not Set");
		// Encryption key
		sdpFields.setProperty("k", "Not Set");
		// Attribute lines
		sdpFields.setProperty("a", "Not Set");
		// Media information
		sdpFields.setProperty("m", "Not Set");
```

```
            // Attribute lines (2nd)
            sdpFields.setProperty("a2", "Not Set");
            // Media information (2nd)
            sdpFields.setProperty("m2", "Not Set");

            // Attribute lines (3rd)
            sdpFields.setProperty("a3", "Not Set");
            // Media information (3rd)
            sdpFields.setProperty("m3", "Not Set");
      }
}
```

SIP User Agent

```
package mkh.sip;

import java.util.Properties;
import drcl.comp.Port;
import drcl.comp.Contract;

/**
 * This class is a Session Initiation Protocol User Agent.
 *
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class SipUA extends drcl.inet.application.SUDPApplication
{

      /** An intermediate variable which is used for processing. */
      public SIPMessage receivedMessage;

      /** This is a message which is created by different methods of the
class. */
      public SIPMessage toBeSentMessage;

      /** A variable used for setting the content type of the SIP message,
typically set to "application/sdp". */
      public String contentType;

      /** A vaiable to store the response class from one of six possible
classes. */
      public int responseMessageClass;

      /** An intermediate variable which is used for checking whether a
message is a request or it is a response and directing it to the related
processor. */
      public String messageType;

      /** Used for storing the other party's address. */
      public int destination;

      /** Used for storing the transaction ID so that junk messages can be
discarded. */
      public String transactionID=null;

      /** Node address is set during initialization through the TCL
interface. */
      public int nodeAddress;

      /** Used for determining if the node should respond like
acknowledging an OK with ACK only if the node is indeed the initiator of
```

```
the request-response. */
      public boolean transactionInitiator=true;

      /** All requests of the node first goes to this address which is set
during initialization through the TCL interface. */
      public int configuredProxyServerAddress;

      /** Used for alerting the T.38 fax module to start sending the fax.
*/
      public Port faxPort;

      /** It is set during initialization through the TCL interface. */
      public String nodeViaField;

      /** It is set during initialization through the TCL interface. */
      public boolean alwaysUseProxyServer = false;

      /** Constructor. */
      public SipUA()
      {
            super();
            faxPort = addPort ("down","faxPort");
      }

      /** Arrived data first gets processed by this method. */
      protected void dataArriveAtDownPort(Object data, Port downPort)
      {
      /* Internal variable(must be located in this method only): Used for
alerting the User
          Agent module to send the BYE request after receiving the
notification of file
          transfer completion by the T.38 receiver (ftpd) at its notify@
port. */

            String faxCompletionNotification = "Not Set";

            try {
            SIPMessage receivedMessage = (SIPMessage)getContent(data);

            System.out.println("An SIP message has been received and is
being processed by "
                                    + getName());

            messageType = receivedMessage.headers.getProperty("Start-
Line");

            // Redirecting the request processing:
            if (messageType.equals("INVITE"))
                  processINVITE(receivedMessage);
            else if (messageType.equals("ACK"))
                  processACK(receivedMessage);
            else if (messageType.equals("BYE"))
                  processBYE(receivedMessage);

            // Redirecting the message to the response processor:
            // If it is an unsupported request message, it will be handled
there.
            else processResponse(receivedMessage);
            }catch (Exception ex)
            {
            // Checking to see if the arrived data is a fax completion
notification issued
            // by T.38 Receiver (ftpd); ftpd's notify@ port is attached to
down@ port of UA.

                  faxCompletionNotification = (String)data;
                  if (faxCompletionNotification.equals ("done"))
                  {
                        System.out.println("\n" + "Fax reception is
```

```
complete now..." );
                        sendBYE();
                        return;
                    }
            }
      }

//////////////////////////////////////////////////////////////////
// Methods for processing/sending responses :
//////////////////////////////////////////////////////////////////

      /** This method first checks to see whether the message is a valid
one then checks to see if it's a response or an unsupported request, after
that if the message is a response it goes on to handle each type of
response classes. */
      public void processResponse(SIPMessage receivedMessage)
      {
            // transactionID must have been set when sending a request.
            if (! transactionID.equals(
receivedMessage.headers.getProperty("Call-ID")))
            {
            System.out.println("This transaction does not exist." );
            return;
            }
            try{
            responseMessageClass =Integer.parseInt(
receivedMessage.headers.getProperty("Start-Line"));
            }catch(Exception ex) {
            System.out.println("The message headers have not been properly
set or" +
                                    " this type of SIP request is not
supported."+"\n" );
            printMessageContent(receivedMessage);
            return;
            }
            // Detemining the response class ...
            if ( (responseMessageClass >= 100) && (responseMessageClass <
200))
                  System.out.println("This is an informational class
response." );

            else if (responseMessageClass == 200)
            {
            System.out.println("This is a success class response." );
            destination =
Integer.parseInt(receivedMessage.headers.getProperty("Contact"));
            printMessageContent(receivedMessage);

