Republic of Iraq
Ministry of Higher Education
and Scientific Research
Al-Nahrain University
College of Science

Security in Voice over Internet Protocol (VoIP)

A Thesis
Submitted to the College of Science, Al-Nahrain University
In Partial Fulfillment of the Requirements for
The Degree of Master of Science in Computer Science

By

Kawthar Abed Al-Elah Abed Al-Rasul

Mashkour

(B.Sc. 2004)

Supervisor

Dr. Ban N. Al-Kallak

April 2008 ***Rabi' Thani 1429***

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

(ٱلْحَمْدُ للهِ ٱلَّذِي هَدَٰنَا لِهَٰذَا
وَمَا كُنَّا لِنَهْتَدِىَ لَوْلآ أَنْ
هَدَٰنَا ٱللهُ)

صدق الله العظيم
الاعراف (43)

Dedication

I would like to dedicate this work to the member of my beloved family and all those who supported me in finishing this thesis.

Kawthar

Acknowledgement

First, I would like to thank God, for all the blessings that have given us.

Second, I would like to express my sincere gratitude and appreciation to my supervisor Dr. Ban N. Al-Kallak for her valuable guidance, supervision and untiring efforts during the course of this work.

Grateful thanks for the Head of Department of Computer Science Dr. Taha S. Bashaga, all staff and employees.

Finally, my very special thanks to my family, my friends for their encouragement during the period of my studies.

Abstract

In today’s environment nearly all end-to-end telephone connections are set up via circuit-switching using Public Switched Telephone Network (PSTN), whereby node-to-node links in an origin/destination connection are set up via interconnects, and the connection is maintained exclusively for exchanges of information between the origin and destination until it is torn down. An alternate way of setting up end-to-end connections that is widely used for transmission of data is packet-switching, whereby origin-to-destination connections are effected by node-to-node, store-and-forward relay of small segments of data sets that are reassembled at the destination; this technique is called Voice over Internet Protocol (VoIP). VoIP is considered as the third generation of telecommunication telephony after the analog and digital telecommunication technology.

This thesis study the architecture of packet-switched telephone networks and then analyzes the structure of VoIP technology, which is the Transmission Control Protocol/Internet Protocol (TCP/IP) model, some protocols reside in the application layer (i.e., Session Initiation Protocol (SIP) for call control, Session Description Protocol (SDP) for description media stream, and Real-time Transport Protocol (RTP) for media exchange).

Over years, Interest of security is increasing. To provide privacy for user's conversation in VoIP, there is a need to implement a security for media transmission. A Secure Real-time Transport Protocol (SRTP) is designated to provide security for real-time media transmission using an encryption method,

but it does not provide key agreement between participants. This thesis implemented SIP protocol and a key agreement using pre-shared key protocol within SDP protocol used by SIP. The implementation is done using **UNICON** language.

List of Abbreviations

AALx	Asynchronous transfer mode Adaptation Layer (1,2, or 5)
ACK	ACKnowledgement
AES-CTR	Advanced Encryption Standard in Counter Mode
AH	Authentication Header
AIN	Advanced Intelligent Network
ALG	Application Layer Gateway
AOR	Address Of Record
API	Application Programming Interface
APP	Application-sPecific Packet
ARPANET	Advanced Research Projects Agency Network
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BES	Back-End Service
BNF	Backus-Naur Form
CAN	Content Addressable Network
CODEC	Coder/Decoder
CRLF	Carriage Return Line Feed
CVE	Collaborative Virtual Environment
DCCP	Datagram Congestion Control Protocol
DHT	Distributed Hash Table
DSL	Digital Subscriber Line

EBCDIC	Extended Binary Coded Decimal Interchange Code
EP	EndPoint
ESP	Encapsulating Security Payload
FoIP	Fax over IP
FreeBSD	Free Berkeley Software Distribution
FSM	Finite State Machine
FSR	Feedback Shift Register
FTP	File Transfer Protocol
GC	Gateway Controller
GK	GateKeeper
GW	GateWay
HP-UX	Hewlett Packard UniX
HTAB	Horizontal TAB
HTTP	HyperText Transfer Protocol
ID	IDentifier
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPsec	IP security protocol
ISDN	Integrated Services Digital Network
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union-Telecommunication

IV	Initialization Vector
JMF	Java Media Framework
JRTPLIB	Jori's Real-time Transport Protocol LIBrary
JVOIPLIB	Jori's Voice Over Internet Protocol LIBrary
LAN	Local Area Network
LFSR	Linear Feedback Shift Register
LocalSI	Local Service Interface
LWS	Linear White Space
MCU	Multipoint Control Unit
MG	Media Gateway
MGC	Media Gateway Controller
MGCP	Media Gateway Control Protocol
MPEG II video	Moving Picture Experts Group II video
MS-Windows	MicroSoft-Windows
NAT	Network Address Translation
NVP	Network Voice Protocol
P2P	Peer-to-Peer
PBX	Private Branch Exchange
PC	Personal Computer
PCM	Pulse Code Modulation
PESQ	Perceptual Evaluation of Speech Quality
POTS	Plain Old Telephony Service
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network

PT	Payload Type
QoS	Quality of Service
RAS	Registration Admission and Status
RFC	Request For Comment
RR	Receiver Report
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SDES	Source DEScription
SDK	Software Development Kit
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SIPS	Secure Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SP	SPace
SR	Sender Report
SRTP	Secure Real-time Transport Protocol
SS7	Signaling System 7
SSRC	Synchronization SouRCe
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDM	Time-Division Multiplexed
Telnet	TELecommunications NETwork
TFRC	Transmission control protocol Friendly Rate Control

TLS	Transport Layer Security
TU	Transaction User
TV	TeleVision
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UDP/IP	User Datagram Protocol/Internet Protocol
Unicon	Unified Extended Dialect of Icon
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
VoATM	Voice over Asynchronous Transfer Mode
VoFR	Voice over Frame Relay
VoIP	Voice over Internet Protocol (Voice over IP)
VPN	Virtual Private Network
vTEL	video TELephony
WAN	Wide Area Network
WSP	White Space
XOR	eXclusive-OR

List of Figures

Contents

Chapter One
Overview of VoIP Technology

1.1 Introduction

The impact of continuing advances in communications technology on our ability to exchange information in new ways, places us at threshold of a new era. The promise of ubiquitous high-speed networks carrying voice, data, and multimedia services is happening today [Bil00].

The idea of using packet networks such as the Internet to transport voice is not new. Experiments with voice over packet networks stretch back to the early 1970s. The first Request For Comment (RFC) on this subject, the Network Voice Protocol (NVP) dates from 1977. The initial developers of NVP were researchers transmitting packet voice over the Advanced Research Projects Agency Network (ARPANET), the predecessor to the Internet, but today voice is transported over packet networks using individual transport technologies: Voice over IP (VoIP), Voice over Asynchronous Transfer Mode (VoATM), and Voice over Frame Relay (VoFR). However, VoFR and VoATM technologies are not used as widely as IP [Col03, Jon00].

This thesis is related to the VoIP technology. In the late 1990s, VoIP was lauded as a way to save on long-distance charges by calling Grandma and Grandpa using a Personal Computer (PC) with a headset and a microphone. VoIP is a generic term that refers to all types of voice communication using Internet Protocol (IP) technology. As a technology, VoIP is a pretty simple ideas packet-switched data encapsulation instead of the tried-and-true Time-Division Multiplexed (TDM), circuit-switched methods that telephony has

used since its creation. Also VoIP is a family of technologies that has sweeping implications for everybody who uses telephones, the Internet, fax machines, email, and the Web. VoIP borrows from, and enhances, many disciplines of communications technology [Jim02, Ted05].

Advances in packet communication technologies are now making that model obsolete, permitting more efficient use of bandwidth resources while providing mobility and the integration of voice, video, and other information, saving cost, and possibility of adding a new features [Ted05].

When finishing this chapter, the following points will be clear:

1. What is VoIP?
2. General VoIP components and there functions.
3. General functional components of VoIP technology.
4. How VoIP technology is different than other data protocol.
5. The most common applications of this new technology.

1.2 Components of a VoIP Network

This topic introduces the basic components of a packet voice network as shown in Figure (1.1).

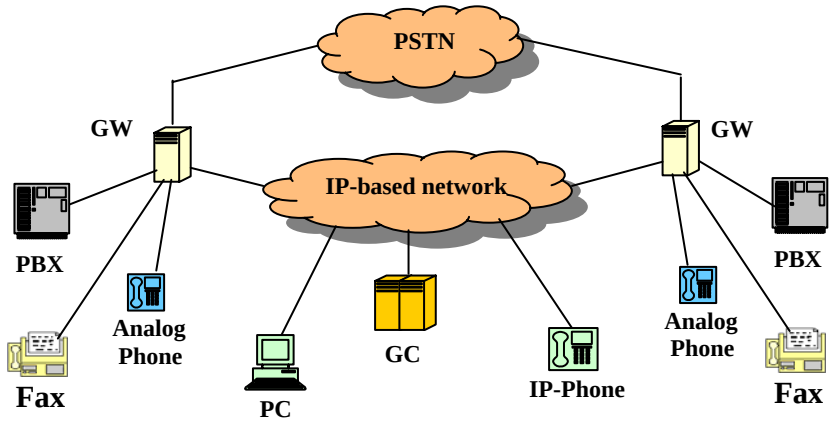

Figure 1.1 Basic Components of Packet Network [Ant06]

1.2.1 IP-based Network

From the viewpoint of telephony, IP is the major Protocol that can be used to deliver voice to the desktop [Dav01].

1.2.2 Gateway (GW)

Gateways are called different things by different people. The IP community calls them Media Gateway (MG). MG is characterized by a collection of endpoints and connections. The MG is a box of various morphologies depending on the number of users, trunks and services it supports. The gateway provides translation between VoIP and non-VoIP networks such as the Public Switched Telephone Network (PSTN). Gateways also provide physical access for local analog and digital voice devices, such as telephones, fax machines, and Private Branch eXchange (PBX) [Ant06, Bil00].

1.2.3 Gateway Controller (GC)

The most important components of distributed architecture are gateways and gateway controllers. The gateway controllers are sometimes called Media Gateway Controller (MGC) or ‘softswitches’, even though they are not actually switches in the sense of switching the voice traffic from input ports to output ports. They are servers that control the gateways. The gateway controller focuses on making routing decision and communicating them to the gateway. The voice traffic never entered the gateway controller, instead it is switched by the gateway. It is only the signaling messages that enter the gateway controller. [Dav01].

1.2.4 Endpoints (EPs)

Endpoint is defined as a point of entry and exit of media flows, such as

IP-Phone, analog phone, softphone, PC, fax [Bil00].

1.3 VoIP Functional Components

In the traditional PSTN, all the elements that are required to complete the call are transparent to the end user. Migration to VoIP necessitates an awareness of these required elements and a thorough understanding of the protocols and components that provide the same functionality in an IP network [Ken07].

Required VoIP functionality includes the following features:

- Signaling
- Bearer control
- CODECs
- Database service

1.3.1 Signaling

Signaling is the ability to generate and exchange control information to establish, monitor, and release connections between two endpoints. Voice signaling requires the ability to provide supervisory, address, and alerting functionality between nodes. VoIP presents several options for signaling, including H.323, Session Initiation Protocol (SIP), and Media Gateway Control Protocol (MGCP) [Ant06].

Signaling protocols are classified either as peer-to-peer or client/server architectures. SIP and H.323 are examples of peer-to-peer signaling protocols where the end devices or gateways contain the intelligence to initiate and terminate calls and interpret call control messages. MGCP is an example of client/server protocol where the endpoints or gateways do not contain call

control intelligence but send or receive event notifications to the server commonly referred to as the call agent [Ken07].

1.3.2 Bearer Channel Control

Bearer channels are the channels that carry voice calls. Proper Supervision of these channels requires that the appropriate call connect and call disconnect signaling be passed between end devices. Correct signaling ensures that the channel is allocated to the current voice call and that the channel is properly de-allocated when either side terminates the call. These connect and disconnect messages are carried in SIP, H.323, or MGCP within an IP network [Ken07].

1.3.3 Coders/Decoders (CODECs)

A CODEC (which stands for Coder/Decoder or Compress/Decompress) is the hardware or software that samples analog sound, converts it to digital bits, and outputs it at a predetermined data rate. Each CODEC type defines the method of voice coding and the compression mechanism that is used to convert the voice stream. For example: G.711 creates a 64-kbps digitized voice stream and the most widely used CODEC in the Wide Area Network (WAN) environment is G.729, which compresses the voice stream (that is, the voice payload only) to 8 kbps [Joh04, Ken07].

1.3.4 Database Service

Access to services such as toll-free numbers or caller IDentifier (ID) requires the ability to query a database to determine whether the call can be placed. This database must be loaded in the MG in order to make efficient use of messaging with the MGC during digit collection. [Ant06, Bil00].

1.4 VoIP Isn't Just Another Data Protocol

VoIP utilizes the Internet architecture, similar to any other data application. However, particularly VoIP is different. There are three significant reasons for this [Tho06]:-

1. Separation of data and signaling. Sessions, particularly unknown inbound sessions, which define addressing information for the data (media) channel in a discrete signaling channel do not interact well with Network Address Translation (NAT) and encryption.

2. The real-time nature of VoIP—gets there a second too late, and the packet is worthless. Each VoIP packet represents about 20 ms of voice on average. A single lost packet may not be noticeable, but the loss of multiple packets is interpreted by the user as bad voice quality. The simple math indicates that even a short VoIP call represents the transport of large numbers of packets. Network latency, jitter (interpacket latency variation), and packet loss critically affect the perceived quality of voice communications.

3. Voice conversations can be initiated from outside the firewall. Most client-driven protocols initiate requests from inside the firewall. Figure (1.2) shows the basic message flow of a typical Web browsing, or e-mail, a request is initiated by a client on the internal side of the firewall to a server daemon residing on a host external to the firewall. Firewalls that are capable of stateful inspection will monitor the connection and open inbound ports if that port is associated with an established session. For the firewall administrator and the user, the session completes normally, and is as secure as the firewall's permissions allow. In Figure (1.3), the request-response

topology is different from the message flow shown in Figure (1.2). In this figure, an external host (IP Phone, PC softphone, etc.) attempts to place a call to an internal host. Since no session is established, firewalls will not allow this connection to complete.

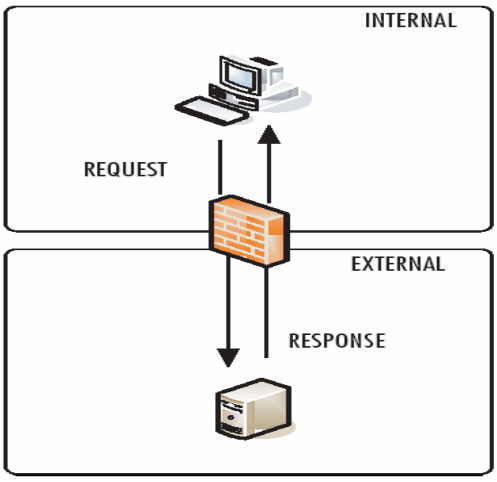

Figure 1.2 Normal Message Flow [Tho06]

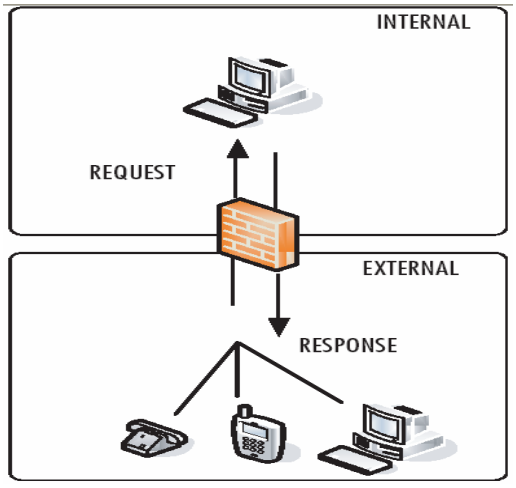

Figure 1.3 VoIP Message Flow [Tho06]

1.5 VoIP Applications

For any new technology to be accepted by the telecommunications industry, it must have a clear business benefit not just to a single player in the industry, but also to many other players as well. VoIP promises to

revolutionize the most familiar of technologies, the telephone. The Internet Protocol, analog telephony, digital telephony, digital audio signal processing, high-availability networking, and a host of other concerns are all touched by growing borders of the vast, ambitious realm of VoIP [Dav01, Ted05].

The first kind of application is the 'telephone alternative'. This means that you would use some kind of VoIP system to make a voice call to another person. When voice is packetised all the way to the desktop, it opens up vast possibilities for new applications and features to be added onto Plain Old Telephony Service (POTS), such as instance messaging [Dav01, Jor00].

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

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

Note that the list of applications presented here is certainly not complete. A wide range of applications using VoIP related techniques are conceivable, but many of them will resemble the ones discussed [Jor00].

1.6 Literature Survey

[Jor00] Voice over IP in Networked Virtual Environments

The PH.D. thesis goal is to create a VoIP application includes an Internet Telephony application and a 3D environment. This done by first, developing a Real-time Transport Protocol (RTP) library, named by JRTPLIB (Jori's RTP LIBrary). It is proved to be both fast and simple to use. Second, developing a VoIP framework in which different VoIP components can be easily used, tested, and made the VoIP part of an application portable to several platforms (MS-Windows, Linux, Free Berkeley Software Distribution (FreeBSD), Solarism, Hewlett Packard UniX (HP-UX), and VxWorks). This is because normally only the reconstruction components need to be rewritten for a new platform.

The use of Internet telephony application and the 3D environment are quite useful. But when using a dial-up link, the necessary bandwidth might simply not be available, even when severe compression is used. So as a conclusion, the solution of this problem is to place a machine which mixes the signals for a specific participant before the link.

[Fre00] SIP, NAT, and Firewalls

The problem is extended with this MS.c. thesis by the fact that it is common today to use “private addresses” on the Local Area Network (LAN). These addresses are not allowed to exist on the Internet and thus the firewall software must remove this address and replace it with an address that is allowed on the Internet. NAT in the firewall normally does this together with Application Level Gateways (ALGs).

Two goals of this thesis have been achieved. The first one was to study what was required by the firewall in order to pass SIP signaling in and out of a private network that uses NAT. This is a problem that can only be solved by having some software analyzing all the SIP messages that pass the firewall on the well-known SIP port and then letting that software what it should do with packets to/from a certain address, e.g. let it through or deny and drop it. The second goal was to implement an application layer gateway for SIP. In the case of incoming SIP INVITE messages from external clients intended for internal clients it is clear that some kind of location service must be available in the SIP ALG in order to direct the message to the correct host on the inside. Implementing a simple SIP registrar server together with the SIP ALG solves this problem. The internal clients would have to register at the registrar server in order for the server to get the clients internal IP address. For outgoing calls it is not necessary for the internal client to be registered at the server.

[Siv01] Voice over Internet Protocol (VoIP)

The MS.c. thesis aims to investigate Voice over IP based on the RTP and Real-Time Control Protocol (RTCP). It studies the key technical issues and develops software to demonstrate how Voice over IP can be implemented using Java (2) Software Development Kit (SDK) and Java Media Framework 2.1.1 Application Programming Interface (JMF 2.1.1 API). Also, It has examined the Quality of Service (QoS) associated with voice transfer through Internet (Internet Protocol).

Taking the above into account, this thesis used and made an enhancement to the functional capabilities of the vTEL (video TELephony) implementation from just merely transmitting the real-time video to both voice and video transmissions with enhanced user friendliness. Apparently,

from the users' point of view, what they are much concerned about is the performance of the voice and video transfer between the end users. Therefore, the working on "vTEL" and "VideoConference" programs was provided everything that is deemed necessary, and also developed the in-depth knowledge of VoIP principles.

[Joh02] Security in VoIP-Telephony Systems

This MS.c. thesis deals with the security of a VoIP telephony system. The involved components are EPs, GateKeeper (GK) and the Back-End Service (BES). The underlying protocol of these components is H.323, a standard for IP telephony.

The goals of the thesis are (1) the used protocol had to be analyzed according to existing threats, which was mainly the ability of attackers to call for free, (2) the implementation of mutual authentication on the connection between two GKs.

As a result all interfaces require the security services authentication and data integrity. The interface between a GK and the BES additionally demands privacy because of the sensitivity of the transmitted data. The result for the proposals of the security framework must be seen for each interface or connection separately. The Registration Admission Status (RAS) protocol between EP-GK was already equipped with authentication and data integrity according to H.235v2. The authentication is implemented for H.235 based authentication between two GKs. The key management has been kept simple since the GKs are using a Local Service Interface (LocalSI), which is a service interface realized by local sockets or UNIX sockets. A connection to these interfaces does not impose much overhead to the execution time.

[Lin04] Speech Quality Prediction for Voice over Internet Protocol Networks

The main goal of this MS.c. thesis is to develop novel and efficient models for non-intrusive speech quality prediction to overcome the disadvantages of current subjective-based methods and to demonstrate their usefulness in new and emerging VoIP applications.

An important conclusion of this thesis is that it is possible to exploit perceptually more accurate intrusive speech quality measurement (e.g. Perceptual Evaluation of Speech Quality (PESQ)) for non-intrusive applications. This is an important development as it avoids time-consuming subjective tests and removes a major obstacle in the development of models for non-intrusive prediction of voice quality. This is applicable to audio, image and video applications over packet networks. The novelty in this work is in a new methodology to predict voice quality non-intrusively, nonlinear regression and neural network models for speech quality prediction, adaptive and perceived quality optimized jitter buffer algorithms, a new QoS control scheme that combines the strengths of adaptive rate and speech priority marking QoS control techniques, and Internet based subjective test methodology.

[Zia05] High Level Audio Communications API for the Unicon Language.

This MS.c. thesis presents a VoIP facility developed for unicon, a high level language to simplify the task of writing programs, reducing their development cost, and programming time.

Unicon’s VoIP interface is made as part of an audio communications API and designed to be minimal and consistent with the rest of the language. These goals keep the Virtual Machine (VM) size reasonable and reduce time a

programmer spends learning how to write VoIP applications by building a set of extended and added built-in functions (open(), close(), VAttrib(), PlayAudio(), StopAudio()). These functions were accomplished using an open source cross platform library called “Jori’s VoIP LIBrary” (JVOIPLIB).

Also unicon has been extended by the thesis to support VoIP application, a Collaborative Virtual Environment (CVE).

[Jos05] Using the Session Initiation Protocol as a Networking Protocol for Home Applications

Home appliances have evolved from being single task devices to integrating several tasks in a particular device. The next step for home appliances is to transform into networked appliances and accelerate the development of home automation.

This work has demonstrated the capabilities of the SIP and how, by the addition of a single method, called DO, in order to control these networked appliances. The ability of SIP being a protocol independent of the transport protocol underneath it, as well as the benefit that it uses existing infrastructure, makes it an ideal protocol to interwork devices that use different networking protocols. This is critical for home appliances since there are existing home automation protocols but none of them can interact with one another.

The study suggests that the only additional hardware needed is an appliance controller that works as a SIP user agent to connect an appliance or a network of appliances of similar technology to the home LAN. This controller, besides being SIP compliant, should have some kind of routing capability and it needs to translate the action received in a SIP message to the given technology.

[Dav05] Analysis and Implementation of TCP Friendly Rate Control in the Context of VoIP

In order to provide to User Datagram Protocol (UDP) a suitable congestion control mechanism for the media flow using it, a mechanism called Transmission Control Protocol (TCP) Friendly Rate Control (TFRC) has been developed. The main characteristics of this mechanism are throughput steadiness in short periods of time and long term fairness in the bandwidth sharing with TCP.

Based on this study TFRC represents a better option for VoIP applications than other TCP-LIKE smoothed mechanisms, but the proper implementation of it is through a new transport protocol such as Datagram Congestion Control Protocol (DCCP). Because of the lack of a standardized transport protocol that implements TFRC, so the most particular way to implement TFRC over a VoIP is through an RTP. The main problem of TFRC on the VoIP context is the difference in size between the VoIP packets and the TCP packets. This difference in the size makes extremely difficult to get the right values of the parameters that models the TCP behavior. If the parameters are not accurate then fairness would not be achieved. For the specific case of the virtual packets alternative, it is conclusive to say that this solution will not scale very well on the Internet because of the heterogeneous configuration of the routers. Another conclusion is that the suitability of TFRC regarding steadiness in relative short periods of time was verified, and it is very sensitive to variations in the delay.

[Jua07] Patterns for VoIP Signaling Protocol Architectures.

The paper presented two patterns that describe the architectures implied by the two main VoIP protocols. The Hybrid Signaling Protocol pattern

allows architectural and protocol flexibility by supporting both H.323 and SIP. These patterns complement the work in VoIP security patterns and provide a model of the environment where specific VoIP security patterns can be implemented, thus adding security to the structure. Patterns describing generic architectures can guide systems development.

1.7 Aim of the work

Study and analyze the architecture of new telecommunication telephony technology, which is VoIP. Specify the more suitable signaling protocol to be implemented on TCP/IP model, and apply a pre-shared key encryption method on the key field in the session description of signaling part.

1.8 Thesis Outlines

The thesis is organized in five chapters summarized as follows:

Chapter two analyzes the structure of TCP/IP model for VoIP technology and explains VoIP protocols place on the application layer. Chapter Three explains the three levels of VoIP security. Chapter Four illustrate the implementation of this thesis. Chapter Five includes the conclusions drown from this work followed by a list of suggestions for future works.

Chapter Two
VoIP Network Architecture and Protocols

2.1 Introduction

At the beginning of VoIP study is important to know the basic structure of the network with all the related protocols, which fall within the application layer.

When finishing this chapter, the following points will be clear:

1. Requirements for voice transport in data networks.
2. The functionality of TCP/IP model layers in context of using VoIP.
3. The best encapsulation in data link layer for the new technology.
4. Characteristics of IP protocol that makes it the base for VoIP Technology.
5. Which transport protocol suitable for sending signaling, and data packets?
6. Responsibility of application layer for signaling and media processing.
7. Signaling part
    a. Define the concept of call signaling in VoIP.
    b. Identify how Session Initiation Protocol (SIP) works?
    c. Identify Session Description Protocol (SDP), which used by SIP signaling protocol.
    d. Identify H.323 signaling protocol, as another option for signaling.
8. Media transport part
    a. Identify Real-time Transport Protocol (RTP).

b. Identify Real-time Transport Control Protocol (RTCP).

9. Clarifying the idea behind the separation of call control and media transport.

2.2 VoIP Network Architecture

The data network differs from other networks in that its sole purpose is providing connectivity. The purpose of the PSTN, for instance, is providing telephone services and the purpose of the TeleVision (TV) network is providing broadcasts. A variety of services such as e-mail, the World Wide Web, videoconferencing, and file transfer are implemented based on end-to-end IP connectivity [Gon02].

The requirements transporting voice over data networks drive the choice of transport protocol. It should be clear that TCP/IP is not appropriate because it favors reliability over timeliness, and our applications require timely delivery. A UDP/IP-based transport should be suitable, provided that the variation in transit time of the network can be characterized and loss rates are acceptable [Col03].

2.2.1 Requirements of Voice Transmission

When transmitting packets containing voice data, there must be some mechanism to preserve synchronization within the speech signal. The consecutive packets should be played at the right time, in the right order. This type of synchronization is called intra-media synchronization [Jor00].

The primary requirement of real-time media places on the transport protocol is for predictable variation in network transit time. Consider, for example, a VoIP system transporting encoded voice in 20-millisecond frames: The source will transmit one packet every 20 milliseconds, and ideally we

would like those to arrive with the same spacing so that the speech they contain can be played out immediately. Some variation in transit time can be accommodated by the insertion of additional buffering delay at the receiver, but this is possible only if that variation can be characterized and the receiver can adapt to match the variation.

A lesser requirement is reliable delivery of all packets by the network. Clearly, reliable delivery is desirable, but many audio and video applications can tolerate some loss: In previous VoIP example, loss of a single packet will result in a dropout of one-fiftieth of a second, which, with suitable error concealment, is barely noticeable. Because of the time-varying nature of media streams, some loss is usually acceptable because its effects are quickly corrected by the arrival of new data [Col03].

The speech data which has to be sent is typically generated at regular small intervals. It is possible that a receiving end cannot cope with this data flow, so somehow the sender should know whether the receiver can handle the incoming stream or not. A method that does this is often called a flow control method. Also, due to the fact that data is sent at a regular basis, it is not unlikely that a link becomes overloaded and congestion occurs. In turn, congestion causes the loss of packets and an increase in delay which are not desirable features for voice communication. The transmission component should be able to detect an arising congestion and take appropriate actions. The mechanism to prevent and control congestions is called congestion control.

The appropriate action for flow and congestion control is to decrease the amount of data sent. Typically, this is done in cooperation with the compression module. This will usually result in a degradation of speech

quality, but it is still better than having a lot of lost packets and a large delay [Jor00].

2.2.2 Network Multimedia Protocol Stack

The TCP/IP family of protocols forms the basis of the Internet and most current corporate networks, where the layered design of TCP/IP is not followed very strictly. Computer programs send and receive data over an IP network by making program calls to the TCP/IP software, known as the protocol stack, in their local computer. The TCP/IP stack in the local computer exchanges information with the TCP/IP stack in the target computer to accomplish the transfer of data from one side to the other [Joh04, Jor00].

Figure 2.1 shows the four-layer Internet Multimedia Protocol stack. The layers shown and protocols identified will be discussed.

