

MINIMIZING COMMUNICATION DELAY IN TCP/IP NETWORKS

By

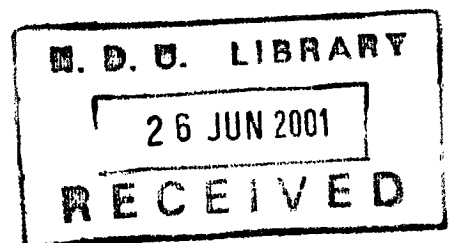
Pascale Y. Ghanem

A Thesis

**Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science**

**Department of Computer Science
Faculty of Natural and Applied Sciences
Notre Dame University**

June 2001

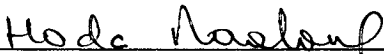


Minimizing Communication Delay In TCP/IP Networks

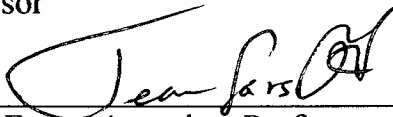
By

Pascale Y. Ghanem

Committee Members



Hoda Maalouf: Assistant Professor of Computer Science
Advisor



Jean Fares: Associate Professor of Mathematics. Dean of Faculty
Member of Committee



Fouad Chedid: Associate Professor of Computer Science. Chairperson
Member of Committee



Khaldoun El-Khalidi: Assistant Professor of Computer Science
Member of Committee

ABSTRACT

The global Internet has experienced many years of sustained exponential growth doubling in size every nine months or faster [8]. Millions of users at tens of thousands of sites around the world depend on the global Internet as part of their daily work environment.

This massive use of the Internet as well as the continuous interconnection of new groups arises many problems such as: packet loss, network congestion, insufficient bandwidth, increase in delay...

In this thesis, we focus mainly on the problem of communication delay and bandwidth allocation. Our main goal is to find a way to minimize the communication delay in the TCP/IP network. Therefore we introduced a new technique that of adopting hop-by-hop packet reassembly with resequencing in opposition to the current method, which consists on end-user resequencing and reassembly. In addition, we introduced a new technique in routing based on finding the fastest path with high MTU. Furthermore, in our study, we alternate the job of the gateway to have the ability of reassembling and resequencing in addition to fragmentation.

With these new techniques, we have shown that communication delay was reduced approximately by 45% comparing to the delay calculated during the end-user resequencing and reassembly adopted by the traditional TCP/IP networks as well as we resulted an increase of 55% in throughput comparing to the common method.

TABLE OF CONTENTS

List of Figures.....	v
List of Equations.....	vi
List of Abbreviations.....	vii
Chapter 1 Introduction and Problem Definition	09
1.1 Introduction	09
1.2 Problem Definition	11
1.3 Research Goals	12
1.4 Approach	13
1.5 Outcomes	14
1.6 Thesis Organization	14
Chapter 2 Background and Motivation	15
2.1 Introduction	15
2.2 TCP/IP Protocol: Structure and Functionality	15
2.2.1 Internet Protocol (IP)	16
2.2.2 Transmission Control Protocol (TCP)	18
2.3 Packet Delivery Services	20
2.3.1 Internet Routing	20
2.3.2 Queue Management	22
2.3.3 Fragmentation and Reassembly	23
2.3.4 Disordering and Resequencing	26
2.4 Motivations	26
Chapter 3 Hop-By-Hop Resequencing and Reassembly	29
3.1 Introduction	29
3.2 Packet Handling	30

