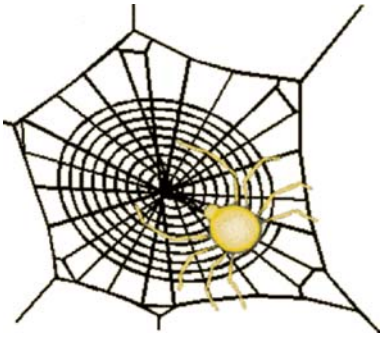
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

MSc THESIS

Session Initiation Protocol Benchmark Suite

Jiangbo Yin



CE-MS-2004-05

Abstract

In current-day networks, the bottleneck in achieving high-speed and high-bandwidth networks has shifted from the physical link limitations to the network processing performance within network nodes. The need for such networks is driven by the fact that increasingly more applications are introduced to send huge amounts of (multimedia) data over the networks, e.g., voice-over-IP (VoIP), videoconferencing, network gaming, etc. Consequently, the network processor – a programmable (general-purpose) processor incorporating specialized hardware – has been under investigation to provide both high performance and high flexibility (due to the varying support of different and emerging applications). At the same time, other design constraints must be met, such as cost, port density, and real-time processing. Therefore, an important aspect in the design of network processors is understanding its functionalities that are mostly defined in network protocols. By analyzing/profiling the protocols, we can determine the most time-consuming functionalities and decide on the most appropriate hardware/software mapping.

Protocol analysis is the part of the network processor design. The main purpose of the protocol analysis is measuring the functionalities of the protocol and determining the most complex and time-consuming functionality that should be implemented in hardware of the network processor to speed up the performance of the network processor. In this thesis, we investigate the Session Initiation Protocol (SIP) and benchmarks for SIP. SIP is an application level control protocol that is used to set up, modify, and tear down the multimedia session between the participants. We modify an exist SIP implementation (SIP Express Router) to make the benchmarks to measure the performance of the protocol. The benchmark suite consists of three benchmarks: message parser benchmark, action benchmark, and forward benchmark. They cover nearly all important functionalities of the SIP. A Message Generation Tool (MGT) was created to generate the SIP messages automatically. It is very useful to the simulation of the SIP benchmarks. The results of the benchmark suite present the profiling performance (in terms of cycles) and the architectural characteristics of the SIP. It helps the designer to determine that which functionalities should be involved in network processor design. Which is the main contribution of this thesis.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Session Initiation Protocol Benchmark Suite

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Jiangbo Yin
born in Wrumqi, China

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Session Initiation Protocol Benchmark Suite

by Jiangbo Yin

Abstract

In current-day networks, the bottleneck in achieving high-speed and high-bandwidth networks has shifted from the physical link limitations to the network processing performance within network nodes. The need for such networks is driven by the fact that increasingly more applications are introduced to send huge amounts of (multimedia) data over the networks, e.g., voice-over-IP (VoIP), videoconferencing, network gaming, etc. Consequently, the network processor – a programmable (general-purpose) processor incorporating specialized hardware – has been under investigation to provide both high performance and high flexibility (due to the varying support of different and emerging applications). At the same time, other design constraints must be met, such as cost, port density, and real-time processing. Therefore, an important aspect in the design of network processors is understanding its functionalities that are mostly defined in network protocols. By analyzing/profiling the protocols, we can determine the most time-consuming functionalities and decide on the most appropriate hardware/software mapping.

Protocol analysis is the part of the network processor design. The main purpose of the protocol analysis is measuring the functionalities of the protocol and determining the most complex and time-consuming functionality that should be implemented in hardware of the network processor to speed up the performance of the network processor. In this thesis, we investigate the Session Initiation Protocol (SIP) and benchmarks for SIP. SIP is an application level control protocol that is used to set up, modify, and tear down the multimedia session between the participants. We modify an exist SIP implementation (SIP Express Router) to make the benchmarks to measure the performance of the protocol. The benchmark suite consists of three benchmarks: message parser benchmark, action benchmark, and forward benchmark. They cover nearly all important functionalities of the SIP. A Message Generation Tool (MGT) was created to generate the SIP messages automatically. It is very useful to the simulation of the SIP benchmarks. The results of the benchmark suite present the profiling performance (in terms of cycles) and the architectural characteristics of the SIP. It helps the designer to determine that which functionalities should be involved in network processor design. Which is the main contribution of this thesis.

Laboratory : Computer Engineering
Codenumber : CE-MS-2004-05

Committee Members :

Advisor:	Stephan Wong, CE, TU Delft
Mentor:	Yunfei Wu, CE, TU Delft
Member:	Ben Juurlink, CE, TU Delft
Member:	Sorin Cotofana, CE, Delft

To my families, friends, and lover

Contents

List of Figures

List of Tables

Acknowledgements

It is a wonderful experience to study in Delft University of Technology (TU Delft) for two years. And I am so lucky to do my Master's thesis following the Prof. Stephan Wong in the Computer Engineering (CE) Laboratory. During this period of time, I met many intelligent and enthusiastic people, I would like express my respect and thanks to all of them. Without their helps, I can't finish my thesis at all.

First of all, I would like express my greatest appreciation to my supervisor Prof. Stephan Wong. He is really an earnest and professional people. His stimulation and instruction are the great helps to me to finish my thesis. Every time I fell in the morass, he always helps me to analyze the problems and find the correct direction. And he spend much time to help me, especially to revise my thesis, which is a hard work indeed. His manner of the work and his attitude to the work are impressed in my mind deeply.

I have to appreciate the help from Yunfei Wu also. In the last nine months, she paid attention to my thesis all the time. During my working, she gave me many good suggestions and supports and expressed the great circumspection and patience to me.

I also want to say thanks to all of my friends both in China and other countries. They are indispensable to my life. Especially to all the members of Delft China Football Team (DCF), playing game with them is the part of the most happy time in these two years.

At last, the special thanks are gave to my families and my girl friend Jie Zhang. Their endless love and support are the motivation of my progress. They comprehend my career and suffer long time waiting for me. I feel that they are the most lovely and trustful people in the world.

Jiangbo Yin
Delft, The Netherlands
August 6, 2004

1 Introduction

In the last decade, the network techniques were developed very fast with the boom of Internet. At the beginning of this phase, the bottleneck of network is the bandwidth that means the capacity of the physical links in network. But with the advances of optical technology the capacity of physical links grow greatly. The bottleneck moves to processing elements in network devices. Especially for some applications such as voice over IP (VoIP), Videoconferencing, network games etc., speed up the performance of network becomes a critical problem. To fix this problem, designing a network processor is an attractive way.

In this chapter, first we introduce the basic knowledge of network processor in Section 1.1. Then why we need make the benchmark for protocols, which is Section 1.2. Section 1.3 presents the problems and goals. Section 1.4 presents an overview of this thesis.

1.1 Network Processor

Before we start this topic, we should make it clear that what is network processor? And what is the difference between network processor and general-purpose processor? *Network processor (NP)* is a high-performance, programmable device designed to efficiently execute communication workloads[2]. In the last 25 years, with the rapid developing in VLSI, it is not difficult to make cost-efficient high-performance embedded processors for communication functions. In fact, a number of factors have contributed to the development of the NP and the NP industry.

Figure 1.1 is an example of an NP in a typical router line card application. In this case, the NP has to examine packets at line speed and perform a set of operations including quality-of-service (QoS) processing.

The network processor design involves many aspects, such as external memory bandwidth, power dissipation, pin limitations, packaging, and verification. But in the principle of design, a designer should handle three key elements which are:

- **Real-time processing constraints** The real-time processing demands the NP to use advanced and novel computer architectures to achieve high performance. For example, videoconferencing has the minimum requirements of network performance.

- **Flexibility** The NP should have the flexibility to deal with the changing of communication protocols and increasing demands for new complex network services.

- **Cost-efficiency** Considering the physical constraints of VLSI technologies, packaging, and power, finding a cost-efficient way to implement NP.

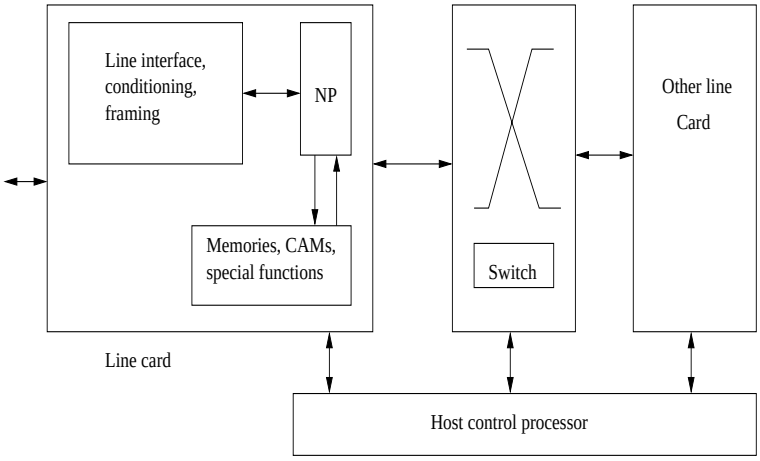


Figure 1.1: Network processor in a typical router application

Additionally, compared to a general-purpose processor, the NP has more challenges because of the real-time processing, port density, and power efficient. The high levels of device integration (on-chip interfaces and controllers for external memories, switch fabrics, co-processors, network interfaces, etc.), critical shared resources and software design for high performance system are challenges also.

1.2 Benchmark for Protocols

At the beginning of this thesis, we should ask a question: Why do we need an SIP benchmark? Compared to a general-purpose processor, the network processor (NP) support some form of a distributed, parallel programming model and optimized for fast packet processing and I/O. These features demand the designer to take the performance requirements of protocols into account. When we design a NP working in the SIP network, we have to know the related performance of SIP. The benchmark for SIP can provide us such information. That is the reason to why we should make benchmark for protocols.

A *benchmark* is a standard by which others can be measured and judged. In the computer realm, any program that is used to measure performance can be called a benchmark[3]. The principle of benchmarking for a protocol is using this protocol in a simulation environment and getting the performance of each functions of protocol and the characteristics of the protocol.

There are many network protocols involved in an NP, such as TCP, UDP, SDP,TLS, SIP etc. In this thesis, we only focus on SIP (Session Initiation Protocol) that is an application level signaling protocol. This signaling protocol is so simple and efficient that suitable to be used in VoIP, videoconferencing and network mobility. We will introduce SIP in detail in Chapter 2. By now, the SIP is not popular and is hard to find exist benchmarks for it.

1.3 Problems and Methods

When we plan to create the benchmarks for protocols, there are several problems have to be solved. We list the problems and the methods to deal with them.

1. *What is the procedure to create the benchmark for a protocol?*
   Before we create the benchmark for a protocol, we have to have the implementation of the protocol. In this project, an existing implementation of SIP - SIP Express Router (SER) is used. We modify the source code and create our benchmarks on it. Based on the analysis of the implementation of the protocol, we determine which functionalities of the protocol should be measured. To profile each functionality, we have to define what kind of the input data for it and what is the expected results. The benchmark runs in a simulation environment and the results reflect the performance and architectural characteristics of the protocol.

2. *How to determine the most important functionality in SIP?*
   First, from the study of the protocol description files, we can learn the different operations of each functionality. The functionality that is necessary in any case and contains more complex operations than others should be the important one. Second, after reading the source codes of implementation of SIP, the functions that contain complex computations or many comparisons should be focused on.

3. *Which characteristics of SIP are related to the NP design?*
   Our benchmarks measure the architectural characteristics of SIP that is needed when designing an NP for SIP. We should get four characteristics about architecture design, they are:

   - **Instruction level parallelism:** presents the efficiency of the program. We get the number of Instruction Per Cycle branch (IPC) to indicate it.
   - **Branch prediction accuracy:** is an important parameter to NP design also. We investigate branch address-prediction rate (APR) and branch direction-prediction rate (DPR) for it.
   - **Instruction distribution** indicates the number of instruction and the instruction load/store rate to each functionality. We get load/store rate from benchmarks.
   - **Cache behavior** relates to the cache organization. We know that both the instruction cache and the data cache can effect the performance of the NP greatly, so we investigate different sizes of instruction and data caches. Additionally, the number of set and the block size in the cache are also investigated.