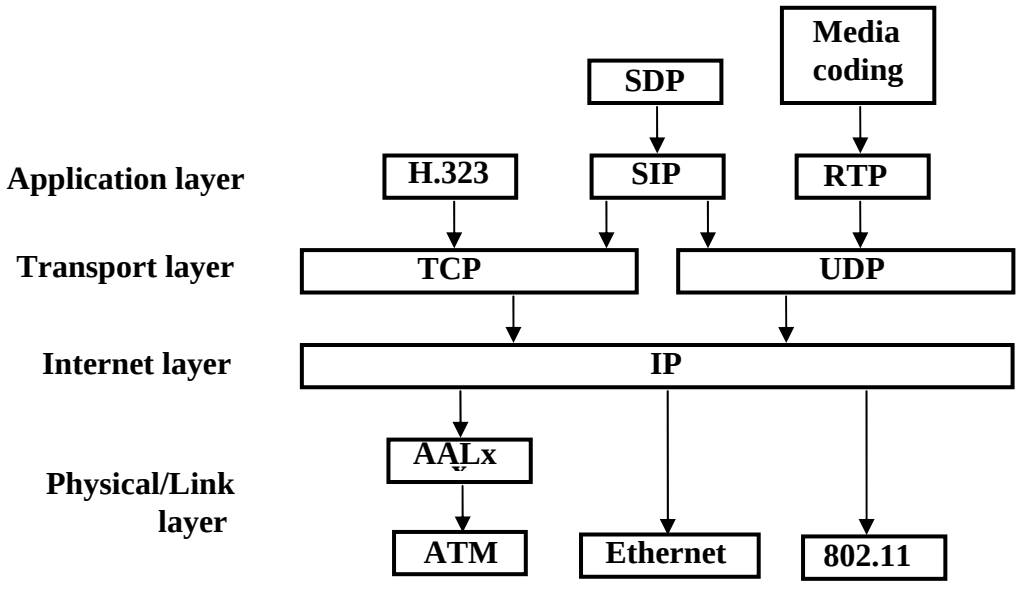

Figure 2.1 Network Multimedia Protocol Stack [Ala04].

2.2.2.1 Physical Layer/Link Layer

The lowest layer is the physical and link layer, which could be an Ethernet LAN, a telephone line (V.90 or 56k modem) running Point-to-Point Protocol (PPP), or a Digital Subscriber Line (DSL) running Asynchronous Transport Mode (ATM), or even a wireless 802.11 network. This layer performs such functions as symbol exchange, frame synchronization, and physical interface specification [Ala04].

ATM was created for time-sensitive traffic, providing simultaneous transmission of voice, video, and data. ATM uses cells that are a fixed 53 bytes long instead of packets. It also can use isochronous clocking (external clocking) to help the data move faster. Transit delays are reduced because the fixed-length cells permit processing to occur in the hardware. ATM is designed to maximize the benefits of high-speed transmission media [Ken07].

2.2.2.2 Internet Layer

The network layer connects links, unifying them into a single network.

It provides addressing and routing of messages through the network. It may also provide control of congestion in the switches, prioritization of certain messages, billing, and so on [Col03].

In order to achieve true end-to-end connectivity, a common end-to-end protocol is implemented at the network layer, the IP protocol. IP itself is a connectionless protocol. A connectionless packet switched network is the packet header contains the address of the ultimate destination to which the packet should be sent, on the other side of the network. The intermediate switches figure out the output port for the packet from routing tables based on

this ultimate destination address, which means that no reliability mechanisms, flow control, sequencing, or acknowledgments are present [Dav01, Gon02].

At this stage someone may be wondering why anyone would consider a packet-based audio or video application over an IP network. Such a network clearly poses challenges to the reliable delivery of real-time media streams. Although these challenges are real, an IP network has some distinct advantages that lead to the potential for significant gains in efficiency and flexibility, which can outweigh the disadvantages [Col03].

This protocol was designed mostly for data transport, and it has only limited QoS support. The main reason IP is so important is because of its omnipresence. The TCP/IP architecture has proved to be very popular and nowadays it is very widely used. This fact gives IP a great advantage over other protocols [Jor00].

You can address an IP packet in three general ways: through unicast, multicast, or broadcast mechanisms. Briefly explained, these three mechanisms provide the means for every IP packet to be labeled with a destination address, each in its unique way. Unicast packets enable two stations to communicate with each other, regardless of physical location. Broadcast packets are used to communicate with everyone on a subnetwork simultaneously. Multicast packets enable applications, such as videoconferencing, that have one transmitter and multiple receivers.

IP networks also offer the potential for higher reliability because IP networks automatically re-route packets around problems such as malfunctioning routers or damaged lines. Also, IP networks do not rely on a separate signaling network, which is vulnerable to outages [Jon00].

IP is used by both routers and hosts, relegating intelligence to the end systems. It tends to keep the state information stored inside the network at minimum in order to scale better and to be more robust. This lack of state in the network makes node failures less dramatic because they do not store any state information necessary for end-to-end communication [Gon02].

Another advantage is the possibility of compression. With the compression methods available today, it is possible to reduce the requirement of (64 kbps) for uncompressed telephone-quality voice communication to amounts which are far lower. However, a high compression ratio often means that the voice signal will be of lesser quality [Ken07].

2.2.2.3 Transport Layer

The transport layer is the first real end-to-end layer. It takes responsibility for delivery of messages from one system to another, using the services provided by the network layer. This responsibility includes providing reliability and flow control if they are needed by the application layer and not provided by the network layer [Col03, Jor00].

There are two commonly used transport layer protocols, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP).

A. TCP

TCP is known as a connection-oriented protocol because the two sides of the data exchange maintain strong tracking of everything that is sent and received. Connection-oriented messages are sent through the network from source to destination requesting a connection to be set up. These may be signaling messages from the customer or messages initiated by the network management system. [Col03, Joh04].

If TCP were utilized for VoIP, the latency incurred waiting for acknowledgments and retransmissions would render voice quality unacceptable. TCP transport makes the assumption that packet loss is a signal that the bottleneck bandwidth has been reached, congestion is occurring, and it should reduce its sending rate. A TCP flow will increase its sending rate until loss is observed, and then back off, as a way of determining the maximum rate a particular connection can support. Of course, the result is a temporary overloading of the bottleneck link, which may affect other traffic. Also, since TCP preserves the order of the packets. The application has to output speech data at regular intervals, so if one packet stays lost for a sufficient amount of time, this will block the playback of other packets, even when they have already arrived [Col03, Jor00].

TCP is used to ensure the reliability of the setup of a call. Due to the methods by which TCP operates, it is not feasible to use TCP as the mechanism to carry the actual voice in a VoIP call. With VoIP and other real-time applications, controlling latency is more important than ensuring the reliable delivery of each packet. Besides, the protocol is designed for communication between two hosts, so it only supports unicasting. If data has to distributed to several destinations, it has to be done using separate TCP connections. This, of course, wastes a lot of bandwidth [Jon00, Jor00].

B. UDP

UDP is called a connectionless protocol, since there is no acknowledgment of sent datagrams. Most of the complexity of TCP is not present, including sequence numbers, acknowledgments, and window sizes. UDP does detect errored datagrams with a checksum. It is up to higher layer

protocols, however, to detect this datagram loss and initiate a retransmission if desired [Ala04, Joh04].

The protocol has the advantage of not having to wait for retransmissions of lost packets. Also, since it is only a small extension to IP, it can make use of the IP multicasting features and save bandwidth when data has to be sent to multiple destinations. As good as all this may seem, there are also some disadvantages: UDP provides no mechanism for synchronization whatsoever and there are no means for flow or congestion control [Jor00].

Applications which do not require the functionality that TCP provides can use UDP. To transmit data, the UDP module simply passes a UDP header followed by that data to the internet layer which then sends the datagram on its way. This means that just like IP itself, UDP is a best-effort service. No guarantees about delivery are given, datagrams can get reordered and datagrams can be duplicated. So that UDP is used in VoIP to carry the actual voice traffic (the bearer channels) [Jon00].

2.2.2.4 Application Layer

The top layer shown in Figure (2.1) is the application layer. This includes signaling protocols such as Session Initiation Protocol (SIP) and media transport protocols such as Real-time Transport Protocol (RTP), which is introduced in chapter three. Figure (2.1) includes H.323, which is an alternative signaling protocol to SIP developed by the International Telecommunication Union (ITU). Session Description Protocol (SDP) is shown above SIP in the protocol stack because it is carried in a SIP message body. HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and TELecommunications NETwork

(Telnet) are all examples of application layer protocols. Because SIP can use any transport protocol, it is shown interacting with both TCP, and UDP in Figure 2.1 [Ala04].

Using VoIP, those silent intervals can be detected. The VoIP application can examine each packet and detect whether it contains speech information or only silence. If the latter is the case, the packet can simply be discarded [Jor00].

2.2.3 Client\Server versus Peer-to-Peer Architecture

When users communicate with one another, they send requests and responses to one another directly or through a specific server. However, this reflects two different types of architectures used in network communications [Tho06]:

1. Client\Server
2. Peer-to-peer

2.2.3.1 Client\Server

The client/server architecture uses a call agent to control signaling on behalf of the endpoint devices, such as gateways. The central control device participates in the call setup only. Voice traffic still flows directly from endpoint to endpoint [Ant06].

In client\server architecture, the relationship of the computers is separated into two roles:

1. The client, which requests specific services or resources.
2. The server, which is dedicated to fulfilling requests by responding (or attempting to respond) with requested services or resources.

An easy-to-understand example of a client/server relationship is seen when using the Internet. When using an Internet browser to access a web site, the client would be the computer running the browser software, which would request a web page from a web server. The web server receives this request and then responds to it by sending the web page to the client computer. In VoIP, this same relationship can be seen when a client sends a request to register with a registrar server, or makes a request to a proxy server or redirect server that allows it to connect with another user agent. In all these cases, the client's role is to request services and resources, and the server's role is to listen to the network and await requests that it can process or pass onto other servers.

The servers that are used on a network acquire their abilities to service requests by the programs installed on it. Because a server may run a number of services or have multiple server applications installed on it, a computer dedicated to the role of being a server may provide several functions on a network. For example, a web server might also act as an e-mail server. In the same way, SIP servers also may provide different services. A Registrar can register clients and also run the location service that allows clients and other servers to locate other users who have registered on the network. In this way, a single server may provide diverse functionality to a network that would otherwise be unavailable

Another important function of the server is that, unlike clients that may be disconnected from the Internet or shutdown on a network when the person using it is done, a server is generally active and awaiting client requests. Problems and maintenance aside, a dedicated server is up and running, so that it is accessible. The IP address of the server generally doesn't change,

meaning that clients can always find it on a network, making it important for such functions as finding other computers on the network [Tho06].

2.2.3.2 Peer-to-Peer

A peer-to-peer (P2P) architecture is different from the client/server model, as the computers involved have similar capabilities, and can initiate sessions with one another to make and service requests from one another. Each computer provides services and resources, so if one becomes unavailable, another can be contacted to exchange messages or access resources. In this way, the user agents act as both client and server, and are considered peers.

Once a user agent is able to establish a communication session with another user agent, a P2P architecture is established where each machine makes requests and responds to the other. One machine acting as the user agent client will make a request, while the other acting as the user agent server will respond to it. Each machine can then swap roles, allowing them to interact as equals on the network. For example, if the applications being used allowed file sharing, a user agent client could request a specific file from the user agent server and download it. During this time, the peers could also be exchanging messages or talking using VoIP, and once these activities are completed, one could send a request to terminate the session to end the communications between them. As seen by this, the computers act in the roles of both client and server, but are always peers by having the same functionality of making and responding to requests [Tho06].

2.3 VoIP Protocols

Over the years, a need was seen for a standard protocol that could allow participants in a chat, videoconference, interactive gaming, or other media to initiate user sessions with one another. In other words, a standard set of rules and services was needed that defined how computers would connect to one another so that they could share media and communicate [Tho06].

The variables in VoIP are the signaling methods. H.323 and SIP define end-to-end call signaling methods. MGCP and H.248 define a method to separate the signaling function from the voice call function. This approach is referred to as client\server architecture for voice signaling.

A constant in VoIP implementation is that voice uses RTP inside UDP to carry the payload across the network. IP voice packets can reach the destination out of order and unsynchronized; the packets must be reordered and resynchronized before playing them out to the user. Because UDP does not provide services such as sequence numbers or time stamps, RTP provides the sequencing functionality [Ant06].

It is necessary to make sure that standards-based protocols are used, so the bearers (RTP streams) are separated from the call-control. Data networking is unique in the fact that multiple protocols can co-exist in a network and you can tailor them to the particular needs of the network [Jon00].

2.3.1 Call Signaling

Signaling is the fundamental to the call establishment, management, and administration of voice communication in an IP network. The term “signaling” is not self-explanatory in VoIP telephony and clarifications are

always necessary, depending on the application. The typical way to make call on the PSTN is to dial digits on the keypad. If the call is going to be successfully completed, it will hear a ringing tone until the party which is trying to reach answers the phone. By introducing VoIP into the call path, the end-to-end path involves at least one call leg that uses an IP internetwork. As in a traditional voice call, support for this VoIP call leg requires two paths [Bil00, Kev07]:

1. A protocol stack that includes RTP, which provides the audio call leg.
2. One or more call control models that provide the signaling path, such as SIP, H.323, or MGCP.

So that the signaling is independent set of actions to the media flow, illustrated in Figure (2.2); it controls the type of media used in a call. Signaling does not necessarily stop when the call is set up, until one or more participants in a call depart. Signaling can occur while a call is active, for example to modify session parameters, and can be concurrent with the media flow [Ala04, Bil00].

The main signaling functions of the protocol are as follows:

1. User location: Location of an end point to be used for communication;
2. User availability: Contacting an end point to determine willingness to establish a session;
3. User capability: Exchange of media information to allow session to be established. Example, SIP uses the Session Description Protocol (SDP) for negotiating media parameters, while H.323 uses Abstract Syntax Notation One (ASN.1);

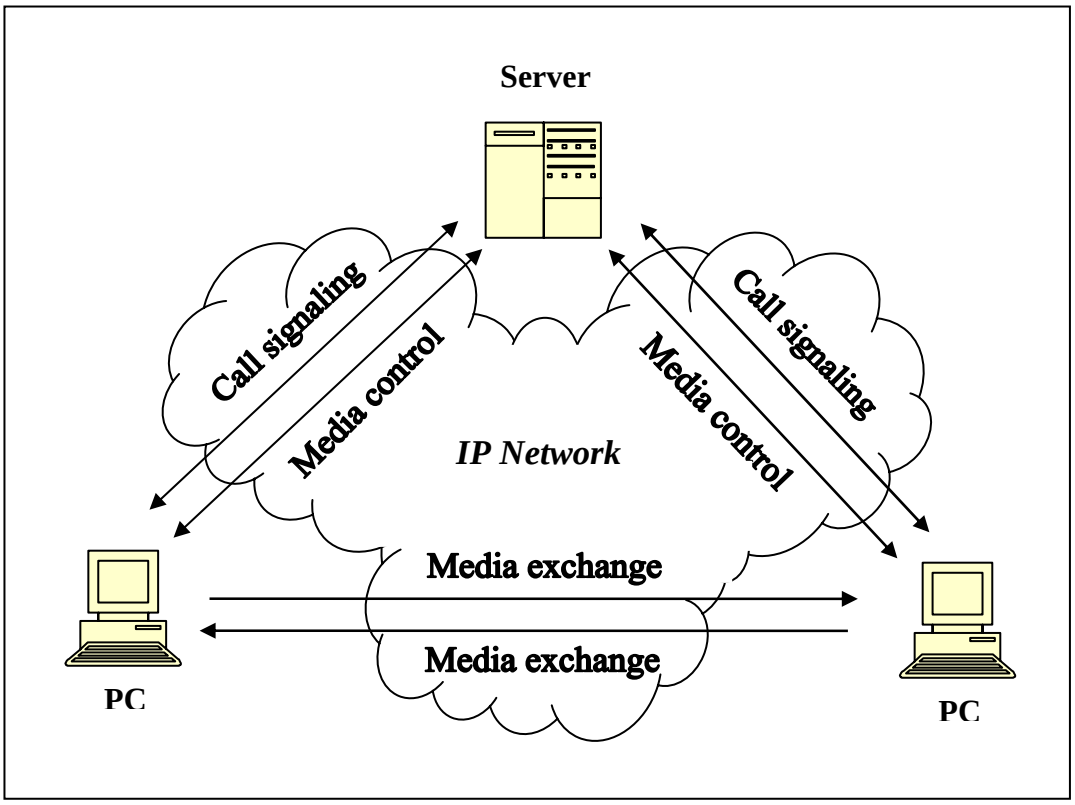

Figure 2.2 Packet-Switched Call [Chr02]

4. Session setup: "ringing", establishment of session parameters at both called and calling party;
5. Session management: including transfer and termination of sessions, modifying session parameters, and invoking services.

2.3.2 H.323

H.323 is developed by International Telecommunication Union-Telecommunication (ITU-T)**.** H.323 is an umbrella-like specification that encompasses a large number of state machines that interact in different ways depending relationship of participating entities and the type of session (for example, audio or video). There are many subprotocols within the H.323 specification. In order to understand the overall message flows within an

H.323 VoIP transaction, figure 2.3 shows the relevant protocols and their relationships.

H.323 protocols are binary protocols. There functionalities are [Eri02]:

1. **H.225/Registration Admission and Status (RAS):** used over UDP to transmit registration, admission, bandwidth changes, and status messages to the GateKeeper.

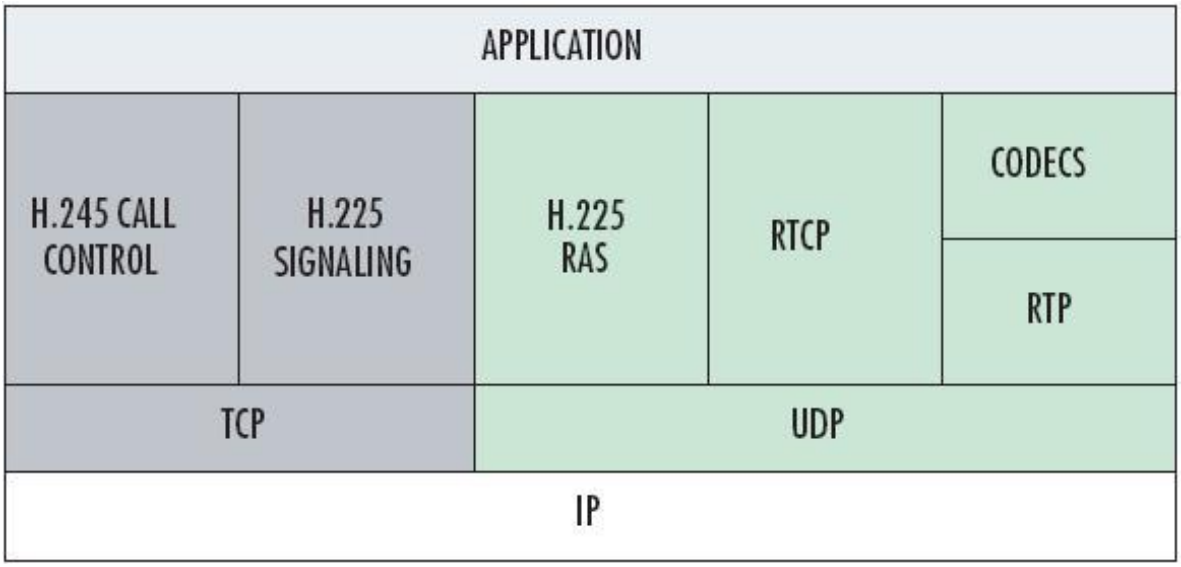

Figure 2.3 H.323 Protocol Stack [Eri02]

2. **H.225 signaling:** Defines signaling for call setup and teardown, including source and destination IP addresses, ports, country code, and H.245 port information.
3. **H.245:** Specifies messages that negotiate the terminal capabilities set, the master/slave relationship, and logical channel information for the media streams.

2.3.2.1 H.323 Components

The overall H.323 protocol has been introduced; let's now turn the attention to components that make up the H.323 protocol. These components are [Tho06]:

1. **Terminals**: it also referred to as endpoints, provide the user interface into the H.323 protocol and provide real-time, two-way multimedia communications. Typically, the devices (telephones, softphones, voice mail, etc.) are that end-users interact with; Microsoft NetMeeting is an example of an H.323 endpoint.
2. **GateKeepers (GKs):** which are optional provide call control functions such as address translation and bandwidth management and are often considered to be the most important component in the H.323 stack.
3. **Multipoint Control Units (MCUs):.**provide conference facilities for users who want to conference three or more endpoints together.

2.3.2.2 H.323 Operation

H.323 signaling exchanges typically are routed via gatekeeper or directly between the participants as chosen by the gatekeeper. Media exchanges normally are routed directly between the participants of a call. H.323 data communications utilizes both TCP and UDP. TCP ensures reliable transport for control signals and data, because these signals must be received in proper order and cannot be lost. UDP is used for audio and video streams, which are time-sensitive but are not as sensitive to an occasional dropped packet. Consequently, the H.225 call signaling channel and the H.245 call

control channel typically run over TCP, whereas audio, video, and RAS channel exchanges rely on UDP for transport [Eri02].

2.3.3 Session Initiation Protocol (SIP)

SIP was originally developed by the Internet Engineering Task Force (IETF) working group. 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. The Internet-Draft containing bug fixes and clarifications to SIP were submitted beginning in July 2000, referred to as RFC (2543) "bis". This document was eventually published as RFC (3261), which obsoletes (or replaces) the original RFC (2543) specification [Ala04].

SIP is described as an application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. Although no real assumptions are made about the underlying network and protocols, SIP has been designed with the TCP/IP architecture in mind; SIP is a text based signaling protocol. It incorporates elements of two widely used Internet protocols: HTTP used for web browsing and SMTP used for e-mail. From HTTP, SIP borrowed a client-server design and the use of URLs and Uniform Resource Identifiers (URIs). From SMTP, SIP borrowed a text-encoding scheme and header style. For example, SIP reuses SMTP headers such as To, From, Date, and Subject [Ala04].

It supports features of the Advanced Intelligent Network (AIN). Such as name mapping, call forwarding and call redirection. This is very useful if SIP is to gain acceptance as a signaling protocol in the public network, where

telephony feature offering is major part of the business of telephone companies. Another significant feature of SIP is support for user mobility. So, SIP is the early signaling protocols intended for serious VoIP telephony signaling in the wide area [Bil00].

2.3.3.1 SIP Entities

The SIP protocol defines several entities, and it's vital to understand their role inside any architecture that uses SIP [Gon02]. These entities are:

A. User Agents

A User Agent (UA) is an application which resides at a SIP end station. A SIP UA must support UDP transport and also TCP if it sends messages greater than 1,000 octets in size, also must support SDP for media description. Other types of media description protocols can be used in bodies, but SDP support is mandatory [Ala04].

UA consists of two parts: a User Agent Client (UAC) and a User Agent Server (UAS). A UAC is capable of generating a request based on some external stimulus (the user clicking a button, or a signal on a PSTN line) and processing a response. The UAS is a server application which contacts the user when there is an incoming request and responds to it, it is capable of receiving a request and generating a response based on user input, external stimulus, the result of a program execution, or some other mechanism. In most cases, the user will be a human, but the user could be another protocol, as in the case of a gateway. A user agent must be capable of establishing a media session with another user agent [Ala04, Ros02].

B. SIP Servers

SIP servers are applications that accept SIP requests and respond to them. 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, Transport Layer Security (TLS), and UDP for transport [Ala04].

Three types of network servers are defined. The first one is a redirect server. A user can send a call invitation request for another person to a redirect server. This server will then locate the user and return the necessary information to enable the caller to establish a call with the intended person.

The second type of server is a proxy server. SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients. It is like with a redirect server, a user can send an invitation request to a proxy server. The proxy server will also try to locate the destination of the call, but unlike with a redirect server, it will not simply return possible locations of the called person. Instead, based upon that information, a proxy server will try to establish a connection on behalf of the caller. A proxy can operate in either a stateful or stateless mode for each new request. When stateless, a proxy acts as a simple forwarding element. But in a stateful proxy remembers information about each incoming request and any requests it sends as a result of processing the incoming request. It uses this information to affect the processing of future messages associated with that request [Gon02].

Finally, the last server type is a called a registrar. A registrar is usually co-located with a redirect server or a proxy server. A user can send

information about its current location to a registrar; the user can register himself. This information can then be used to contact him. Thanks to registration information, personal mobility is allowed, which means that a person should be able to accept calls directed to him at any end system. The information sent to a registrar describes at which system a user should be contacted [Jor00].

2.3.3.2 How SIP Works

Any user must specify to whom he want to make a call. A SIP user is identified by a SIP-URI, example: ‘sip:user2@there.com’. SIP has two broad categories of URIs: ones that correspond to a user, and ones that correspond to a single device or end point. The user URI is known as an Address Of Record (AOR) and a request sent to an address of record will require database lookups and service and feature operations and can result the request being sent to one or more end devices. A device URI is known as a contact, and typically does not require database lookups [Ala04].

When a user wants to invite someone into a session or wants to make a call to someone, the user can send an invitation request to the end system specified in the destination's SIP-URI. The request would be sent to ‘there.com’. If the called user is available at that system, he can send a response, indicating whether he wants to participate in the communication or not. When the caller receives this response, he sends an acknowledgement to the other system.

As shown in figure (2.4), the caller could also send its invitation request to a proxy server. This proxy server then looks for possible locations of the other user and tries to invite that user itself. When the proxy knows that

the invitation was either accepted or denied, it can send an appropriate response back to the caller. This way, a proxy acts as both a client and a server.

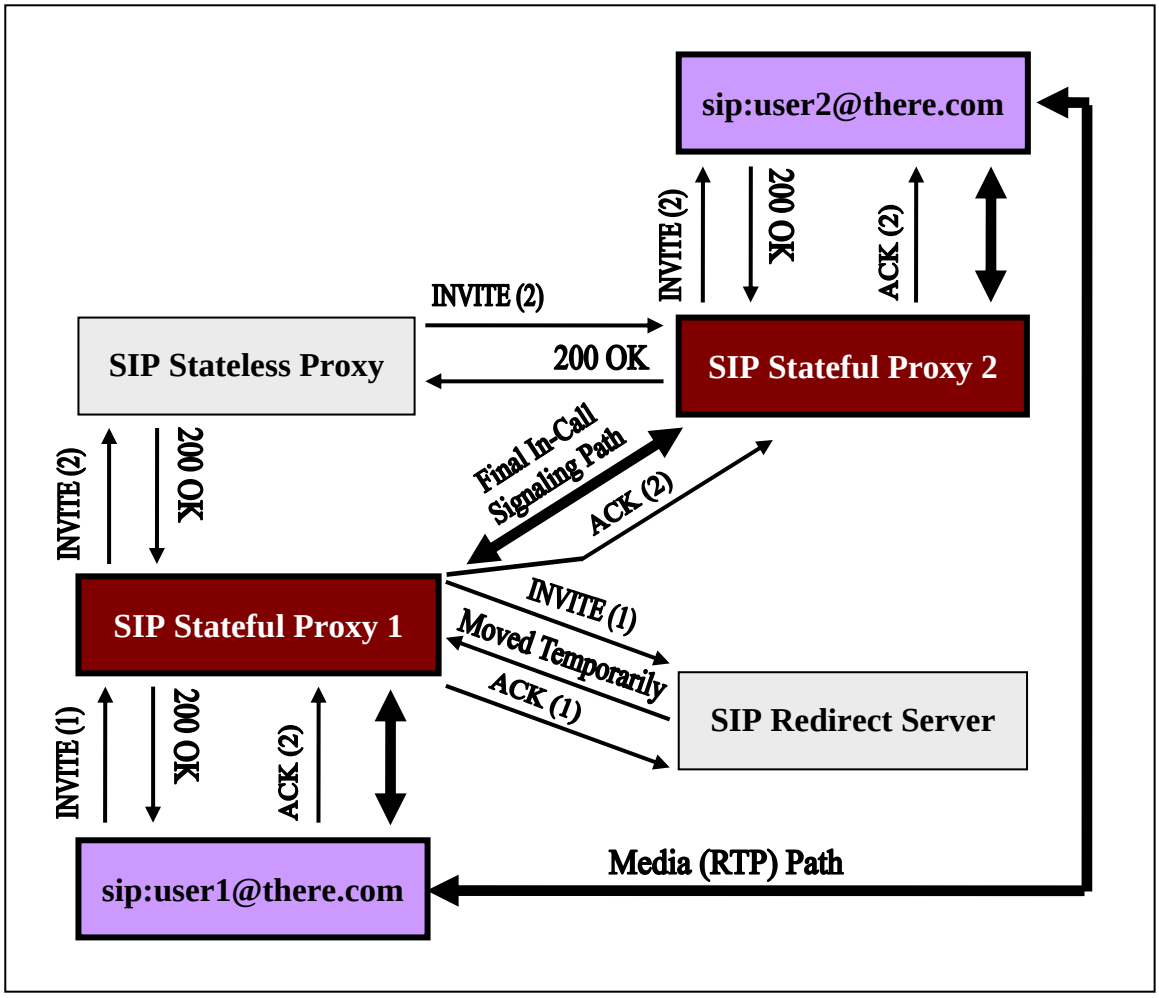

Figure 2.4 SIP Protocol Operations [Gon02]

Finally, the invitation request could also be sent to a redirect server. This redirect server would then look for possible locations of the called user and send the corresponding SIP-URIs back to the caller. Based upon this information, the caller could then try to contact the other user directly, as described in figure (2.4).