            // If this node is the initiator of request-response then it
sends an ACK
            if (transactionInitiator) sendACK(receivedMessage);

            }
            else if ( (responseMessageClass >= 300) &&
(responseMessageClass < 400))
                  System.out.println("This is a redirection class
response." );

            else if ( (responseMessageClass >= 400) &&
(responseMessageClass < 500))
                  System.out.println("This is a client error class
response." );

            else if ( (responseMessageClass >= 500) &&
(responseMessageClass < 600))
                  System.out.println("This is a server error class
response." );

            else if ( (responseMessageClass >= 600) &&
```

```
(responseMessageClass < 700))
                  System.out.println("This is a global error class
response." );

            else
            {
                  System.out.println("The message headers have not been
properly." );
                  printMessageContent(receivedMessage);
            }
      }

      public void sendOK(SIPMessage receivedMessage)
      {
            int address;
            toBeSentMessage=receivedMessage;
            toBeSentMessage.headers.setProperty("Start-Line", "200");
            toBeSentMessage.headers.setProperty("Contact",
String.valueOf(nodeAddress));
            toBeSentMessage.headers.setProperty("Request-URI", "");

            if (! transactionInitiator)  address =
configuredProxyServerAddress;
            else address = destination;

            if (alwaysUseProxyServer)  address=
configuredProxyServerAddress;

            sendmsg(toBeSentMessage, 10/*size*/, address, 5060);
            System.out.println("An OK message is being sent now by "+
getName() +"\n" );
      }
//////////////////////////////////////////////////////////////////
// Methods for processing/sending requests :
//////////////////////////////////////////////////////////////////

      public void processINVITE(SIPMessage receivedMessage)
      {
            System.out.println("The following INVITE request has been
received by the"
                                    +" request processor...");
            printMessageContent(receivedMessage);

            // transactionID is saved to identify this session in later
messages.
            transactionID = receivedMessage.headers.getProperty("Call-ID");

            System.out.println("\n" + "The response of that is being
issued...");

            // destination is saved to use for directly sending messages to
the other party.
            destination =Integer.parseInt(
receivedMessage.headers.getProperty("From"));

            // The other party is the initiator, hence the value given.
            transactionInitiator=false;

            sendOK(receivedMessage);
      }

      public void sendINVITE(SIPMessage toBeSentMessage)
      {
            transactionID = toBeSentMessage.headers.getProperty("Call-ID");

            toBeSentMessage.headers.setProperty("Contact",
toBeSentMessage.headers.getProperty("From"));
            toBeSentMessage.headers.setProperty("Request-URI",
toBeSentMessage.headers.getProperty("To"));
```

```
            sendmsg(toBeSentMessage, 10/*size*/,
configuredProxyServerAddress, 5060);
            System.out.println("The SIP message is being sent now by "+
getName() +"\n" );
      }

      public void processACK(SIPMessage receivedMessage)
      {
            System.out.println("An ACK message has been received...");
            printMessageContent(receivedMessage);

            /* If this node is not the initiator of request-response then
receiving
             this ACK means that now it's the time for alerting the other
party to
                send the fax by faxPort which is connected to sender's
T.38 Sender
                module. Not a nice implementation, but the only way I
could make the
                fax start automatically. There is an implementation of
user agent
                which sends fax and BYE request by scheduling them in TCL
as well.*/

            if (! transactionInitiator)
            {
                  faxPort.doSending ("Ready");
            }
      }

      public void sendACK(SIPMessage receivedMessage)
      {
            int address;
            toBeSentMessage = receivedMessage;
            toBeSentMessage.headers.setProperty("Start-Line","ACK");
            toBeSentMessage.headers.setProperty("Content-Type","No Body" );
            toBeSentMessage.headers.setProperty("Contact",
String.valueOf(nodeAddress));
            toBeSentMessage.headers.setProperty("To",
String.valueOf(destination));
            toBeSentMessage.headers.setProperty("Request-URI",
receivedMessage.headers.getProperty("To"));
            toBeSentMessage.headers.setProperty("Via 1","Not Set");
            toBeSentMessage.headers.setProperty("Via 2","Not Set");

            if (alwaysUseProxyServer)  address=
configuredProxyServerAddress;
            else address= destination;

            sendmsg(toBeSentMessage, 10/*size*/, address, 5060);
            System.out.println("\n"+"An ACK message is being sent now by "+
getName() +"\n" );

      }

      public void processBYE(SIPMessage receivedMessage)
      {
            System.out.println("The following BYE request has been received
by the request"+
                                    " processor...");
            printMessageContent(receivedMessage);

            if (transactionID.equals(
receivedMessage.headers.getProperty("Call-ID")))
            {