Based on the problems and solutions, we can see the goal of this thesis is investigation of the characteristics of SIP to get the information for NP design. Which includes the profiling performance and architectural characteristics of SIP.

1.4 Structure of the Thesis

In this section, we present an overview of the whole thesis.

In Chapter 1, we introduce the thesis from three aspects: the challenges of the network processor design, the requirement of the benchmark, and the SIP overview.

In Chapter 2, we discuss the background of the benchmark for SIP. First, we described the main aspects of SIP. Then we describe the definition of the SIP network elements and explain the function of each element. The following part discusses the SIP message and transaction models. The SIP message is about SIP message structure and types. The transaction models contain many operations based on different types of SIP messages. We integrate all aspects in the models to make the functionalities of SIP clearly.

In Chapter 3, we discuss the benchmark for SIP in detail. First, we introduce the implementation of SIP, which is an exist application that is named SIP Express Router (SER)[9]. We divide the application into several parts based on the functionalities of SIP. Then we introduce the benchmark with the definition, methodology. Our work includes three benchmarks: message parser benchmark, action benchmark and forward benchmark. Each benchmark consists of input data, simulation processing and expected results. Additional, a message generation tool is created for convince to generate input data to benchmarks automatically. We introduce the simulation tool sets and the simulation environment in the last part of this chapter.

Chapter 4 discusses the results of our benchmarks. First is the overview of the results . We compare the performance of the three benchmarks in terms of the clock cycles to determine which is the most time consuming one. Then, we do the extreme testing for these three benchmarks to get raw performance[7]. The sub functions of each benchmark are tested also. We investigated architectural characteristics in several aspects. The results can help us to find the cost-efficient way when design a network processor to implement SIP.

Chapter 5 presents the conclusions of this thesis. It describes the conclusions of each chapter and the contribution of this thesis. We also present several topics that should be investigated in the future.

2 SIP Introduction

SIP stands for Session Initiation Protocol. It is an application-layer control protocol which has been developed and designed within the IETF[8]. The protocol has been designed with easy implementation, good scalability, and flexibility in mind.

The specifications are described in RFC3261[15]. The protocol is used for creating, modifying, and terminating sessions between participants. Here, the session is considered to be a set of senders and receivers that communicate and the states kept in those senders and receivers during the communication. Examples of a session are Internet telephone calls, distribution of multimedia, multimedia conferences, distributed computer games, etc.

In this chapter, we describe the structure and characteristics of SIP. Section 2.1 discusses the basic features of SIP. Section 2.2 presents the definitions of the SIP net work elements. These definitions are used in the remainder of this thesis; Section 2.3 discusses the SIP message, the categories and functions; Section 2.4 presents the SIP transaction models, which show how the SIP work between different elements. Section 2.5 presents the conclusions

2.1 SIP in Network

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences), e.g., Internet telephony calls. With SIP, one can also invite others to participate in an already existing session, e.g., multi-cast conferences. Media can be added to (and removed from) an existing session[15]. As specified in RFC3261, SIP provides four basic functions[6]:

- **User location:** This function is responsible for translating a user name to user's current network address. For example, translating an e-mail-like address to an IP address. This function includes user mobility features, that means keeping track of a user's address during the user's movement to different locations in the SIP network. For example, when a user move into the other subnets, his SIP URI (IP address) should be changed according to the subnet he is in. This function is finding the latest address of a user.

- **Feature negotiation:** The responsibility of this function is to ensure that all participants in a session agree on the features to be supported among them before the session is set up. Since not all of them have the same capabilities, they exchange information with each other in order to reach an agreement for all.

- **Call management:** It takes responsibility for management of participants of a session. It includes adding, dropping, transferring, and placing participants in a

session on hold. For example, allows a new user to receive the video media stream during a videoconferencing.

- **Feature modification:** This function is in charge of changing the features of a session during the session. For example, adding a video channel to a session that is holding only a voice channel.

2.1.1 Position in Network Protocol

SIP can operate over UDP or TCP. When sending over TCP or UDP, multiple SIP transactions can be carried in a single TCP connection or UDP datagrams. Note that UDP datagrams, including all headers should not be larger than the path maximum transmission unit (MTU)[14]. From RFC2543, the MTU normally is 1500 bytes but it is not the only option. Figure 2.1 depicts the position of SIP in the Internet layers and the communication between two SIP users. SIP is an application layer protocol that is located at the top of the OSI (Open System Interconnection) layers.

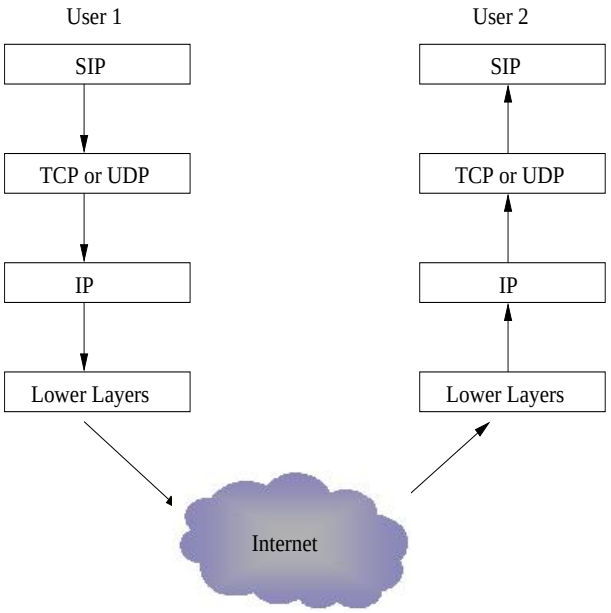


Figure 2.1: The SIP position in network

2.2 SIP Network Elements

In this section, we present the definition of the SIP network elements. Basic SIP elements are user agents, proxies, registrars, and redirect servers. Though the simplest communication can be set up by only two user agents, a typical SIP network always contains several SIP elements.

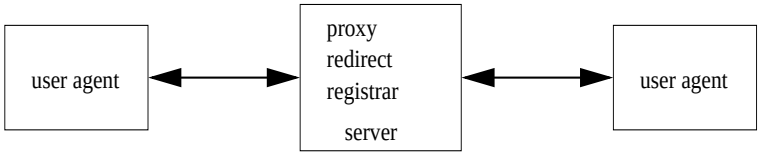


Figure 2.2: The elements in SIP network

We should note that the SIP elements that we mentioned above are often logical entities[1]. That means different elements can be configured together, for example, a computer can act as both a proxy server and a registrar. In a proper configuration, the integration of SIP elements can save the implementation cost and increase the speed of processing between the network elements. In previous example, the proxy server doesn't need to communicate with other machines when it needs the information of the registrar.

2.2.1 User Agent

Internet end point that use SIP to find each other and to negotiate a session characteristics are called *user agent*[10]. Usually, it is an application that is installed in a computer. Alternatively, it can be an individual device that implements SIP functions, such as cellular phones, PSTN gateways, PDAs etc.

User agent can be classified into *User Agent Client* (UAC) and *User Agent Server* (UAS) in logic functions. UAC is the function that sends requests and receives responses. UAS is the function that receives requests and sends responses. Normally, a user agent has both of these two functions. For example, in Figure 2.3, when a user agent receives request and sends response, we call it UAS; when a user agent sends request and receives response, we call it UAC.

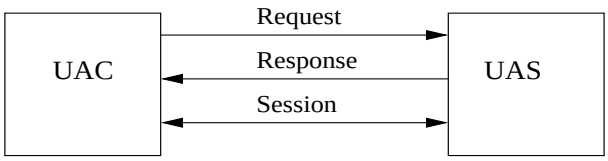


Figure 2.3: The simplest communication model between two UAs

2.2.2 Proxy

The *proxy* is an intermediary entity that acts as both a server and a client for the purpose of translating requests to other clients[14]. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity that is "closer" to the destination. As dispected in Figure 2.4, the UAC generates an INVITE request and sends to its SIP domain server-proxy A. Because the UAS is not in the same

[1] An entity that is defined by logic function not by physical characteristics.

domain, so the proxy A forward the INVITE request to the server of SIP domain B. The proxy B finds the UAS and forward INVITE request to it. The response of UAS should go through the same path back to UAC. The proxy is also useful for enforcing policy (for example, making sure a user is allowed to make a call) and performing authentications . In addition, a proxy interprets and, if necessary, rewrites specific parts of a request message before forwarding it.

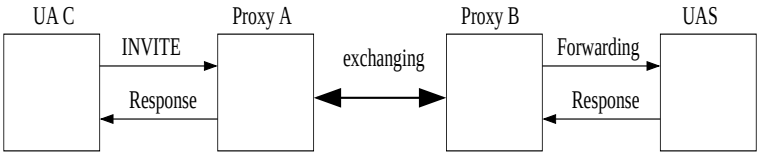


Figure 2.4: The example communication with proxies

2.2.3 Registrar

The *registrar* is a server that accepts requests for registration and places the information it receives from those requests into the location service for the domain it handles[14]. In Figure 2.5, the registrar receives a register request message from the user agent and stores the information in the message into the location database. If the operation is successful, it reply to user agent with a 200 OK response. A user agent has to make a successful register in its SIP domain, otherwise, it can't receive any message from the SIP server. The information in location service is used by proxy and redirect server. When a proxy or redirect server receives an INVITE request, it has to consult with location service to get the information for the request, e.g., addresses required by the request. A registrar is typically integrated with a proxy or redirect server. Sometimes registrar server can be a simple table that contains the records of all registered users.

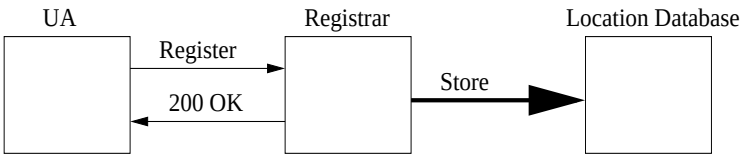


Figure 2.5: The example communication with registrar

2.2.4 Redirect Server

The *redirect server* is a server that accepts SIP requests from clients, maps the new addresses and sends required addresses back to the clients[14]. The difference between redirect server and proxy server is that the redirect server generates 3xx responses (the classification of response messages will be discussed in Section 2.3.2) to requests but does not initiate its own SIP request. The redirect server never forwards the request to UAS

or other SIP servers. In Figure 2.6, the UAC sends INVITE request message, which is same as the message sent to proxy server, to redirect server. The redirect server consult with location service and get the address of UAS and sends it back to UAC. Then the UAC can communicate with UAS through the address returned from redirect server.