The invitation request normally contains information about the media that will be sent. If the invitation was successful, the response will also contain a description about the media that the other user will use. The SIP specification does not demand a specific format, but the SDP was designed for this purpose [Jor00].

2.3.4 Session Description Protocol (SDP)

The SDP protocol was developed by the IETF work group. The original purpose of SDP was to describe multicast sessions set up over the Internet's multicast backbone. SDP specifies how the information necessary to describe a session should be encoded. It does not include any transport mechanism or any kind of parameter negotiation. A SDP description is simply a chunk of information that a system can use to join a multimedia session [Gon02]. It contains the following information about the media session:

1. IP Address (IPv4 address, IPv6 address, or host name);
2. Port number (used by UDP or TCP for transport);
3. Media type (audio, video, fax, and so forth);
4. Media encoding scheme (Pulse Code Modulation (PCM), Moving Picture Experts Group II video (MPEG II video), and so forth).

In addition, SDP contains information about the following:

1. Subject of the session;
2. Start and stop times;
3. Contact information about the session.

Like SIP, SDP uses text coding. A 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 [Ala04].

2.3.5 Real-time Transport Protocol (RTP)

The Real-time Transport Protocol is defined as a protocol which provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. So this protocol can also be used for VoIP applications. It is the key standard for audio/video transport in IP networks along with its associated profiles and payload formats [Col03].

As shown in the protocol stack of Figure (2.1), RTP is an application layer protocol that uses UDP for transport over IP. RTP is not text encoded, but uses a bit-oriented header similar to UDP and IP [Ala04].

The RTP specification actually defines two separate protocols: the RTP protocol, and Real-time Transport Control Protocol (RTCP). The protocols themselves do not provide mechanisms to ensure timely delivery. They also do not give any QoS guarantees. These things have to be provided by some other mechanism. Flow and congestion control are not directly supported [Jor00].

The strength of RTP is that it provides a unifying framework for real-time audio/video transport, it has proven useful for a range of other applications: in H.323 video conferencing, and TV distribution; and in both wired and cellular telephony [Col03].

2.3.5.1 RTP

RTP aims to provide services useful for the transport of real-time media, such as audio and video, over IP networks. These services include

timing recovery, loss detection and correction, payload and source identification, media synchronization, and membership management. RTP was originally designed for use in multicast conferences, using the lightweight session's model.

A session consists of a group of participants who are communicating using RTP. A participant may be active in multiple RTP sessions—for instance, one session for exchanging audio data and another session for exchanging video data. For each participant, the session is identified by a network address and port pair to which data should be sent, and a port pair on which data is received. The send and receive ports may be the same. Each port pair comprises two adjacent ports: an even-numbered port for RTP data packets, and the next higher (odd-numbered) port for RTCP control packets. The RTP Payload data in a typical voice-over-IP application sending an audio in 20-millisecond packets [Ala04, Col03].

2.3.5.2 RTCP

The RTP is accompanied by a control protocol, RTCP protocol. The control protocol supplies information about the participants in the session. Each participant of a RTP session periodically sends RTCP packets to all other participants in the session. RTCP has three basic functions:

1. The primary function is to provide feedback on the quality of data distribution. Such information can be used by the application to perform flow and congestion control functions.
2. RTCP distributes an identifier which can be used to group different streams audio and video for example together. Such a

mechanism is necessary since RTP itself does not provide this information.

3. By periodically sending RTCP packets, each session can observe the number of participants. The RTP data cannot be used for this since it is possible that somebody does not send any data, but does receive data from other participants. For example, this is the case in an on-line lecture.

Also, RTCP provides information about reception quality which the application can use to make local adjustments. For example if a congestion is forming, the application could decide to lower the data rate [Jor00].

RTCP defines several different packet types [Joh02]:

1. Sender Report (SR): is a sender report and conveys statistical data of an active sender.
2. Receiver Report (RR): is a report of a receiver for statistics of a participant that does not actively send data.
3. Source DEScription (SDES): it is contains source description items.
4. (BYE): it is indicates the end of participation.
5. Application-sPecific Packet (APP): it is consists of application specific data.

Chapter Three
SIP-Based VoIP Security

3.1 Introduction

The security becoming an increasingly important issue in modern day computer environment, it is becoming vital to consider how to protect a system before incorporating it in daily business operations. Due to the increasing interest in computer based media communication. (e.g., VoIP application) there is need for security solutions which makes these technologies reliable enough to carry important information.

The ideas produced by this chapter are:

1. Identify four levels of VoIP security that can be implemented.
2. Producing two common key agreement protocols
3. Clarify the point of using key agreement protocol with SDP.
4. Identify the reason of using SDP in security media description.

3.2 Encryption Protocols

One encryption protocol can be used in different context for slightly different purposes. There are some common encryption protocols for encryption at different layers as described in later sections.

3.2.1 IPsec: Network Layer Encryption

The IP security protocol suite known as (IPsec), which provides a security level at the IP level (in network layer). The preferred form of Virtual Private Network (VPN) tunneling across the internet, IPsec defines two basic protocols: Encapsulating Security Payload (ESP) and Authentication Header

(AH). IPsec provide connectionless integrity, source authentication, confidentiality and replay protection. Given that IPsec can provide these services to an entire IP packet, including the header [Tha98].

The SIP protocol does not specify a framework for the use of IPsec and no key management is suggested. The most common use of IPsec is in collaborative with the Internet Key Exchange (IKE) protocol to provide automated cryptographic key exchange and management mechanisms [Ros02].

3.2.2 TLS: Transport Layer Encryption

Transport Layer Security (TLS) is a security protocol which provides encryption at the transport layer. The protocol specifically requires a connection oriented, reliable delivery transmission protocol which means it will not work with protocols using UDP transmission. TLS provides integrity protection, authenticity and confidentiality of sent data without needing additional key management.

TLS is compromised of two layers; the TLS Record Protocol and the TLS Handshake Protocol. The Handshake Protocol is used to authenticate the participants and negotiate security parameters while the Record Protocol provides confidentiality and integrity to the actual data transfer [Die99].

The SIPS (Secure SIP) in SIP specification implies the use of TLS there is a great probability that an application securing SIP messages makes use of TLS. This is might make TLS seem like a good alternative for encryption of media stream because no other security protocol would need to be implemented. This would be false assumption since TLS uses the reliable transport protocol, such as TCP. While this might not be a great problem

where SIP messages is concerned it is devastating in the context of real-time media encryption. The nature of streaming media simply does not allow the use of a reliable delivery protocol such as TCP for transportation due to time constraints [Ros02].

3.2.3 SRTP: Application Layer Encryption

The Secure Real-time Transport Protocol (SRTP) is an extension to the RTP Audio/Video profile and provides confidentiality, authenticity, integrity and replay protection for RTP and RTCP packets, providing all the important elements to secure the media stream. The RTP packets are used to carry the session contents while the control packets, RTCP, are used for session statistics and control. A secure session key derivation function is used to produce pseudo-random session keys using only a master key and an optional (highly recommended) master salt [SC03, Bau04].

The key derivation function in SRTP enables session keys to be created using a master key, and to protect against pre-computation attacks, a master salt. This function is used to create a session encryption key, session authentication key and session salt to use when processing packets. In fact, both the SRTCP and SRTP stream can be provided with session keys using only the master key and salt. This function also enables the definition of the optional key derivation rate in the SRTP protocol which specifies how often new keys are to be generated. This is useful because an application sending data for along period of time might wish to use several session keys so that one leaked or cryptographically broken key will only compromise part of the packet stream. This is then easily done by defining a key derivation rate higher than zero [Bau04].

3.2.3.1 Default Encryption Algorithms

In principle any encryption scheme can be used with SRTP. As default algorithms the NULL cipher (no confidentiality) and the Advanced Encryption Standard in Counter Mode (AES-CTR) are defined. The AES-CTR encryption setup is shown in Figure (3.1).

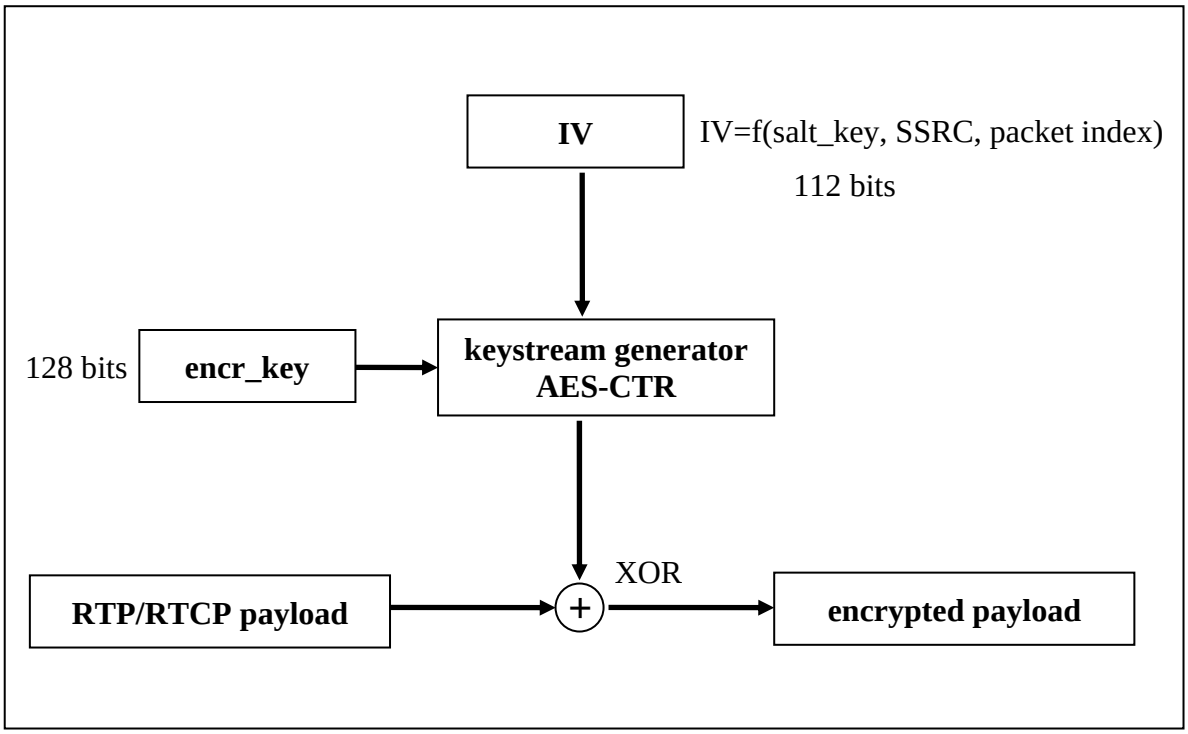

Figure 3.1 Encryption using AES in Counter Mode [And04].

AES in counter mode acts as a keystream generator producing a pseudo-random keystream of arbitrary length that is applied in a bit-wise fashion to the RTP/RTCP payload by means of a logical XOR function, thus working as a classical stream cipher. AES itself is a block cipher with a block size of 128 bits and a key size of 128, 192, or 256 bits. In order to work as a pseudo-random generator AES is loaded at the start of each RTP/RTCP packet with a distinct Initialization Vector (IV) that is derived by hashing a 112 bit salt_key, the Synchronization SouRCe identifier (SSRC) of the media

stream and the packet index (header fields of the media packet header). Encrypting this IV results in an output of 128 pseudo-random bits. Next the IV is incremented by one and again encrypted, thus generating the next 128 bits of the keystream. By counting the IV up by increments of one as many keystream blocks can be generated as are required to encrypt the whole RTP/RTPC payload. Any remaining bits from the last keystream block are simply discarded.

AES used in counter mode instead of the more common cipher block chaining mode (CBC) has the big advantage that the keystream can be precomputed before the payload becomes available thus minimizing the delay introduced by encryption. And of course by using a stream cipher instead of block cipher there is no need to pad the payload up to a multiple of the block size which would add 15 overhead bytes to the RTP/RTCP packet in the worst case [And04].

3.2.3.2 Session Key Derivation

The encryption algorithm described in sections (3.2.3.1) require secret symmetric session keys that must be known to all user agents participating in a SIP session. This raises the logistical problem of session key generation and distribution.

The SRTP standard offers a partial solution by deriving all needed session keys from a common master key but leaves open the distribution of the master key itself. Figure (3.2) shows how the session keys are computed starting out from a single master key. Again the AES block cipher is used in counter mode to generate the necessary keying material. The master key which can have a size of 128, 192, or 256 bits plays the role of the AES

encryption key. The pseudo-random generator is loaded with an IV that is itself a function of a 112 bit master_salt value, a one byte label and a session key number. By applying the labels 0x00 up to 0x05, encryption, authentication and salting keys for both SRTP and SRTCP are derived from the same master key. If a key derivation rate has been defined then every time a number of packets equivalent to the key derivation rate have been sent, a new set of either SRTP or SRTCP session keys are computed. If the key derivation rate is set to zero then the same set of keys is used for the whole duration of the session.

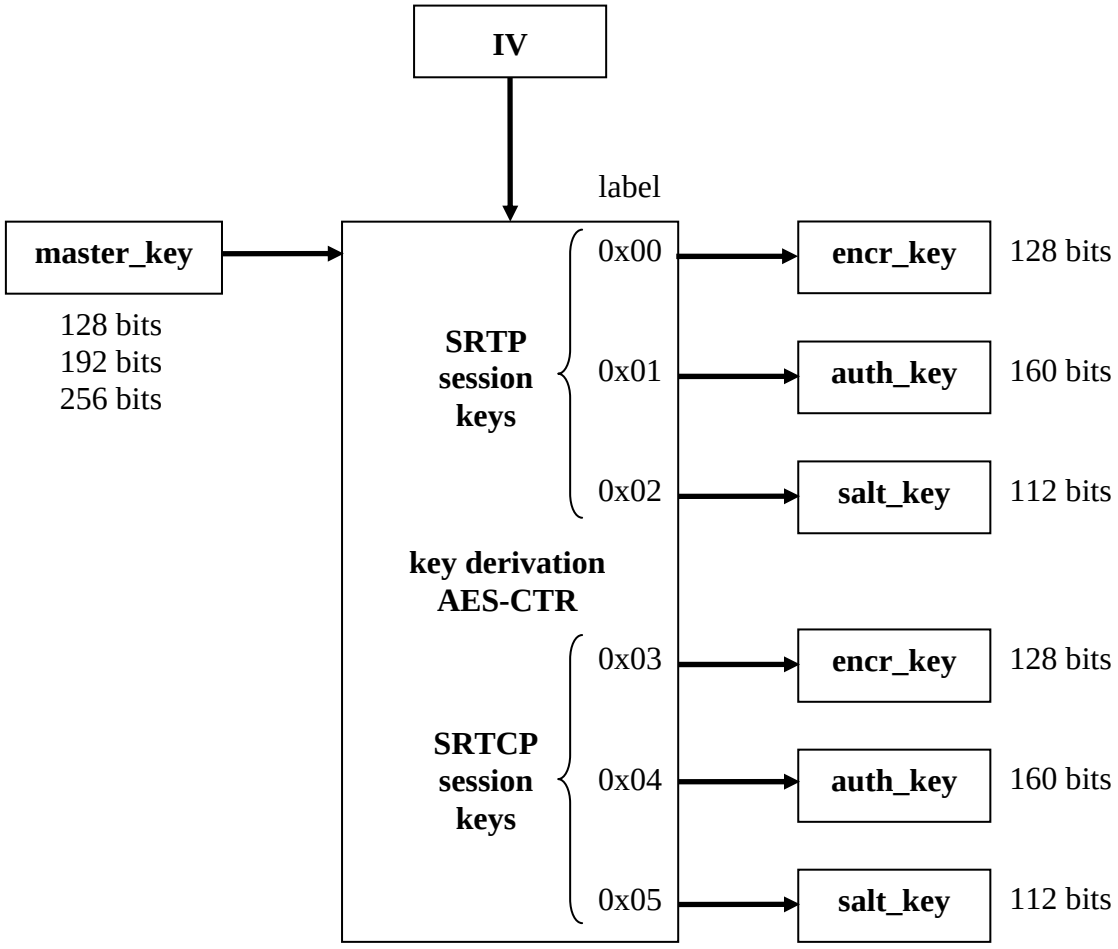

Figure 3.2 Session Key Derivation [And04]

3.2.3.3 Master Key Distribution

We turn now to the crucial issue of distributing the master key to the user agents as part of the session initiation where no key management is defined in SRTP, external key management mechanisms are used to exchange keys and cipher-suite information and parameters. This makes SRTP used within a SIP context reliant on external key management protocol or SDP negotiation. Because of this dependency, and because SRTP is such a useful protocol for secure real time media transmission purposes, the SDP protocol is geared towards providing this information to SRTP. This is not to say it cannot be used with other protocols, but the intended use upon their creation was with SRTP [Ark05].

By requiring external key management using SRTP creates requirement for an encrypted exchange of key data which cannot be satisfied by SRTP itself. However, since SDP tunneling of this information is possible the encryption of SDP data or SIP messages, which should be present, would also serve to protect tunneled key agreement information. The benefit of doing this, in addition to piggyback in the SIP security, is that using the SDP information makes configuration of or changes to the security mechanisms easy to implement [Ark05].

3.3 Key Agreement

All types of encrypted communication require both participants to agree upon how to perform encryption and decryption. Specifically, participants need to know what crypto-suite, keys and crypto-suite parameters are being used. To provide these services to protocols that do not themselves cover this area (such as SRTP or IPsec) key agreement protocols are needed.

When it comes to exchange of the actual keying information there are a few well known and common mechanisms used within key management protocols. These mechanisms are needed because of the paradox that the key information must not be sent in the clear but the information itself is needed for encryption. Exactly how these mechanisms are applied is up to the key management protocol that uses them, but the basics of the methods are described below [Ark05].

3.3.1 Pre-Shared Key Agreement

It is the most basic of key agreement protocol, pre-shared key requires that both participants share a secret key. Both participants use this key to provide an encryption key. The initiator creates and sends a randomly generated session key encrypted with the generated encryption key and thus gets access to the session key included in the message.

This mechanism as in figure (3.3) requires of maintaining a shared secret key with an intended participant. This scheme does not scale well because each user would need to maintain a key for each possible recipient, in addition to requiring some means of obtaining these keys. A pre-shared key scheme is therefore only suitable in a scenario where few users need to communicate.

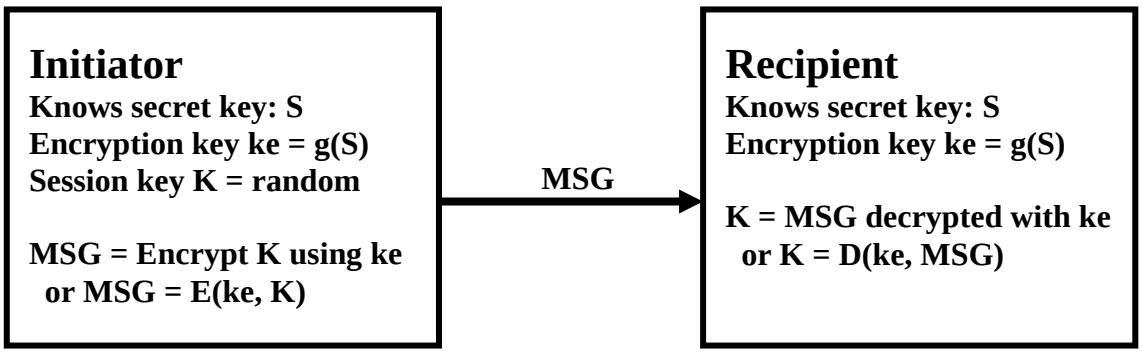

Figure 3.3 Pre-Shared Key Agreement Protocol [Ark05]

Because a pre-shared key can be used for a long period of time, and because refreshing this key can be arduous, many key exchanges can be performed using the same key. What this means is that if the secret key is disclosed to an authorized party all future transactions using that key are compromised until the key is replaced [Ark05].

3.3.1.1 Feedback Shift Registers

A finite state machine (FSM) consists of finite sets of (internal) states $\{s\}$, input and output alphabets $\{a\}$ and $\{b\}$, an output function T determining the output

$$T : (s, a) \rightarrow b,$$

and a state function $\sum$ determining the successor state.

$$\sum : (s, a) \rightarrow s^* = \sum(s, a).$$

Given an initial internal state s_0, and sequence of input states $a_0, a_1, \ldots$, the functions T and $\sum$ determine the output sequence $b_0, b_1, \ldots$, according to the recursion

$$bi = T(si, ai) \qquad si+1 = \sum(si, ai), \qquad i = 0, 1, \ldots$$

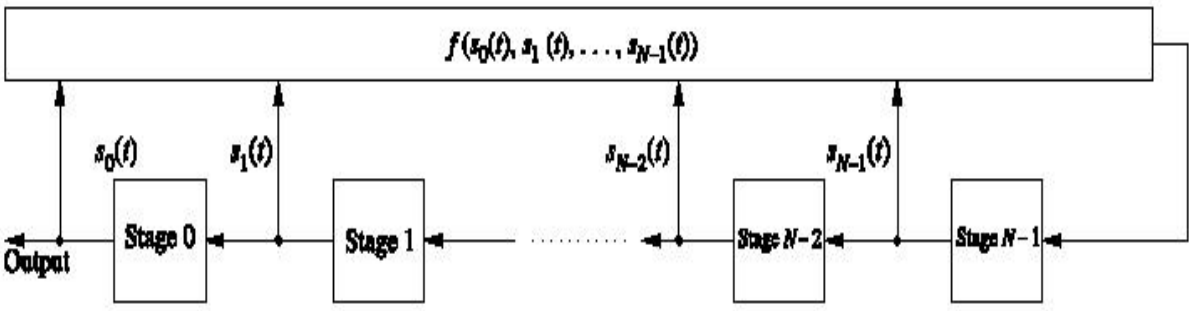

Figure 3.4 Feedback Shift Register

Figure (3.4) depicts a feedback shift register (FSR) with feedback function f, an FSM with null input consisting of N stages (each capable of storing one bit), a feedback register, and a single output port, where

1. The content of Stage *i* at time *t* is $s_i(t)$ = 0 or 1,
2. The output $s_0(t)$ is the content of Stage 0 at time *t*,
3. The state of the FSR at time *t* is the *N*-vector $\underline{s}(t) = (s_0(t), s_1(t), \ldots, s_{N-1}(t)) \in Z_{N,2}$ (where $Z_{N,2}$ is the set of 2^N vectors of length *N* with components 0 or 1), and
4. The feedback value at time t is $f(s_0(t), s_1(t), \ldots, s_{N-1}(t))$.

The states of the FSR change only when a clocking signal is applied and then as follows:

1. The content $s_i(t)$ of Stage i+1 at time t is shifted to the left, meaning it becomes the new content of Stage *i* at time t+1; $s_i(t+1) = s_{i+1}(t)$ for 0 <= i < N-1, and
2. The value f $(s_0(t), s_1(t), \ldots, s_{N-1}(t))$ in the feedback register at time *t* becomes the new content of Stage *N*-1 at time t+1; $s_{N-1}(t+1) = f(s_0(t), s_1(t), \ldots, s_{N-1}(t))$.

Figure (3.5) depicts a linear feedback shift register (LFSR), the special case of a feedback shift register with linear feedback function f

$$f(s_0(t), s_1(t), \ldots, s_{N-1}(t)) = \sum_{n=0}^{N-1} (c_{N-n} s_n(t)),$$

where

1. $c_0, c_1, \ldots, c_N$ are the feedback coefficients or taps [c_0 = 1],
2. The output of the AND-gate A[*j*] is the (current) content of Stage *j* if c_{N-j} = 1, and 0, otherwise, and
3. The feedback bit entering Stage *N*-1 when a clock pulse is applied is the eXclusive-OR (XOR) of the current outputs of the *N* AND-gates.

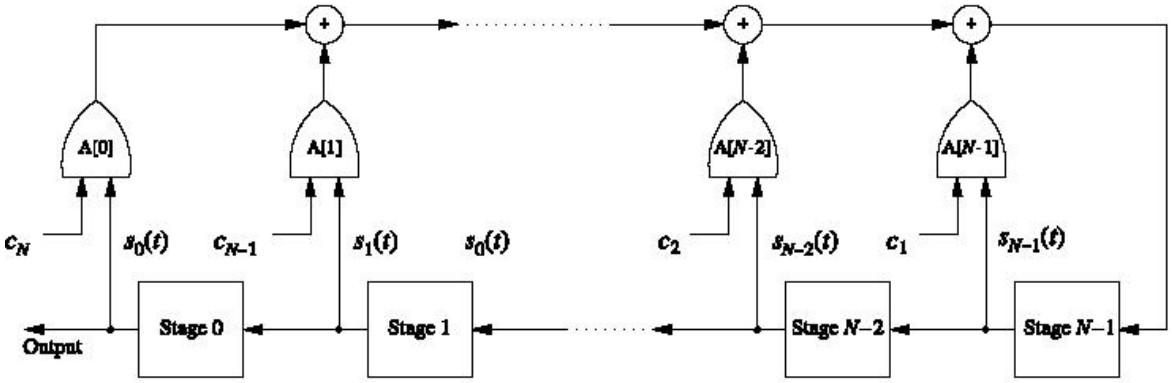

Figure 3.5 Linear Feedback Shift Register

The state of the LFSR at times t and t 1 are related by

$$\underline{s}(t) = (s_0(t), s_1(t), \ldots, s_{N-1}(t))$$

$$\underline{s}(t+1) = (s_0(t+1), s_1(t+1), \ldots, s_{N-1}(t+1)). \qquad (4.1)$$

As $s_i(t+1) = s_{i+1}(t)$ for $0 <= i < N-1$

$$\underline{s}(t+1) = (s_1(t), s_2(t), \ldots, s_{N-2}(t), s_{N-1}(t), s_{N-1}(t+1)) \qquad (4.2)$$

where

$$s_{N-1}(t+1)) = \sum_{n=0}^{N-1} (c_{N-n} s_n(t)), \qquad (4.3)$$

the addition in Equation (4.3) being modulo 2. As $s_0(t+k) = s_k(t)$ for $0<= k <N$, Equations (4.2) and (4.3) give

$$S_0(t+N)) = \sum_{n=0}^{N-1} (c_{N-n} s_0(t+n)), \qquad 0 <= t < \infty \qquad (4.4)$$

Equation (4.4) is a forward recursion, because the future output $s_0(t+N)$ is determined by the most recent N outputs $(s_0(t), (s_0(t+1), \ldots, s_0(t+N-1))$. When $c_N = 1$, Equation (4.4) may be rearranged such that

$$\boldsymbol{S_0(t))} = \sum_{n=1}^{N} (c_{N-n} s_0(t+n))\boldsymbol{.} \qquad (4.5)$$

Equation (4.5) is a backward recursion, because the N outputs ($s_0(t+1)$, ($s_0(t+2)$, . . ., $s_0(t+N)$) from time $t+1$ on determine the past output $s_0(t)$.

It may always assume that $c_N = 1$, for if $c_N = c_{N-1} = \ldots = c_{N-(k-1)} = 0$, $c_{N-k} = 1$, the LFSR essentially contains N-k active stages and the output sequence consists of the k-bit prefix determined by the contents of the leftmost k-stages concatenated with the output of a (N-k)-stage LFSR [Ala07].

Chapter Four
Implementation and Result

4.1 Introduction

Voice over Internet Protocol (VoIP) is a method of using the Internet to talk to people. It result from implementation of a common networking infrastructure to carry voice trafic over TCP/IP. This technology can be either software or hardware; for example, Skype and IP-Phone.

This thesis use SIP as the signaling protocol, SDP as a descriptor protocol and RTP as the media transport protocol in the application layer. For the network layer is use absolutely the IP protocol of version (4). As mentioned previously, the work use only one signaling protocol, which means the network type is homogenous. The application made by this work is a voice conversation between two PCs over LAN network. It's programming is done using unicon version (11.4) (Unified Extended Dialect of Icon*)* language, which is high level, goal directed, object oriented, general purpose applications programming language. More details are found in appendix A.

4.2 Call Control

Call control with SIP gives more facilities, such an easy to scalability especially when applied on public networks.

Although SIP is a client-server protocol, but it also work as P2P. This thesis built the application using SIP version (2.0) with P2P communication. More details are found in (RFC 3261).