3.2.1	Packet Size	30
3.2.2	Packet Arrival Process	31
3.3	Queueing Management	32
3.3.1	Queueing Delay	32
3.3.2	Queueing Theory	33
3.3.3	M/M/1 Queueing Model	34
3.4	Packet Routing	34
3.4.1	Transmission Delay	34
3.4.2	Path MTU Discovery	35
3.4.3	Fastest Path Routing	36
3.5	Gateway	37
3.5.1	Internet Gateways	37
3.5.2	Gateway Functionality	38
3.6	Hop-By-Hop Resequencing and Reassembly Technique	38
3.6.1	Fragmentation Process	39
3.6.2	Buffer Management	41
3.6.3	Resequencing and Reassembly Process	42
Chapter 4	Simulation Analysis	44
4.1	Introduction	44
4.2	Network Topology	44
4.3	Packet Generation	47
4.4	Queueing Delay	49
4.5	Transmission Delay	51
4.6	Fragments Tracing	52
Chapter 5	Conclusion	54
	Bibliography	55
	Appendix A Simulation Code	58

List of Figures.....		Page
Fig 1.1	The structure of networks and routers that provide interconnection	10
Fig 2.1	Format of an Internet datagram	16
Fig 2.2	The format of a TCP segment	19
Fig 2.3	Routing Algorithm	20
Fig 2.4	Routing Table	22
Fig 2.5	Format of an Internet datagram	24
Fig 2.6	Fragment 1	24
Fig 2.7	Fragment 2	25
Fig 3.1	Poisson Process	31
Fig 3.2	Birth-Death Process	33
Fig 3.3	Packet Transmission	34
Fig 3.4	Gateways	37
Fig 3.5	Level Partitioning of our simulation network	39
Fig 3.6	Sequence number settings in Hop-By-Hop Resequencing and Reassembly	40
Fig 3.7	Resequencing Buffer Allocation	41
Fig 3.8	Hop-By-Hop Resequencing and Reassembly Algorithm	43
Fig 4.1	Network Topology	46
Fig 4.2	Mean Arrival Delay for $\mu = 1.5$	48
Fig 4.3	Throughput for different Arrival Rates λ ($\mu = 1.5$)	48
Fig 4.4	Mean Service Delay for $\lambda = 0.5$	50
Fig 4.5	Routing Table	52
Fig 4.6	Throughput vs Mean Delay for $\lambda = 0.5$ and $\mu = 1.5$	52
Fig 4.7	Fragments Tracing in Normal TCP/IP Networks	53
Fig 4.8	Fragments Tracing in our simulated TCP/IP Networks	53

List of Equations.....Page

Eq 1.1 TCP Performance 19

Eq 1.2 Buffer size 26

Eq 1.3 $P(T_1 > t) = P_0(t) = e^{-\lambda t}$ 31

Eq 1.4 $P(T_2 > T_1 + t | T_1 = s) = e^{-\lambda t}$ for $s, t > 0$ 32

Eq 1.5 Transmission Delay 35

Eq 1.6 Traffic Intensity 45

Eq 1.7 Service Time 49

Eq 1.8 Queuing Delay for packet i 49

Eq 1.9 $\rho =$ Intensity Traffic 49

Eq 2.0 Transmission Time 51

Eq 2.1 Transmission Delay 51

List of Abbreviations

BSD	Berkley Software Distribution
DF	Don't Fragment
EGP	Exterior Gateway Protocol
FTP	File Transfer Protocol
HLEN	Header Length Field
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IGP	Interior Gateway Protocol
IHL	Internet Header Length
IP	Internet Protocol
ISO	International Standards Organization
LAN	Local Area Network
LFN	Long Fat Pipe
MF	More-Fragments Flag
MTU	Maximum Transmission Unit
NFB	Number of Fragment Blocks
PMTU	Path MTU
RED	Random Early Detection
SMTP	Simple Mail Transfer Protocol
SPF	Shortest Path First Protocol
TCP	Transportation Control Protocol
TELNET	Remote Login
TTL	Time To Live
WAN	Wide Area Network
WWW	World Wilde Web

ACKNOWLEDGEMENTS

This thesis is the fulfilment of one full year of work. I want to thank all those who helped in the realization of this achievement.