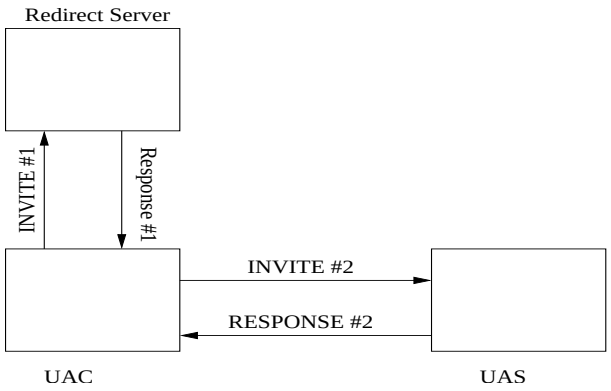


Figure 2.6: The example communication with redirect server

2.3 SIP Messages

In SIP, messages are utilized as means of communication between SIP network elements. The SIP messages are classified into two types: REQUEST and RESPONSE. Both types use a basic format to construct the message. It consists of a start-line, a message header that contains one or more header fields, an empty line indicating the end of the header, and an optional message-body[12]. The structure of message can be shown as:

generic-message = start-line
[message-header]
CRLF
[message-body]

The following is an example of a real SIP message:

```
INVITE SIP:office@tudelft.nl SIP/2.0
From: sip:home@container.com
To: sip:office@tudelft.nl
Via: SIP/2.0/UDP 135.180.130.133
Contact: ¡client¿friend3@tudelft.nl
Call-ID: 0077@10.0.0.1
CSeq: 1 INVITE
Expires: Fri, 01 Jan 2010 16:00:00 EST
Accept-Language:en
Content-Type: application/sdp
Content-Length: 174

v=0
o=mhandley 29739 7272939 IN IP4 126.5.4.3
s=SIP Call
t=3149328700 0
c=IN IP4 135.180.130.88
m=audio 49210 RTP/AVP 0 12
m=video 3227 RTP/AVP 31
a=rtpmap:31 LPC/8000
```

2.3.1 SIP Request

An SIP *request* is the SIP message sent from a client to a server for the purpose of invoking a particular operation. We present the structure of the request message below, it consists of three parts:

Request = Request-line
[message-header]
CRLF
[message-body]

- **Request-line** The Request-line begins with a method token, such as INVITE, and following the SIP version number. Then, the different types of headers contains the information of sender and the properties of the session requested[13]. For example, an INVITE request-line can be:

  INVITE SIP:office@tudelft.nl SIP/2.0

  The most important elements in request-line is method. The *method* states the primary function of a request that decides what type of the message is and what operations should be executed. SIP uses six methods[3]:

  1. **INVITE**: This method indicates that a user or server is being invited to participate in a session. The message contains a description of the session, using Session Description Protocol (SDP), and the type of media that is to

be used for the session, caller[2] and callee[3] addresses, user location, caller preferences, and desired features for the response. An INVITE request may be sent during a session on holding to modify the session characteristics.

2. **ACK**: This method is used with INVITE method. The purpose of ACK is to confirm that the client has received a final response to an INVITE request. The ACK can contain the message body which confirm the final session description to be used by the callee. But it is not necessary. If there is no message body with ACK message, which means the callee accepts the session description in INVITE request.
3. **OPTIONS**: This method Queries a server about the capabilities of the called party. A called user agent may also send an OPTIONS message reflecting how it would respond to an INVITE if it is in urgency. A server may respond to this request if believes it can contact the user.
4. **BYE**: The BYE can be generated by all communicating parties to indicate other one to end the call. A BYE is forwarded after a party has released the call. The other party that receives BYE has to release transmitting streams.
5. **CANCEL**: The CANCEL is used to end a pending INVITE request but does not affect completed requests. For example, a proxy server has received a response to one of its parallel searches, the other response should be canceled. In practice, only the INVITE is canceled.
6. **REGISTER**: This method is used by a client to register its address to an SIP server. The client may be in any locations in the same domain under the server. Usually, a user agent may register with a local server on startup by sending an IP multi-cast to "all SIP servers" (sip.mcast.net,224.0.1.75).

- **Header fields** An *SIP header* is the description of the references in a SIP message, for example, the host address, the destination address, call sequence number etc. Each of the headers in the SIP message contains a number of fields. The contents of these fields are decided by UAC to inform the UAS that what is the proposition for the required session. These header fields form the basic architecture of SIP message. There are many header fields defined in RFC3261, but not all the fields have to present in a header. Only several header fields are obligatory to be included in all the headers, they are Via, To, From, CSeq and Call-ID. We give the description of these header fields and other several useful header fields.
  - To: This field identifies the recipient of the request message. It can be the name-address (URL) or number-address (numeric IP address).
  - From: This field indicates the initiator of the SIP request message. It is copied from the request to the response by server. It is name-address or number-address.
  - Via: This field indicates the path that the request has traversed so far, and used to ensure that the response message takes the same inverse path as the

[2]The participant who initializes a call
[3]The participant who receives a call from the caller.

request message. The client makes the request with a Via field containing its host name or network address and the port number at which it wishes to receive responses. Each subsequent proxy server that forwards the request adds its own additional Via field before any existing Via fields.

- CSeq: The name of the field comes from the Command Sequence. It contains the request method (for example, INVITE), and a sequence number.
- Call-ID: It identifies a particular SIP invitation or all registrations for a specific client uniquely. A multimedia conference results in several calls with different Call-IDs. The REGISTER and OPTIONS methods use this parameter to match requests and responses.
- Contact: This parameter provides a URI (Universal Resource Identifier) that the user can be used for further communications. For example, when the INVITE request is forwarded, the request message is sent to both *To* address and *Contact* address. Both parties can response the request and set up the communication.
- Content-Length: This field indicates the length of the message body, in decimal number of octets.
- Proxy-Authenticate: This field is used to support a proxy authentication operation.Its value is verified by authentication scheme, and the parameters that are applicable to the proxy for the operation.

- **Message Body** A *message body* is the data that describes the properties of the session. Before we set up a session, the participants must agree on the media they will use to communicate with each other. This media is described in the message body using SDP (Session Description Protocol), which is one of the most important supporting protocols to IP Call processing.

  SDP defines a session as a set of media streams. Due to the properties of the media streams, a SDP description should contain the following information about a session:

  - The name of the session and its purpose.
  - The time during which the session will be active.
  - The information needed to build a session, such as media type, transport protocol, media format etc.

  Here is an example of a message body:

```
v=0
o=mhandley 29739 7272939 IN IP4 126.5.4.3
s=SIP Call
t=3149328700 0
c=IN IP4 135.180.130.88
m=audio 49210 RTP/AVP 0 12
m=video 3227 RTP/AVP 31
a=rtpmap:31 LPC/8000
```

A message body contains several optional fields. The normally used optional fields are explained in the following:
v=Protocol version
o=owner/creator and session identifier
s=session name
c=connection information
u=URI of description
e=e-mail address
t=time the session is active
r=repeat times
m=media and transport address

2.3.2 SIP Response

The *SIP response* is the SIP message to indicate the state to a request. The difference between a request and a response in the message structure is only the start line. So, the response message can be shown as:

Response = Response-line
(general-header — response-header — entity-header)*
CRLF
[message-body]

The response line consists of SIP-Version, Status-Code and Reason-Phrase. Where the SIP-Version is the version of the protocol being used by the message, for example, SIP/2.0. The Status-Code is a 3 digit value. The Reason-Phrase is a short text string that explains Status-Code[13].