4.3 Proposed Structure

The proposed structure of the implementation of this thesis is illustrated in figure (4.1). This structure is installed in each UA

4.2.1 SIP Structure

To understand how SIP programming is done, it is important to know that the protocol is consists of four layers, these are:

- ***Syntax and encoding:*** it is the lowest layer of SIP. It specifies the SIP message syntax. Its encoding is specified using an augmented Backus-Naur Form grammar (BNF) in RFC 2234.
- ***Transport layer:*** It defines how a client sends requests and receives responses and how a server receives requests and sends responses over the network. Both UAC and UAS have this layer.
- ***Transaction layer:*** The transaction layer has a client component (referred to as a client transaction) and a server component (referred to as a server transaction). A transaction is a request sent by a client transaction (using the transport layer) to a server transaction, along with all responses to that request sent from the server transaction back to the client. Both UAC and UAS have this layer.
- ***Transaction User layer (TU):*** The layer of protocol processing that resides above the transaction layer. Transaction users include the UAC core, UAS core, and proxy core (Core designates the functions specific to a particular type of SIP entity, i.e., specific to either a stateful or stateless proxy, a user agent or registrar. All cores, except those for the stateless proxy, are transaction users). The behavior of UAC and UAS

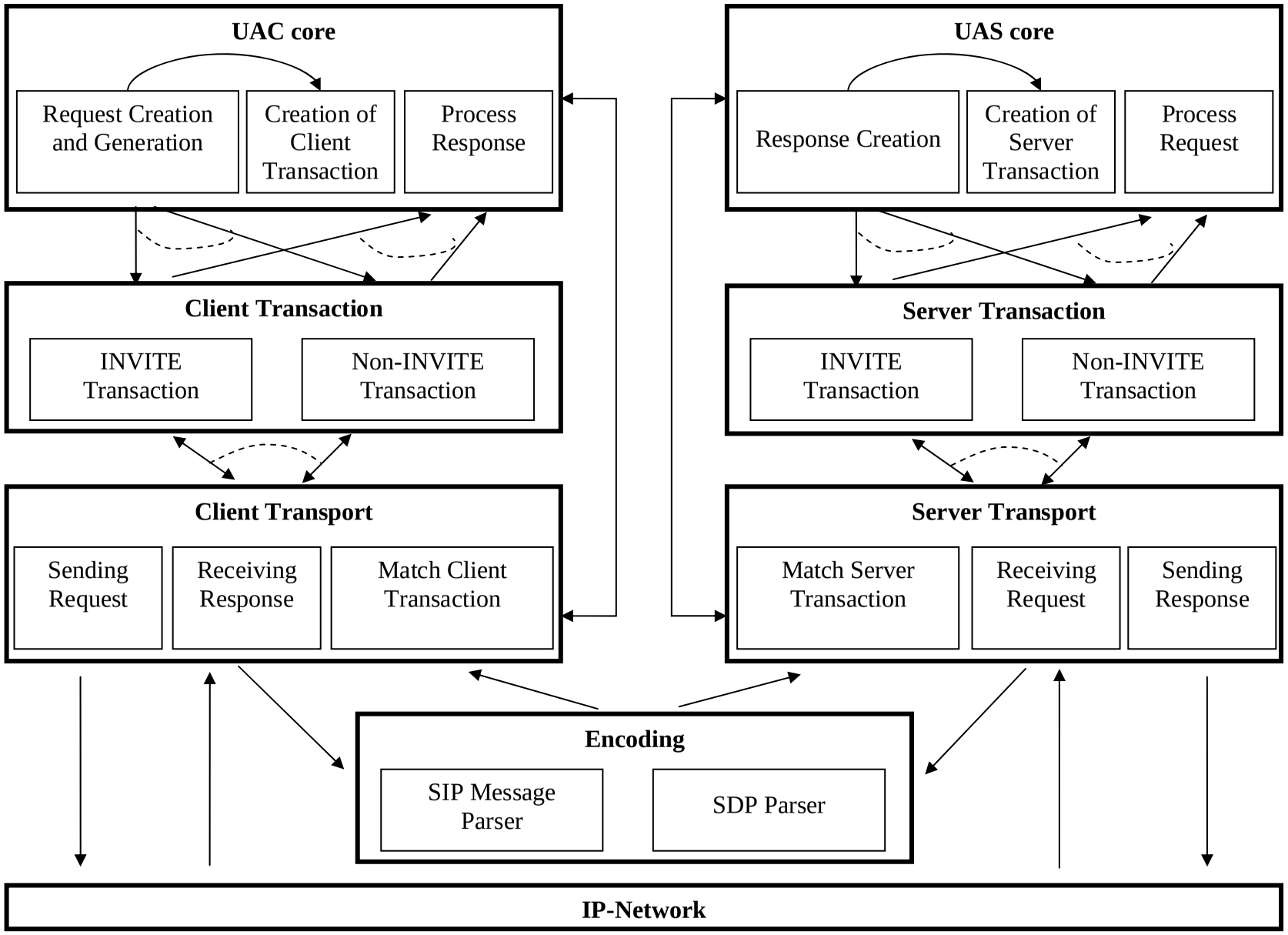

Figure 4.1 Implementation Structure

cores are method dependent, there are some common rules for all methods as it will discuss later.

4.2.1.1 Syntax and Encoding Layer

This layer helps understanding the parse construction of any SIP message using a standard rules, these rules are described as a tree for better understand. The doted lines means an optional field while a solid lines refer to a non-optional field, and the doted curves means (OR) while solid curves refer to (AND). More details are found in appendix B.

In this layer, trees must be read carefully, it identify how can messages (request, response) be distinguished as shown in figure (4.2) also identify SIP methods of requests. The purposes of request methods are indicated in table (4.1). This work uses only three methods (INVITE, ACK, and BYE).

Response types and there meanings are identified in figure (4.3). Figure (4.4) through figure (4.9) shows the message header fields.

Table 4.1 Methods Purpose

Method	**Purpose**
INVITE	Invites a user to join a call.
ACK	Confirms that a client has received a final response to an INVITE.
BYE	Terminates the call between two of the users on a call.
OPTIONS	Requests information on the capabilities of a server.
CANCEL	Ends a pending request, but does not end the call.
REGISTER	Provides the map for address resolution, this lets a server know the location of a user.

SIP message header fields are all case-insensitive except some parameters for some header fields (ex: value of "branch" parameter for "Via" header field).

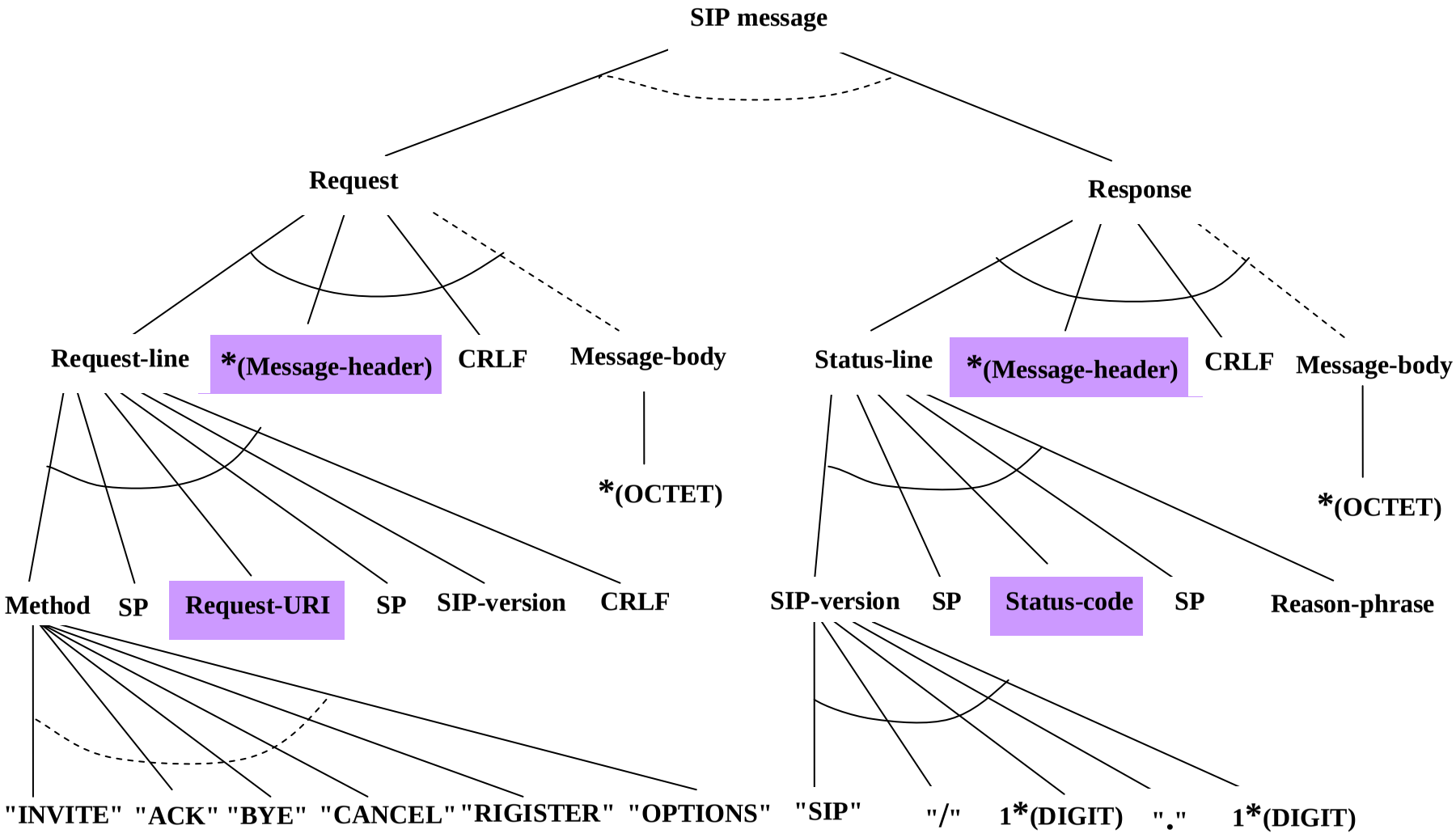

Figure 4.2 SIP Message

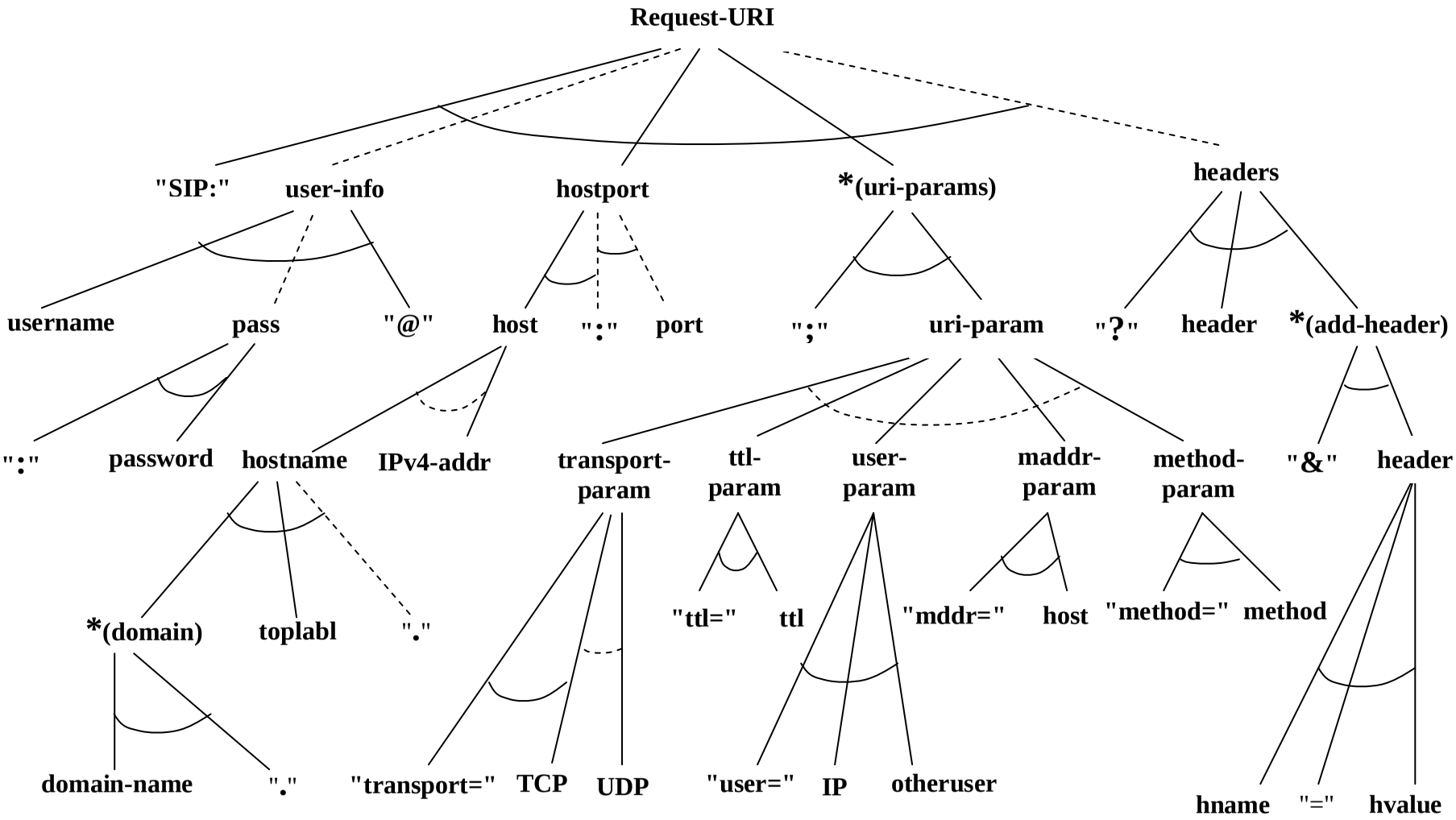

Figure 4.3 SIP-URI

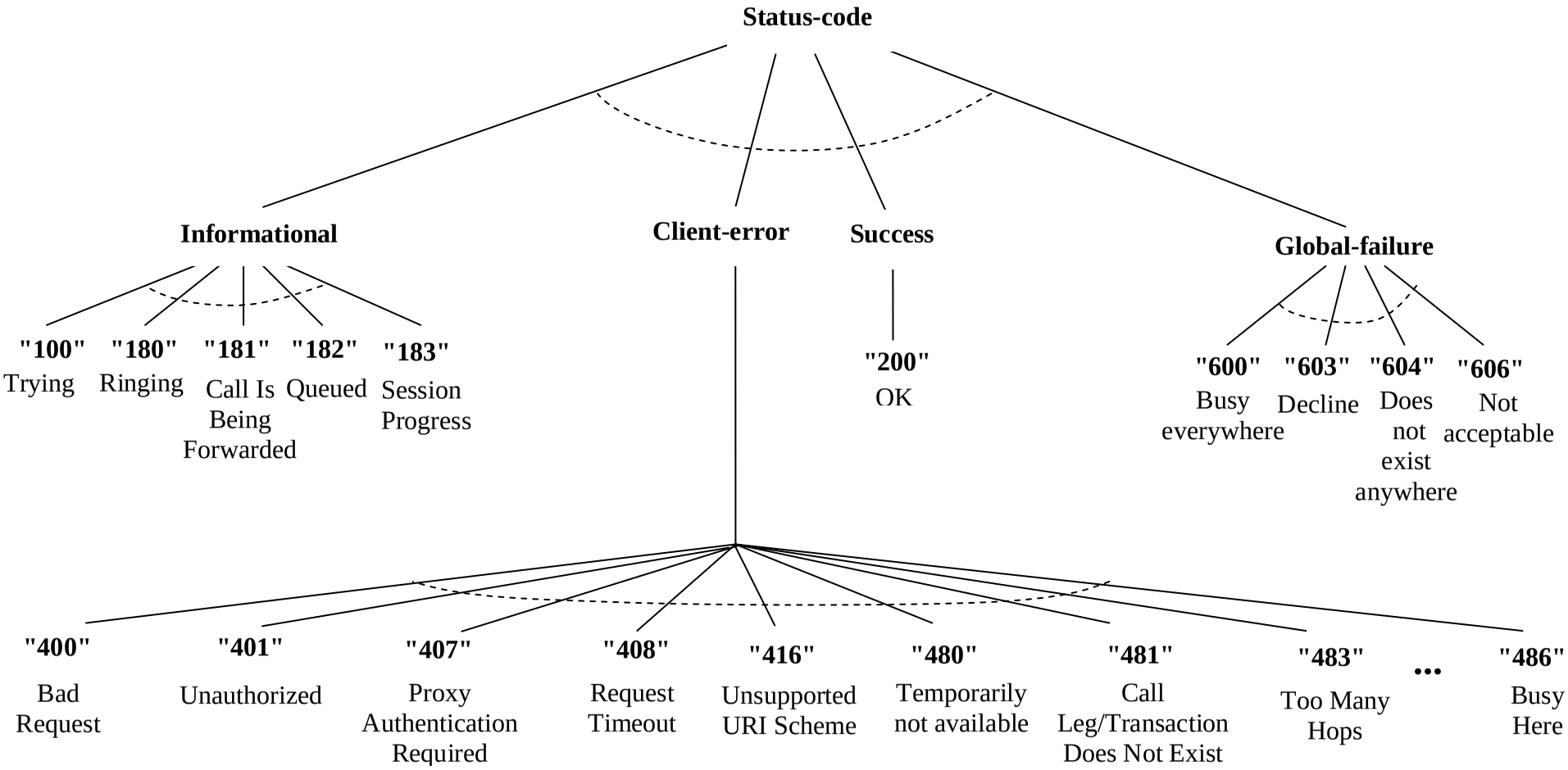

Figure 4.4 SIP Response Types

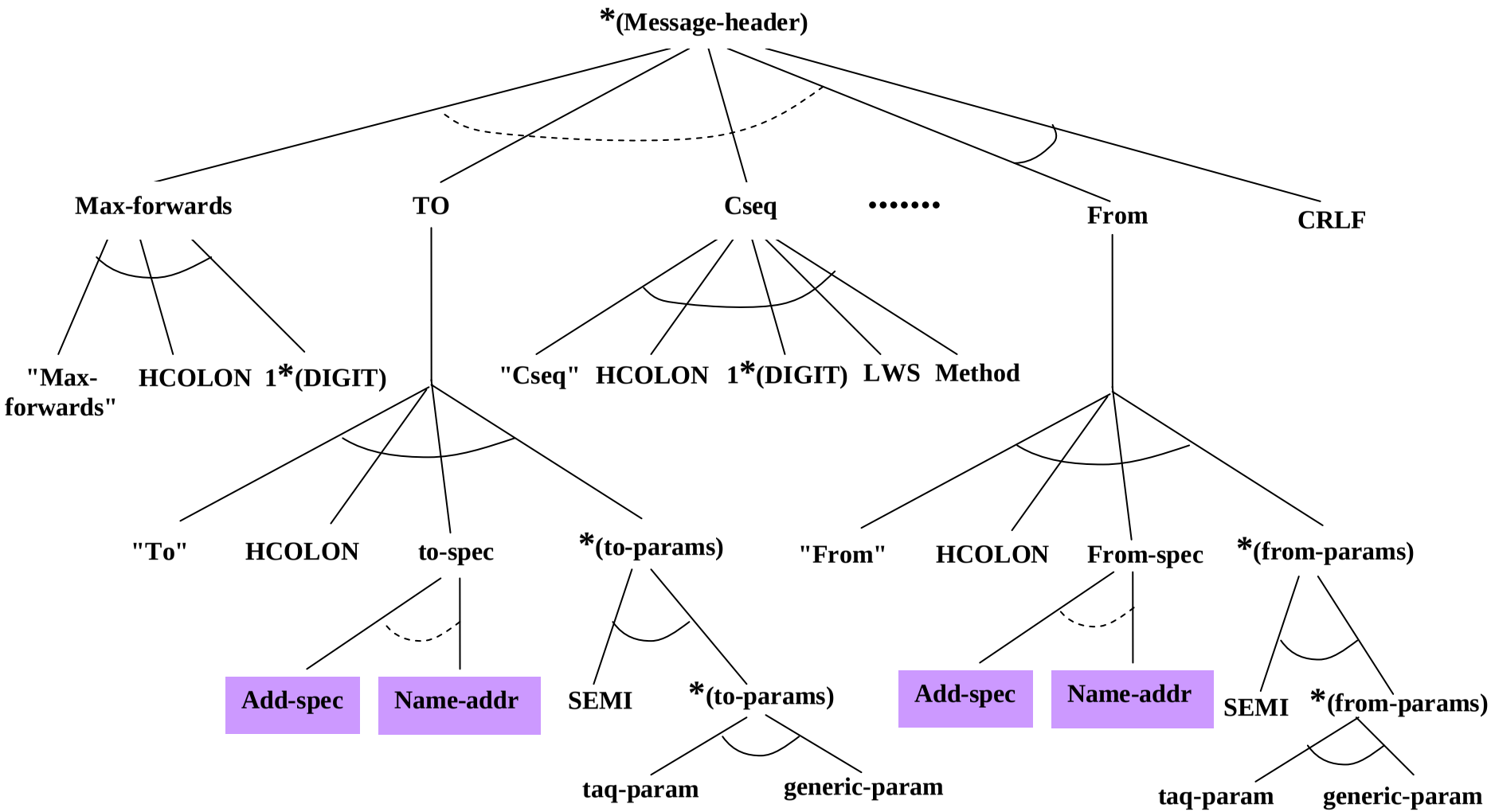

Figure 4.5 SIP Message Headers (Part1)

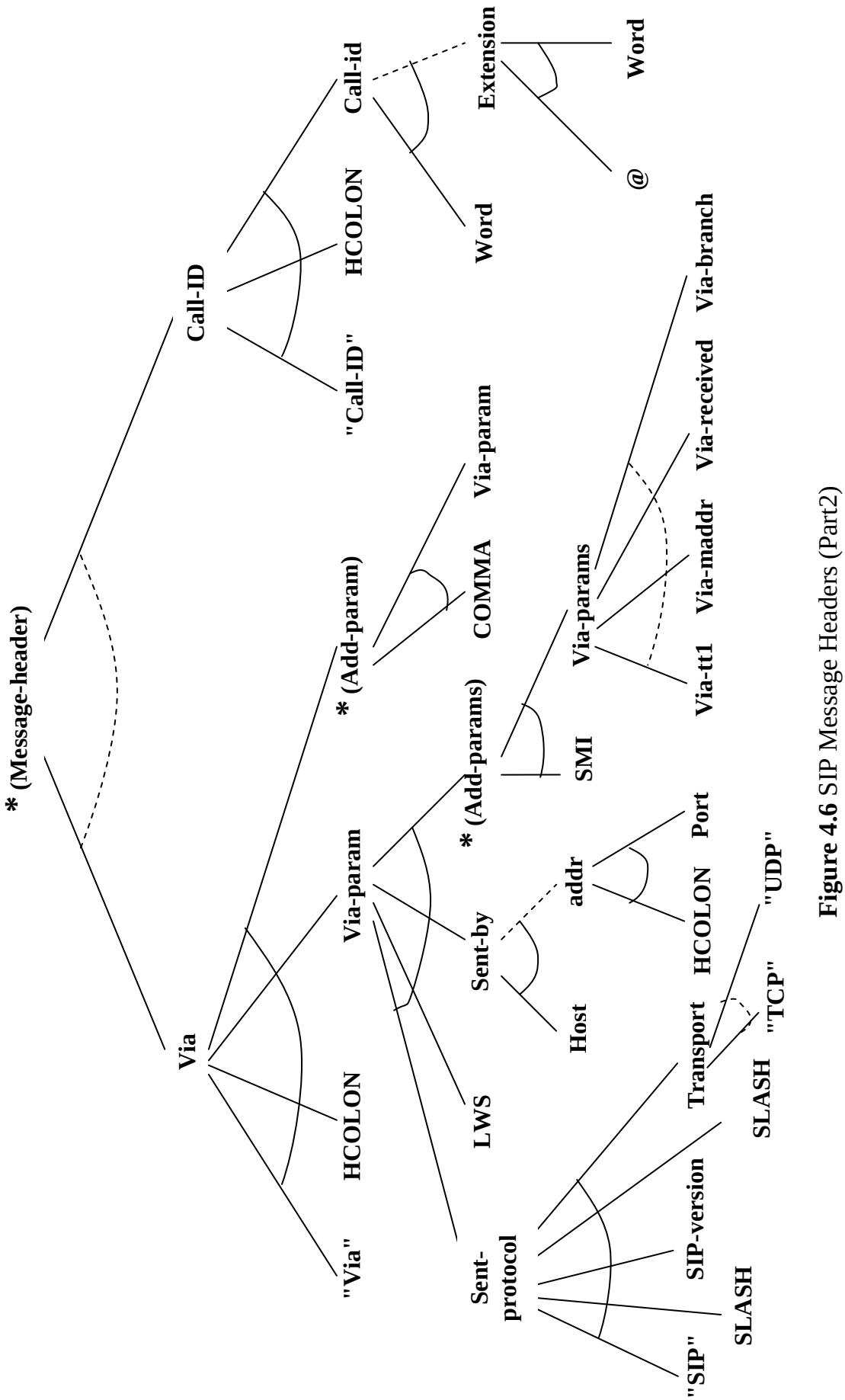

Figure 4.6 SIP Message Headers (Part2)

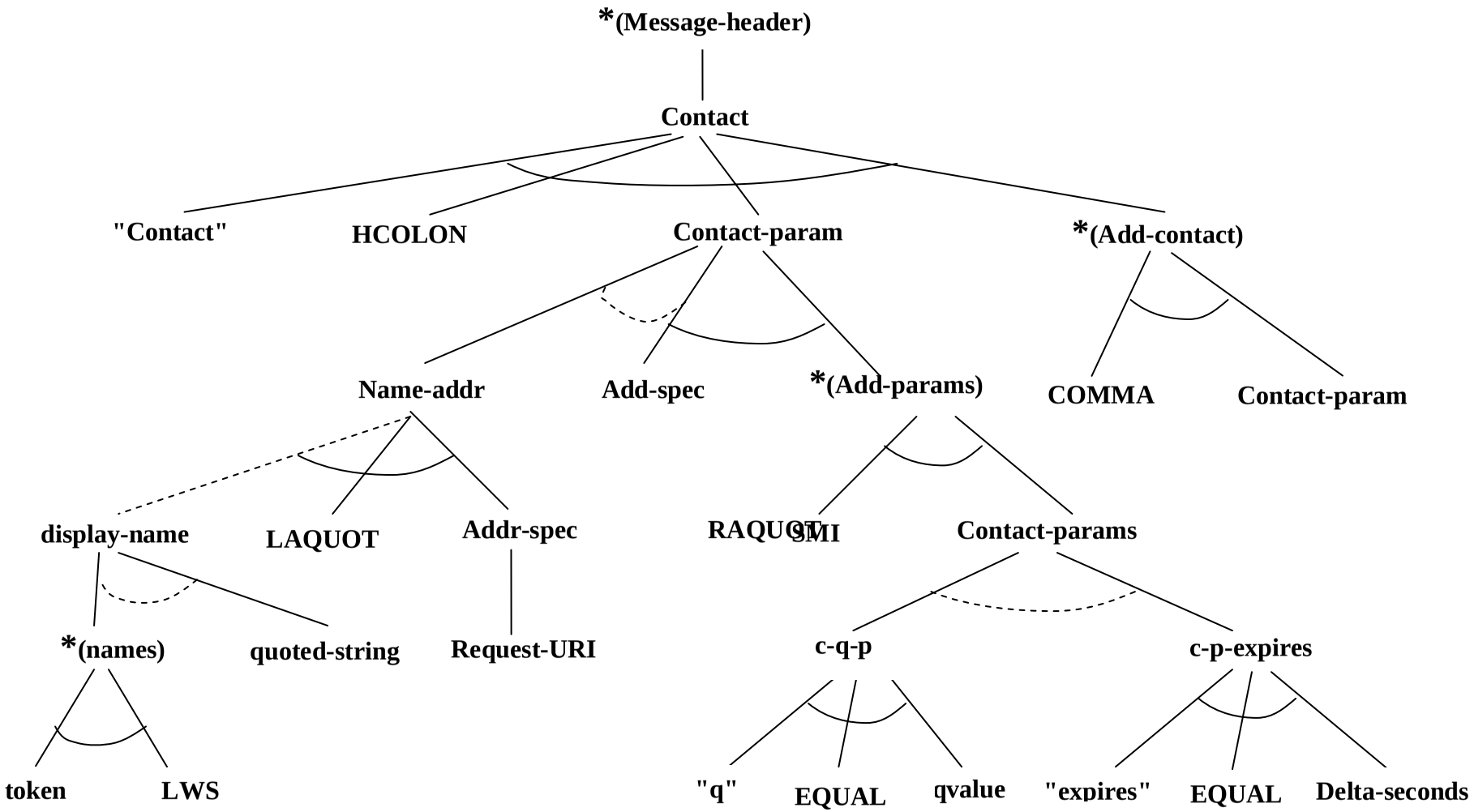

Figure 4.7 SIP message Headers (Part3)