            System.out.println("\n" + "The response of that is being
issued...");
```

```
            destination =Integer.parseInt(
receivedMessage.headers.getProperty("From"));

            sendOK(receivedMessage);
            }else System.out.println("A request for terminating a non-
existent connection "+
                                          "has been received !...");
      }

      public void sendBYE()
      {
            int address;
            SIPMessage toBeSentMessage = constructMessage("BYE",
nodeViaField, String.valueOf(destination), String.valueOf(nodeAddress),
transactionID, "No Body" ) ;

            toBeSentMessage.headers.setProperty("Contact",
String.valueOf(nodeAddress));
            toBeSentMessage.headers.setProperty("Request-URI",
String.valueOf(destination));

            if (alwaysUseProxyServer)  address=
configuredProxyServerAddress;
            else address= destination;

            sendmsg(toBeSentMessage, 10/*size*/, address, 5060);
            System.out.println("A BYE request is being sent now by "+
getName() +"\n" );

      }

////////////////////////////////////////////////////////////////
// Other utility methods :
////////////////////////////////////////////////////////////////

      public SIPMessage constructMessage(String startLine, String via,
String to, String from, String callID, String contentType)
      {
            SIPMessage toBeSentMessage= new SIPMessage(contentType);

            toBeSentMessage.headers.setProperty("Start-Line", startLine);
            toBeSentMessage.headers.setProperty("Via", via);
            toBeSentMessage.headers.setProperty("To", to);
            toBeSentMessage.headers.setProperty("From", from);
            toBeSentMessage.headers.setProperty("Call-ID", callID);
            toBeSentMessage.headers.setProperty("Subject", "A Fax
session");

            return toBeSentMessage;
      }

      public void printMessageContent(SIPMessage message)
      {
            System.out.println("\n" + "The message content:");
            System.out.println("Start-Line: "
                                    +message.headers.getProperty("Start-
Line"));
            System.out.println("Request-URI: "
                                    +message.headers.getProperty("Request-
URI"));
            System.out.println("Via: "
                                    +message.headers.getProperty("Via"));
            System.out.println("Via 1: "
                                    +message.headers.getProperty("Via 1"));
            System.out.println("Via 2: "
                                    +message.headers.getProperty("Via 2"));
            System.out.println("To: "
                                    +message.headers.getProperty("To"));
```

```
            System.out.println("From: "
                                    +message.headers.getProperty("From"));
            System.out.println("Call-ID: "
                                    +message.headers.getProperty("Call-
ID"));
            System.out.println("Subject: "

+message.headers.getProperty("Subject"));
            System.out.println("Contact: "

+message.headers.getProperty("Contact"));
            System.out.println("Content-Type: "
                                    +message.headers.getProperty("Content-
Type"));

            contentType = message.headers.getProperty("Content-Type");

            if (contentType.equals("application/sdp"))
            {
                  System.out.println("SDP fields: ");
                  System.out.println("SDP Protocol Version Number: "+

message.sdpMessage.sdpFields.getProperty("v"));
                  System.out.println("SDP Protocol Session Information: "
                                                      +
message.sdpMessage.sdpFields.getProperty("i"));
            }
      }

      public void setAddress(int address)
      {
            nodeAddress = address;

      }

      public void setConfiguredProxyServerAddress(int address)
      {
            configuredProxyServerAddress=address;
      }

      public void setNodeViaField(String s)
      {
            nodeViaField = s;

      }

      public void setAlwaysUseProxyServer (boolean x)
      {
            alwaysUseProxyServer =x ;
      }
////////////////////////////////////////////////////////////////
// Other module methods :
////////////////////////////////////////////////////////////////

      public void reset()
      {
            super.reset(); // Let super class reset its fields.
      }

      public void duplicate(Object source)
      {
            super.duplicate(source);
      }

      public String info()
      {
            return getName();
      }
}
```

SIP Proxy Server

```
package mkh.sip;

import java.util.Properties;
import drcl.comp.Port;
import drcl.comp.Contract;

/**
 * This class is an SIP Proxy Server.
 *
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class SipPS extends drcl.inet.application.SUDPApplication
{

      /** An intermediate variable which is used for processing. */
      public SIPMessage receivedMessage;

      /** This is a message which is created by different methods of the
class. */
      public SIPMessage toBeSentMessage;

      /** A variable used for setting the content type of the SIP message,
typically set to "application/sdp". */
      public String contentType;

      /** A vaiable to store the response class from one of six possible
classes. */
      public int responseMessageClass;

      /** An intermediate variable which is used for checking whether a
message is a request or it is a response and directing it to the related
processor. */
      public String messageType;

      /** Used for storing the transaction ID so that junk messages can be
discarded. */
      public String transactionID=null;

      /** Node address is set during initialization through the TCL
interface. */
      public int nodeAddress;