The example of the an SIP response message as following:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 135.180.130.133
From: sip:home@container.com
To: sip:office@tudelft.nl
Call-ID: 0077@10.0.0.1
CSeq: 1 INVITE
Content-Length: 0
```

SIP has six classes of response status code, using the first digit to indicate the different classes.

- **1xx:** Information responses. It indicates that the request has been received and is being processed. Note at this time, it does not know if the request is forwarded successful.

  Typically, the proxy server send a response message with the status code 100 (Trying) when it starts processing an INVITE and user agents send response message

with the status code 180 (Ringing) that means the callee has received the request. This is similar to the ringing of a traditional telephone.

- **2xx:** Success response. The request has been successfully processed and accepted. For example, when the callee pick up the phone, the response with code 200 (OK) will be sent.

  A UAC may receive several 200 messages to a single INVITE request because of forking proxy. The forking proxy can fork the request so it will reach several UAS and each of them will accept the invitation and reply 200 response. In this case, each response is distinguished by the tag parameter in *To* header field.

- **3xx:** Redirection responses. It means the server has to do more actions to complete the request process. A redirection response presents information about the user's new location or an alternative service that the caller might use to reach the callee. Usually, it it sent by proxy server. For example, when a proxy server receives a request but can't process it for any reason, it will send a redirection response to the caller and put another location into the response. The caller can use this new location information to find the callee.

- **4xx:** Client error responses. The request had an error or could not be processed by the server.

- **5xx:** Server error responses. The request is valid but the server failed in processing.

- **6xx:** Global failure responses. The request could not be processed by any server.

The message body of a response message is same as the message body that we have introduced in request message. Normally, the response message doesn't has the message body.

2.4 SIP Transaction Models

A *transaction* is a sequence of SIP messages exchanged between the SIP network elements. It consists of one request and all responses to that request. The different models can be used for different purposes. We show a typical transaction model to explain the basic operations in an SIP communication.

From the Figure 2.7, we can see that there are six steps to finish a session:

1. The proxy server receives an INVITE request from a UAC and responses a 100 Trying message, which means that the INVITE request is valid and is forwarded to the UAS.

2. When the UAS receives the INVITE request, it replies a 180 Ringing response to the server. This message will be forwarded to the UAC in the same inverse path.

3. When the UAS decides to accept the request, it sends a 200 OK response. This message is also forwarded by the proxy server to the UAC.

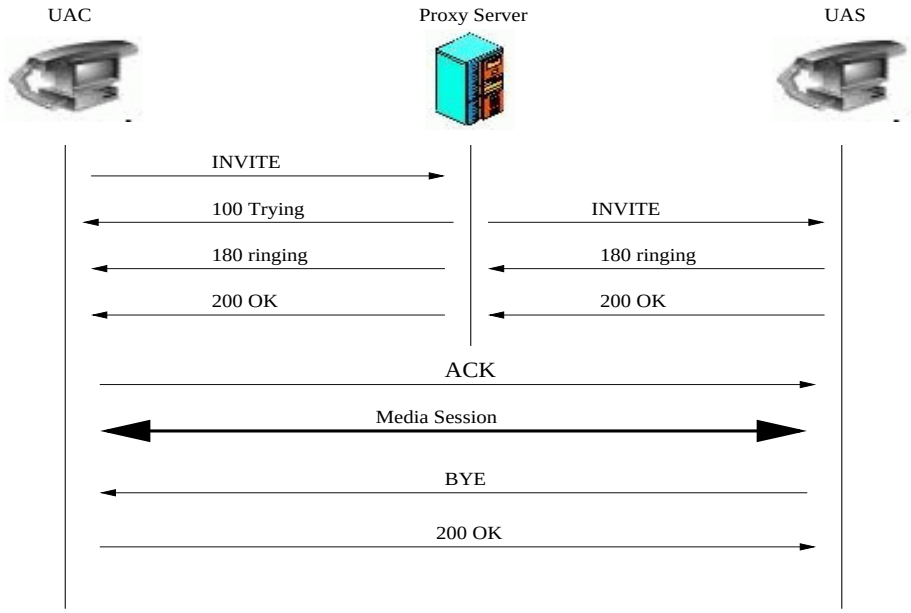


Figure 2.7: A typical SIP transaction model

4. The UAC sends an ACK request message to the UAS directly, because at this time, the UAC obtained the location of the UAS from the information in previous responses.

5. The two parties start the session following the agreement in the INVITE message body.

6. When the UAS closed the session, it send a BYE message to the UAC. Note that both parties can send a BYE message to inform the other party that it wants to close the session.

2.4.1 Transaction with Proxy Server

The main operations of the proxy server are depicted in Figure 2.4. Here we make a simple explanation and describe another function of the proxy server, which is forking.

First, the UAC sends an INVITE request to its SIP service provider, which is the proxy server A. The server reads the message and gets the URI from the To header field. Because the callee is not in the same domain, the server looks up the callee's domain address from its database and forwards the request to the domain B. With this operation, server A has to put its address in the Via field as second Via header field.

Second, proxy server B receives an INVITE request from proxy server A and determines that the UAS is in its domain. It looks up the location service for the UAS by name. In this case, the INVITE request is forwarded to UAS. The proxy server B has to add its address in the Via header field too.

Third, the UAS generates the response to the request. Such response is forwarded to the UAC following the same way. Note that when the proxy server is forwarding the response to the UAC, it must remove its address from the Via header field, then forwarding it.

After these three steps, the proxy server quits from the communication between the UAC and the UAS. The response contains the location information of the UAS, so the UAC can find UAS directly.

The other function of the proxy server is forking a request to several parties. In this case, we can call the proxy server a fork server. In the following, we present an example of forking message by a proxy server.

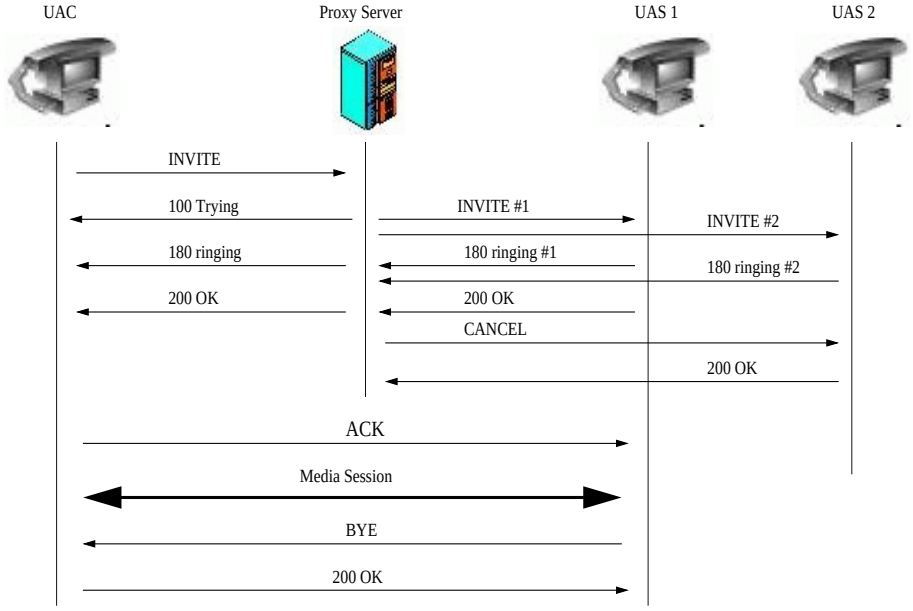


Figure 2.8: Forking SIP request through proxy server

The basic operations of the proxy server the same as described above. Only two operations are different. The first one is when the proxy server forwards the request to the UAS, it forwards a copy of the INVITE request to all destinations. The second one is that if one of the UAS accepts the request and replies 200 OK already, it will send a CANCEL message to all other UASs to cancel the pending requests.

2.4.2 Transaction with Redirect Server

The redirect server does not forward the request to other parties. It only receives the request and generates a response. As the shown in Figure 2.6, when the redirect server receives an INVITE request, it consults a location server to determine where to redirect the invitation to. The response should be the address of the next hop to find the destination. The other operations in this model are the same as the operations in proxy server.

Considering the properties of redirect server, it is suitable to be used in SIP mobility[5]. In Figure 2.9 , we depicted a simple example for SIP mobility.

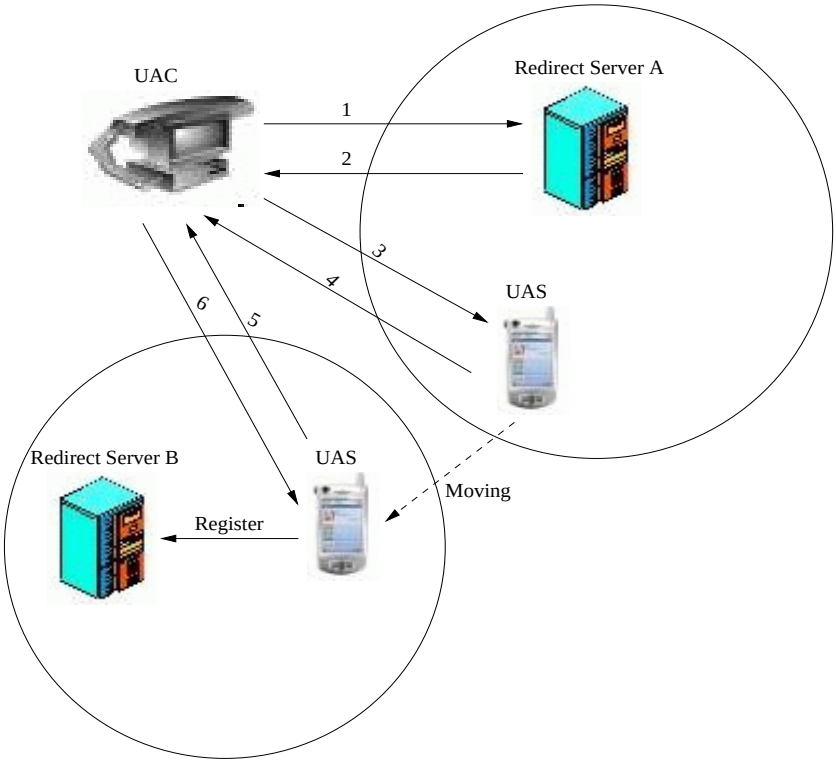


Figure 2.9: The model of using redirect server in SIP Mobility

In Figure 2.9, the UAC sends an INVITE request to the redirect server and gets the current location of the UAS. During the communication, when the UAS moves to the subnet B, it should send a re-invite message to the UAC to inform it to change location where the media sent to. In this case, the UAS should register in the redirect server in the domain B, and the redirect server B informs the redirect server A that the UAS in its domain now. After that, if there is a request for the UAS, the redirect server A will return the address of the redirect server B.

2.4.3 Transaction of Register

From the previous part, we know that no matter what kind of the server is used, it has to use location service to determine the location of the callee. In fact, the location service can simply be a table handled by an SIP server. The function of the location service is to return the address of the callee. The question is how to implement location service?

In SIP, the solution to this problem is using the REGISTER request to construct the database of location service. Following is an example of a REGISTER request:

```
REGISTER sip:tudelft.nl SIP/2.0
From: sip:home@container.com
To: sip:office@tudelft.nl
Via: SIP/2.0/UDP 135.180.130.133
Call-ID:0077@10.0.0.1
CSeq: 1 REGISTER
Contact: ¡client¿friend3@tudelft.nl
Content-Length: 0
```

In this example, the REGISTER request registers a user *home* at the *tudelft.nl* server. After the server processed this request successful, any invitation it receives for *sip: home@tudelft.nl* will be redirected or forwarded to the address in the Contact header field: *friend3@tudelft.nl*.

The server returns a 200 OK response that lists all the current registration records for the user *home*. Because there is no Expires header field, this record will last until another registration request to override it.

2.5 Conclusion

In this chapter, we described the main aspects of Session Initiation Protocol. First, we learned the basic features of SIP and what it can do. Then we defined the basic elements in SIP network and explained the function of each element. The following part discussed the SIP message and transaction models. Learning the detail of the SIP message could help us to understand different operations in transaction models. The transaction models are the implementations of SIP, they contain many operations with different SIP messages. We integrate all aspects in the models to understand SIP clearly.

3 SIP Benchmark

SIP is regarded as the next generation signaling protocol, it is very useful in Voice over IP (VoIP), Videoconferencing and mobility network. To the network processor design, we have to know the performance and characteristics of the protocols that are used in the network. The SIP benchmark can provide us the performance and architectural characteristics of SIP. It helps us to determine the complex and time-consuming functions of the SIP protocol that should be implemented in hardware in order to speed up the performance of the network processor.

In the previous chapter, we discussed in detail the Session Initiation Protocol (SIP). In this chapter, we present the SIP benchmark suite that cover several main aspects of SIP. This chapter is organized as follows. Section 3.1 presents the description of SIP implementation, which is an exist SIP application. Section 3.2 discusses the implementation of benchmark in detail. It goes from the basic methodology to each benchmark implementation for each function. Section 3.3 introduces the simulation environment, we use SimpleScalar Tools for simulation. Section 3.4 presents the conclusions.

3.1 SIP Implementation

Based on RFC3261, there are several SIP implementations available now, such as siproxy, OSIP, SIP Express Router (SER) etc.. In our benchmark, we use SER as the original application that is an open source software developed by iptel company[11]. We modify the source code to implement our benchmarks. The source code can be download from the web site of iptel company.

3.1.1 Architecture of the Software

SIP Express Router is a full functional SIP server application. It implements almost all the functions that are described in RFC3261. The program has two parts. One part is the main program that includes program startup, main function, parameters definition and module interface. The other part is the modules that implement more functions as modules. The module can be loaded into the server to execute some special operations or removed from the server if a module is not used any more.

Here we focus on the first part, this part is more important to a server and has more complicated functions than the second part. We give the program flow chart first:

1. **Startup:** Startup is the initialization of the program[11]. It includes:

    - Processing Command Line Parameters: Such function was done by *getopt* to get the parameters in.

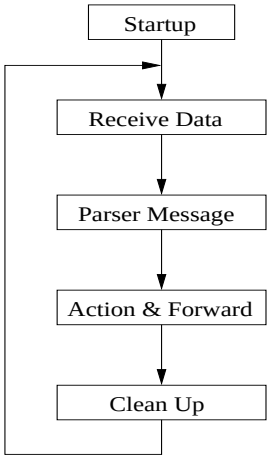


Figure 3.1: The architecture of the SIP implementation

- Parser Initialization: It contains the function *init_hfname_parser* that initializes hash table in header field name parser and the function *init_digest_parser* that initializes hash table in digest authentication parser.
- Malloc Initialization: It uses a modified *malloc* function to make the server faster. The initialization creates internal data structures and allocates memory region to be partitioned.
- Timer Initialization: The server need a timer for various subsystems of the server which must be called periodically regardless of the incoming request.
- FIFO Initialization: SER has built-in support of FIFO control. It means that the running server can accept commands over a FIFO pipe.
- Built-in Module Initialization: The modules can be either loaded at runtime or compiled in statically. In this step, the required modules are compiled statically.
- Server Configuration: In this step, the server is configured through a configuration file. The configuration file is C-shell like script which defines how incoming requests should be processed.
- Interface Initialization: The server will try to obtain list of all configured interfaces of the host which it is running on.

2. **Receive Data:** In the server, a request or response is represented as sip_msg structure. In this step, the structure is allocated. The received message was copied from original field to a backup field. In the next steps, the operations are only made on original received message.

3. **Parse Message:** Parsing message is very important and one of the most time-consuming operations of a SIP server. A header field parser can be either

in server core or in a module. Normally, only most often used field parsers are configured in the server core. We will discuss some functions of parsing message later.

4. **Action & Forward:** After the message parser, a received message is parsed and a sip_msg structure is filled. The server has to do some operations to meet the requirement of the message, for example, if it is a response, the server should forward message to UAC or other proxy. We will give some detail of this part later.

5. **Clean Up:** When the operations to a message are finished, the server should destroy the structures and release the memory.

3.1.2 Underlying Functions

In this part, we focus on several important functions of SIP server. We chose the functions in a simple way that is the most time-consuming and complex functions. We discuss the function *parse_msg* first, subsequently discuss *run_action* and *forward_reply*. After message parser, if the message is a request then basic sanity checks will be performed (make sure there is a first Via and parsing was successful) and the message will be passed to routing engine (run_action). If the message is a response, it will be forwarded to destination. This operation is more simple than routing engine.

- **Parse_msg:** We have known that a SIP message consists of message header and optional message body. The message header consists of the first line and a number of header fields. The first line determines the type of the message and the header fields provide additional information that is needed by client or server to process the message successfully. The structure of this function can be shown as Figure 3.2:

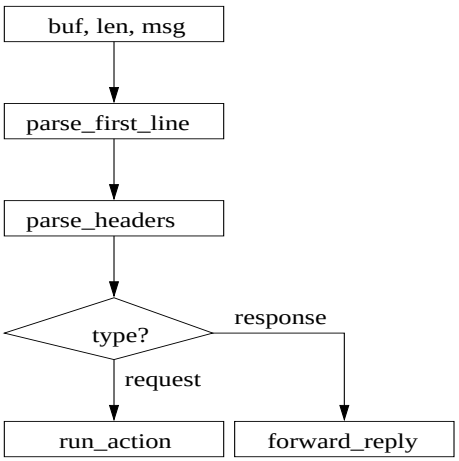


Figure 3.2: The flow chart of *parse_msg*

  - *Parse_first_line* function only operates on the first line of SIP message to find what type it is. It uses matching key word to find the types. For example, the first character of the first line is I(NVITE) or R(EGISTER) can dedicates the type of the message. If the type is response, the program should get the status code that is a three digital number and save it. This function will fill in msg_start structure.
  - *Parse_headers* parses all header fields of SIP message. Only the fields that presented in a message header will be parsed. In SER, not all header fields have been implemented. Only following header fields can be recognized:
    Via, To, From, CSeq, Call-ID, Contact, Max-Forwards, Route, Record-Route, Content-Type, Content-Length, Authorization, Expires, Proxy-Authorization, WWW-Authorization, Supported, Require, Proxy-Require, Unsupported, Allow, Event.

- **Run_action:** This function is one of most complicated part in the server. After the server received a SIP message, it was converted into sip_msg structure. If the message is a request, it will be passed to *run_action*. The function accept two parameters, one is the list of actions to be processed ,an other is sip_msg structure that to be processed.
  There is a big *switch* statement in this function. Each case of the statement is one command that handled by the server core itself. They are:
  drop, forward, send. log, append_branch, len_gt, setflag, resetflag, isflagset, error, route, exec, revert_uri, set_host, set_hostport, set_user, set_userpass, set_port, set_uri, prefix, strip, if, module.
  Each command will do operation on sim_msg structure, we don't discuss detail of each item here.

- **Forward_reply:** After message parser, if the message is a response, the server simply forward it to destination. It includes several operations, such as basic check, get socket, translate sip_msg structure to buffer. In SER, the message was forwarded stateless, so the server has to put its socket information in a Via header field.

3.2 Benchmark Implementation

As mentioned before, SIP is a next generation application layer signaling protocol. It is important to know how good it is and which part of SIP should be focused on during the implementation. For this purpose, the benchmark for SIP seems necessary to measure the performance of different parts of SIP. In our project, we make benchmark based on SIP Express Router (SER) that we have described in last section. The benchmark includes several parts: The first one is making input data for benchmark. Because the benchmark has to run in a simulation environment, so we need make input data to simulate the situation in real world. The input data should follow some rules to make it like the data in real world mostly. Which makes the result of benchmark reasonable. The second part is rewrite the functions that need to be measured. We have to design

a strategy to get input and give the output without affecting the original function very much. The last part of a benchmark is result analysis. Before the benchmark is run, we should get the expected result based on the functionalities we learned. Then we compare the expected result with real result to find if the benchmark is reasonable. From the simulation tool that we will introduce later, we can get more results to analysis the different aspects of the functions.
In this section, we only discuss the first two parts. The result analysis will be put into the next chapter.

3.2.1 Benchmark Introduction

Before we create benchmark for SIP, we should introduce the general knowledge of benchmark. A *benchmark* is a standard by which others can be measured and judged[7]. In the computer realm, any program that is used to measure performance can be called a benchmark. It usually done as a basis for comparison. For example, how fast the system A is compared with system B? Typically, several aspects of computers can be studied with a benchmark[3]:

- Raw performance of a complete system.
- Raw performance of a specific subsystem (microprocessor, memory, disk interface etc.).
- Performance of a computer running a typical suite of applications.
- Performance of a computer running a specialized application.
- Performance of a server on a network.

The benchmark can be classified in two categories: Artificial and Living. Artificial benchmarks measure raw performance. They use fixed size of data types as input data to individual parts of a system. The results reflect the difference between these individual parts on that fixed input data. It can be used to compare the different parts of a system. Living benchmarks apply performance analysis to real world task. They can be changed to meet the requirement of customers.

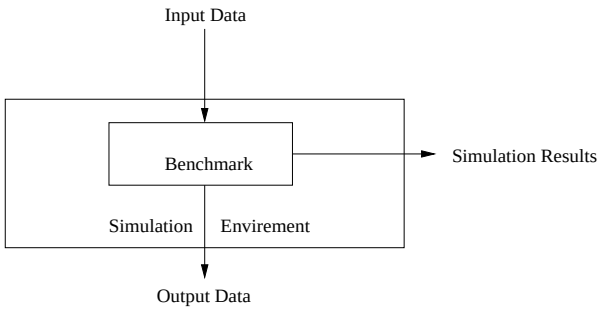


Figure 3.3: The general working model of benchmark

Additionally, a benchmark should be accuracy, scalability and representativeness. That means a benchmark should reflect the situation in real world; It should be suitable for different requirements without large modification; It should test almost all functionalities of a testing system and present each aspect accurately.

3.2.2 Benchmark for SIP Implementation

When we design the benchmark for SIP, we should make it clear that several steps should be implemented. First, we make the benchmark only in function level. Each benchmark measures or tests one or more functions. The second step is defining input data and how to generate it. Then we decide which functions should be involved. In our project, we make benchmarks for three functions which will be described later. The last step is result analysis, the result should reflect the characteristics of SIP that we want to know.