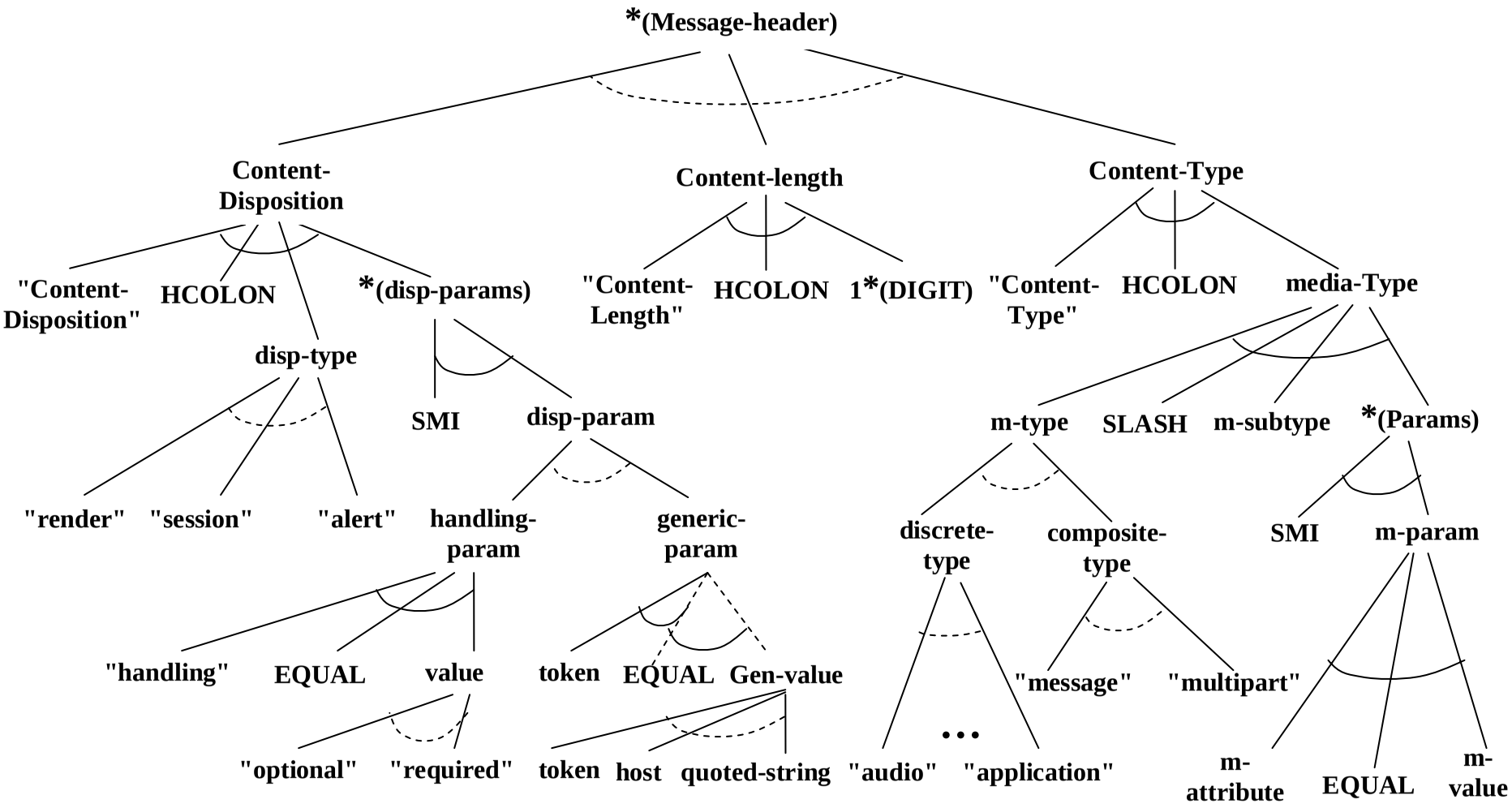

Figure 4.8 SIP message Headers (Part4)

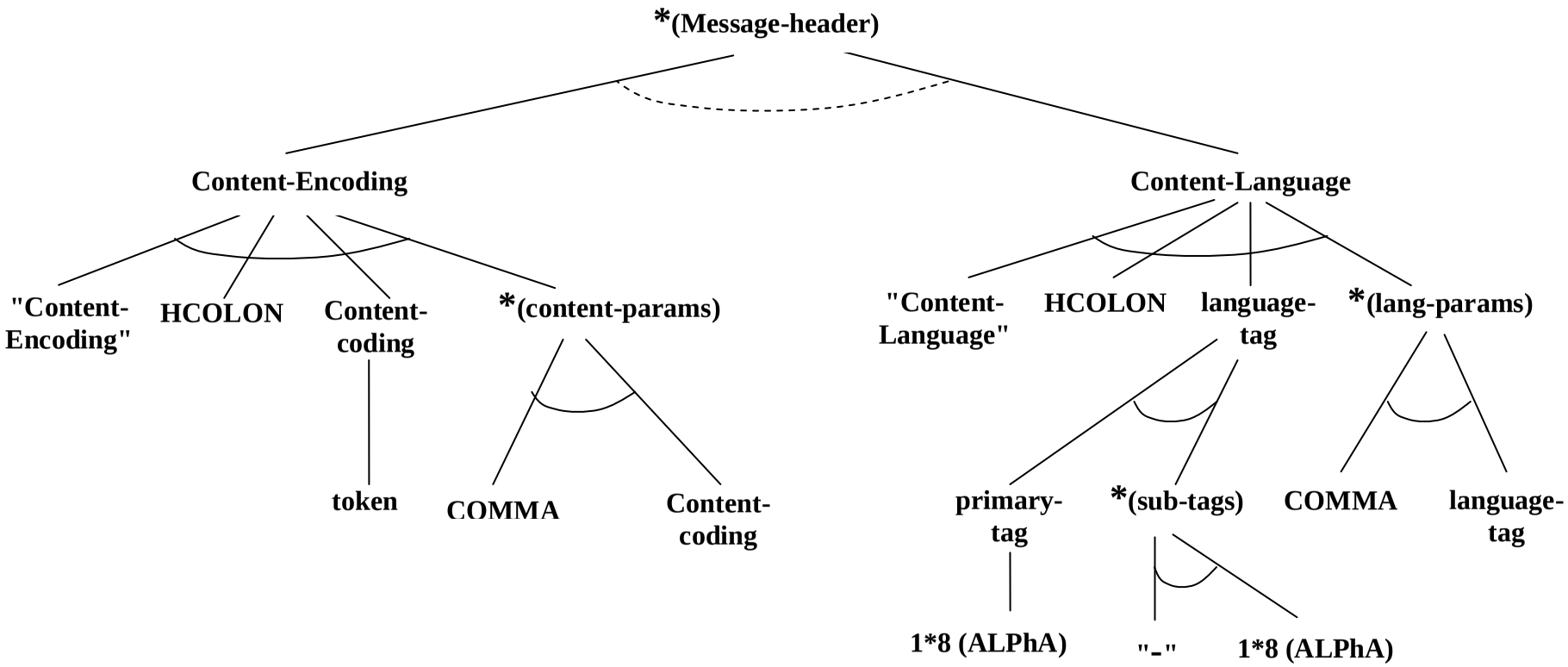

Figure 4.9 SIP message Headers (Part5)

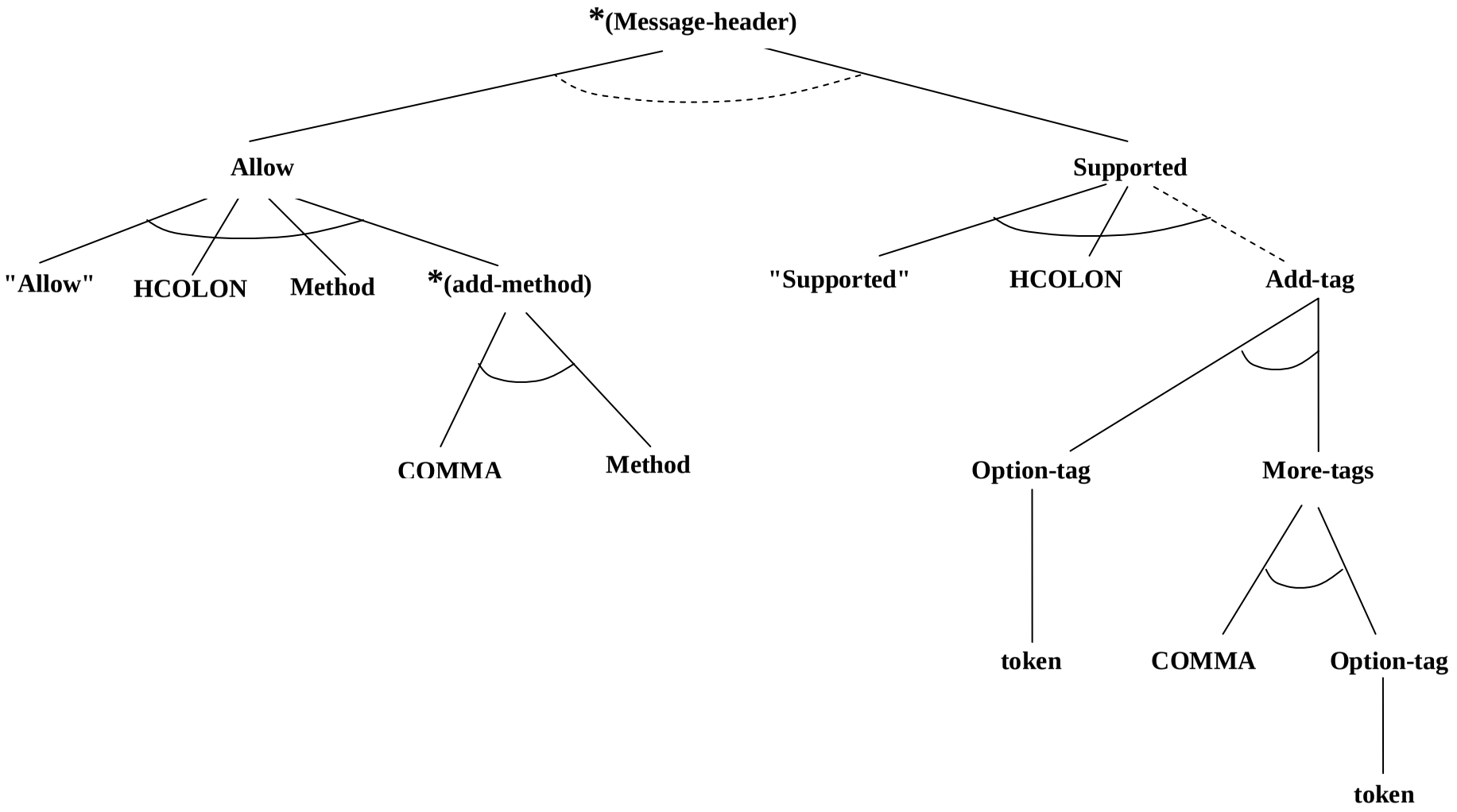

Figure 4.10 SIP message Headers (Part6)

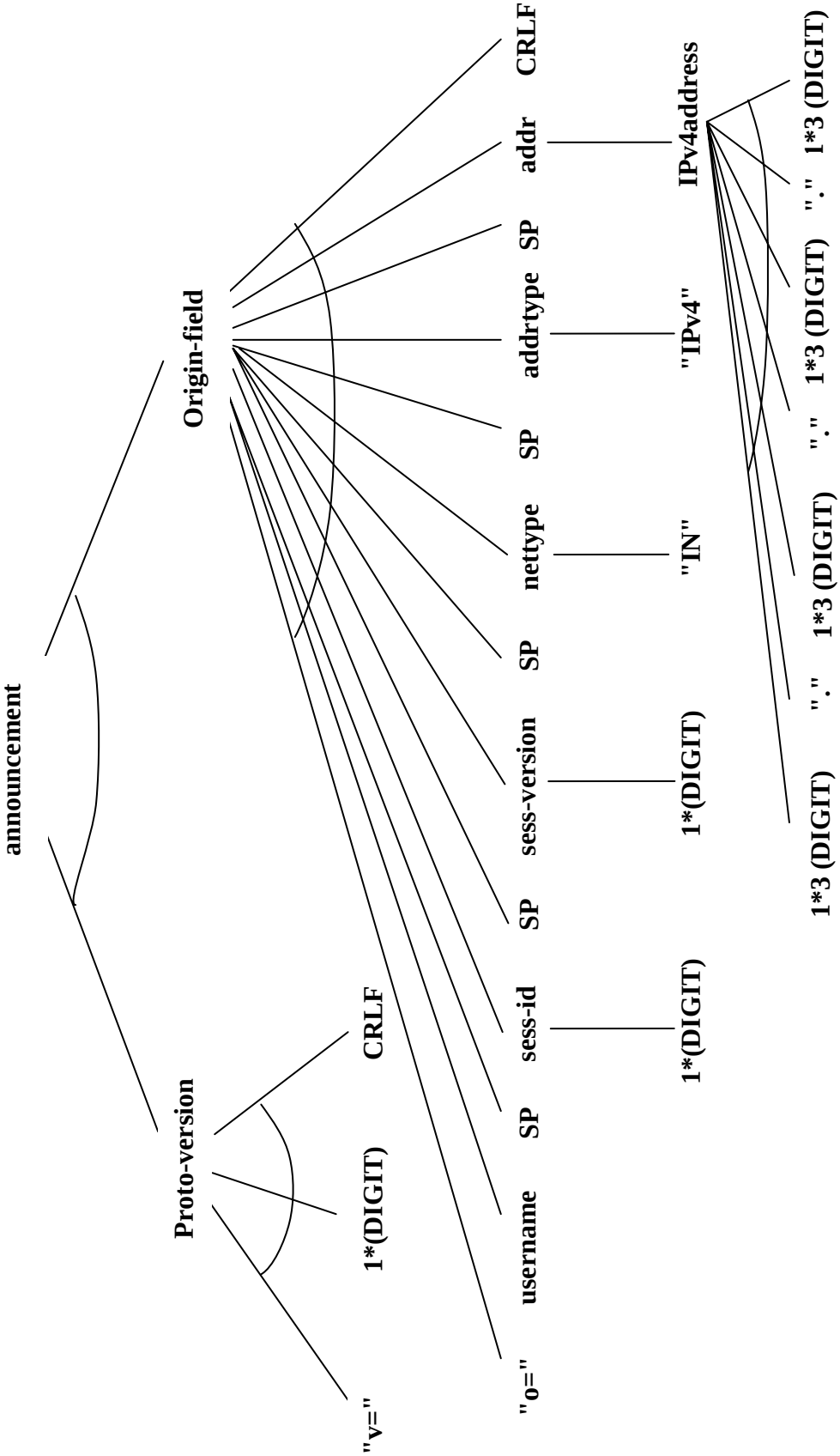

Figure 4.11 SDP Message (Part1)

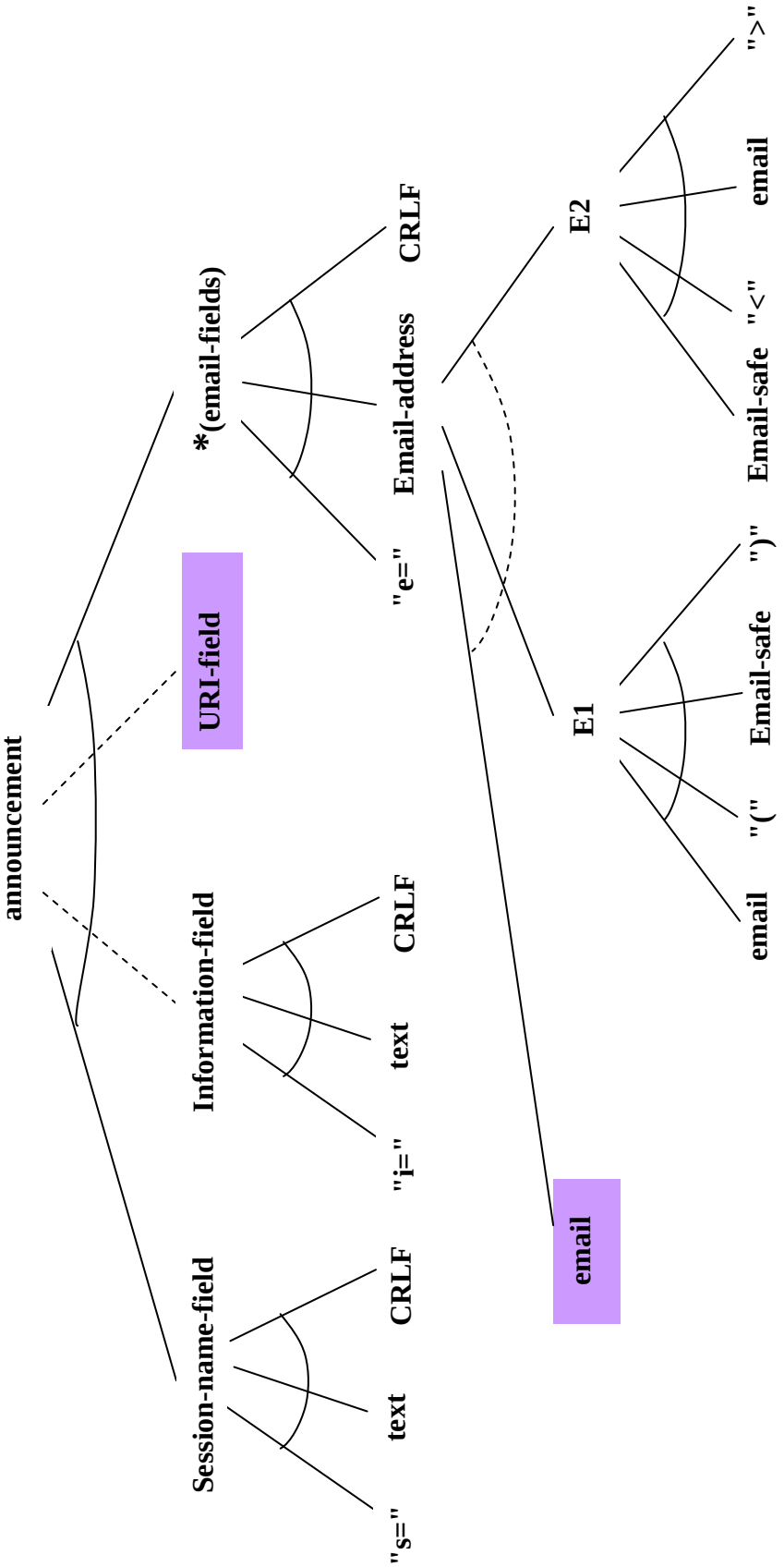

Figure 4.12 SDP Message (Part2)

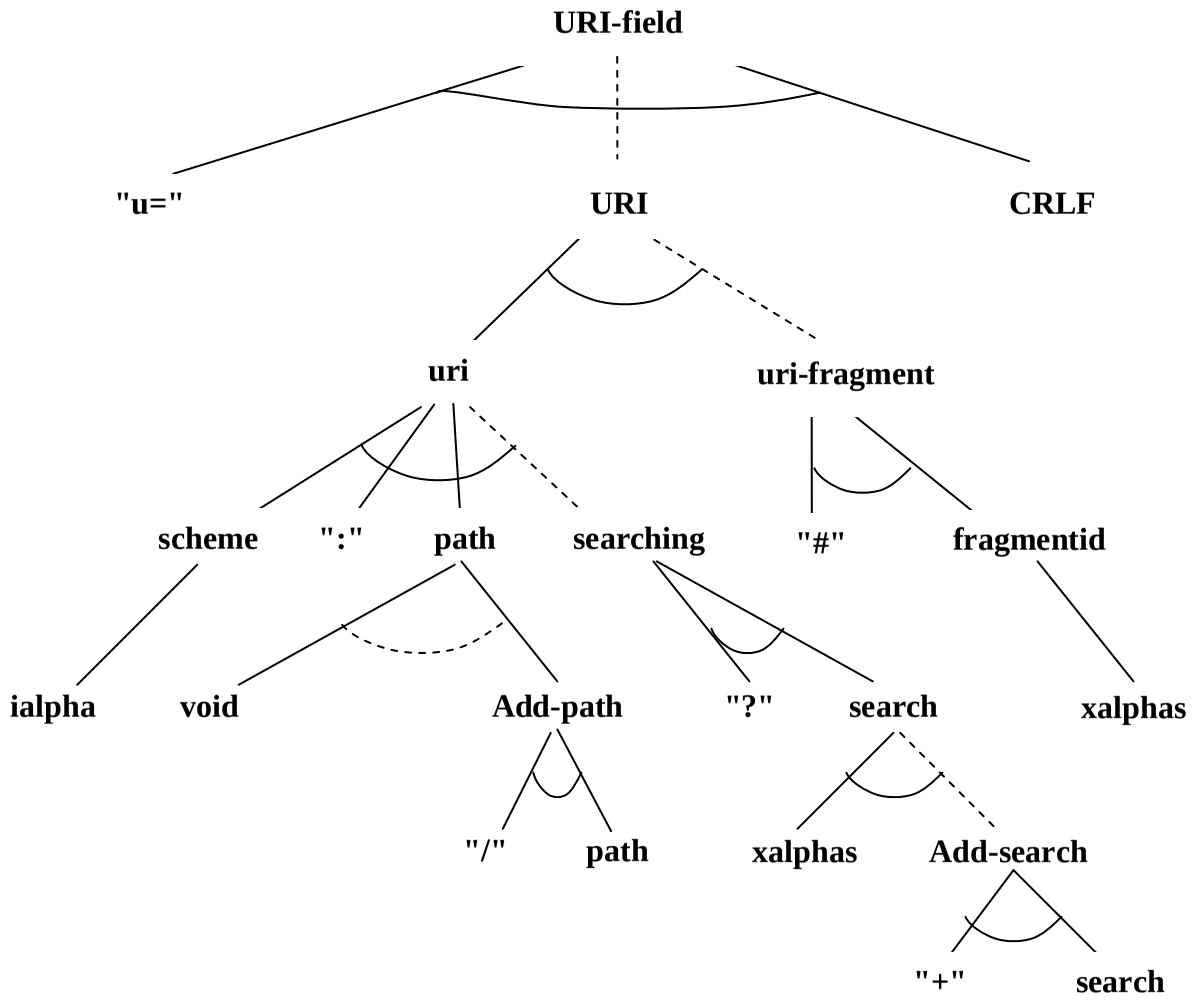

Figure 4.13 SDP Message (Part2) Continued

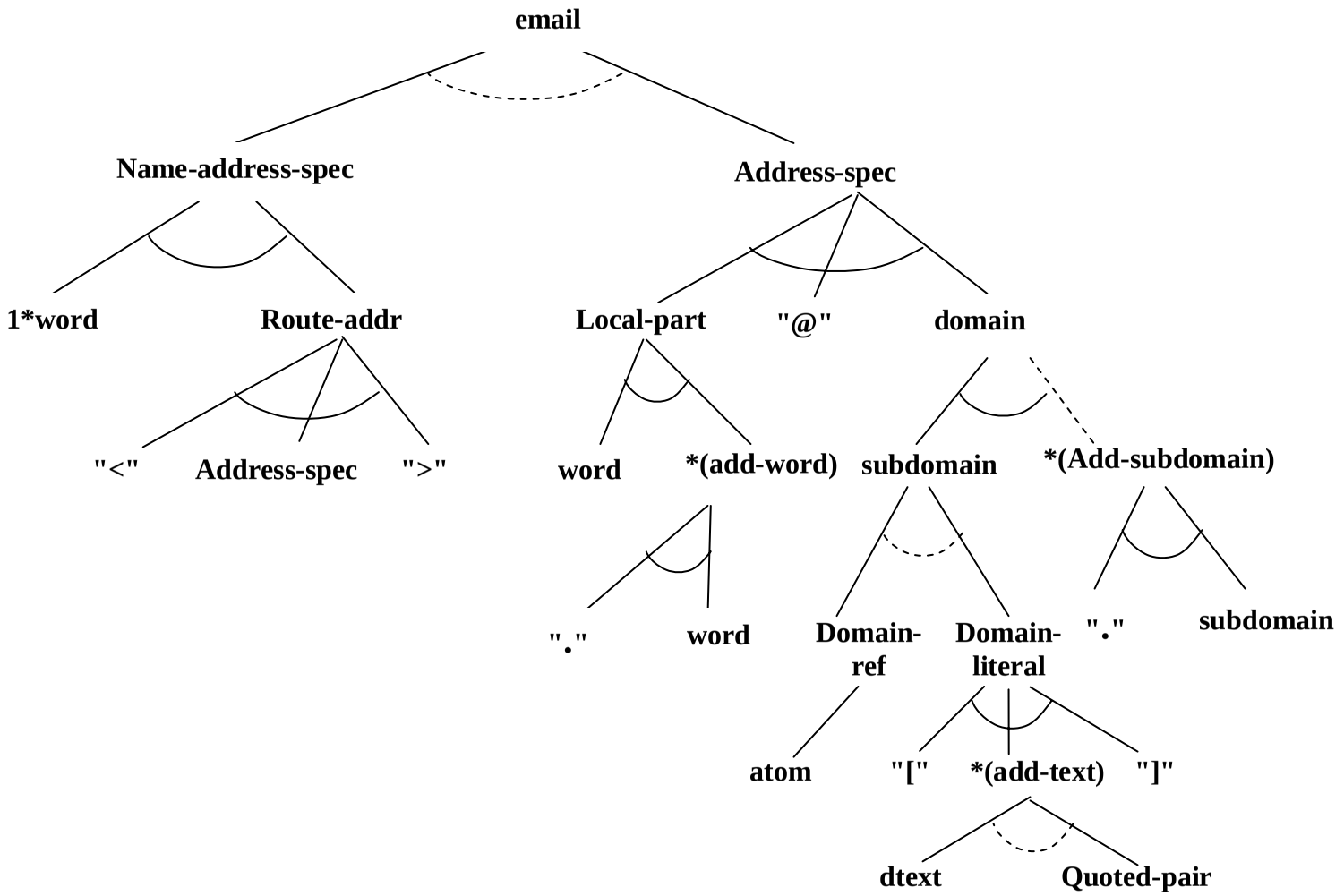

Figure 4.14 SDP Message (Part2) Continued

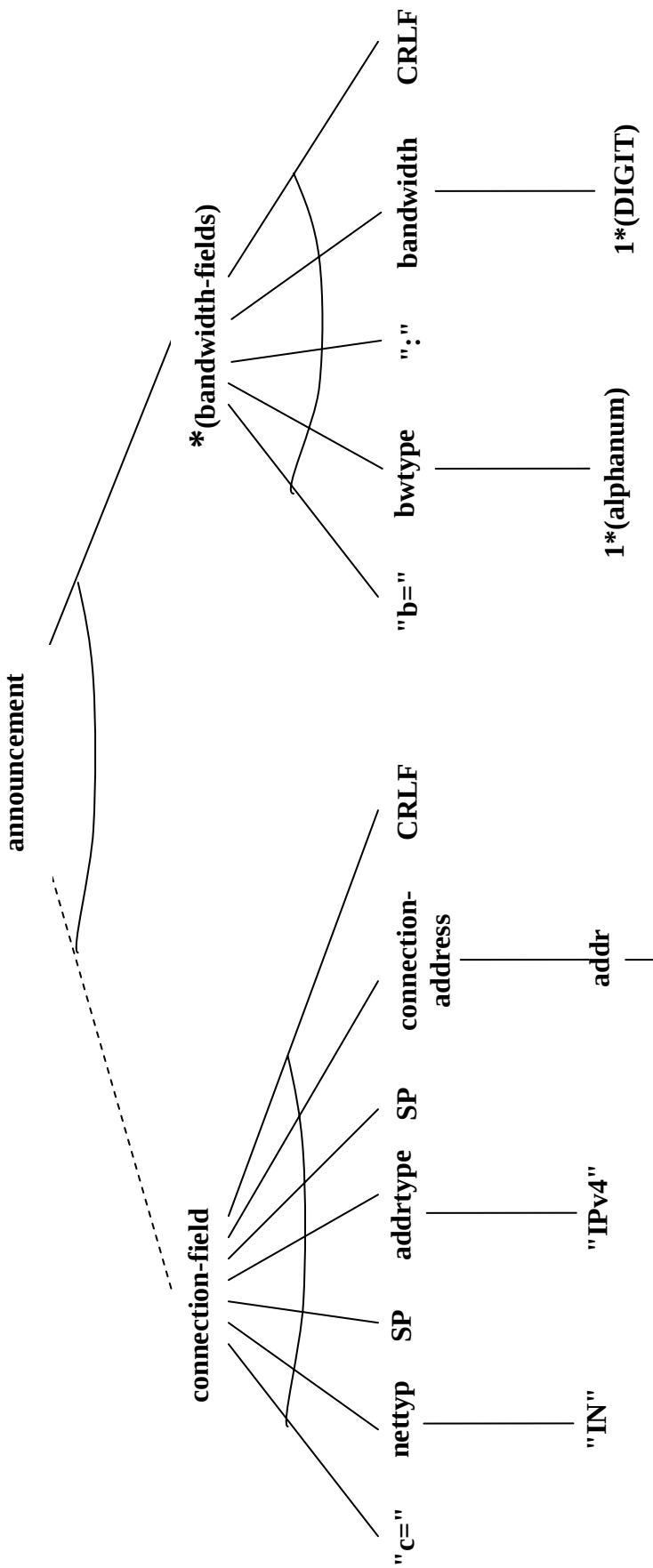

Figure 4.15 SDP Message (Part3)