      /** All requests of the set node first goes to this proxy server and
it is set during initialization through the TCL interface. */
      public int registeredNode;

      /** It is set during initialization through the TCL interface. */
      public String nodeViaField;

      /** It is set during initialization through the TCL interface. */
      public int  otherNetworkProxyServerAddress;

      /** Constructor. */
      public SipPS()
      {
            super();
      }

      /** Arrived data first gets processed by this method. The proxy
server in this version only can accept/direct SIP packets. Fax data itself
should be directed using other routers.*/
      protected void dataArriveAtDownPort(Object data, Port downPort)
      {
```

```
            SIPMessage receivedMessage = (SIPMessage)getContent(data);

            System.out.println("An SIP message has been received and is
being processed by "+ getName());

            messageType = receivedMessage.headers.getProperty("Start-
Line");

            // Redirecting the request processing:
            if (messageType.equals("INVITE"))
                  processINVITE(receivedMessage);
            else if (messageType.equals("ACK"))
                  processACK(receivedMessage);
            else if (messageType.equals("BYE"))
                  processBYE(receivedMessage);

            // Redirecting the message to the response processor:
            // If it is an unsupported request message, it will be handled
there.
            else processResponse(receivedMessage);
      }

////////////////////////////////////////////////////////////////
// Methods for processing/sending responses :
////////////////////////////////////////////////////////////////

      /** This method first checks to see whether the message is a valid
one then checks to see if it's a response or an unsupported request, after
that if the message is a response it goes on to handle each type of
response classes. */
      public void processResponse(SIPMessage receivedMessage)
      {
      // transactionID must have been set when receiving the first INVITE
request.
      if (! transactionID.equals(
receivedMessage.headers.getProperty("Call-ID")))
      {
      System.out.println("This transaction does not exist." );
      return;
      }
      try{
      responseMessageClass =Integer.parseInt(
receivedMessage.headers.getProperty("Start-Line"));
      }catch(Exception ex) {
      System.out.println("The message headers have not been properly set
or" +
                                    " this type of SIP request is not
supported."+"\n" );
      printMessageContent(receivedMessage);
      return;
      }
      // Determining the response class ...
      if ( (responseMessageClass >= 100) && (responseMessageClass < 200))
            System.out.println("This is an informational class response."
);

      else if ( (responseMessageClass >= 300) && (responseMessageClass <
400))
            System.out.println("This is a redirection class response." );

      else if ( (responseMessageClass >= 400) && (responseMessageClass <
500))
            System.out.println("This is a client error class response." );

      else if ( (responseMessageClass >= 500) && (responseMessageClass <
600))
            System.out.println("This is a server error class response." );

      else if ( (responseMessageClass >= 600) && (responseMessageClass <
```

```
700))
            System.out.println("This is a global error class response." );

      else if (responseMessageClass == 200)
      {
            System.out.println("This is a success class response." );
            printMessageContent(receivedMessage);

            // If the message is received from a registered user agent, it
is forwarded to
            // the other network's proxy server

      if(Integer.parseInt(receivedMessage.headers.getProperty("Contact"))==
registeredNode)
            {
                  System.out.println("\n" + "The message is being forwarded
to the other"+
                                          " network's proxy server
now...");

            // Proxy server removes its Via header from the message.
            if
(nodeViaField.equals(receivedMessage.headers.getProperty("Via 1")) )
                  receivedMessage.headers.setProperty("Via 1","Removed");
            else if
(nodeViaField.equals(receivedMessage.headers.getProperty("Via 2")))
                  receivedMessage.headers.setProperty("Via 2","Removed");

                  // The message is configured and sent
                  sendOK(receivedMessage, otherNetworkProxyServerAddress);
            }else{
            // If the message is received from a non-registered user agent,
it is forwarded
                // to the intended registered node.

                  System.out.println("\n" + "The message is being forwarded
to the"+
                                          " intended node now...");

            if
(nodeViaField.equals(receivedMessage.headers.getProperty("Via 1")) )
                  receivedMessage.headers.setProperty("Via 1","Removed");
            else if
(nodeViaField.equals(receivedMessage.headers.getProperty("Via 2")))
                  receivedMessage.headers.setProperty("Via 2","Removed");

                  sendOK(receivedMessage, registeredNode);
            }
      }

      else System.out.println("The message headers have not been properly."
);

      }

      public void sendOK(SIPMessage receivedMessage, int address)
      {
            toBeSentMessage = receivedMessage;
            toBeSentMessage.headers.setProperty("Start-Line", "200");
            sendmsg(toBeSentMessage, 10/*size*/, address, 5060);
            System.out.println("An OK message is being sent now by "+
getName() +"\n" );
      }
////////////////////////////////////////////////////////////////
// Methods for processing/sending requests :
////////////////////////////////////////////////////////////////

      public void processINVITE(SIPMessage receivedMessage)
      {
```

```
            System.out.println("The following INVITE request has been
received by the "
                                    + "request processor...");

            printMessageContent(receivedMessage);