- **Input and Output** How to define the input data type and structure is one of the most important problems of a benchmark. Generally, we see that the input data to a SIP server is SIP message and the output is SIP message too. But from the previous chapter, we know there are a number of types of SIP messages. Different type of messages should involve different operations. So we define the type and structure of input data first.

  – **Input Data Structure:** To a real SIP proxy server, the input data (SIP message) should be came from input port one by one. The number of input data depends on several conditions, such as the number of users, the network capacity, buffer size etc. Normally, a server accept thousands of SIP messages per second. In our benchmark we create an input file to instead of the real incoming SIP messages. The input file contains thousands of SIP messages. The structure of the input file is shown as Figure 3.4.

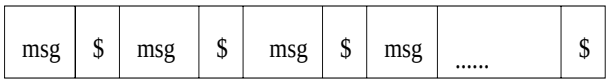


Figure 3.4: The structure of the input file

In the Figure 3.4, we can see that we use a symbol " $ " at the end of each message. It can be considered as a stop symbol, which helps the program to recognize the messages one by one. We make a tool to generate such input file automatically before we start running benchmark in simulation environment.

In the simulation, the program reads the input file first and put all content of the file into buffer. If we read file each time when we parse each message, it will cost a lots of time for reading during the simulation. So the better way is loading the input file into the buffer. When the program parses the SIP messages, it starts from the beginning address of the buffer where the input file saved. When it meets a stop symbol, the program knows that the first

message is over. In the next time, it skips the stop symbol and starts to read the next message till meets an other stop symbol.

In Figure 3.2, we see the input data to function *parse_msg* is a serial parameters, i.e. buf, len, sip_msg. Where the *buf* is the start address of the buffer where a received message is stored in. The *len* is the length of the message, in decimal number of octets. The *sip_msg* is a SIP structure that is the most important structure in this program. The result of the message parser is filling this structure.

– **Input Data Types:** As we mentioned before, there are a number of types of SIP messages. But from the functionalities of the program, we see that the different operations only based on the request, response or register messages. So, we classify the input data in the three classes: request, response and register though there are many different types of messages in each class. What percentage of each class occupation in an input file is the most important problem here.

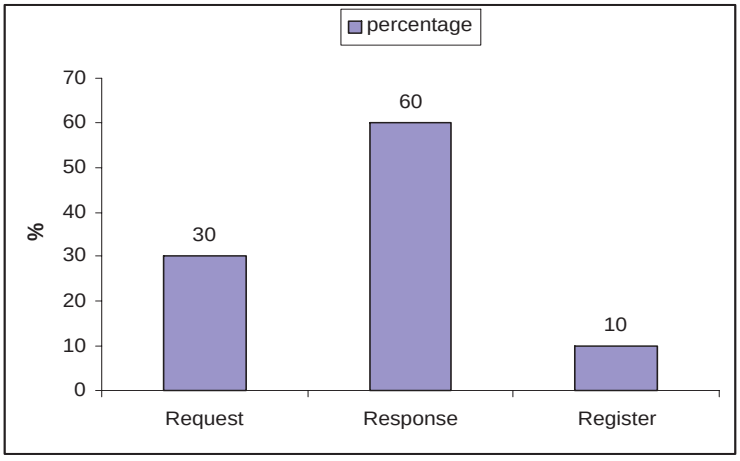


Figure 3.5: The distribution of the each class of messages in an input file

In Figure 3.5, we see the distribution of each class in an input file. The request message occupies 30% and response message occupies 60%, the register message only takes 10% in an input file. The decision comes from the following rules:

1. When we review the typical transaction model of SIP in Figure 2.6, we find that it only needs one request (INVITE) to set up a session. Note that the following request ACK or BYE doesn't pass the proxy server. And there are two responses (180 ringing, 200 OK) need to be passed by server in the model. Consider a forking server, it receives more than two responses during setting up a session.
2. We know every UA should register in server when it startup or when it moves to other SIP domains. That means if the UA moves only in one

SIP domain, it only needs to register once. So the number of register operation is small.

- **message parser** Message parser is one of the most time-consuming part in SIP server. The main function of this part is *parse_msg*. The Figure 3.2 shows the work flow of this function. The input of this function are three arguments, which are buf, len and sip_msg. We make benchmark for this function in several steps:

  1. Making a program to Generate the input file automatically. The program is designed to generate SIP messages following the rules of the message distribution and types that we defined before. The structure of the file should meet the requirement exactly. That means the program has to write a symbol " $ " at the end of each message. We make several input files that contain 500, 1000, 2000, 3000, 5000 messages individually. The program is wrote to get the input file as an argument and read the content of the file into buffer. There are two pointers. One is start pointer that point to the beginning address of the file. The other is end pointer that keeps increasing from the beginning address till meets a stop symbol. The program calculates the number of characters between these two pointers as *len*. The start pointer is assigned to *buf*. The program creates a sip_msg structure and initializes it as zero. Then it can start to parse the message.
  2. The first step of parse message is parsing the first line. Because the different types of messages are determined by the first line, so this function is the most important function in *parse_msg*. In this function, the program reads the first character of the line and compares it with several key words. The examples are given in Figure 3.6.

First Char.	Msg. Type	Subsequent Operation
I or i	Request	INVITE, get target address
S or s	Response	get status code
R or r	Register	get domain name or address
C or c	Request	CANCEL, get target address

Figure 3.6: The examples of matching the first character

After this operation, the program should set a flag to mark the type of this message. If the first character is not matched, the flag will be set to ERROR. Based on different types of the message, the specific operations has to be done. For example, to a response, the program has to get the status code and verify if it is valid. The return data of the function is the address to the end of the first line. So in the next step, the program can start after the first line.

  3. Parse header fields is the following work. The header fields of a SIP message includes five compulsory fields and a random number optional fields. Because of this property, the program should only parse the presented header fields in a message. The function is done in *parse_headers*. The input data to this function is *sip_msg, flag, next*, where *sip_msg* is a SIP structure, *flag* indicates

what type it is and *next* is to show if there is an other header field doesn't be parsed yet. The program invokes the sub-function *get_hdr_field* for each header field. This sub-function parses a header field per time and gets the type of the field, verifies the content of the header field. The return data is the type of the header field. Then the program fills each items in SIP structure *sip_msg*.

- **Action and Forward** Based on the result of the message parser, the program does two operations to finish its work: Action and Forward. We put these two function together because they are the parallel forks in program flow. As we discussed before, if the message is a request, it will goes to **Action**, or it will goes to the other way - **Forward**.

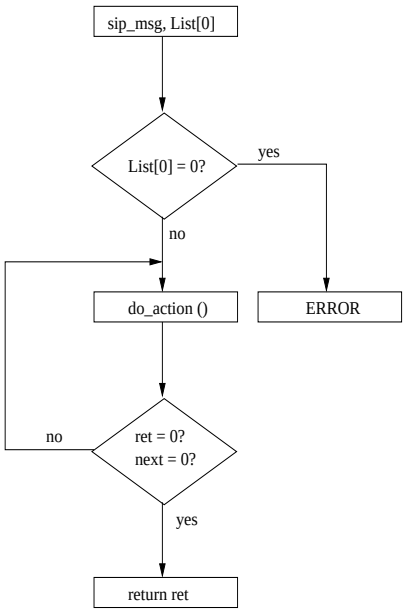


Figure 3.7: The work flow chart of action

– Action is the operation that executes on the SIP message following the user commands. The operation is done in function *run_action* with the input arguments – sip_msg and command list. Here the sip_msg is the result of message parser.

First, the program check if the command list is null, if it finds command items,then go to *do_action*. There is a big switch statement in this function. Each case of the statements presents a type of command, such as drop, forward, log, set_uri etc. For example, if the first command is forward, the message will be either forwarded to Request URI of the message or to IP or host that given as parameter. If the command is executed successful, the function returns *true* and the program executes the next command if exist. If the return status is *false* or no command left in the list, the program return to

main function. Note that the command list is translated from configuration file in the server startup.

– Forward is a simple function to forward response message to destination. The only thing that we should take care is the updating *Via* header field. The function *forward_reply* does all these things about forwarding. The input data of this function is sip_msg that is the result of message parser. If the program configures a module to do this operation, the program invokes the module *response_f*, otherwise, it has to forward message stateless. Which means it needs the second *Via* header field. The program first checks the second *Via* header field, then removes the first one and calls *msg_send* to send message out.

3.2.3 Message Generation Tool

When we run the benchmark for simulation, we need thousands SIP messages as input data. For convince, we make a tool to generate SIP messages automatically. The generated messages are write in a file in a required order. Which means we insert a stop symbol " $ " at the end of each message. It helps program reads the message individually like we mentioned before.

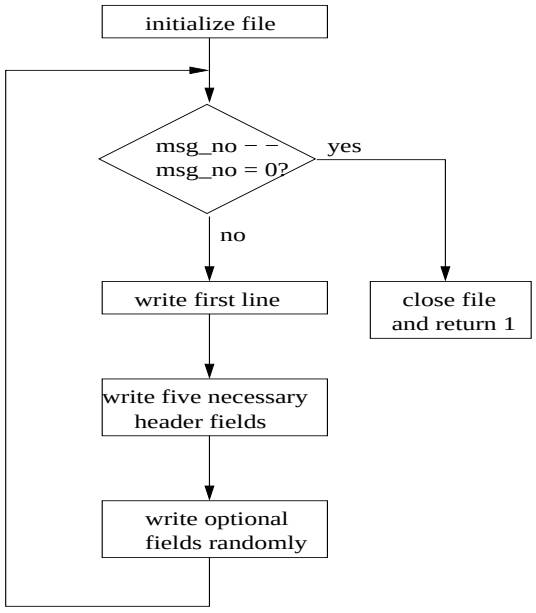


Figure 3.8: The flow chart of message generation tool

First, how to create the first line is very important because the first line determines the type of the message. We use random number to control the percentage of each class of message. For example, we put random seed as 10, so each number from 0 to 9 has 10% probability to be chosen. We assume that the register message is chosen if the returned random number is 3, which means the probability to create register message is 10%.

Random(10)	Percentage	Types
0, 1, 2	30%	Request
3	10%	Register
4 – 9	60%	Response

Figure 3.9: The definition of percentage of each type of the message

The second part is how to create header fields in a reasonable way. Following the first line, five necessary header fields have to be created, they are *To, From, Via, Call-ID, CSeq*. We should note that the response message needs the second *Via* field, so if the message is a response, we add an other *Via* header field in the message.To other optional header fields, we divide them into two parts. One group is often used header fields named as A, which consists of *Contact, Subject, Content-Length, Content-Type, Max-Forward* and *Route*. The other optional header fields are grouped as B, it includes *User-Agent, Supported, Unsupported, Require, Priority, Recontact, Accept-Language, Organization, Record-Route, Accept, Content-Disposition, Allow, Proxy-Require, Authorization, My-State, Event*. The fields in group A are often used because they give the normal and useful information of the user. We also use random number to control the occupancy percentage of these two groups. In each time the program chooses a header field to write in the file, the fields in group A has double probability to be chosen than the fields in group B.

3.3 SimpleScalar Tool Set

The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification[1]. It can emulate the Alpha, PISA, ARM, and x86 instruction sets. The tool set consists of an assembler, a linker, a simulator and a visualization tools for the SimpleScalar architecture[4].

Figure 3.10 depicts the working flow of SimpleScalar tool set. For example, our benchmark is wrote in c language, we have to compile it by SimpleScalar GCC that is retargeted toward to SimpleScalar architecture. The SimpleScalar GCC generates assembly that to be assembled by SimpleScalar GAS and generates object files. The SimpleScalar GLD links object files with libraries to generate SimpleScalar executables. This executable can be run in simulators provided by SimpleScalar tool set, such as sim-fast, sim-ourorder, sim-safe etc..

The SimpleScalar architecture is derived from the MIPS-IV ISA. The tool suite defines both little-endian and big-endian versions of the architecture[4]. Additional, there are three instruction encodings of SimpleScalar instructions: *register, immediate and jump* formats. All instructions are 64 bits in length. Shown as Figure 3.11. The register format is used for computational instructions. The immediate format supports the inclusion of a 16-bit constant. The jump format supports specification of 24-bit jump targets. Each instruction format has a fixed-location that is a 16-bit opcode field, it

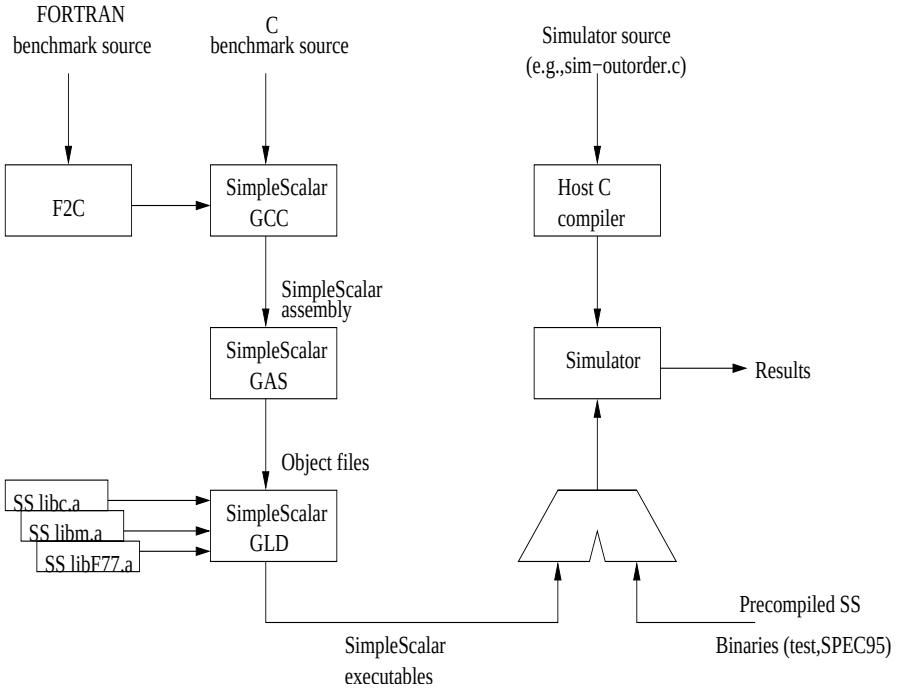


Figure 3.10: The overview of SimpleScalar tool set

provides a fast instruction decoding.

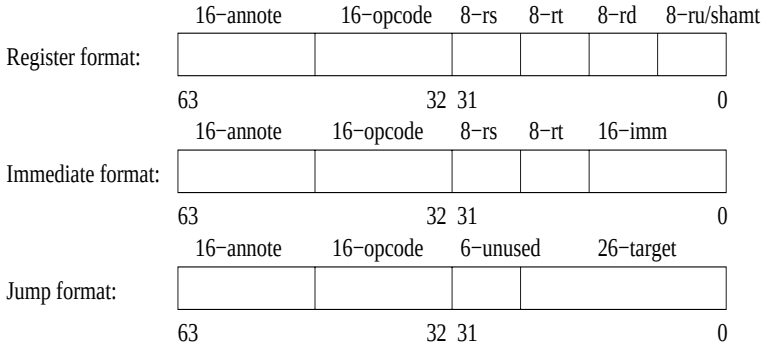