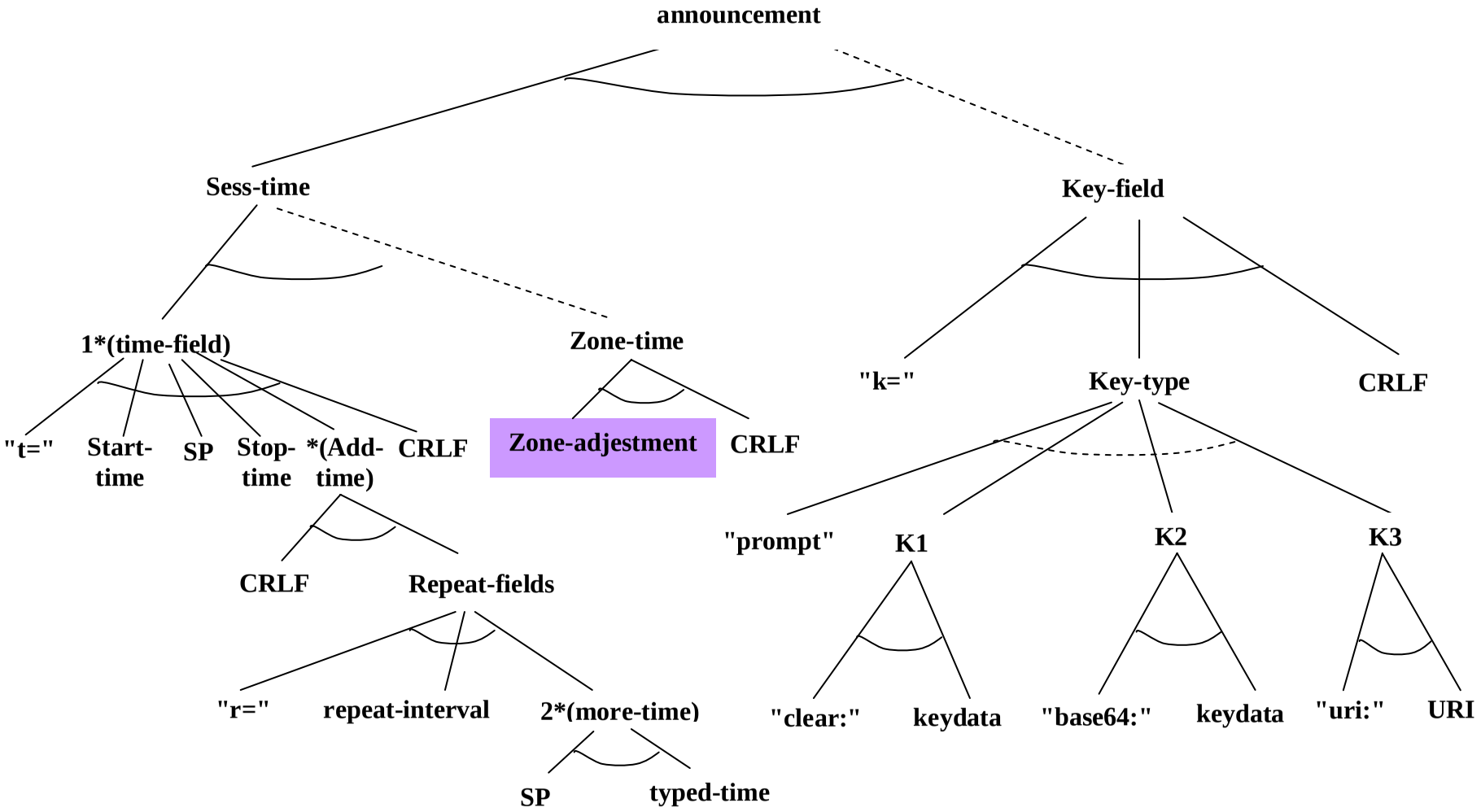

Figure 4.16 SDP Message (Part4)

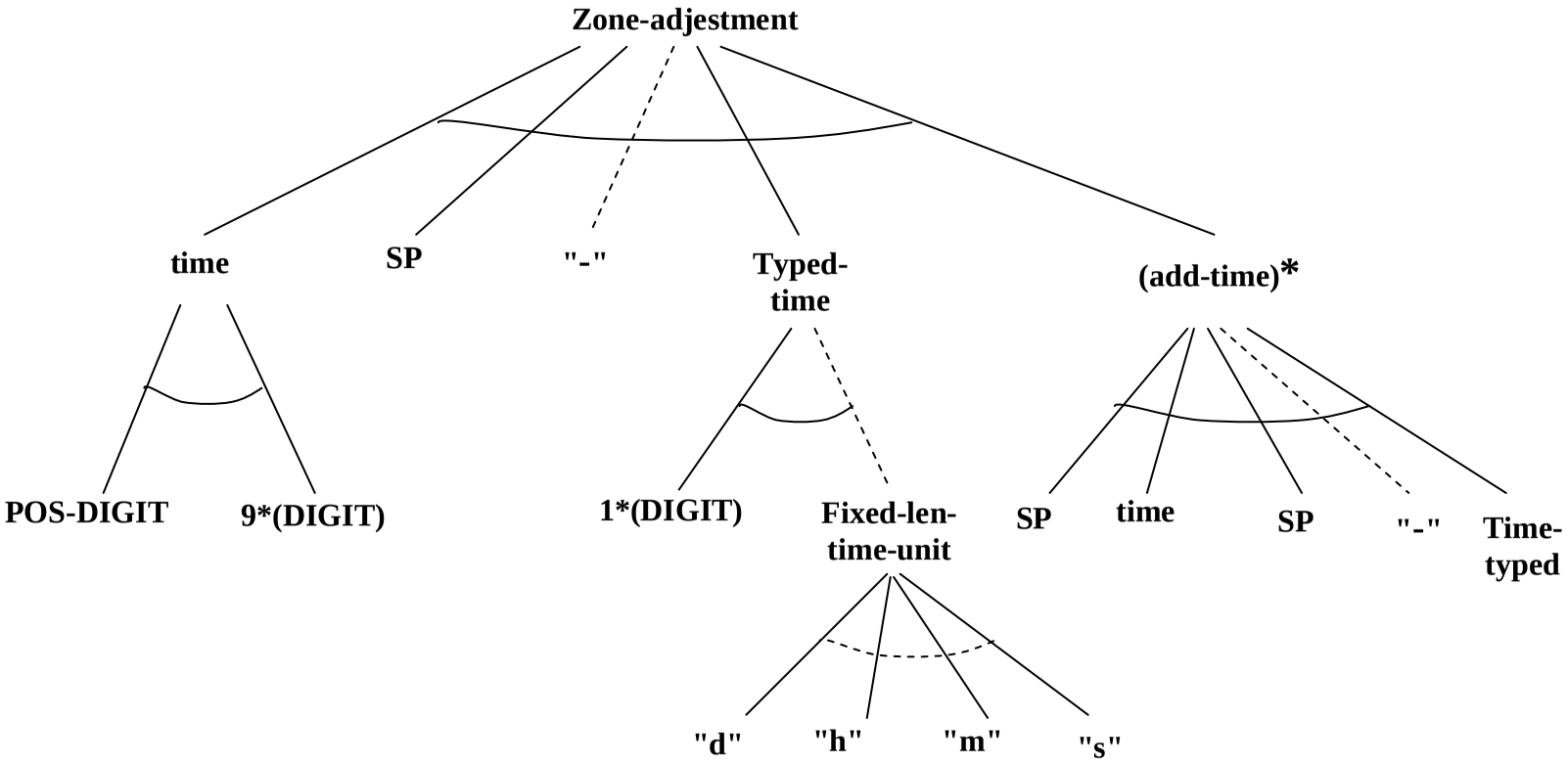

Figure 4.17 SDP Message (Part4) Continued

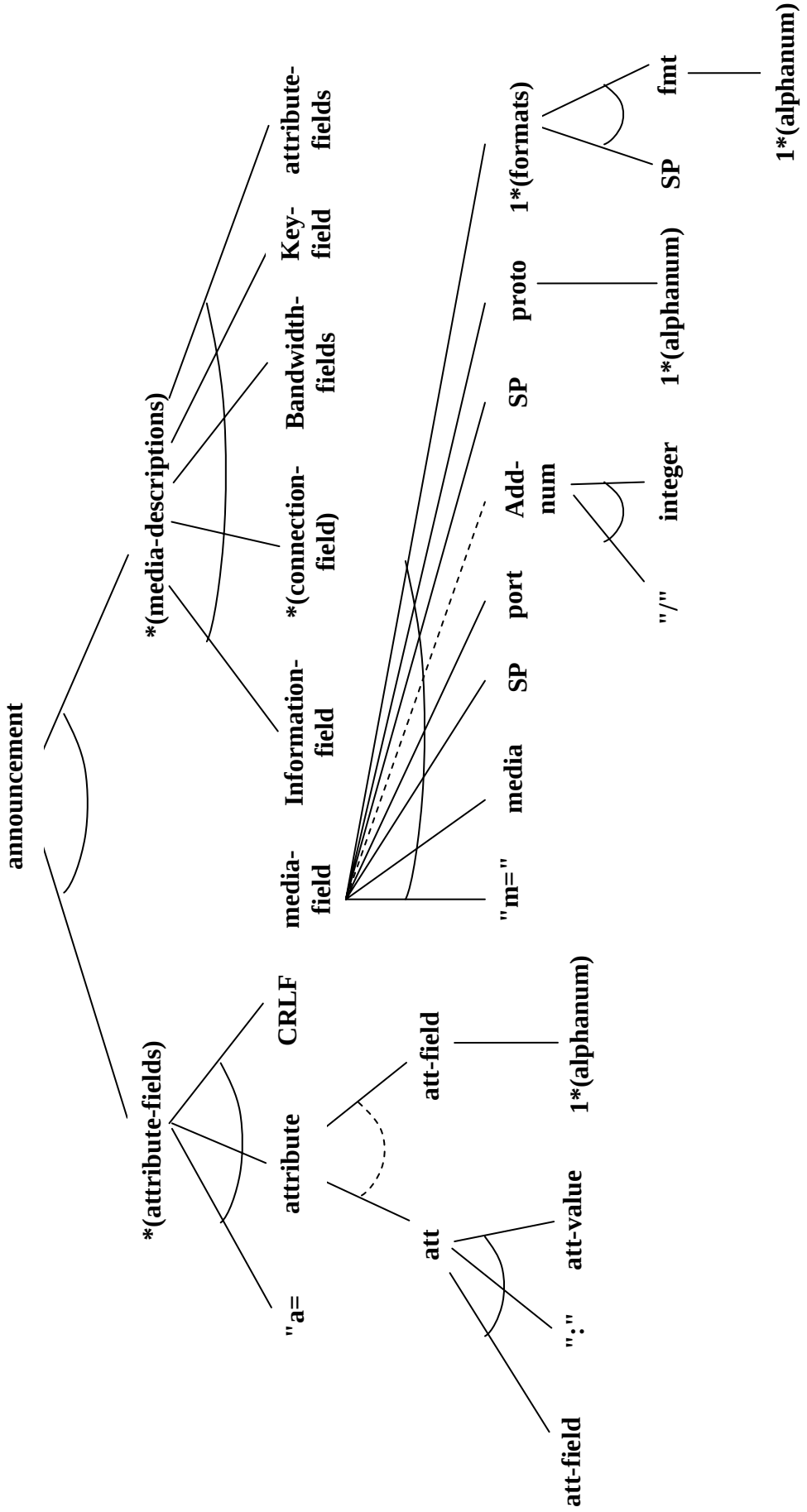

Figure 4.18 SDP Message (Part5)

SIP message body might be a descriptor (SDP). Its syntax tree is shown in figure (4.11) through figure (4.17). SDP fields' names are all small letters and fields' bodies are case-insensitive.

To parse SIP protocol messages, it can use the procedure (4.1).

4.1 SIP Message Parser

Step A: Create an SIP message structure which consist of several header fields' structures, the first line structure (request line, or status line), and field to save the Message body contents.

Step B: Parse the first line (request-line, or status-line) to distinguish if it is a request or response message respectively.

Step C: Because of each header field is in only one line ending with Carriage Return Line Feed (CRLF), so parse the first token of each line. From this it will have a list of all header field names included in the message header. Also the condition below must be executed:

If not understanding a header field name and it is not mandatory header field Then

Ignore this header field.

Else Silently discard the message**.**

Step D: Parse the body for first "Via" header field and the other mandatory header fields.

If any parameter of a header field is not understood Then

Ignore the parameter.

Step E: To parse SIP message body, check the following:

If "Content-Disposition" header field is exist Then

```
If "Content-disposition" header field body doesn't parsed
    Then
        Parse "Content-disposition" header field body.
If content-disposition is "session" Then
        Parse the body using SDP syntax.
Else If "Content-Type" has media type is "application" and media
    subtype is "sdp" Then
        Parse the body using SDP syntax.
Else if "Content-Type" has media type is "audio" Then
        Copy the content of SIP message body to a
        (.wav) file type.
```

A valid SIP request formulated by a UAC must, at a minimum contains the following header fields: To, From, CSeq, Call-ID, Max-Forwards, and Via; all of these header fields are mandatory in all SIP requests. These six header fields are the fundamental building blocks of a SIP message. Parsing only mandatory header fields as a beginning, this is because of other header fields may not needed, so parsing only the necessary header fields and the rest can be requested to be parsed as needed.

Step (D) is carried out when SIP message body information is needed in processing. SIP protocol must support SDP protocol, because it is the default descriptor protocol. So that, parsing process of SDP is will be put it within the parsing layer of SIP.

To parse an SDP messages that resides in SIP message body, follow the steps in procedure (4.2).

4.2 SDP Parser

Step A: create two data structure, one for saving session description and the other is for media description of that session.

Step B: parse all SDP header fields in the same order as shown in SDP message tree.

4.2.1.2 Transport Layer

The transport layer is responsible for the actual transmission of requests and responses over network. It is used for managing persistent connections for transport protocols like TCP, and UDP. This includes connections opened by the client or server transports, so that connections are shared between client and server transport functions. Also, this layer filters requests and responses according to procedures (4.3, 4.4). These procedures matching the request or response to a transaction they belong to.

4.3 Match Response to Client transaction (match_CT)

If "branch" in response = "branch" of the first "Via" in request that creates the transaction and "Cseq" method of response = "Cseq" of the request Then

Response is match to that client transaction.

4.4 Match Request to Server Transaction (match_ST)

If "branch" in first "Via" in received request = "branch" in the transaction

and "sent-by" in first "Via" in received request = "sent-by" in the transaction and request method in received request = request method in the transaction of the original request Then

Request is matched to that server transaction.

Now Procedures described later are the processing steps in transport layer for UAC and UAS.

- **UAC:** client side of the transport layer is responsible for:

Sending request

The request is received from the client transaction of caller to be sending to the transport layer of UAS of callee. The process is as in procedure (4.5).

4.5 Sending Request

Step A: Insert "sent-by" parameter in first "Via" header field (IP address of caller, and port=5060).

Step B : If destination address is multicast Then

Add "maddr" parameter in "Via" header field.

"maddr" = destination multicast address.

If IPv4 Then

Add "ttl" parameter

"ttl" = 1

Step C : If message size > 1300 byte Then

"transport" parameter in "Via" header field = "TCP".

Else "transport" parameter in "Via" header field = "UDP".

Step D : Open connection using destination (IP, port, transport).

Step E : Send the request.

If error happens in sending Then

Inform transaction layer.

"sent-by" is useful when UAS send response in a new connection (different than connection that send request). Port (5060) is used with TCP, or UDP transport protocol. It is considered as a well-known port for VoIP application.

The opened connection is preferred to still opened and waiting to receive response on it. This is to made something called a transaction

Receiving response

The response is received from the transport layer of the callee and sends it to the client transaction of the caller. The process is as in procedure (4.6).

4.6 Receiving Response

Step A: Receive response from an opened connection that used to send the request. UAC might prepare to receive any response in any IP, port, and transport. When this connection accepted by the transport layer for receiving the response, it almost does not used again.

Step B: If "sent_by" of the response = "sent_by" of the original request Then

If match_CT ("sent-by") Then

Pass to the matched transaction. Else Pass to the core in TU layer. Else Discard the response.

- **UAS:** server side of the transport layer is responsible for:

Receiving Request

The request is received from the transport layer of UAC of caller to be sent to the client transaction or to TU layer of the callee. Sometimes requests are received from transport layer of callee to the transport layer of the caller (ex: BYE request). The process is as in procedure (4.7).

4.7 Receiving Request

Step A: UAS must ready to receive any request in any IP, port, and transport. It is preferred to listen for requests on default port (5060). **Step B:** If "sent-by" has IP address <> source packet IP address Then Add "received" and set to IP of source packet. **Step C:** If match_ST Then Pass to the matched transaction. Else Pass to TU.

In step (C) is used to send to TU layer for ACK request, where were no transaction for ACK, because client transaction is destroyed after receiving final responses (2xx). So ACK request has its own transaction other than INVITE request transaction.

Sending Response

Responses are received from client transaction of callee. Through the transport layer of callee, responses will send to transport layer of caller. The process is as in procedure 4.8.

4.8 Sending Response

Step A: If sent-protocol = "TCP" Then
If connection that sends the request is still exist Then
Send response on the same connection.
Else Open new connection using IP in "received", port from "sent-by" or using default port (5060).
Send response.
Else If first "Via" has "maddr" Then
Send response using IP in "maddr", port in "sent-by" or default port (5060).
Else If "received" is exist Then
Send response using IP in "received, port in "sent-by" or default port (5060).
Else send response using IP and port in "sent-by".
Step B: If error happens in sending Then
Inform transaction layer.

4.2.1.3 Transaction Layer

UAC executes the client transaction, which sends the request to a server transaction that is executed in UAS. Processing of both client transaction and server transaction are method dependent (INVITE, or Non-INVITE).

- **Client Transaction**

The client transaction is responsible for:

1. Receive request from TU, then reliably deliver the request to a server transaction.
2. Receiving responses and delivering them to TU.
3. Filter any response (retransmission or disallowed).
4. In case of INVITE request, it will generate (ACK request) for any final response accepting 2xx response

INVITE Client Transaction

The INVITE transaction consists of a three-way handshake:

1. The client transaction sends an INVITE.
2. Receive responses from server transaction.
3. The client transaction sends an ACK.

The procedure (4.9) which illustrated in figure (4.19) uses some timer as in table (4.2) that summarizes the meaning and defaults of the various timers used by this specification.

Table 4.2 Table of Timer Values

Timer	Value	Meaning
T1	500ms	Estimate of round-trip time
Timer_A	initially T1	INVITE request retransmit interval, for UDP only
Timer_B	64*T1	INVITE transaction timeout timer
Timer_D	> 32s for UDP, 0s for TCP	Wait time for response retransmits

4.9 INVITE-Client Transaction

Step A: Calling state

Step A.1: Receive INVITE request and its transaction from TU

T1 = 500 ms

Timer_B = 64*T1

Step A.2: If "transport" in first "Via" = "UDP" Then

Send the received request to transport layer

If error in sending request Then

Inform TU

Go to Terminated state

Timer_A = T1

Repeat until (Timer_B is fires or receive a response)

If Timer_A is fires Then

Resend the request to the transport layer.

If error in sending request Then

Inform TU.

Go to Terminated state.

Timer_A = T1*2

T1 = Timer_A

Go to Step A.4.

Else Send the request to the transport layer.

Step A.3: If a response (100-199) is received Then

Send the response to TU.

Go to Proceeding state.

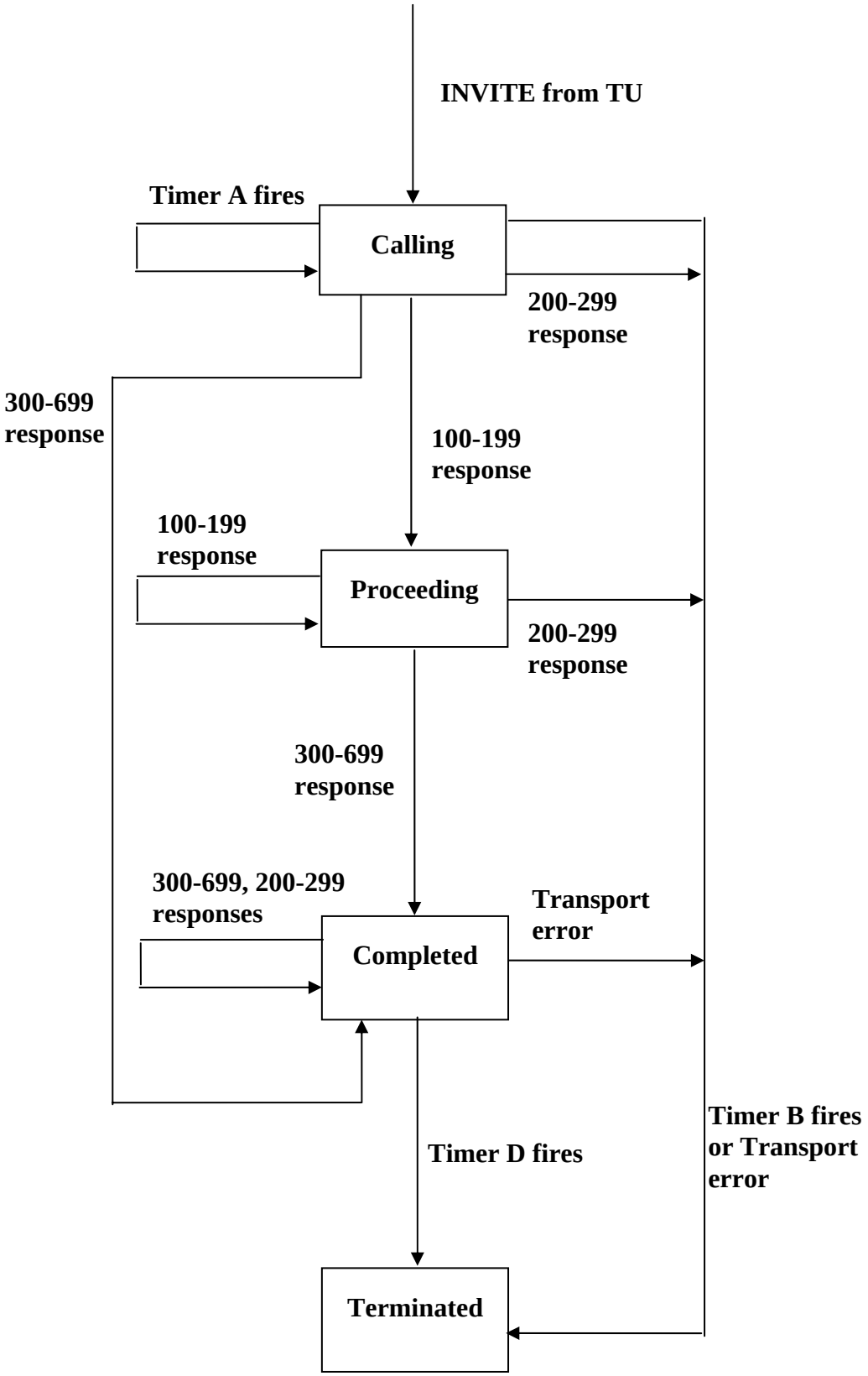

Figure 4.19 INVITE Client Transaction

If a response (300-699) is received Then

Create_ACK.

Send ACK to transport layer.

If error in sending request Then

Inform TU.

Go to Terminated state.

Send the response to TU.

Go to Completed sate.

If a response (200-299) is received Then

Send the response to TU.

Go to Terminated state.

Step A.4: If Timer_B is fires Then

Advertise TU of transaction timeout.

Go to Terminated state.

Else Go to Step A.3

Step B: Proceeding state

Step B.1: If a response (100-199) is received Then

Send the response to TU.

If a response (300-699) is received Then

Create_ACK.

Send ACK to transport layer.

Send the response to TU.

Go to Completed state.

If a response (200-299) is received Then

Send the response to TU.

Go to Terminated state.

Step C: Completed state

Step C.1: If "transport" in first "Via" = "UDP" Then

Timer_D = 32 sec.

Else Timer_D = 0 sec.

Step C.2: If a response (300-699) is received for the first time Then

Pass the response to TU.

Create_ACK.

Save ACK.

Send ACK to transport layer.

If error in sending request Then

Inform TU.

Go to Terminated state.

Else Resend ACK.

If error in sending request Then

Inform TU.

Go to Terminated state.

Discard the response.

If a response (200-299) is received Then

Pass the response to UAC core in TU.

Step C.3: If Timer_D is fires Then

Advertise TU for timeout of transaction.

Go to Terminated state.

Step D: Terminated state

Step D.1: Destroy the client transaction.

Non-INVITE Client Transaction

The procedure (4.10) which illustrated in figure (4.20) uses some timer as in table (4.3) that summarizes the meaning and defaults of the various timers used by this specification.

Table 4.3 Timer Values

Timer	Value	Meaning
T1	500 ms.	Estimate of round-trip time
T2	4 s.	The maximum retransmit interval for non-INVITE requests
T4	5 s.	Maximum duration a message will remain in the network
Timer_E	initially T1	Non-INVITE request retransmit interval, for UDP only
Timer_F	64*T1	Non-INVITE transaction timeout timer
Timer_K	T4 for UDP, 0s for TCP	Wait time for response retransmits

4.10 Non-INVITE Client Transaction

Step A: Trying state

Step A.1: Receive Non-INVITE request and its transaction from TU.

T1 = 500 ms

Timer_F = 64*T1

Send the request to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

Step A.2: If "transport" in first "Via" = "UDP" then

Timer_E = T1

T2 = 4 sec.

Repeat until (Timer_F is fires or receive a response)

If Timer_E is fires Then

Resend request to transport layer

If error happens in sending Then

Inform TU

Go to Terminated state

T1 = Min(2*T1,T2)

Timer_E = T1

If Timer_F is fires Then

Advertise TU of transaction timeout

Go to Terminated state

Else Send request to transport layer

If error happens in sending Then

Inform TU.

Go to Terminated state.

Step A.3: If a response (100-199) is received Then

Send the response to TU.

Go to Proceeding state.

If a response (200-699) is received Then

Send the response to TU.

Go to Completed state.

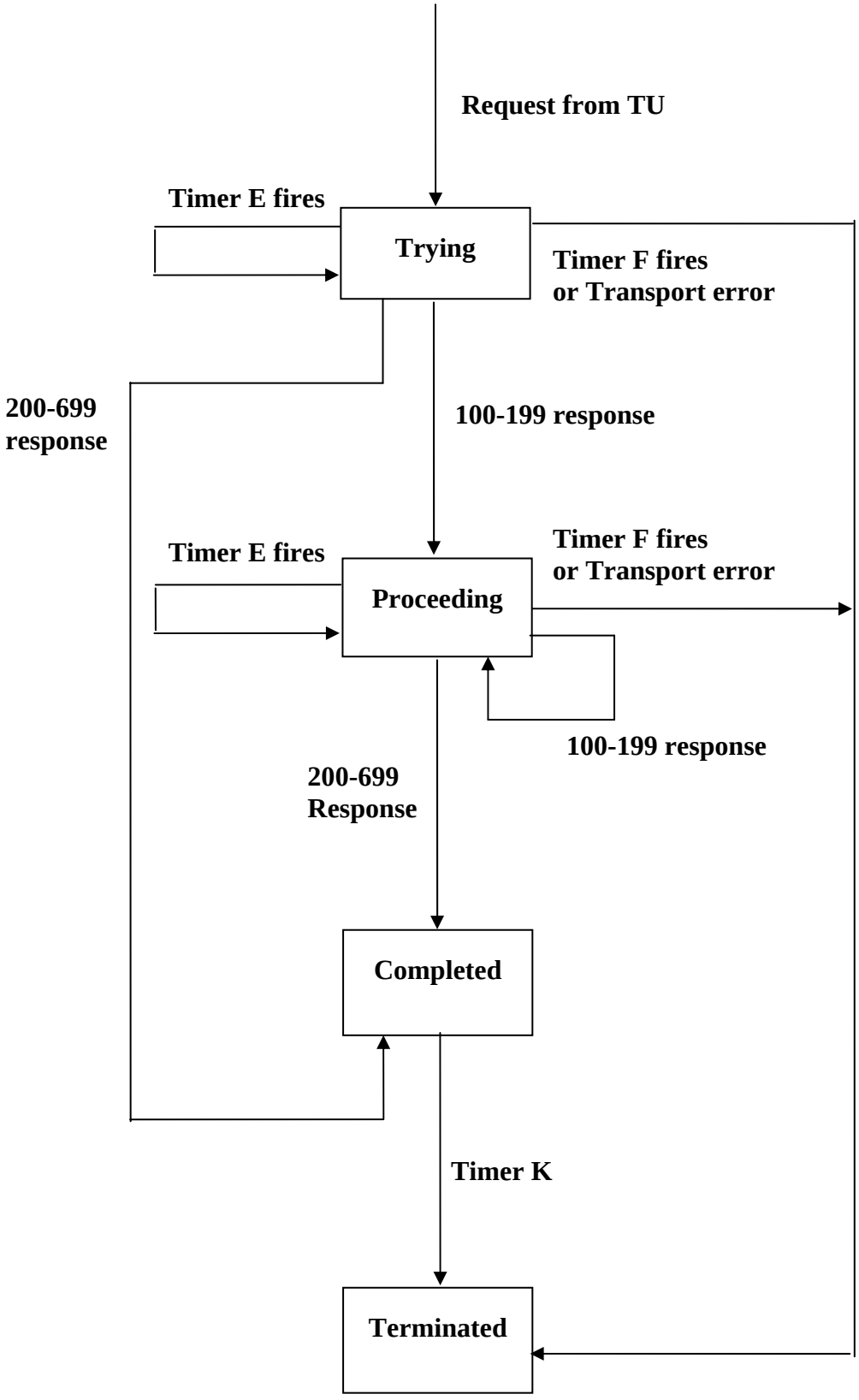

Figure 4.20 Non-INVITE Client Transaction

Step B: Proceeding state

Step B.1: If Timer_E is fires Then

Resend the request to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

Timer_E = T2

Step B.2: If a response (200-699) is received Then

Send the response to TU.

Go to Completed state.

If a response (100-199) is received Then

Send the response to TU.

Step B.3: If Timer_F is fires Then

Advertise TU of transaction timeout.

Go to Terminated state.

Else Go to Step B.1

Step C: Completed state

Step C.1: T4 = 5 sec.