            // transactionID is saved to identify this session in later
messages.
            transactionID = receivedMessage.headers.getProperty("Call-ID");

            // If the message is received from a registered user agent, it
is forwarded to
            // the other network's proxy server

      if(Integer.parseInt(receivedMessage.headers.getProperty("Contact"))==
registeredNode)
            {
                  System.out.println("\n" + "The message is being forwarded
to the other"+
                                           " network's proxy server
now...");

                  receivedMessage.headers.setProperty("Via 1",
nodeViaField);
                  sendINVITE(receivedMessage,
otherNetworkProxyServerAddress);
            }else{
            // If the message is received from a non-registered user agent,
it is forwarded
                // to the intended registered node.
                  System.out.println("\n" + "The message is being forwarded
to the"+
                                           " intended node now...");

                  receivedMessage.headers.setProperty("Via 2",
nodeViaField);
                  sendINVITE(receivedMessage, registeredNode);
            }
      }

      public void sendINVITE(SIPMessage toBeSentMessage, int nextHop)
      {
            // The following three statements are necessary only when a
proxy server
                // initiates a transaction.

      /*
            transactionID = toBeSentMessage.headers.getProperty("Call-ID");

            toBeSentMessage.headers.setProperty("Contact",
toBeSentMessage.headers.getProperty("From"));
            toBeSentMessage.headers.setProperty("Request-URI",
toBeSentMessage.headers.getProperty("To"));

      */
            sendmsg(toBeSentMessage, 10/*size*/, nextHop, 5060);
            System.out.println("The SIP message is being sent now by "+
getName() +"\n" );
      }


      public void processACK(SIPMessage receivedMessage)
      {

      if (transactionID.equals( receivedMessage.headers.getProperty("Call-
ID")))
      {
            System.out.println("An ACK message has been received...");
            printMessageContent(receivedMessage);
```

```
            // If the message is received from a registered user agent, it
is forwarded to
            // the other network's proxy server

      if(Integer.parseInt(receivedMessage.headers.getProperty("Contact"))==
registeredNode)
            {
                  System.out.println("\n" + "The message is being forwarded
to the other"+
                                          " network's proxy server
now...");

                  receivedMessage.headers.setProperty("Via 1",
nodeViaField);
                  sendACK(receivedMessage, otherNetworkProxyServerAddress);
            }else{
            // If the message is received from a non-registered user agent,
it is forwarded
                // to the intended registered node.
                  System.out.println("\n" + "The message is being forwarded
to the"+
                                          " intended node now...");

                  receivedMessage.headers.setProperty("Via 2",
nodeViaField);
                  sendACK(receivedMessage, registeredNode);
            }
      }else System.out.println("An ACK request for a non-existent
connection "+
                                       "has been received !...");
      }

      public void sendACK(SIPMessage receivedMessage, int nextHop)
      {
            toBeSentMessage = receivedMessage;
            toBeSentMessage.headers.setProperty("Start-Line", "ACK");
            toBeSentMessage.headers.setProperty("Content-Type", "No Body");

            sendmsg(toBeSentMessage, 10/*size*/, nextHop, 5060);
            System.out.println("An ACK message is being sent now by "+
getName() +"\n" );

      }

      public void processBYE(SIPMessage receivedMessage)
      {
      if (transactionID.equals( receivedMessage.headers.getProperty("Call-
ID")))
      {
            System.out.println("The following BYE request has been received
by the request"+
                              " processor...");
            printMessageContent(receivedMessage);

            // If the message is received from a registered user agent, it
is forwarded to
            // the other network's proxy server

      if(Integer.parseInt(receivedMessage.headers.getProperty("Contact"))==
registeredNode)
            {
                  System.out.println("\n" + "The message is being forwarded
to the other"+
                                          " network's proxy server
now...");

                  receivedMessage.headers.setProperty("Via 1",
nodeViaField);
```

```
                  sendBYE(receivedMessage, otherNetworkProxyServerAddress);
            }else{
            // If the message is received from a non-registered user agent,
it is forwarded
                // to the intended registered node.
                  System.out.println("\n" + "The message is being forwarded
to the"+
                                           " intended node now...");

                  receivedMessage.headers.setProperty("Via 2",
nodeViaField);
                  sendBYE(receivedMessage, registeredNode);
            }
      }else System.out.println("A request for terminating a non-existent
connection "+
                                          "has been received !...");
      }

      public void sendBYE(SIPMessage toBeSentMessage, int nextHop)
      {
            sendmsg(toBeSentMessage, 10/*size*/, nextHop, 5060);
            System.out.println("The SIP message is being sent now by "+
getName() +"\n" );
      }