Figure 3.11: SimpleScalar architecture instruction formats

In our simulation, we use the simulator *sim-outorder* that is the most complicated and detailed simulator in this tool set. This simulator supports out-of-order issue and execution.

In Figure 3.12, it is clear that the work of sim-outorder simulator can be divided into six stages. We give the briefly description for each stage.

- The first is fetch stage. In this stage, it takes the following inputs: the program counter, the predictor state, misprediction detection from the branch execution

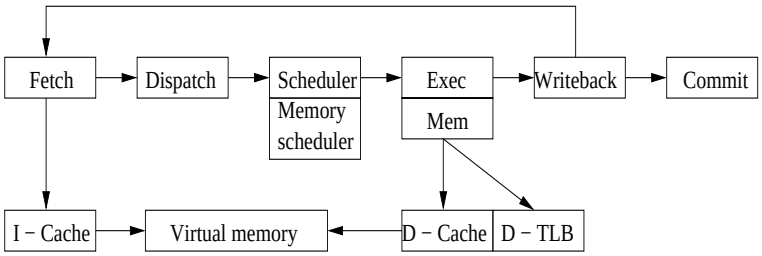


Figure 3.12: Pipeline for sim-outorder simulator

units. It fetches instructions each cycle and put them in the dispatch queue.

- In dispatch stage, the program decodes the instruction and renames the register. It takes as many instructions as possible from the fetch queue per cycle and places them in the scheduler queue. Additional, the program enters and links instructions into the register update unit (RUU) and the load/store queue (LSQ).

- In third stage, if the register inputs are all ready, the program will locates the instructions for it. The issue of ready loads is stalled if there is an earlier store with an unresolved effective address in the LSQ. If the address of the earlier store is same as the waiting one, the store value will be loaded or the load is sent to the memory system.

- The fourth stage is execute stage. The routine gets as many ready instructions as possible from the scheduler queue each cycle. And the routine also schedules writeback operations using the latency of the functional units.

- In writeback stage, the routine scans the operation queue that scheduled in last stage and gets instruction completions. When finding a completed instruction, it goes to the dependence chain and mark the instructions that are dependent on the completed instruction.

- The last stage is commit stage. The routine does in-order committing of instructions, updating of the data caches and data TLB miss handling. The instruction at the head of the RUU will be committed when it is ready. The result of committed instruction is stored in the architected register file.

3.4 Conclusion

In this chapter, we discuss the benchmark for SIP in detail. First, we introduce the implementation of SIP, which is an exist application that is named SIP Express Router (SER). Then we describe the detail functionalities of SER. It helps us to determine which parts of the application are more important. This is the fundamental of our benchmark. Then we introduce the benchmark with the definition, the methodology. Based on this knowledge, we make our benchmark in function level. The whole benchmark suite

for SIP consists of three benchmarks: message parser benchmark, action benchmark and reply benchmark. Each benchmark consists of input data, simulation processing and expected results. Additional, a message generation tool is created for convince to generate the input data for the benchmarks. In the last part of this chapter, we introduce the simulation environment.

4 Simulation Results Analysis

In the previous chapter, we discussed the implementation of SIP benchmarks and the simulation environment. The three benchmarks are created, they are: the message parser benchmark, the action benchmark and the forward benchmark. The three benchmarks cover nearly all important functions in an SIP proxy server. The simulation environment includes the cycle-accurate simulator sim-outorder from the SimpleScalar tool sets. The profiling results are obtained by compiling the benchmarks and running the benchmarks on the above mentioned simulator with the appropriate input data. We distinguish two types of profiling results, namely performance-related results (in terms of cycles) and architectural characteristics (e.g., IPC, cache miss rate etc.).

In this chapter, Section 4.1 introduces the simulation rules that describe the definitions and simulation environment. Section 4.2 presents an overview of the the performance of each benchmark in SIP server. Section 4.3 describes in detail the profiling results of each benchmark. Section 4.4 presents the results on architectural characteristics. Section 4.5 concludes this chapter with some concluding remarks.

4.1 Description of Simulation Rules

Before we run the benchmarks on the simulator, we have to state our assumptions. The assumptions details the parameters utilized in the simulation environment and are needed to meaningfully interpret the simulation results. A detailed description of the assumptions is presented in Chapter 3, therefore, only a short explanation of each assumption is presented in the following:

- **Definition of input data** The input data to message parser is a file that contains thousands of different types of SIP messages. The file is created in different sizes, we use five files which contains 500, 1000, 2000, 3000, 5000 SIP messages respectively. The input data for the action benchmark are an SIP structure and a command list. We use three commands here, they are *Len_GT, Forward_UDP, Drop*. The input data for the forward benchmark is only an SIP structure. Note that the SIP structure is filled by message parser.

- **Sim-outorder simulator** The simulator entails a 2-way superscalar processor. It has 64 KB direct-mapped level 1 (L1) data and instruction cache with 1 clock cycle latency and 1 MB unified level 2 (L2) cache with 6 clock cycles latency.

- **The base machine** It consists of 4 integer ALUs, 1 integer MULT/DIV unit, 4 floating-point adders, 1 floating-point MULT/DIV unit, 2 memory ports (read/write).

- **Simulation times** We run each benchmark three times using each input file, then we get the average of the three times as the results.

- **The limitation** We have to note that we have chosen to benchmark the proxy server as it provides the widest range of functionalities and is the most complex compared to other servers (i.e., redirect server and registrar). Finally, we must also note that we utilized synthetically generated input data instead of the real-world data. However, the generation of SIP messages serving as input for the benchmarks is based on the statistics gathered from the real SIP message.

4.2 Results Overview

We give the overview of the results first. These results are measuring the performance of each benchmark in an SIP server. It gives the number of clock cycles of each benchmark comparison to the total cycles of the application.

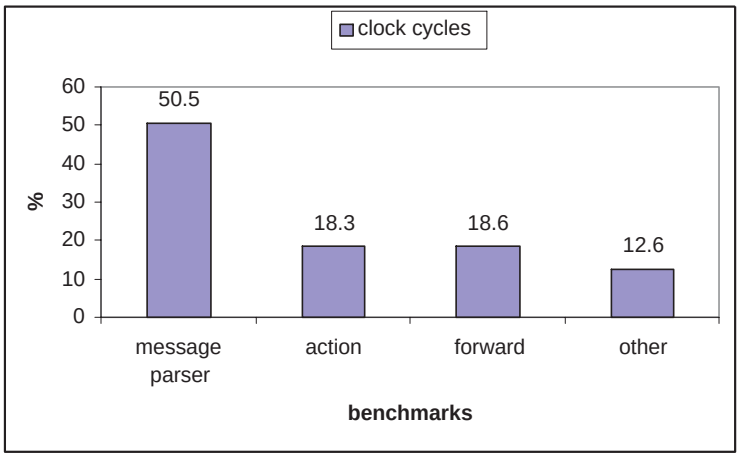
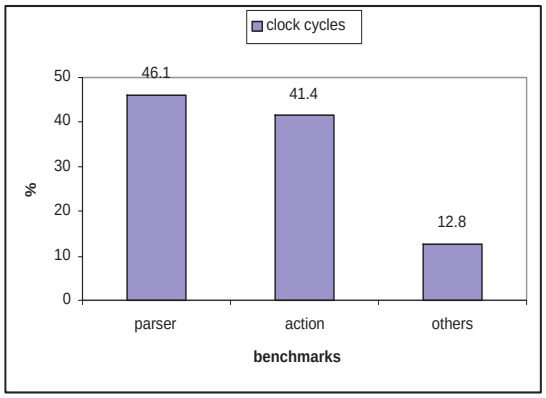


Figure 4.1: The results of each benchmark comparison to the whole application

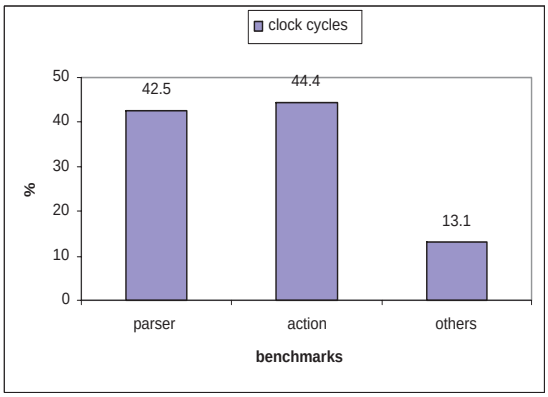
The results of each benchmark are presented by percentage in Figure 4.1. We see that the message parser benchmark takes 50.5% that is the biggest part in the total clock cycles. From previous chapter, we know that the message parser parses each line of an SIP header, it does many comparisons to match the type of the message and the types of the header fields. This is the most time consuming work in a server. The action benchmark occupies 18.3% because we use three commands in our benchmark including Forwarding command. The number of cycles are used in this benchmark is based on the size of command list and what operations are involved. Normally, a request should be transferred to other side also, so the forwarding command is used. The forward benchmark only takes 18.6% in whole cycles though the response messages occupy 60% of the input data. The reason is that the operations in this part is not complex. The other operations including post-script callbacks take 13.1%. From the result, we can see

that our benchmarks cover 86.9% operations in an executing server except the part of startup.

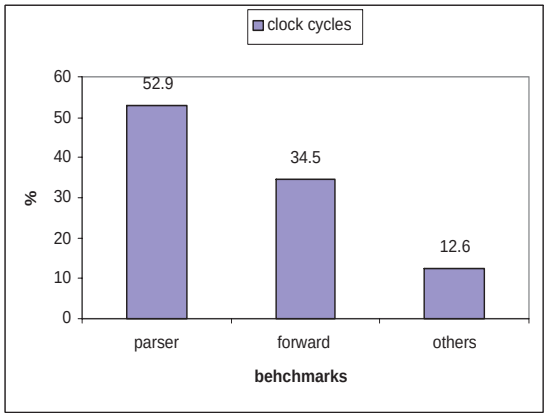
In this section, we also give the results of the extreme testing, which means we give server only one type of messages to see if the result is changed. In fact, this part is very important because we run each benchmark under the almost same conditions. The comparison of the results can give us the raw performance of each benchmark. In this testing, each benchmark runs in equal times. For example, when we use the file that contains 3000 request messages, the message parser benchmark and action benchmark are all running 3000 times. The results of the three classes messages are given in Figure 4.2(a), 4.2(b), 4.2(c) that are concerned with the request messages, the response messages and the register messages respectively.



(a) The extreme testing using request messages



(b) The extreme testing using register messages



(c) The extreme testing using response messages

Figure 4.2: The extreme testing of different types of the messages

In Figure 4.2, it is clear that the number of clock cycles for executing message parser benchmark is the biggest one when using request messages and response messages. But when using register messages, the action benchmark takes more cycles than message parser benchmark. From the analysis of functionalities, we know that a register message

will invoke an operation of looking up hash table and comparing, changing, adding the items. But in our benchmarks, it doesn't involve the operations of the registrar, the register messages are marked as request but unknown types. It is executed as a request and invokes extra operations to delete it. That is why the action costs more time in this case. In these three figures, the forward benchmark takes the least number of cycles, which is as same as the expected result we analyzed before. The other result is in Figure 4.1, the forward benchmark takes more cycles than action benchmark only because the response messages occupy 60% of the input data. To the functionality itself, the forward benchmark is the simplest one of the three benchmarks.

4.3 Results of Functions

After the overview of the results, we focus on each benchmark to find the performance of the sub-functions in it. In this part, the message parser is also the most complicated one, and the forward benchmark is so simple that we only give a description of it.

4.3.1 Message Parser

In Section 3.2, we gave the structure of message parser. There are two main sub-functions in it, *parse_first_line* and *parse_headers*. The *parse_first_line* only does the work to determine the type of the message and gets some related information. For example, to a response message, it gets the status code. The *parse_headers* does more work, it parses all header fields that presented in the message header. Figure 4.3 depicts the number of clock cycles for these two functions respectively. It is clear that *parse_headers* is the most time consuming function in a server.

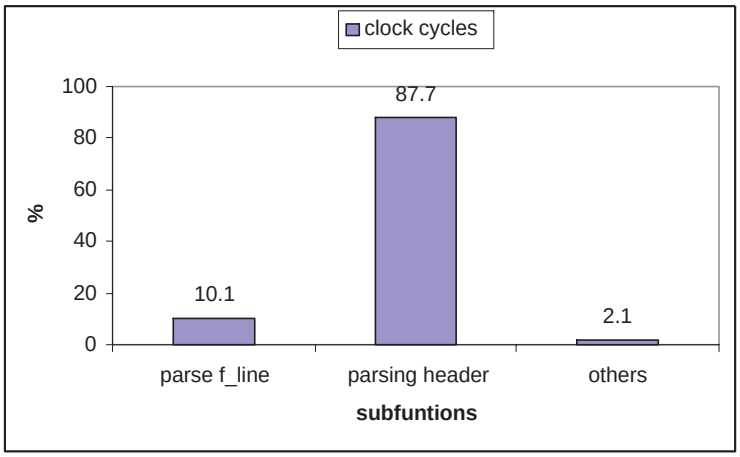


Figure 4.3: The results of sub-functions in *parse_message*

4.3.2 Action and Forward

To the action benchmark, most of the work is done in main function, so we only give description of it. To the forward benchmark, it consists several sub-functions. The operations of these sub-functions are simple, we give details following.

- **Action benchmark** In the benchmark, there is a big switch statement and each case of the statements presents a command. Each command involves some operations. So the number of cycles are used in this benchmark depends on how many commands in command list and what operations are involved by the commands. In previous section, we only use several often used commands, but in fact, this benchmark can be the most time consuming one if more commands are executed.

- **Forward benchmark** From the analysis of the functionality, we know that the operations of this function is not complex. It includes the socket update (remove the second Via header field) and the sending out. We know the most important function is the updating header field but the operation is not complicated. The sub-function for sending out only transfers the structure of an SIP message to the buffer, then waiting for the lower application to send it out. From the Figure 4.4, we see that more time is spent on the memory and the buffer operations.

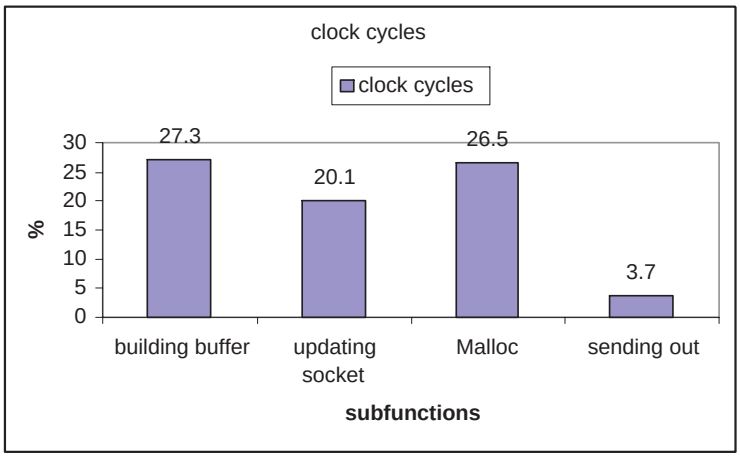


Figure 4.4: The results of sub-functions in *forward_reply*

4.4 Results on Architectural Characteristics

In the work of designing network processor, we have to take the characteristics of the protocol into account. For example, how much cache should be used in a network processor? We have to know the performance of the implementation of the protocol under different sizes of the cache. Which helps the designer to design the network processor in a cost-efficient way.

types	IPC	iL1	dL1	uL2	load	store	APR	DPR
normal	1.1485	0.0547	0.0087	0.0056	24.5	15.7	0.9314	0.9413
request	1.1168	0.0576	0.0094	0.0054	24.7	15.4	0.9300	0.9390
response	1.1876	0.0519	0.0082	0.0061	24.9	15.4	0.9351	0.9461