I want to express my sincerest appreciation for the help and supervision of Dr. Hoda Maalouf, my thesis advisor, to my work. Without her patience, helpful advices and encouragement, it would have been very difficult to achieve this thesis on time and complete.

I am also grateful to the faculty and staff of the Natural and Applied Sciences department who helped me along these three years of my graduate study and made my years at Notre Dame University fruitful and enjoyable.

I would like to thank all the friends who helped me achieving this study in a way or another. A special appreciation to Samira Riachi, my boss for her patience and encouragement, which gives me the motivation to continue, especially in the difficult periods of this thesis. I would like also to thank Nathalie and Camille, my best friends, for standing by me and helping me in my work.

Finally, the gratefulness is to my parents and my sisters. Without their priceless love, their continuous support and their sacrifices, I would not have been able to realize this achievement. Without them, I would not have realized my Masters study.

Chapter 1

INTRODUCTION AND PROBLEM DEFINITION

1.1 Introduction

The Internet concept aims to connect different types of networks at different locations and with different architecture. The concept of Internet arises in the early 1960s. It moved from a military project known as ARPANET, which was under the supervision of the US department of defense, to a wider interconnection of several hosts belonging to different sectors of the economy forming what is commonly known today as INTERNET [30].

The Internet concept is extremely powerful since it consists of an interconnection scheme that hides the low level details from the user and makes the collection of networks appears to be a single large network. This interconnection allows communication between a new network added and all other existing networks even though there is no physical connection between them. Of course, this communication will take place only if both computers agree on a set of universal identifiers for moving data to its destination [8]; the International Standards Organization (ISO) sets these standard identifiers. Those ISO standards are rules that each network willing to adhere to the Internet should satisfy despite the structure of this network. (i.e.: LAN, WAN, ...) [30].

Architecture wise, the Internet topology consists of multiple physical networks such as LANs or WANs interconnected to each other by routers. (See Fig 1.1) Those Internet routers are computers that interconnect two networks and pass packets from one network to the other.

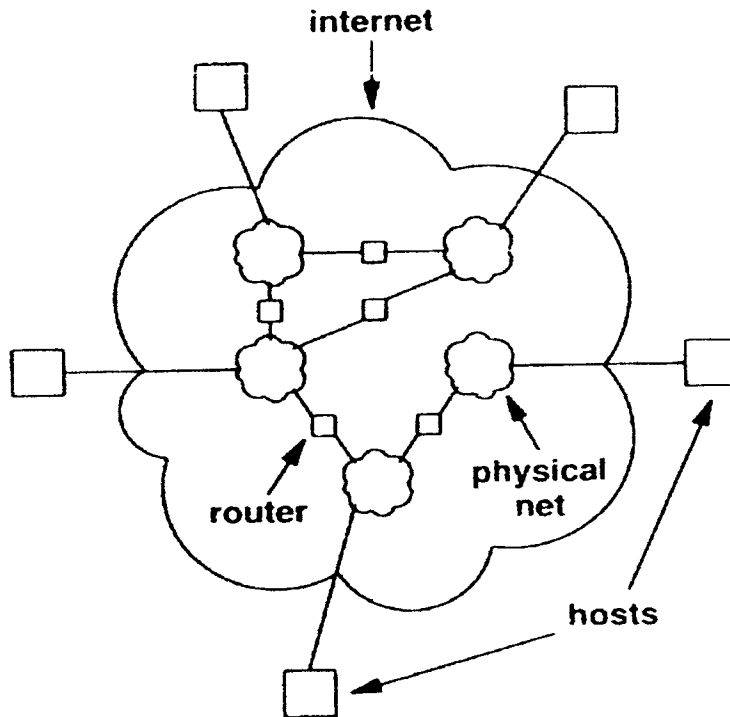


Fig 1.1: The structure of networks and routers that provide interconnection.