////////////////////////////////////////////////////////////////
// Other utility methods :
////////////////////////////////////////////////////////////////

            // The following function is necessary only when a proxy server
                // initiates a transaction.

      public SIPMessage constructMessage(String startLine, String via,
String to, String from, String callID, String contentType)
      {
            SIPMessage toBeSentMessage= new SIPMessage(contentType);

            toBeSentMessage.headers.setProperty("Start-Line", startLine);
            toBeSentMessage.headers.setProperty("Via", via);
            toBeSentMessage.headers.setProperty("To", to);
            toBeSentMessage.headers.setProperty("From", from);
            toBeSentMessage.headers.setProperty("Call-ID", callID);
            toBeSentMessage.headers.setProperty("Subject", "A Fax
session");

            return toBeSentMessage;
      }

      public void printMessageContent(SIPMessage message)
      {
            System.out.println("\n" + "The message content:");
            System.out.println("Start-Line: "
                                    +message.headers.getProperty("Start-
Line"));
            System.out.println("Request-URI: "
                                    +message.headers.getProperty("Request-
URI"));
            System.out.println("Via: "
                                    +message.headers.getProperty("Via"));
            System.out.println("Via 1: "
                                    +message.headers.getProperty("Via 1"));
            System.out.println("Via 2: "
                                    +message.headers.getProperty("Via 2"));
            System.out.println("To: "
                                    +message.headers.getProperty("To"));
            System.out.println("From: "
                                    +message.headers.getProperty("From"));
            System.out.println("Call-ID: "
                                    +message.headers.getProperty("Call-
```

```
ID"));
            System.out.println("Subject: "

+message.headers.getProperty("Subject"));
            System.out.println("Contact: "

+message.headers.getProperty("Contact"));
            System.out.println("Content-Type: "
                                    +message.headers.getProperty("Content-
Type"));

            contentType = message.headers.getProperty("Content-Type");

            if (contentType.equals("application/sdp"))
            {
                  System.out.println("SDP fields: ");
                  System.out.println("SDP Protocol Version Number: "+

message.sdpMessage.sdpFields.getProperty("v"));
                  System.out.println("SDP Protocol Session Information: "
                                                      +
message.sdpMessage.sdpFields.getProperty("i"));
            }
      }
      public void setAddress(int address)
      {
            nodeAddress = address;

      }


      public void setOtherNetworkProxyServerAddress(int address)
      {
            otherNetworkProxyServerAddress=address;
      }

      public void setRegisteredNode(int address)
      {
            registeredNode = address;
      }

      public void setNodeViaField(String s)
      {
            nodeViaField = s;

      }


////////////////////////////////////////////////////////////////
// Other module methods :
////////////////////////////////////////////////////////////////

      public void reset()
      {
            super.reset(); // Let super class reset its fields.
      }

      public void duplicate(Object source)
      {
            super.duplicate(source);
      }

      public String info()
      {
            return getName();
      }
}
```

T.38 Sender

```
package mkh.sip;

import drcl.comp.Port;
import java.io.*;
import drcl.comp.lib.bytestream.ByteStreamContract;

/**
 * This class is a T.38 Sender (Basically an FTP Server). "helper" in
SApplication.java Line
 * 59 must be declared public and the file gets recompiled and stored in
the class folder if one
 * wants to compile the T38Sender.java
 *
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class T38Sender extends drcl.inet.application.ftp
{

      public Port faxPort;

      /** Constructor. faxPort is initialized for listening to "Ready"
notifications. To-be-sent fax (file) name is also set to "ToBeSentFax.JPG"
(The to-be-sent fax has to have exactly this name and extention, otherwise
the code has to be recompiled using the preferred names).*/
      public T38Sender() throws IOException
      {
            super();
            faxPort = addPort ("down","faxPort");
            super.setup("ToBeSentFax.JPG" , 32768);
      }


      /** The process() method of drcl.inet.application.SApplication has
been overridden.
            and "data" is first examined to see if it is a fax start
notification from the user agent module and then if it's not, it is
directed to the "helper" variable of SApplication.*/
      public void process(Object data, drcl.comp.Port inPort)
      {
            if (inPort.getID().equals("faxPort"))
            {
                  String faxReadyNotification = (String) data;
                  if (faxReadyNotification.equals("Ready"))
                  super._start();
                  System.out.println("\n"+"Starting the T.38 fax
transmission..." +"\n" );
            }else
            {
                  // "helper" in SApplication.java Line 59 must be declared
public and the
                        // file gets recompiled and stored in class folder

                  helper.handle((ByteStreamContract.Message)data);
            }

      }

//////////////////////////////////////////////////////////////////
// Other module methods :
//////////////////////////////////////////////////////////////////

      public void reset()
      {
```

```
            super.reset(); // Let super class reset its fields.
      }

      public void duplicate(Object source)
      {
            super.duplicate(source);
      }

      public String info()
      {
            return "T.38 Sender Class.";
      }
}
```

T.38 Receiver