If "transport" in first "Via" = "UDP" Then

Timer_K = T4

Else Timer_K = 0

Step C.2: Discard all received responses from server transaction

Step C.3: If Timer_K is fires Then

Go to Terminated state

Step D: Terminated state

Step D.1: Destroy the client transaction

- **Server Transaction**

The server transaction is responsible for:

1. Receive request from (transport layer) and deliver them to TU.
2. Filter any request retransmissions from the network.
3. Accepts responses from TU and deliver them to transport layer.
4. In case of (INVITE transaction), it absorbs the ACK request for any final response (non-1xx) excepting response (2xx).

INVITE Server Transaction

The procedure (4.11) as shown in figure (4.21) has number of timers there meaning in table (4.4)

Values of timers

Timer	Value	Meaning
T1	500 ms.	Estimate of round-trip time
T2	4 s.	The maximum retransmit interval for non-INVITE requests
T4	5 s.	Maximum duration a message will remain in the network
Timer_G	initially T1	Non-INVITE request retransmit interval, for UDP only
Timer_H	64*T1	Non-INVITE transaction timeout timer
Timer_I	T4 for UDP, 0s for TCP	Wait time for response retransmits

4.11 INVITE Server Transaction

Step A: Proceeding state

Step A.1: T1 = 500 ms

If receive new INVITE request from transport layer Then

Pass the request to TU.

T = 200 ms

If T is fires Then

Create response (100).

Send response (100) to transport layer.

If rereceive INVITE request from transport layer Then

Resend last response received from TU to transport layer.

Step A.2: If a response (100-99) is received from TU Then

Send the response to transport layer.

If a response (200) is received from TU Then

Send the response to transport layer.

Go to Terminated state.

If a response (300-699) is received from TU Then

Send the response to transport layer.

If "transport" in first "Via" = "UDP" Then

Timer_G = T1

Else Timer_G = 0

Go to Completed state

Step B: Completed state

Step B.1: Timer_H = 64 * T1

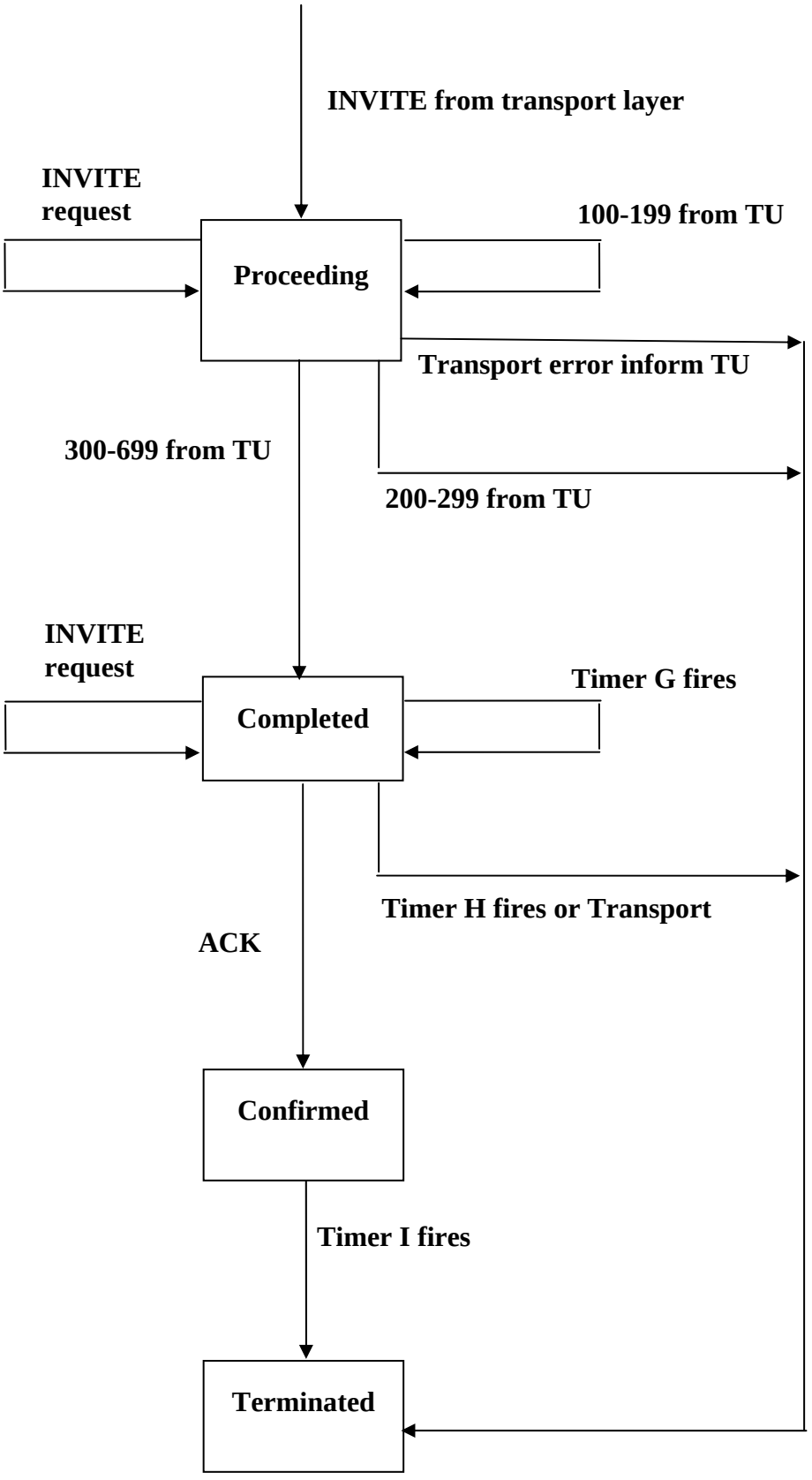

Figure 4.21 INVITE Server Transaction

If Timer_G is fires Then

Resend response (300-699) to transport layer.

Repeat until (Timer_H is timeout or receive a ACK request)

If Timer_G is fires Then

Resend response (300-699) to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

T1 = Min(2*T1,T2)

Timer_G = T1

Go to Step B.4

Step B.2: If rereceive INVITE request from transport layer Then

Resend last response received from TU to transport layer

Step B.3: If receive ACK request Then

Timer_G = 0

Go to Confirm state

Step B.4: If Timer_H is fires Then

Advertise TU for transaction failure.

Go to Terminated state.

Else Go to Step B.2

Step C: Confirm state

Step C.1: Timer_I = T4

If Timer_I is fires Then

Go to Terminated state.

Step D: Terminated state

Step D.1: Destroy the server transaction

Non-INVITE Server Transaction

The procedure (4.12) as shown in figure (4.22) has one timer (Timer_J is useful for controlling transaction timeout).

4.12 Non-INVITE Server Transaction

Step A: Trying state

Step A.1: Receive Non-INVITE (not an ACK request) from transport layer along with transaction it belongs to.

Step A.2: If a response (100-199) is received from TU Then

Send the response to transport layer.

Go to Proceeding state.

If a response (200-699) is received from TU Then

Send the response to transport layer.

Go to Complete state.

Step B: Proceeding state

Step B.1: If Non-INVITE request (not ACK request) is received Then

Send last response to transport layer

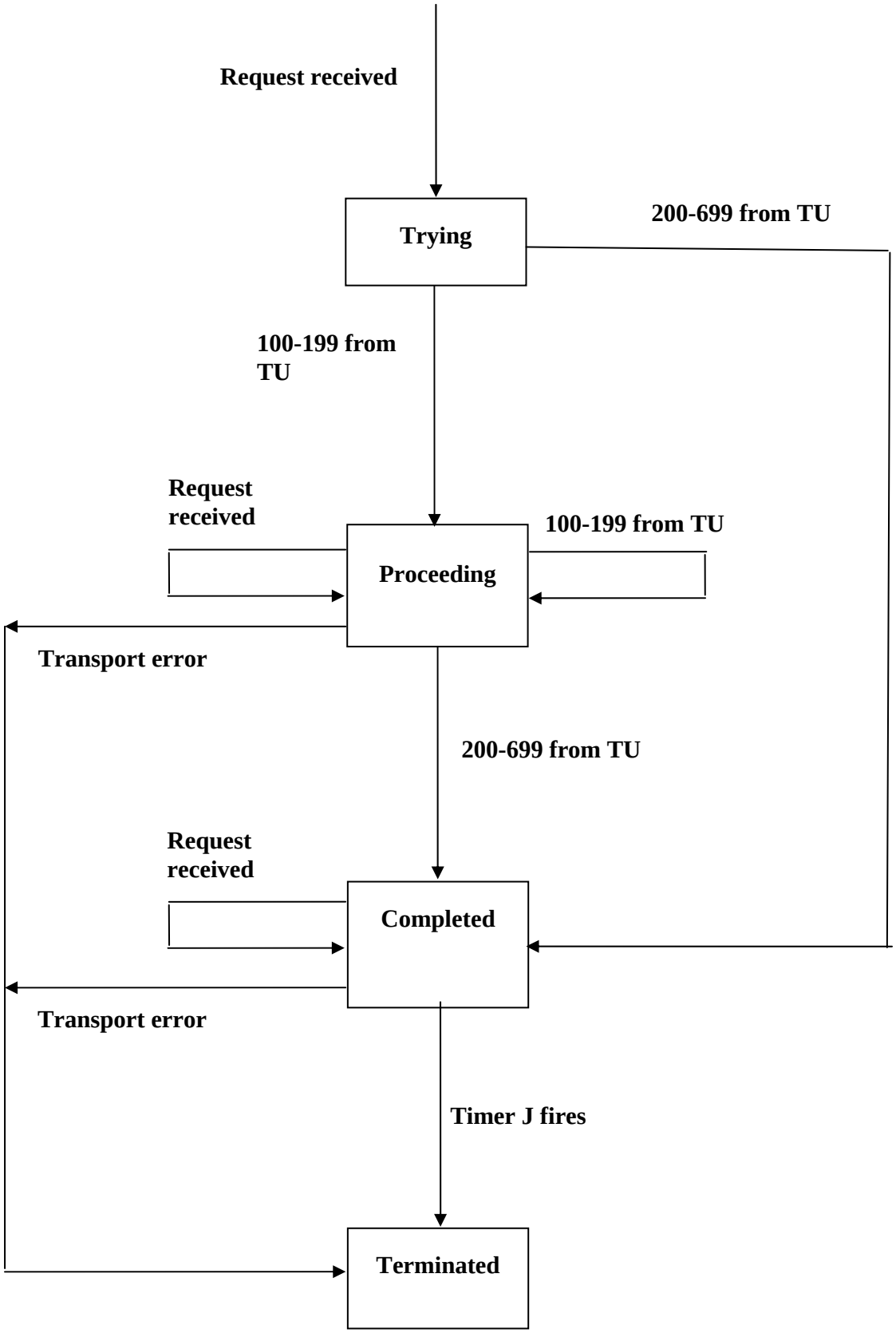

Figure 4.22 Non-INVITE Server Transaction

Step B.2: If a response (100-199) is received from TU Then

Send the response to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

If a response (200-699) is received from TU Then

Send the response to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

Go to Complete state.

Step C: Completed state

Step C.1: If "transport" in first "Via" = "UDP" Then

Timer_J = 64*T1

Else Timer_J = 0

Step C.2: If receive Non-INVITE request Then

Send final response to transport layer.

If error happens in sending Then

Inform TU.

Go to Terminated state.

Discard any other received responses (200-699).

Step C.3: If Timer_J is fires Then

Inform TU of transaction timeout.

Go to Terminated state.

Step D: Terminated state

Step D.1: Destroy the server transaction

4.2.1.4 Transaction User Layer

Here is the processing behavior of UAC and UAS.

UAC : client side of the transaction user layer is responsible for:

- **Creating or Generating Request**

For creating a request for the first time, follow the steps of procedure (4.13).

4.13 Create a Request

Step A: Case "Request-URI": Initiated by same value of "To"

Set version by 2.0

Case "To": take value by different ways

- Human interface, or
- Choose from address book

Case "From": Initiated by the current user address

Add "tag" parameter

Case "Call-ID": Generate a unique text

Case "Cseq": Choose unsigned integer 32-bit random number>0

Initiate method by the same method of the request

Case "Max-Forwards": initiated by 70

Case "via": Set the protocol name by "SIP

Set version by 2.0

A:= Generate a unique text

Add "Branch" parameter and set it with value of

("z9hg4bk" && A)

Case "Contact": initiated by same value of "To"

Case "Supported": list the supported features

Step B: if creating an initial INVITE request , follow these steps

Step B.1: construct a general request.

Step B.2: Allow = "INVITE", "ACK", "BYE"

Supported = empty

Step B.3: Add headers for description body

Content-Disposition = "session"

Parameter "handling" = "required"

Content-Length = no. >= 0

Content-Type = "application/sdp"

Step B.3: Add an offer SDP description to body of INVITE request. Add key field in media description, it has value of master key, encrypted by linear feedback shift register (LFSR) as illustrated in chapter (3) section (3.3.1.1).

Step C: Create a client transaction by saving values of ("branch" of the first "Via", and "Cseq" of the first "Via"), add it to the list of client transactions.

For generating a request, usually an ACK request message, do the following:

4.14 Generating a Request

Step A: Add request line, which means adding method name, request URI = destination IP address, and SIP version (2.0).

Step B: Add mandatory header fields

Add "Via" header field

To = add user name and (may or may not add final destination IP address)

From = IP address of user that created the request

Add "tag" parameter

Call-ID = unique text for each call (case-sensitive)

Cseq = add same method name that added in first line

add random number <=(0< I >= 2^31)

Max-Forwards = 70

Contact = final destination IP address

4.15 Generate ACK Request (Create_ACK)

Step A: Set "Call-ID", "From", and "Request-URI" in ACK request to same values exist in the request sent by client transaction.

Set "To" in ACK request to same value of "To" in the response to be acknowledged.

Set "Via" of ACK to the value of the first "Via" in the original request.

Set "Cseq" number to that "Cseq" in the original request.

Set "Cseq" method to "ACK".

Step B: Add "Tag" parameter to "To" header field of ACK request.

If INVITE request has "Route" header field Then

Set "Route" of ACK request to the value if "Route" in INVITE request.

Step C: Create a client transaction by saving values of ("branch" of the first

"Via", and "Cseq" of the first "Via"), add it to the list of client transactions.

- **Processing Response**

UAS process all responses as in procedure (4.16) except the retransmitted responses.

4.16 Processing Response

```
If there is more than one "Via" header field Then
                                Discard the response
                                Exit
Else If unrecognized provisional response different than 100 Then
                        Process responses (100 and 183)
                        Exit
If response is within 3xx Then
        Initiate target set with "Request-URI" value of the
        original request
        to make request based on this response do the following
        While target set is not empty do
                Set "Request-URI" of request by the value of URI in
                target set
                If connection is failed Then
                                Take new value from target set
                                Begin a new transaction for request
        If all connections are failed Then
                        The request is failed
```

Generate response with status code >399
"To" of response = "To" of original request
"From" of response = "From" of original request
"Call-ID" of response = "Call-ID" of original request
Construct the new request
Send the Request
If response within (400-499) Then
If response (401 or 407) Then
UAC must use authorization procedure
If response (413) Then
Retry request by omitting msg. body OR use smaller body
If response (415) Then
Retry send request within media type, encoding, and languages that supported by UAS
If response (416) Then
Retry send request within using SIP-URI
If response (420) Then
Retry send request by omitting any extension listed in "Unsupported"

UAS: server side of the transaction user layer process request and create responses as in procedure (4.17).

4.17 Process Request or Create a Response

Step A: method inspection
If request method doest not supported by UAS Then

Generate response (405)

Add "Allow" header field listing the allowed methods

Else Go to step3

Step B: header inspection

If extension header field is not supported or understood Then

Ignore this header field

If malformed header field & it not necessary in process Then

Ignore this header field

If "Require" header field is present Then

Ignore this header field

If not recognize URI of "To" or it not known add. Or it not the current user of this UAS Then

Accept the request

If reject the request Then

Generate response (403)

Pass response to server transaction

If "Request-URI" use scheme not supported Then

Reject request with response (416)

Else if "Request-URI" does not identify add. Accepted by UAS Then

Reject request with response (404)

If no "tag" in "To" Then

If ("tag" of "From" & "Call-ID" & "Cseq") match with ongoing transaction & not matched that transaction Then

Generate response (482)

Pass response to server transaction

If it is the proper UAS to process request &

"Require" is present & "option-tag" does not understood Then

Generate response (420)

Add "Unsupported" and list it with unsupported options

Step C: Content processing

If "Content-type" is not understood & msg. body is not optional as indicated in "Content-disposition" Then

Reject request with response (415)

Add "Accept" and list the accepted types

Else If "Content-encoding" is not understood & msg. body is not optional as indicated in "Content-disposition" Then

Reject request with response (415)

Add "Accept-encoding" and list the accepted encodings

Else If "Content-language" is not understood & msg. body is not optional as indicated in "Content-disposition" Then

Reject request with response (415)

Add "Accept-language" and list the accepted languages

Step D: Apply extensions

If no "Supported" in request Then

Generate response (421)

Add "Require" to the response

Step E: Processing request

If INVITE request Then

If user accepts the call Then

If INVITE req. has session description Then

If all description parameters is understood Then

Accept the request

Generate response (200)

Add "Allow" header field

Add "Supported" header field

Pass response (200) to transport layer, this is done directly to not destroy the server transaction.

Else Generate response (200) and must contain an offer (SDP description contain Key field encrypted using LFSR as illustrated in chapter (3) section (3.3.1.1).

Add "Allow" header field

Add "Supported" header field

Pass response (200) to transport layer, this is done directly to not destroy the server transaction.

If user not willing or able to answer the call Then

Generate response (486)

Pass the generated response to server transaction

If user reject the call Then

Generate response (488)

Pass the generated response to server transaction

If ACK request Then

Open media connection

If BYE request Then

Close the media connection

Generate response (200)

Step F: Generating response

If non-invite request Then

Generate final response (200) and add the SDP answer

If response (100) Then

Copy "timestamp" of request to response (100)

If there is a delay in response generation Then

Delay time + timestamp

"From" of response = "from" of request

"Call-ID" of response = "Call-ID" of request

"Cseq" of response = "Cseq" of request

"Via" values of response = "Via" values of request

If request has "tag" in "To" Then

"To" of response = "To" of request

Else URI of "To" of response = URI of "To" of request

Add "tag" to "To" of response

Step G: Create a server Transaction by saving values of ("branch" of the first "Via", "sent-by" of the first "Via", and request method), add it to the list of server transactions.

Step H: send response to transport layer

4.3 Result of Work

Simple version of UserA invites UserB to an SIP session. It begins by examining the details of session setup as in figure (4.19).

Message1:

INVITE sip:UserB@192.168.0.2 SIP/2.0

Via: SIP/2.0/UDP 192.168.0.1:5060;branch=z9hG4bK74bf9

Max-Forwards: 70

From: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

To: UserB <sip:UserB@192.168.0.2>

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 INVITE

Contact: <sip:UserA@192.168.0.1>

Content-Type: application/sdp

Content-Length: 151

v=0

o=UserA 2890844526 2890844526 IN IP4 192.168.0.1

s= Meeting

c=IN IP4 192.168.0.1

t=0 0

m=audio 49172 RTP/UDP 0

k=clear:910bc4defa71eb6190008762fca6ae2f1d959e87cdf3c0c5c5076ad38e8

a=rtpmap:0 PCM/8000

Message 2:

SIP/2.0 180 Ringing

Via: SIP/2.0/ UDP 192.168.0.1:5060;branch=z9hG4bK74bf9;

received=192.168.0.1

From: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

To: UserB <sip:UserB@192.168.0.2>;tag=8321234356

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 INVITE

Contact: <sip:UserB@192.168.0.2>

Content-Length: 0

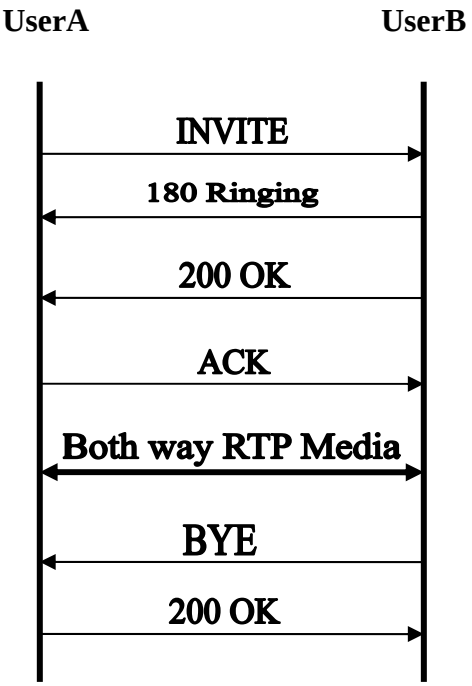

Figure 4.23 Invite Session

Message 3:

SIP/2.0 200 OK

Via: SIP/2.0/ UDP 192.168.0.1:5060;branch=z9hG4bK74bf9;

received=192.168.0.1

From: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

To: UserB <sip:UserB@192.168.0.2>;tag=8321234356

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 INVITE

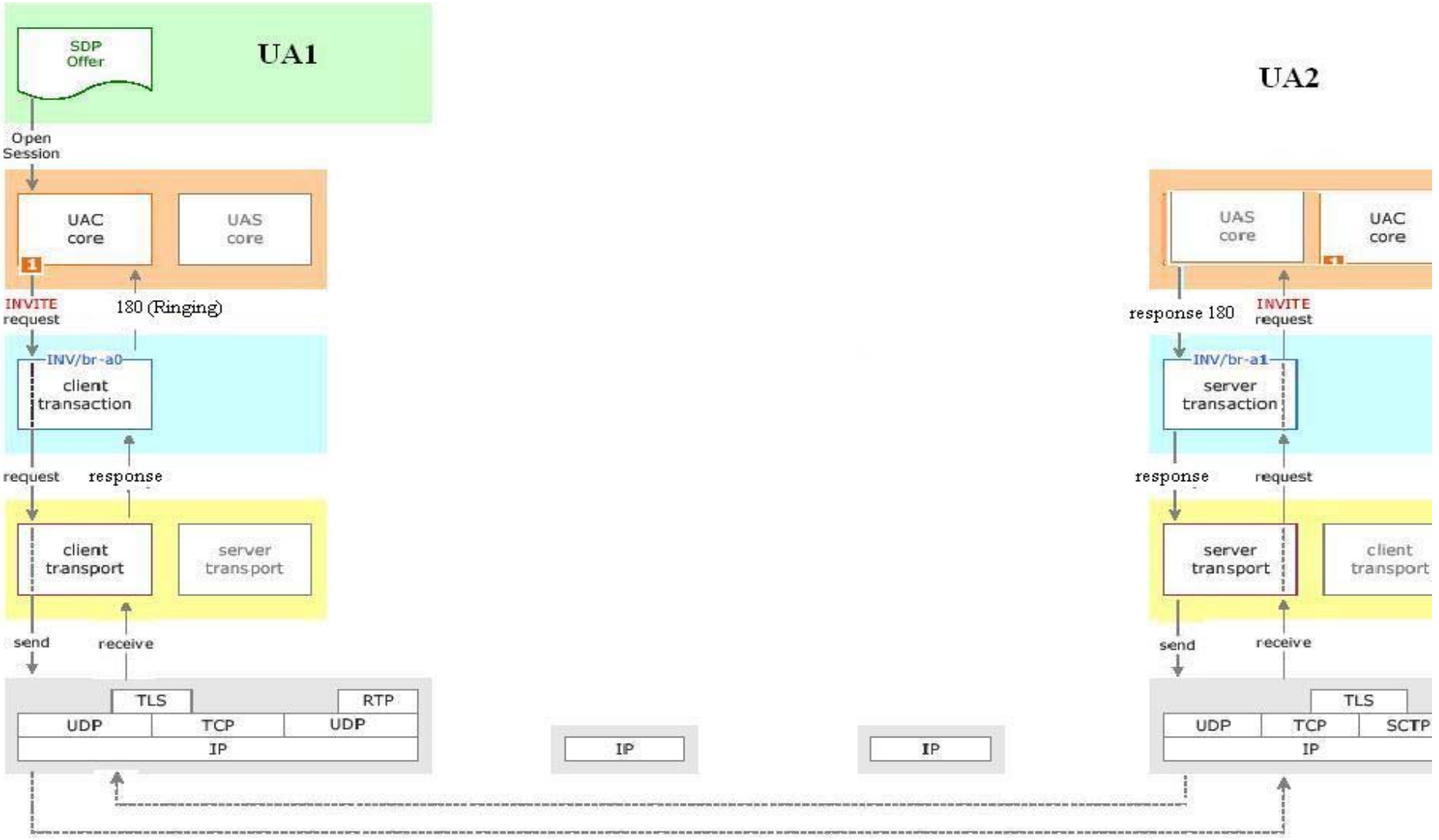

Figure 4.24 INVITE Request

Contact: <sip:UserB@192.168.0.2>

Content-Type: application/sdp

Content-Length: 147

v=0

o=bob 2890844527 2890844527 IN IP4 192.168.0.2

s= Meeting

c=IN IP4 192.168.0.2

t=0 0

m=audio 3456 RTP/UDP 0

a=rtpmap:0 PCM/8000

Message 4:

ACK sip:UserB@192.168.0.2 SIP/2.0

Via: SIP/2.0/ UDP 192.168.0.1:5060;branch=z9hG4bK74bd5

Max-Forwards: 70

From: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

To: UserB <sip:UserB@192.168.0.2>;tag=8321234356

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 ACK

Content-Length: 0

Now a successful set-up sequence is done. INVITE is the only method in SIP that involves a 3-way handshake with ACK.

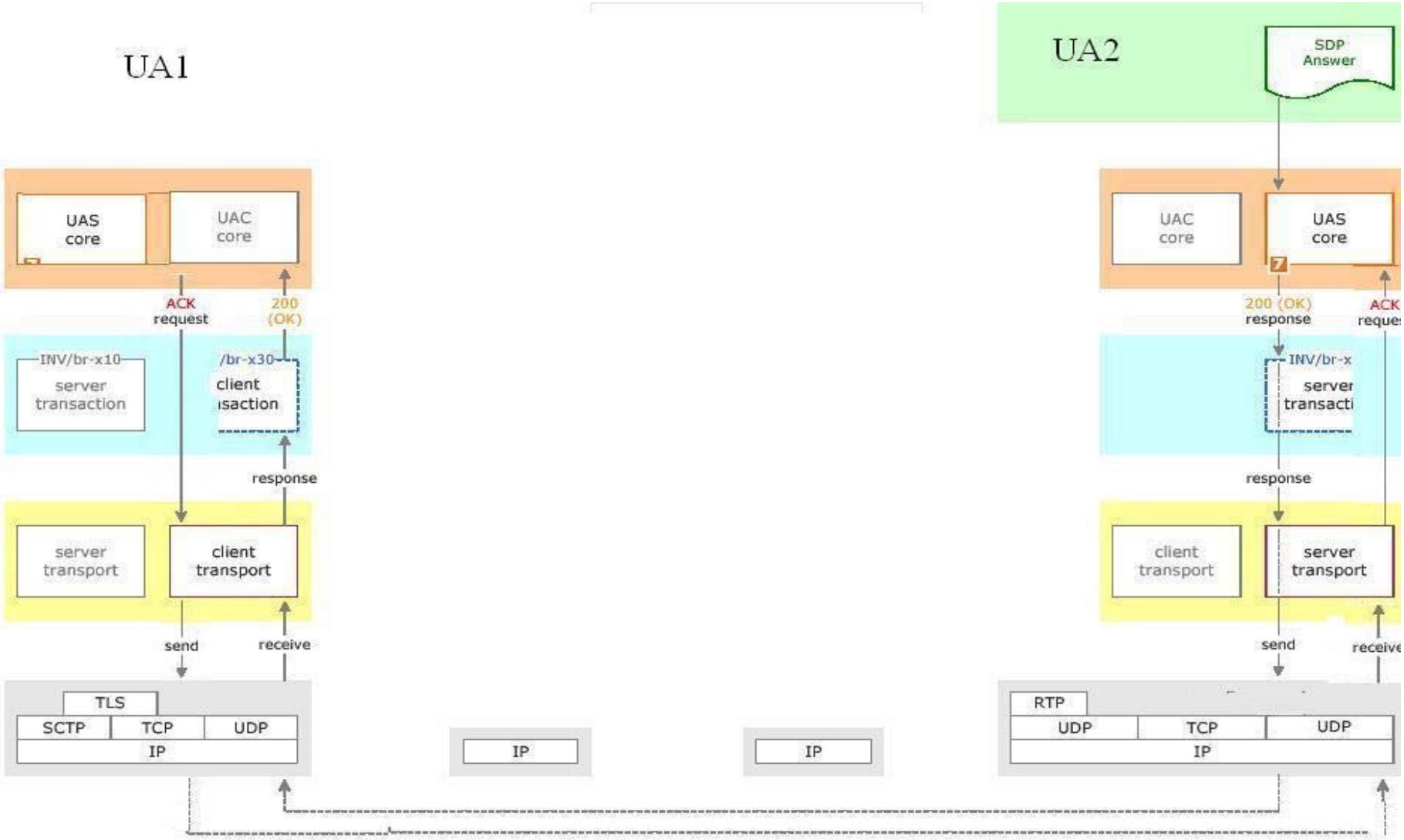

Figure 4.25 Accepting INVITE Request

Message 5:

BYE sip:UserA@192.168.0.1 SIP/2.0

Via:SIP/2.0/ UDP 192.168.0.2:5060;branch=z9hG4bKnashds7

Max-Forwards: 70

From: UserB <sip:UserB@192.168.0.2>;tag=8321234356

To: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 BYE

Content-Length: 0

Message 6:

SIP/2.0 200 OK

Via:SIP/2.0/UDP 192.168.0.2:5060;branch=z9hG4bKnashds7

;received=192.168.0.2

From: UserB <sip:UserB@192.168.0.2>;tag=8321234356

To: UserA <sip:UserA@192.168.0.1>;tag=9fxced76sl

Call-ID: 384827629822018851l@192.168.0.1

CSeq: 1 BYE

Content-Length: 0

Chapter Five
Conclusions and Suggestions for Future works

5.1 Conclusions

1. While there are some similarities between the protocols in call setup, there is a philosophy of purpose design for each one. In taking place, H.323 and SIP as two different encoding protocols. H.323 design and implementation reflects its PSTN background and heritage, utilizing binary encoding and reusing parts of Integrated Services Digital Network (ISDN) signaling, this is because H.323 is base on Signaling System 7 (SS7) signaling protocol of the PSTN. On the other hand, SIP was developed with an Internet perspective, this is because of SIP is based on HTTP 1.1 protocol, designed to be scalable over the Internet and work in an interdomain way utilizing the full set of Internet utilities and functions. So SIP with its text encoding, and Internet architecture, is poised to be the signaling and “rendezvous” protocol of choice for TCP/IP model.