To communicate between those interconnected networks we use Internet protocols. The Internet protocols are protocols that define the connectionless delivery mechanism [8]. They can be used to communicate across any set of interconnected networks. They suit well both LANs and WANs communications. The two best-known protocol suites are: Internet Protocol (IP) and Transmission Control Protocol (TCP). Other protocols exist such as: File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and Remote Login (TELNET). TCP/IP protocol was included with Berkley Software Distribution (BSD) UNIX and has since become the foundation on which Internet and World Wide Web (WWW) are based.

The Internet Protocol (IP) is defined as an unreliable, connectionless delivery mechanism. This service is unreliable because delivery is not guaranteed; packets may be lost, duplicated, delayed or delivered out of order. It is said connectionless because each packet is treated independently from all others. While IP protocol is responsible for routing individual packets, Transmission Control Protocol (TCP) is responsible for breaking up the message into IP level packets, reassembling them at the other end, resending anything that gets lost and putting things in the right order. The TCP protocol takes a file that has to be sent to another computer and breaks it into size-suitable packets. TCP adds a header to each

of those packets containing the source “Port Number”, the destination “Port Number” and a “sequence number”. The “Port Number” of the destination is used to keep track of all the packets belonging to the same destination. The destination user reorders and reassembles those packets according to their “sequence number”. A “More Fragment” (MF) bit is used with each packet. This MF is set to zero in the last packet of the file. The destination user uses this MF to identify the last packet and starts reassembling the file.

When we talk about reassembly in the Internet we talk about disordered arrival of packets. When the packet rate sent over the Internet increases, the communication delay between source and destination increases as well as the resequencing and reassembly delay of messages at the end-user increases. Furthermore, all the packets received for a specific file by the destination host will be discarded if one of the packets does not reach the destination host. Our main objective in this thesis is to find techniques to minimize communication delay and maximize throughput for real time applications.

1.2 Problem Definition

If we consider the fact that billions of microprocessors were fabricated since 1997 and many of these embedded microprocessors were connected to Internet since then, we can see the size of the load that is occurring on the Internet since 1997 [29].

On the Internet whenever hosts want to communicate with each other, they divide their data into sequenced packets (i.e.: datagrams) and those packets will be delivered from source to destination using Internet gateways and following different routing algorithms. The communication delay, we are focusing on in our study, results while delivering packets from source to their destination on the Internet. It consists of queueing delay, propagation delay, transmission delay and resequencing delay. The cumulative of those delays increases the overall Communication Delay during packet delivery over the Internet as well as decreases throughput.

As we have said previously Internet consists of several types of networks interconnected to each other by gateways. Each of those networks adapts different size for a packet and have different Path MTU sizes. In the usual TCP/IP network, the datagrams are subdivided into packets of 576-bytes, which are transmitted over a certain path. If those

packets are routed over a path with large MTU, the network utilization is low leading to a decrease in the throughput. On the other hand, if those packets pass through a small MTU path, this will lead to additional fragmentation. Those additional fragmentations increase the transmission delay as well as the resequencing delay because the increase in the number of fragments lead to an increase in network disorder and hence, an increase in the resequencing delay.

In the TCP/IP network, the job of routers was limited to the routing of packets between two hosts and fragmenting those packets when needed, leaving the resequencing and reassembly process to the destination host. This end-user resequencing and reassembly process has two main disadvantages. First, the reassembly of the fragments at the destination will lead to inefficiency witnessed by the decrease of throughput. In fact, although some of the networks at the point of fragmentation have large MTU, datagrams will be divided into small size packets as a result we will not benefit from the resources of these large networks.

Second, since the reassembly takes place at the destination host, this host has to wait until all the fragments of a certain packet reach this host in order to start the reassembly process. It is a major factor of the increase in communication delay especially in a heavily disordered network. Furthermore, if one of the fragments is lost on the way and does not reach the destination, the packet cannot be reassembled and will be discarded.

Our main objective is to suggest better algorithms for packet sizing, packet delivery as well as for the fragmentation and reassembly process. In this thesis, we introduced new technique for resequencing and reassembling packets at the gateways. In addition, we proposed a new technique for packet sizing and routing.