```
package mkh.sip;

import drcl.comp.Port;
import java.io.*;
import drcl.comp.lib.bytestream.ByteStreamContract;

/**
 * This class is a T.38 Receiver (Basically an FTP Client). "helper" in
SApplication.java Line
 * 59 must be declared public and the file gets recompiled and stored in
the class folder if one
 * wants to compile the T38Receiver.java
 *
 *
 * @author Masood Khosroshahy ( www.m-kh.info )
 * @version 1.0, 14/07/2004
 */

public class T38Receiver extends drcl.inet.application.ftpd
{

      public Port faxPort;

      /** Constructor. faxPort is initialized for listening to "Ready"
notifications. Received fax (file) name is also set.*/
      public T38Receiver() throws IOException
      {
            super();
            faxPort = addPort ("down","faxPort");
            super.setup("ReceivedFax.jpg", 32768);
      }

      /** The process() method of drcl.inet.application.SApplication has
been overridden.
            and "data" is first examined to see if it is a fax start
notification from the user agent module and then if it's not, it is
directed to the "helper" variable of SApplication.*/

      public void process(Object data, drcl.comp.Port inPort)
      {
            if (inPort.getID().equals("faxPort"))
            {
                  String faxReadyNotification = (String) data;
                  if (faxReadyNotification.equals("Ready"))
                  super._start();
            }else
            {
                  // "helper" in SApplication.java Line 59 must be declared
public and the
                        // file gets recompiled and stored in class folder
```

```
                  helper.handle((ByteStreamContract.Message)data);
            }
      }

////////////////////////////////////////////////////////////////
// Other module methods :
////////////////////////////////////////////////////////////////

      public void reset()
      {
            super.reset(); // Let super class reset its fields.
      }

      public void duplicate(Object source)
      {
            super.duplicate(source);
      }