2. Signaling part of VoIP consist of SIP and it description protocol (SDP), both of them are text encoding. It observed from the programming perspective, syntax and encoding layer of SIP takes more time than other layers. From processing perspective, parsing the total packet for (new received, or rereceived) makes consuming much more time for parsing than other layers, so it have to limit parsing process to only the necessary header fields.

3. SIP protocol does not have a robust on private networks because of retransmission of request and responses when using UDP transport protocol. UDP protocol is the default transport protocol even with SIPv1.

5.2 Suggestions for Future Works

1. Implement a P2P registration.
2. Implement the SRTP protocol then calculate the QoS that affect for the implementation of application layer security.
3. Implement SIPS then analyze the security strength.

References

[Ala04] Alan B., ***Understanding the Session Initiation Protocol***, second edition, Artech House, 2004.

[Ala07] Alan G., ***Computer Security and Cryptography***, John Wiley & Sons, 2007.

[And04] Andreas S., Daniel K., ***SIP Security***, Security Group, 2004.

[Ant06] Antonio R., and Michael C., ***Cisco Voice over IP***, Cisco Press, 2006.

[Ark05] Arkko J., Carrara E., Lindholm F., et.al., ***Key Management Extensions for Session Description Protocol (SDP) and Real Time Stream Protocol (SRTP)***, Internet Engineering Task Force (IETF), March 2005.

[Bau04] Baugher M., McGrew D., Naslund M., et.al., ***The Secure Real-time Transport Protocol (SRTP)***, RFC 37, Internet Engineering Task Force (IETF), March 2004.

[Bil00] Bill D., ***IP Telephony: Integration of Robust VOIP services***, Hewlett-Packard, 2000.

[Chr02] Chris P., Johne A., Anne S., et.al., ***Cisco CallManager Fundamentals***, Cisco Press, 2002.

[Cli06] Clinton J., and Ziad S., ***Adding High Level VoIP Facilities to the Unicon Languag***, IEEE, 2006.

[Col03] Colin P., ***RTP: Audio and Video for the Internet***, Pearson Education, 2003.

[Dav01] David J., ***Voice over Packet Networks***, John Wiley & Sons, 2001.

[Dav05] David A., ***Analysis and Implementation of TCP Friendly Rate Control in the Context of VoIP***, Royal institute of technology, M.Sc. thesis, 2005.

[Die99] Dierks T., and Allen C., ***The TLS Protocol Version 1.0***, Internet Engineering Task Force (IETF), January 1999.

[Eri02] Eric K., and Paul J., ***Configuring Cisco Voice over IP***, second edition, Syngress Publishing, 2002.

[Fre00] Fredrik T., ***SIP, NAT, and Firewalls***, Royal institute of technology, department of Teleinformatics, M.Sc. thesis, 2000.

[Gon02] Gonzalo C., ***SIP Demystified***, McGraw-Hill, 2002.

[Jam05] James E., ***Talk is Cheap***, O'Reilly Media, 2005.

[Jim02] Jim D., Gerard J., et.al., ***Voice-over-IP: The Future of Communications***, Global Internet Policy Initiative (GIPI), 2002.

[Joh02] Johann T., ***Security in VoIP-Telephony Systems***, GRAZ university of technology, department of IP-Telephony, M.Sc. thesis, 2002.

[Joh04] John Q., and Jeffrey T., ***Taking Charge of Your VOIP Project***, Cisco Press, 2004.

[Jon00] Jonathan D., and James P., ***Voice over IP fundamentals***, Cisco Press, 2000.

[Jor00] Jori L., ***Voice over IP in networked virtual environments***, Maastricht university, department of Computer Science, PhD thesis, 2000.

[Jos05] Josué A., ***Using the Session Initiation Protocol as a Networking Protocol for Home Applications***, university of Puerto Rico, department of Computer Engineering, M.Sc. thesis, 2005.

[Jua07] Juan C., Eduardo B., et.al., ***Patterns for VoIP Signaling Protocol Architectures***, Florida Atlantic university, department of Computer Science and Engineering, and university of Oulu, department of Electrical and Information Engineering, 2007.

[Ken07] Kevin W., ***Authorized Self-Study Guide Cisco Voice over IP (CVoice)***, Cisco Press, 2007.

[Lin04] Lingfen S., ***Speech Quality Prediction for Voice over Internet Protocol Networks***, university of Plymouth, department of Communications and Electronics, PH.D. thesis, 2004.

[Pau97] Paul O., and David H. Crocker, ***Augmented BNF for Syntax Specifications: ABNF***, RFC 2234**,** Internet Engineering Task Force (IETF), November 1997.

[Ras99] Rescorla E., ***Diffie-Hellman Key Agreement Method***, RFC 2631, Internet Engineering Task Force (IETF), 1999.

[Ros02] Rosenberg J., Schulzrinne H., Camarillo G., et.al., ***SIP: Session Initiation Protocol***, RFC 3261, Internet Engineering Task Force (IETF), June 2002.

[SC03] Schulzrinne, H.; Casner, S., ***RTP Profile for Audio and Video Conferences with Minimal Control***, RFC 3551, Internet Engineering Task Force (IETF), 2003.

[Sha04] Shamim M., Clinton J., Ray P., et.al., ***Programming with Unicon***, 2004.

http://www.unicon.org/book/ub.pdf

[Siv01] Sivavakeesar S., ***Voice over Internet Protocol (VOIP)***, university of Surrey, department of Electronic and Electrical Engineering, M.Sc. thesis, 2001.

[Ted05] Ted W., ***Switching to VOIP***, O'Reilly Media, 2005.

[Tha98] Thayer R., Doraswamy N., and Glenn R., ***IP Security Document Roadmap***, RFC 2411, Internet Engineering Task Force (IETF)**,** November 1998.

[Tho06] Thomas P., Jan K., Andy Z., et.al., ***Practical VOIP security***, Syngress, 2006.

[Tim94] Tim B., ***Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web***, RFC 1630, Internet Engineering Task Force (IETF),June 1994

[Zia05] Zian S., ***High Level Audio Communications API for the Unicon Language***, New Mexico State university, department of Computer Science, M.Sc. thesis, 2005.

Appendix A: Unicon Language

Unicon is stand for *Unified Extended Dialect of Icon,* The name unicon refers to the descendant of Icon. It can be pronounced it either (a) "lexicon", or (b) is pronounced as if it stands for "un-Icon".

Unicon's predecessor is icon language, a highly portable language; it runs on everything from mainframes to Unix machines to Amigas and Macs. It is a very high level, goal-directed, object-oriented, general purpose applications language. This language is extended with capabilities such as object-oriented programming and database management that are useful in many programs, especially larger programs that use the Internet.

In short these are the features of unicon language:-

- Packaging, it is similar to a class with only one instance.
- Loadable child programs
- Ministering of child programs
- Dynamic loading of C modules (some platforms)
- Multiple inheritance
- DBM files can be used associative arrays
- POSIX system interface.

Be aware that unicon performs automatic storage management, also known as garbage collection. If you have used a language like C or C++, you know that one of the biggest headaches in writing programs in these languages is

tracking down bugs caused by memory allocation, especially dynamic heap memory allocation. unicon transparently takes care of those issues for you.

Another big source of bugs in languages like C and C++ are pointers, values that contain raw memory addresses. Used properly, pointers are powerful and efficient. The problem is that they are easy to use incorrectly by accident; this is true for students and practicing software engineers alike. It is easy in C to point at something that is off-limits, or to trash some data through a pointer of the wrong type.

Unicon has no pointer types. Instead, all structure values implicitly use pointer semantics. A reference is a pointer for which type information is maintained and type safety is strictly enforced.

Unicon's system facilities provide a high-level interface to the most common features of modern operating systems, such as directories and network connections. This interface is vital to most applications, and it also presents the main portability challenges, since unicon has a design goal that most applications should require no source code changes and no conditional code needed to run on most operating systems. Of course some application domains such as system administration are inevitably platform dependent.

This language provides object-oriented programming facilities as a collection of tools to reduce the complexity of large programs. These tools are encapsulation, inheritance, and polymorphism. Since a primary goal of unicon is to promote code sharing and reuse, various specific programming problems have elegant solutions available in the unicon class library.

Also its string processing facilities are extensive. Simple operations are very easy, while more complex string analysis has the support of a special control structure, string scanning. String scanning is not as concise as regular expression pattern matching, but it is fundamentally more general because the code and patterns are freely intermixed [Sha04]. There is some main differences between the C-based lexical analyzer and the Unicon lexical analyzer for this language include:

- The C lexical analyzer has to process string escapes; Unicon does not because its output is processed by icont/iconc.
- The C lexical analyzer does not use hash tables, it uses static arrays and sneaky tricks to make lookups fast.
- The C lexical analyzer has code to worry about Extended Binary Coded Decimal Interchange Code (EBCDIC)!
- Small inhale, character preprocessor interface: the C version reads 1 character at a time from a function named ppch(), while the Unicon version was grabbing input from the preprocessor a line at a time using a single call to a generator (apparently that 1000000 bug was in "dead code"):

A programming language with built-in VoIP functions can accelerate the development of voice enabled applications. Most programmers are familiar with manipulating files; making VoIP connections as simple to create and use as local files simplifies VoIP programming and its applications.

Recently unicon facilitated with VoIP, a very high level supporting peer-to-peer, one-to-many, and many-to-many VoIP sessions. Unicon's VoIP facilities provide full duplex voice conversations by having each client as a destination in the other's voice session [Cli06].

Appendix B: Parsing Rules of SIP and SDP

B.1 SIP

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) defined in RFC (2234). Section 6.1 of RFC (2234) defines a set of core rules that are used by this specification. Implementers need to be familiar with the notation and content of RFC (2234) in order to understand this specification. Certain basic rules are in uppercase, such as SP (SPace), LWS (Linear White Space), HTAB (Horizontal TAB), CRLF (Carriage Return Line Feed), DIGIT, ALPHA, etc. Angle brackets are used within definitions to clarify the use of rule names.

The use of square brackets is redundant syntactically. It is used as a semantic hint that the specific parameter is optional to use [Pau97, Ros02].

ALPHA = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT = %x30-39; 0-9
HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
WSP = SP / HTAB; White SPace (WSP)
CRLF = %d13.10; Internet standard newline
SP = %x20
HTAB = %x09; horizontal tab
OCTET = %x00-FF; 8 bits of data
DQUOTE = %x22
alphanum = ALPHA / DIGIT
reserved = ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+" / "$" / ","
unreserved = alphanum / mark

```
mark                = "-" / "_" / "." / "!" / "~" / "*" / "'" / "(" / ")"
escaped             = "%" HEXDIG HEXDIG
LWS                 = [*WSP CRLF] 1*WSP ; linear whitespace
SWS                 = [LWS] ; sep whitespace
HCOLON              = *(SP / HTAB) ":" SWS
UTF8-NONASCII = %xC0-DF 1UTF8-CONT
                      / %xE0-EF 2UTF8-CONT
                      / %xF0-F7 3UTF8-CONT
                      / %xF8-Fb 4UTF8-CONT
                      / %xFC-FD 5UTF8-CONT
UTF8-CONT     = %x80-BF
token               = 1*(alphanum / "-" / "." / "!" / "%" / "*" / "_" / "+" / "`"
                      / "'" / "~")
separators          = "(" / ")" / "<" / ">" / "@" / "," / ";" / ":" / "\" / DQUOTE
                      / "/" / "[" / "]" / "?" / "=" / "{" / "}" / SP / HTAB
word                = 1*(alphanum / "-" / "." / "!" / "%" / "*" / "_" / "+" / "`"
                      / "'" / "~" / "(" / ")" / "<" / ">" / ":" / "\" / DQUOTE / "/"
                      / "[" / "]" / "?" / "{" / "}")
SLASH               = SWS "/" SWS ; slash
EQUAL               = SWS "=" SWS ; equal
LPAREN              = SWS "(" SWS ; left parenthesis
RPAREN              = SWS ")" SWS ; right parenthesis
RAQUOT              = ">" SWS ; right angle quote
LAQUOT              = SWS "<"; left angle quote
COMMA               = SWS "," SWS ; comma
SEMI                = SWS ";" SWS ; semicolon
```

```
COLON            = SWS ":" SWS ; colon
quoted-string    = SWS DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext           = LWS / %x21 / %x23-5B / %x5D-7E
                   / UTF8-NONASCII
quoted-pair      = "\" (%x00-09 / %x0B-0C / %x0E-7F)

SIP-URI          = "sip:" [userinfo] hostport uri-parameters
                   [headers]
SIPS-URI         = "sips:" [userinfo] hostport uri-parameters
                   [headers]
userinfo         = user [":" password] "@"
user             = 1*(unreserved / escaped / user-unreserved)
user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password         = *(unreserved / escaped / "&" / "=" / "+" / "$" / ",")
hostport         = host [":" port]
host             = hostname / IPv4address
hostname         = *(domainlabel ".") toplabel ["."]
domainlabel      = alphanum / alphanum *(alphanum / "-") alphanum
toplabel         = ALPHA / ALPHA *(alphanum / "-") alphanum
IPv4address      = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
port             = 1*DIGIT
uri-parameters   = *(";" uri-parameter)
uri-parameter    = transport-param / user-param / method-param / ttl-param
                   / maddr-param
transport-param = "transport=" ("udp" / "tcp")
user-param       = "user=" ("ip" / other-user)
```

```
other-user          = token
method-param        = "method=" Method
ttl-param           = "ttl=" ttl
maddr-param         = "maddr=" host
headers             = "?" header *("&" header)
header              = hname "=" hvalue
hname               = 1*(hnv-unreserved / unreserved / escaped)
hvalue              = *(hnv-unreserved / unreserved / escaped)
hnv-unreserved      = "[" / "]" / "/" / "?" / ":" / "+" / "$"
SIP-message         = Request / Response
Request             = Request-Line *(message-header) CRLF
                      [message-body]
Request-Line        = Method SP Request-URI SP SIP-Version CRLF
Request-URI         = SIP-URI / SIPS-URI
SIP-Version         = "SIP" "/" 1*DIGIT "." 1*DIGIT
message-header      = (Call-ID / Call-Info / Contact
                       / Content-Disposition / Content-Encoding
                       / Content-Language / Content-Length
                       / Content-Type / CSeq / From / Max-Forwards
                       / To / Allow / Supported / Via) CRLF
INVITEm             = %x49.4E.56.49.54.45 ; INVITE in caps
ACKm                = %x41.43.4B ; ACK in caps
OPTIONSm            = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm                = %x42.59.45 ; BYE in caps
CANCELm             = %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm           = %x52.45.47.49.53.54.45.52 ; REGISTER in caps
```

```
Method          = INVITEm / ACKm / OPTIONSm / BYEm
                  / CANCELm / REGISTERm
Response        = Status-Line *(message-header) CRLF
                  [message-body]
Status-Line     = SIP-Version SP Status-Code SP Reason-Phrase
                  CRLF
Status-Code     = Informational / Success / Client-Error / Global-Failure
Reason-Phrase   = *(reserved / unreserved / escaped
                  / UTF8-NONASCII / UTF8-CONT / SP / HTAB)
Informational   = "100" ; Trying
                  / "180" ; Ringing
                  / "181" ; Call Is Being Forwarded
                  / "182" ; Queued
                  / "183" ; Session Progress
Success         = "200" ; OK
Client-Error    = "400" ; Bad Request
                  / "401" ; Unauthorized
                  / "402" ; Payment Required
                  / "403" ; Forbidden
                  / "404" ; Not Found
                  / "405" ; Method Not Allowed
                  / "406" ; Not Acceptable
                  / "407" ; Proxy Authentication Required
                  / "408" ; Request Timeout
                  / "410" ; Gone
                  / "413" ; Request Entity Too Large
```

```
                    / "414" ; Request-URI Too Large
                    / "415" ; Unsupported Media Type
                    / "416" ; Unsupported URI Scheme
                    / "420" ; Bad Extension
                    / "421" ; Extension Required
                    / "423" ; Interval
                    / "480" ; Temporarily not available
                    / "481" ; Call Leg/Transaction Does Not Exist
                    / "482" ; Loop Detected
                    / "483" ; Too Many Hops
                    / "484" ; Address Incomplete
                    / "485" ; Ambiguous
                    / "486" ; Busy Here
                    / "487" ; Request Terminated
                    / "488" ; Not Acceptable Here
                    / "491" ; Request Pending
                    / "493" ; Undecipherable
Global-Failure    = "600" ; Busy Everywhere
                    / "603" ; Decline
                    / "604" ; Does not exist anywhere
                    / "606" ; Not Acceptable
Call-ID           = ("Call-ID" / "i") HCOLON callid
callid            = word ["@" word]
Contact           = ("Contact" / "m") HCOLON (contact-param
                    *(COMMA contact-param))
contact-param     = (name-addr / addr-spec) *(SEMI contact-params)
```

```
name-addr            = [display-name] LAQUOT addr-spec RAQUOT
addr-spec            = SIP-URI / SIPS-URI
display-name         = *(token LWS) / quoted-string
contact-params       = c-p-q / c-p-expires
c-p-q                = "q" EQUAL qvalue
c-p-expires          = "expires" EQUAL delta-seconds
qvalue               = ("0" ["." 0*3DIGIT]) / ("1" ["." 0*3("0")])
delta-seconds        = 1*DIGIT
Content-Disposition = "Content-Disposition" HCOLON disp-type *(SEMI
                        disp-param)
disp-type            = "render" / "session" / "alert"
disp-param           = handling-param / generic-param
generic-param        = token [EQUAL gen-value]
gen-value            = token / host / quoted-string
handling-param       = "handling" EQUAL ("optional" / "required")
Content-Encoding     = ("Content-Encoding" / "e") HCOLON
                        content-coding *(COMMA content-coding)
content-coding = token
Content-Language     = "Content-Language" HCOLON
                        language-tag *(COMMA language-tag)
language-tag         = primary-tag *("-" subtag)
primary-tag          = 1*8ALPHA
subtag               = 1*8ALPHA
Content-Length       = ("Content-Length" / "l") HCOLON 1*DIGIT
Content-Type         = ("Content-Type" / "c") HCOLON media-type
media-type           = m-type SLASH m-subtype *(SEMI m-parameter)
```

m-type = discrete-type / composite-type

discrete-type = "text" / "image" / "audio" / "video" / "application"

composite-type = "message" / "multipart"

m-subtype = extension-token

extension-token = ietf-token / x-token

ietf-token = token

x-token = "x-" token

m-parameter = m-attribute EQUAL m-value

m-attribute = token

m-value = token / quoted-string

CSeq = "CSeq" HCOLON 1*DIGIT LWS Method

From = ("From" / "f") HCOLON from-spec

from-spec = (name-addr / addr-spec) *(SEMI from-param)

from-param = tag-param / generic-param

tag-param = "tag" EQUAL token

Max-Forwards = "Max-Forwards" HCOLON 1*DIGIT

To = ("To" / "t") HCOLON (name-addr / addr-spec)
*(SEMI to-param)

to-param = tag-param / generic-param

Allow = "Allow" HCOLON [Method *(COMMA Method)]

Supported = ("Supported" / "k") HCOLON [option-tag
*(COMMA option-tag)]

option-tag = token

Via = ("Via" / "v") HCOLON via-parm *(COMMA
via-parm)

via-parm = sent-protocol LWS sent-by *(SEMI via-params)

via-params = via-ttl / via-maddr / via-received / via-branch
via-ttl = "ttl" EQUAL ttl
via-maddr = "maddr" EQUAL host
via-received = "received" EQUAL (IPv4address)
via-branch = "branch" EQUAL token
sent-protocol = protocol-name SLASH protocol-version SLASH
transport
protocol-name = "SIP" / token
protocol-version = token
transport = "UDP" / "TCP"
sent-by = host [COLON port]
ttl = 1*3DIGIT ; 0 to 255
message-body = *OCTET

B.2 SDP

This section provides an Augmented BNF grammar for Session Description Protocol (SDP) [Ark05, Tim94].

announcement = proto-version
origin-field
session-name-field
information-field
URI-field
email-fields
connection-field
bandwidth-fields
time-fields
key-field

```
                    attribute-fields
                    media-descriptions
proto-version       = "v=" 1*DIGIT CRLF ;this memo describes version 0
origin-field        = "o=" username SP sess-id SP sess-version SP
                      nettype SP addrtype SP addr CRLF
session-name-field = "s=" text CRLF
information-field   = ["i=" text CRLF]
URI-field           = ["u=" URI CRLF]
email-fields        = *("e=" email-address CRLF)
connection-field    = ["c=" nettype SP addrtype SP connection-address
                      CRLF] ;a connection field must be present
                              in every media description or at the
                              session-level
bandwidth-fields    = *("b=" bwtype ":" bandwidth CRLF)
time-fields         = 1*("t=" start-time SP stop-time *(CRLF
                      repeat-fields) CRLF) [zone-adjustments CRLF]
repeat-fields       = "r=" repeat-interval SP typed-time
                      1*(SP typed-time)
zone-adjustments    = time SP ["-"] typed-time *(SP time SP ["-"]
                      typed-time)
key-field           = ["k=" key-type CRLF]
key-type            = "prompt"
                      / "clear:" key-data
                      / "base64:" key-data
                      / "uri:" URI
key-data            = email-safe / "~"
```

```
attribute-fields      = *("a=" attribute CRLF)
media-descriptions = *( media-field information-field *(connection-field)
                        bandwidth-fields key-field attribute-fields )
media-field           = "m=" media SP port ["/" integer] SP proto 1*(SP fmt)
                        CRLF
media                 = 1*(alphanum) ;typically "audio", "video",
                                      "application";or "data"
fmt                   = 1*(alphanum) ;typically an RTP payload type for
                                       audio ;and video media
proto                 = 1*(alphanum) ;typically "RTP/AVP" or "udp" for
                                      IP4
port                  = 1*(DIGIT) ;should in the range "1024" to "65535"
                                  inclusive for UDP based media
attribute             = (att-field ":" att-value) / att-field
att-field             = 1*(alphanum)
att-value             = byte-string
sess-id               = 1*(DIGIT) ;should be unique for this originating
                                  username/host
sess-version          = 1*(DIGIT) ;0 is a new session
connection-address = addr
start-time            = time / "0"
stop-time             = time / "0"
time                  = POS-DIGIT 9*(DIGIT) ;sufficient for 2 more
                                            centuries
repeat-interval       = typed-time
typed-time            = 1*(DIGIT) [fixed-len-time-unit]
```

```
fixed-len-time-unit = "d" / "h" / "m" / "s"
bwtype              = 1*(alphanum)
bandwidth           = 1*(DIGIT)
username            = safe ;pretty wide definition, but doesn't include space
email-address       = email / email "(" email-safe ")" /
                      email-safe "<" email ">"
email               = address-spec                    ; simple address
                      / 1*word route-addr      ; name & addr-spec
route-addr          =  "<" address-spec ">"
address-spec        =  local-part "@" domain    ; global address
local-part          =  word *("." word)
domain              =  sub-domain *("." sub-domain)
sub-domain          =  domain-ref / domain-literal
domain-ref          =  atom         ; symbolic reference
domain-literal      = "[" *(dtext / quoted-pair) "]"
dtext               = <any CHAR excluding "[", "]", "\" & CR, &
                       including linear-white-space>
atom                =  1*<any CHAR except specials, SPACE and CTLs>
URI                 = uri ["#" fragmentid]
uri                 = scheme ":" path ["?" search]
scheme              = ialpha
path                = void / xpalphas [ "/" path ]
search              = xalphas [+ search]
fragmentid          = xalphas
xalpha              = ALPHA / DIGIT / safechar / extra / escaped
xalphas             = xalpha [xalphas]
```

xpalpha = xalpha / +

xpalphas = xpalpha [xpalpha]

ialpha = ALPHA [xalphas]

safechar = "$" / "-" / "_" / "@" / "." / "&"

extra = "!" / "*" / "'" / "’" / "(" / ")" / ","

void

nettype = "IN" ;list to be extended

addrtype = "IP4" ;list to be extended

addr = unicast-address

unicast-address = IPv4address

text = byte-string ;default is to interpret this as IS0-10646
UTF8 ISO 8859-1 requires a
a=charset:ISO-8859-1" session-level
attribute to be used

byte-string = 1*(0x01..0x09/0x0b/0x0c/0x0e..0xff) ;any byte
except NUL,
CR or LF

integer = POS-DIGIT *(DIGIT)

POS-DIGIT = "1"/"2"/"3"/"4"/"5"/"6"/"7"/"8"/"9"

email-safe = safe / SP / tab

safe = alphanum / "’" / "’" / "-" / "." / "/" / ":" / "?" / "'" /
"#" / "$" / "&" / "*" / ";" / "=" / "@" / "[" / "]" / "^" / "_"
/ "‘" / "{" / "|" / "}" / "+" / "~"

tab = %d9

الملخص

في بيئة اليوم, تقريبا كل الاتصالات الهاتفية (نهاية-الى-نهاية) تقام عبر دائرة-تحويل باستخدام شبكة الهاتف التحويلية العامة (بي أس تي أن)، حيث ان الربط الداخلي (عقده-الى-عقده) ما بين المنشأ والمقصد تقام طريق جهاز رابط، والاتصال يحافظة عليه حصرا لتبادل المعلومات ما بين المنشأ والمقصد حتى انهاء ذلك الاتصال. طريقة اخرى لاقامة اتصالات (نهاية-الى-نهاية) والذي يستخدم على نطاق واسع لنقل البيانات هي (تبادل-حزم)، حيث اتصالات (المنشأ-الى-المقصد) تتم عن طريق الربط (عقده-الى-عقده)، الخزن-و-الارسال يعتمد على قطاعات صغيرة من مجموعة من البيانات التي يعاد تجميعها في المقصد؛ هذا الاسلوب يسمى الصوتِ على الإنترنتِ بروتوكول (ڤي أو آي پي). (ڤي أو آي پي) يعتبر الجيل الثالث من الاتصالات الهاتفية بعد تكنولوجيا الاتصالات التناظريه والرقميه.

هذه الاطروحه تاخذ بدراسة المعماريه (تبادل-حزم) الهاتف وثم تحلل بنية تكنولوجيا (ڤي أو آي پي)، وهو بروتوكول السيطرة على الارسال/بروتوكول الانترنت (تي سي پي/آي پي) باستخدام بروتوكول تلقين الجلسة (اس آي پي) من اجل السيطرة على المكالمه، وبروتوكول وصف الجلسة لوصف الوسائط المتعددة للجلسة، وبروتوكول النقل الفوري(آر تي پي) لتبادل الوسائط, هذه البروتوكولات مستقرة في طبقة التطبيقات للشبكة.

على مدى سنوات، الاهتمام بالجانب الأمني اخذ في الازدياد. لتوفير الخصوصيه لمستخدم المحادثة في (ڤي أو آي پي)، ثمة الحاجة الى تنفيذ الأمن لارسال الصوت. بروتوكول النقل الفوري الآمن (أس آر تي پي) هو مصمم لتوفير الامن لبرامج النقل الفوري باستخدام طريقه لتشفير الوسائط المتعددة (الصوت) ، لكنه لا يقدم مفتاح الاتفاق بين المشاركين. هذه الاطروحة تنفذ بروتوكول تلقين الجلسة (اس آي پي) وموافقة المفتاح باستخدام بروتوكول مفتاح الاتفاق المسبق ضمن بروتوكول وصف الجلسة (اس دي پي) المستخدم من قبل بروتوكول تلقين الجلسة. تم بناء التطبيق باستخدام لغة (أن-ايقون).

جمهورية العراق
وزار التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم

أمن في الصوتِ على الإنترنتِ بروتوكول (قي أو آي پي)

رسالة
جزء من متطلبات نيل درجة كمقدمة الى كلية العلوم في جامعة النهرين
ماجستير علوم في علوم الحاسوب

من قبل

كوثر عبدالاله عبدالرسول مشكور

(بكلوريوس علوم الحاسوب 2004)

المشرف

كد. بان نديم الكلا

نيسان 2008 *ربيع الثاني 1429*