1.3 Research Goals

This research suggests solutions to minimize communication delay on TCP/IP networks and optimize the throughput. Our study will do the following:

- 1 – Optimize Packet size. We will set the packet size equal to the MTU available for the path with the highest bandwidth.
- 2 – Maximize Throughput. Datagrams with sizes smaller than path MTU will waste Internet resources.

3 – Minimize Communication Delay. We will introduce a new hop-by-hop resequencing and reassembly technique.

1.4 Approach

The Communication Delay is the average delay required to deliver a packet from source to its destination. The approach considered in this thesis consists of adopting Hop-By-Hop resequencing and reassembly technique to minimize this communication delay and maximize throughput. Our study is done on a system composed of two networks connected by an exterior gateway where a source host generates packets using a Poisson process with arrival rate λ and routers receive those packets fragmenting and reassembling them according to the MTU of the fastest path before they route them to their destination. (See Figure 1.1)

Our first suggestion concerns the packet size. The usage of small packet size in large MTU paths wastes Internet resources leading to a decrease in the throughput. Therefore, we suggest that the packets will be fragmented into fragments with sizes equal to the MTU of the path they will be routed to. This will minimize the number of fragments and benefit from the resources to the maximum; especially, that throughput increases linearly with packet size.

Second, we are adopting the intranet fragmentation. In this method, the fragmentation and reassembly are performed in every hop of the network. It starts at the source where only fragmentation occurs then at the gateways where both fragmentation and reassembly take place and finally at the destination host where a final reassembly will take place in case some of the packets remain fragmented. This intranet fragmentation allows maximum packet size of each network to be used, since the individual fragments are reassembled by each gateway in the route.

Concerning the resequencing and reassembly process, we introduced a new type of routers having an additional buffer and a new job that of resequencing and reassembling packets which consists of allowing the router to reassemble the packets in a separate buffer called “resequencing buffer”. In addition, we introduced a new Hop-By-Hop resequencing and reassembly technique. This new method consists of allowing the router grouping the fragments belonging to the same packet and checking the MTU of the fastest path available. If the MTU is larger than the packet’s original size, they reassemble them to the original size otherwise to the available MTU. This method sequences the packets according to a new

sequence number assigning techniques. This technique gives each packet 3 numbers: the first number refers to the Packet ID, the second refers to the fragment ID and the last number is a sequence for the order of the fragment. This method will avoid wasting Internet resources by sending small packet size into large MTU paths.

Fourth, in our study, we adopted the Shortest Path method in selecting the path to send the packet through it. This method based on Dijkstra's method for finding Shortest Path. This will allow us to find a path with highest bandwidth, which will deliver the packets faster minimizing the delay, and with largest MTU, which will reduce the number of fragments and maximize the throughput.

1.5 Outcomes

In order to study and analyze our research goals, we used a simulation model imitating a special Internet topology consisting of two networks interconnected by an exterior gateway. In this simulated networks, delay can reach high degree with the presence of several paths with different MTU and bandwidth. However, this simulation showed that our Hop-by-Hop resequencing and reassembly technique as well as our new packet sizing and routing algorithm outperformed the end-user reassembly method adopted in the TCP/IP network. Concerning the delay and throughput calculation, the simulation showed that our method increases 55% the throughput and decreases 50% the delay. (Please, refer to Chapter 4 for results' details)

1.6 Thesis Organization

This thesis consists of five chapters. In Chapter 2, we introduce the TCP/IP protocols as structure and functionalities. In addition, we elaborate on certain packet delivery services such as Internet routing, queueing management, fragmentation and reassembly as well as disordering and resequencing. In Chapter 3, we highlight the different causes of communication delay from packet handling, to transmission delay passing through the queueing delay and resequencing delay. Then, we introduce the new Hop-By-Hop Resequencing and Reassembly technique. In Chapter 4, we analyze the results of the simulation for the new technique and compare it to the End-User resequencing technique. Finally in Chapter 5, we summarize the main results of the thesis.