      public String info()
      {
            return "T.38 Sender Class.";
      }
}
```

References

[1] Bur Goode ”Voice over Internet Protocol (VoIP)” , Proceedings of the IEEE, Vol.90, No.9, September 2002

[2] “Voice and fax over IP”, The International Engineering Consortium, http://www.iec.org

[3] James Irvine and David Harle, “Data Communications and Networks: An Engineering Approach”, John Wiley 2002

[4] Lillian Goleniewski, “Telecommunications Essentials”, Addison-Wesley 2002

[5] Bill Douskalis, “IP Telephony: The Integration of Robust VoIP Services”, Prentice Hall 2000

[6] “Tutorial on Signaling System 7”, Performance Technologies, http://www.pt.com

[7] Steven Shepard, “Telecommunications Convergence”, McGraw-Hill 2002

[8] “Signaling System 7”, The International Engineering Consortium, http://www.iec.org

[9] Kimmo Ahonen, Juha Koskelainen, “Transport Control Protocol”, University of Helsinki, October 1998

[10] Henning Schulzrinne (Columbia University), Jonathan Rosenberg (Bell Laboratories Lucent Technologies), “Internet Telephony: Architecture and Protocols an IETF Perspective”, July 1998

[11] “Understanding Packet Voice Protocols”, Cisco Systems, 2001

[12] Kenneth R. McConnell, Dennis Bodson, Stephen Urban, “Fax: Facsimile Technology and Systems”, Artech House 1999

[13] “An Introduction to IP Telephony”, Mockingbird Networks

[14] Alan B. Johnston, “SIP: Understanding the Session Initiation Protocol”, Artech House 2001

[15] “Voice Performance over packet-based networks”, Alcatel, October 2002

[16] “IP Telephony Design Guide”, Alcatel, April 2003

[17] Asim Karim, “H.323 and Associated Protocols”, Helsinki University of Technology, November 1999, http://www.hut.fi

[18] “H.323”, The International Engineering Consortium, http://www.iec.org

[19] “Use of MEGACO vis-à-vis MGCP to build a gateway solution”, Hughes Software Systems, May 2001, http://www.hssworld.com

[20] F.Andreasen and B. Foster, "Media Gateway Control Protocol (MGCP)",RFC 3435, IETF, January 2003.

[21] B. Foster and C. Sivachelvan, "Media Gateway Control Protocol (MGCP) Return Code Usage", RFC 3661, IETF, December 2003.

[22] Mike Gray, “FAX Technology Tutorial and Testing Issues”, Agilent Technologies, February 2002

[23] ITU-T Recommendation T.4, “Standardization of Group 3 facsimile terminals for document transmission”, Terminals For Telematic Services, July 2003

[24] ITU-T Recommendation T.30, “Standardization of Group 3 facsimile terminals for document transmission”, Terminals For Telematic Services, July 2003

[25] ITU-T Recommendation T.37, “Procedures for the transfer of facsimile data via store-and-forward on the Internet”, Terminals For Telematic Services, June 1998

[26] ITU-T Recommendation T.38, “Procedures for real-time Group 3 facsimile communication over IP networks”, Terminals For Telematic Services, March 2002

[27] ITU-T Recommendation T.38- Amendment 1, “Procedures for real-time Group 3 facsimile communication over IP networks”, Terminals For Telematic Services, July 2003

[28] ITU-T Recommendation T.38-Amendment 3, “Procedures for real-time Group 3 facsimile communication over IP networks”, Terminals For Telematic Services, January 2004

[29] ITU-T Recommendation T.38-Corrigendum 1, “Procedures for real-time Group 3 facsimile communication over IP networks”, Terminals For Telematic Services, July 2003

[30] “T.38 and the Future of Fax”, Intel 2003

[31] K. Toyoda, H. Ohno, J. Murai and D. Wing, " A Simple Mode of Facsimile Using Internet Mail",RFC 2305, IETF, March 1998.

[32] K.Mimura, K.Yokoyama, T.Satoh and C.Kanaide, " Internet FAX Gateway Functions", Internet Draft, IETF, draft-ietf-fax-gateway-protocol-09.txt, April 14 2003.

[33] Y.Rafiq, O.Bashir,S.I.Shah and S,A.Khan, “FoIP gateways-architectures, implementation and QoS issues”, IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings., Page(s): 87 -92, 28-30 Dec. 2001

[34] “eBusiness Companies Can Reap Rewards by Faxing over IP”, DCI 1999, http://www.dci.com

[35] Phelim O’Doherty, “SIP Specifications and the Java Platforms”, 2003 Sun Microsystems

[36] Jean-Francois Mule and Jieying Li, "SIP Support for Real-time Fax: Call Flow Examples And Best Current Practices", Internet Draft, IETF, draft-ietf-sipping-realtimefax-01.txt, August 2003.

[37] Jonathan Cumming, “SIP Market Overview”, Data Connection (DCL), September 2003, jonathan.cumming@dataconnection.com

[38] Dorgham Sisalem and Jiri Kuthan, “Understanding SIP”, Mobile Integrated Services, Sisalem,kuthan@fokus.gmd.de

[39] M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg, "SIP: Session Initiation Protocol",RFC 2543, IETF, March 1999.

[40] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson. R. Sparks, M. Handley and E. Schooler, "SIP: Session Initiation Protocol",RFC 3261, IETF, June 2002.

[41] J. Rosenberg and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers",RFC 3263, IETF, June 2002.

[42] A. B. Roach, " Session Initiation Protocol (SIP)-Specific Event Notification",RFC 3265, IETF, June 2002.

[43] A. Vemuri and J. Peterson, " Session Initiation Protocol for Telephones (SIP-T): Context and Architectures",RFC 3372, IETF, September 2002.

[44] A. Johnston, S. Donovan, R. Sparks, C. Cunningham and K. Summers, "Session Initiation Protocol (SIP) Basic Call Flow Examples", RFC 3665, IETF, December 2003.

[45] M. Handley and V. Jacobson, " SDP: Session Description Protocol",RFC 2327, IETF, April 1998.

[46] J. Rosenberg and H. Schulzrinne, " An Offer/Answer Model with the Session Description Protocol (SDP)",RFC 3264, IETF, June 2002.

[47] Jonathan Rosenberg (Bell Laboratories), Jonathan Lennox and Henning Schulzrinne (Columbia University), “Programming Internet Telephony Services”, jdrosen@bell-labs.com, lennox@cs.columbia.edu and hgs@cs.columbia.edu

[48] Xiaotao Wu and Henning Schulzrinne, ”Programmable End System Services Using SIP”, IEEE International Conference on Communications ICC '03, Volume: 2 , Page(s): 789 -793, 11-15 May 2003

[49] G.Stojsic, R.Radovic and S.Srbljic, “Formal Definition of SIP Proxy Behavior”, EUROCON'2001, International Conference on Trends in Communications, Volume: 2, Page(s): 289 -292, 4-7 July 2001

[50] R.Radovic, I.Crkvenac and S.Srbljic, “Formal definition of SIP end systems behavior”, EUROCON'2001, International Conference on Trends in Communications, Volume: 2, Page(s): 293 - 296, , 4-7 July 2001

[51] “Next-Gen VoIP Services and Applications Using SIP and Java”, The Applied Technologies Group, 2001, http://www.techguide.com

[52] J. Lennox, H. Schulzrinne and J. Rosenberg, "Common Gateway Interface for SIP",RFC 3050, IETF, January 2001.

[53] Phelim O’Doherty and Mudumbai Ranganathan, “JAIN SIP Tutorial", Sun Microsystems 2003

[54] JAIN team, “JAIN Technology”, Sun Microsystems 2003