register	1.1184	0.0575	0.0091	0.0055	24.7	15.4	0.9277	0.9382

Table 4.1: Architectural Characteristics with Different Inputs

size	IPC	iL1	dL1	uL2
8K	1.2520	0.0466	0.0515	0.0067
16K	1.3390	0.0367	0.0361	0.0075
32K	1.4354	0.0271	0.0263	0.0081

Table 4.2: Architectural Characteristics with Different Sizes of Cache

We have two methods to get the results. First, we use different input datum to determine how it changes in each characteristic. Then we set different sizes of caches to test it again.

- **Results of Different Input** In this part, we define four classes input data: normal, request, response and register. The normal message means that it contains 30% request message, 60% response message and 10% register message. The other three classes messages are same as extreme testing. That means the input file contains only one class of messages.

  In the Table 4.1, when the input messages are the response messages, we got the highest IPC. And the iL1 is Level-1 instruction cache miss rate, dL1 is Level-1 data cache miss rate, load/store instruction rate, branch address-prediction rate (APR) and branch direction-prediction rate (DPR). The total number of instructions executed is about 78.7M and total cycles are 60M.

- **Results of Different Size of Caches** We know that the size of the cache can affect the performance of processor greatly. In this part, we use different size of caches to find out the affect to each characteristic.

  Table 4.2 shows the results. We change the size of cache from 8Kb to 32Kb and find that the performance became better with increasing of the cache size.

Additionally, we know that the cache consists of sets and block. When we change the size of cache, we can change the number of sets or the size of block. We find the results are different between these two ways. The input data is normal message, the results are show in Table 4.3. We find the change of block size of L1 instruction cache can get higher IPC under same size of cache.

size	sets	block	IPC	iL1	dL1	uL2
8K	128	64	1.2520	0.0466	0.0515	0.0067
8K	256	32	1.0454	0.0696	0.0531	0.0066
16K	256	64	1.3390	0.0367	0.0361	0.0075
16K	512	32	1.1555	0.0542	0.0345	0.0076
32K	512	64	1.4354	0.0271	0.0263	0.0081
32K	1024	32	1.2825	0.0396	0.0236	0.0082

Table 4.3: Comparison between Different Sets of Cache

4.5 Conclusion

In this chapter, we conclude the results of our benchmarks. First is the overview of the results . We compared the performance of the three benchmarks in terms of the clock cycles to determine which is the most time consuming one. In this step, the message parser benchmark constitutes 50.5% executing cycles of the total cycles. The action benchmark constitutes 18.3% and forward benchmark constitutes 18.6% of the total cycles. Then, we do the extreme testing for these three benchmarks to get raw performance. Under the same condition, the forward benchmark only constitutes 34.5% of the total cycles, which means that this functionality is the simplest one in these three functionalities. The message parser benchmark constitute 47.2% and the action benchmark constitutes 42.9%. Then, we test each benchmark to get performance of the sub-functions. To the action benchmark, there are many sub-functions and each of them presents a command. The sub-function is invoked only when the related command is in the command list. In the forward benchmark, the memory allocation takes more time which means the operations in this part is not complex. We investigated architectural characteristics in several aspects. With the increase of L1 instruction cache, the IPC is increasing and cache miss rate is decreasing. With the same size of the L1 instruction cache, we use small sets and big block size can provide higher performance of the network processor. The results can help us to find the cost-efficient way when design a network processor to implement SIP.

5 Conclusion

Designing cost-effective network processors is one of the most challenging of current computer architecture problems. It includes the investigations of both software and hardware. The benchmark of network protocol is part of the investigation in software. From this work, we can find the characteristics of a protocol, for example, we implement some operations that take complicated computations in hardware to speed up the performance of network processors. From the previous chapters, we know that SIP is a promising protocol that has some outstanding characteristics, such as using URI to define a user, using simple hypertext syntax and formats, feature modification etc. It can be used in many fields such as Voice over IP (VoIP), Videoconferencing, network mobility etc. Though there are only few SIP network exist now, the benchmark of SIP is an expected work.

In this thesis, we discuss such topic from implementation of SIP to the benchmark results analysis. Following we will give conclusion of the work. Section 5.1 discuss the overview of conclusion. Section 5.2 underlines some related work that should be done in the future.

5.1 Overview Conclusion

We give the introduce of network processors at the beginning of this thesis. It answers the questions about what is network processor and why we need it. Then some basic knowledge about network processor design is necessary.

In Chapter 2, we described the main aspects of the Session Initiation Protocol. First, we determined the basic features of SIP and what it can do. Subsequently, we defined the basic elements in an SIP network and explained the functionalities of each element. In this chapter, we also discussed the SIP message and different transaction models. Learning the detail of SIP message could help us to understand different operations in transaction models. The transaction models are the implementations of SIP, they contain many operations based on different SIP messages.

In Chapter 3, we discussed the benchmark for SIP in detail. First, we introduced the implementation of SIP, which is an exist application that is named SIP Express Router (SER). Subsequently, we described the detail functionalities of SER. This helps us to determine which parts of the application are more important, which is the main focus of out investigation. We introduced the benchmark with the definition, the methodology. Based on this knowledge, we created out benchmark suite on functional level. The whole benchmark suite consists of three benchmarks: message parser benchmark, action benchmark and reply benchmark. Each benchmark consists of input data, simulation processing and expected results. Additional, a message generation tool is created for convenience to generate the input data for the benchmarks. In the last part of this chapter, we introduced the simulation tool set and the environment.

In Chapter 4, we presented the results of our benchmarks. First is the overview of the results . We compared the performance of the three benchmarks in terms of the clock cycles to determine which is the most time consuming one. The message parser benchmark takes 50.5% in the whole clock cycles of the benchmarks. Subsequently, we did the extreme testing for these three benchmarks to get raw performance. From the comparison, it is clear that Message parser benchmark is the most time consuming one, the average number of the cycles of the three testing is 48.7% which means that this functionality should be focused on during the implementation of SIP. Then, we tested each benchmark to get performance of the sub-functions. To the action benchmark, there are many sub-functions and each of them presents a command. The sub-function is invoked only when the related command is in the command list. In the forward benchmark, the memory allocation takes more time which means the operations in this part is not complex. We investigated architectural characteristics in several aspects. When using the normal input data, the IPC is 1.1485, level-1 instruction cache miss rate is 0.0547, APR is 0.9314, and DPR is 0.9413. With the increasing of the size of the level-1 cache, the IPC is increased and the iL1 is decreased as our expectation. The results can help us to find the cost-efficient way when design a network processor to implement SIP.

5.2 Main Contributions

First, we defined a procedure to create the benchmark for a protocol. It includes the SIP implementation, functionality analysis, creating benchmarks, simulation environment, and results analysis. The procedure can be used for other benchmarks for protocols.

Second, we described the methodology of determining the most important functionality in a protocol. We divided the whole program into several functionalities and compared them in terms of the priorities, the structure complexity, and the number of the computations. Which helps us to determine the important functionalities.

Third, our benchmarks determined that the message parser is the most time consuming functionality in SIP, and increasing L1 cache size can speed up the performance of the SIP processing. Additionally, With the same size of the L1 instruction size, using big block size and small number of the sets can provides a higher performance of the network processor.

5.3 Further Research

In this thesis, we have presented a benchmark suite that allows us to investigate the performance of network processors in relation to the session initiation protocol (SIP). In this section, we propose several research directions that can be followed to further enhance the results presented in this thesis or widen the application set:

- The input data used in the current benchmark suite are synthetically generated and do not completely match a real-world distribution of SIP messages. We have chosen several distributions of SIP messages that we believe either reflect the real-world or stress the performance of the introduced benchmarks. In order for the

profiling results to even better reflect a real-world distribution of SIP messages, traces of SIP messages must be gathered. We have to note that these traces can immediately serve as input data without major modifications to our benchmarks.

- In this thesis, we only focused on the stateless SIP proxy server. Further investigation is needed to determine the functionalities of the redirect server, registrar, and the stateful servers.
- As mentioned before, the SIP is an application level control protocol. It has to work with many other protocols, e.g., TCP, UDP, SDP, etc. In order to arrive at a single network processor, these protocols must also be further investigated.

Bibliography

[1] *Simplescalar overview*, http://www.simplescalar.com, 2001.

[2] Doug Burger and Todd M. Austin, *The simplescalar tool set, version 2.0*, http://www.simplescalar.com.

[3] Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, and Z. Peter, *Network processor design-issues and practices volume 1*, Morgan Kaufmann Publishers, 2003.

[4] S. Donovan, *The sip info method*, RFC 2976, http://www.ietf.org/rfc/rfc2976.txt, October 2000.

[5] Camarillo Gonzalo, *Sip demystified*, New York: MCGraw, 2002.

[6] Rich Grace, *The benchmark book*, Prentice Hall, 1996.

[7] *The internet engineering task force (ietf)*, http://www.IETF.org.

[8] iptel, *Iptel company*, http://www.iptel.org.

[9] Jan Janak, *Sip introduction*, 2003.

[10] Jan Janak, Jiri Kuthan, and Bogdan Iancu, *Sip express router v0.8.8 - developer's guide*, (2002).

[11] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, and K. Summers, *Session initiation protocol (sip) basic call flow examples*, RFC3665, http://www.ietf.org/rfc/rfc3665.txt, Desceber 2003.

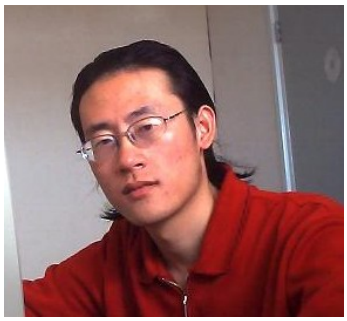
[12] Alan B. Johnston, *Sip: understanding the session initiation protocol*, Artech House, 2001.

[13] Stephen M. Mueller, *Apis and protocols for convergent network services*, McGraw-Hill, 2002.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *Sip: Session initiation protocol*, RFC 3261, http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[15] Eli Wedlund and Henning Schulzrinne, *Mobility support using sip*.

Curriculum Vitae



Jiangbo Yin was born in Wrumqi of China on 05th, December, 1975. He have studied in Electrical Engineering faculty of Shanghai Tiedao University from 1995-1999, and got his bachelor degree in there. From 1999 to 2002, he worked in Beijing Zhikai Official Automatic Equipment Ltd. as a software programmer. His duty is development of lower driver of Pin-printer and deal with the problems as a consultant.

From September, 2002, he started his MSc program in Electrical Engineering faculty of Delft University of Technology (TU Delft) in Netherlands. In November, 2003, he started his MSc project in Computer Engineering (CE) Laboratory. He worked on the topic "benchmark for Session Initiation Protocol (SIP)" with the instruction of Dr. Ir. Stephan Wong. His interesting fields are: digital network, microprocessor, remote control, and embedded system.