Chapter 2

Background and Motivation

2.1 Introduction

To be able to understand the delay that occurs when packets flow the Internet from host to host as well as the delay resulting from the Resequencing and reassembly process, we have to understand the TCP/IP protocol which is responsible for managing and routing packets through the Internet. In this chapter, we start by describing in section 2.2 the TCP/IP protocol as structure and functionality. In section 2.3, we elaborate on some processes involved in the packet delivery through Internet such as Internet Routing, Queuing Process, fragmentation/reassembly process and disordering/resequencing process. Finally, this chapter ends with the presentation of our motivations in section 2.4 based on the communication delay problems faced while delivering packets in TCP/IP networks.

2.2 TCP/IP Protocol: Structure and Functionality

TCP/IP Protocol is designed to provide a universal interconnection among machines independent of the particular networks to which they attach [8]. The TCP/IP protocol is suitable to any interconnected network whether a LAN or a WAN. The TCP/IP Internet protocol suite includes: TCP, IP, TELNET, Our focus will be on TCP and IP. The Internet Protocol (IP) is responsible for the fragmentation and reassembly of packets as well as it is responsible for the routing of those packets. In addition to routing and fragmentation, IP handles security on the Internet. Internet Control Message Protocol (ICMP) and Internet Group Management Protocol (IGMP) are considered integral parts of IP, although they are architecturally layered upon IP. ICMP provides error reporting, flow control, first-hop router redirection, and other maintenance and control functions. IGMP provides the mechanisms by which hosts and routers can join and leave IP multicast groups. Concerning the Transmission Control Protocol (TCP), it is responsible for the reliable data delivery. TCP provides end-to-end retransmission, resequencing and connection control. [1]

2.2.1 Internet Protocol

Internet protocol (IP) is the protocol that defines the unreliable, connectionless delivery mechanism [8]. The IP protocol specifies the exact format of all data as it passes across a TCP/IP Internet. In addition, the IP software performs the routing function, choosing a path over which data will be sent. Furthermore, the IP includes a set of rules that defines how the hosts should process packets, how and when error messages should be generated and the conditions under which packets can be delivered. Finally, IP is responsible of fragmenting and reassembling packets when necessary [14].

As we have mentioned earlier, the Internet protocol implements two basic functions: routing and fragmentation. One important mechanism of the Internet Protocol is the Internet routing, which is the selection of a path for transmission. The Internet modules use the addresses carried in the IP header to transmit the datagrams from one host to another until they reach their destinations. During this process, the Internet protocol treats each datagram as an independent entity. These Internet modules (especially in gateways) have procedures for making routing decisions and other functions. The second functionality of IP is fragmentation. The Internet modules use fields in the IP header to fragment and reassemble the datagrams when necessary for transmission through "small packet" networks [13]. In fact, while routing datagrams from host to another, these datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the Internet Protocol. Fragmentation of Internet datagrams will be elaborated later in this chapter.

Architectural wise, the Internet Protocol layout can be described as follow: the Internet datagram consists of a header part and a data part. (see Fig 2.1)

0	10	20	31
Version	IHL	Type of Service	Total Length
Identification		Flags	Fragment Offset
Time to Live	Protocol	Header Checksum	
Source Address			
Destination Address			
Options			Padding
Data			
...			

Figure 2.1: Format of an Internet datagram

The Internet datagram structure starts with a 4-bit *Version* field indicating the IP protocol version. It is used to verify that the sender, receiver and gateways agree upon the format of the datagram. The current version used is IPv4. Internet Header Length (IHL) follows the *Version* field. It is also a 4-bit field referring to the length of the IP header in 32 bit words, pointing to the beginning of the data. The *Total Length* field is the length of the datagram, measured in octets, including Internet header and data. This field allows the length of a datagram to be up to 65,535 bytes. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 bytes. It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams. The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 bytes plus 64 header bytes to fit in a datagram [13].

The Internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum [13]. *The Type of Service* is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters, which characterize the service choices provided in the networks that make up the Internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an Internet datagram. This *Type of Service* field is subdivided into six parameter zones. Those parameters will be set according to the quality of service desired. In fact, some of the networks treat high precedence traffic as more important than other traffic. Those parameter zones are: Precedence a 2-bit field specifying the priority (0 – Normal, 1 – High), D, T and R a one-bit fields. When set, D bit requires a low delay, T bit requests high throughput and R bit requests high reliability. *The Type of Service* is used to specify the treatment of the datagram during its transmission through the Internet system. *The Time to Live (TTL)* is an indication of maximum lifetime of an Internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the *Time to Live reaches zero* before the Internet datagram reaches its destination, the Internet datagram is destroyed. If a gateway holds a datagram for more than one second, it must decrement the TTL by one for each second [4]. The TTL can be thought of as a self-destruct time limit. *The Option* field is used in selected datagrams to carry additional information relating to security for e.g. the data field may be encrypted, source routing, route recording, timestamp and stream identification. The routing and timestamp

options are the most interesting because they provide a way to control how gateways route datagrams and to monitor network performance. The *Options* provide for control functions needed or useful in some situations but unnecessary for the most common communications. The *Header Checksum* provides a verification that the information used in processing Internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the Internet datagram is discarded at once by the entity that detects the error. The three fields *Identification*, *Flags* and *Fragment Offset*, control fragmentation and reassembly. If *Flags* is set to *don't fragment* *D* bit the datagram will be transferred without fragmentation otherwise discarded. The Source Address and Destination Address contain a 32-bit IP address of the sender and receiver although the datagrams may be routed to several intermediate routers but the source and destination remains the same.

2.2.2 Transmission Control Protocol

TCP is represented as part of the Internet protocol suite. TCP provides full-duplex, acknowledged and flow-controlled service to upper-layer protocols. It is the responsible for transporting the data from source to destination. It defines when and how fast data should be sent. TCP gives each packet a unique 32-bit sequence number. TCP can also support numerous simultaneous upper-layer conversations. TCP insures that a packet is delivered by using one of two protocols: the *3-way handshake* protocol or sliding window protocol. In the *3-way handshake* protocol the sender sends one fragment at a time to a certain receiver and for each fragment it waits for an acknowledgment from receiver before sending another one otherwise it will retransmit the same packet again. Whereas, in the second method, the sender sends several packets at a time; then, he waits for one acknowledgment from the receiver once he receives those packets. The sliding window protocol is faster than the 3-way handshake protocol.

The TCP protocol was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments. Production TCP implementations currently adapt to transfer rates in the range of 100 bps to 10^7 bps and round-trip delays in the range 1 ms to 100 seconds. Recent work on TCP performance has shown that TCP can work well over a variety of Internet paths, ranging from 800 Mbit/sec I/O channels to 300 bit/sec dial-up modems [12]. The introduction of fiber optics is resulting in ever-higher transmission speeds, and the fastest paths are moving out of the domain for which TCP was originally engineered. TCP performance depends not

upon the transfer rate itself, but rather upon the product of the transfer rate and the round-trip delay. This

$$\text{Bandwidth} \times \text{Delay} \quad \text{Eq 1.1}$$

measures the amount of data that would "fill the pipe"; it is the buffer space required at sender and receiver to obtain maximum throughput on the TCP connection over the path. TCP performance problems arise when the Eq 1.1 is large. We refer to an Internet path operating in this region as a "long, fat pipe", and a network containing this path as an "LFN" (pronounced "elephan(t)") [13].

Architectural wise, the TCP Protocol packet format as shown in figure 2.2 consists of the following:

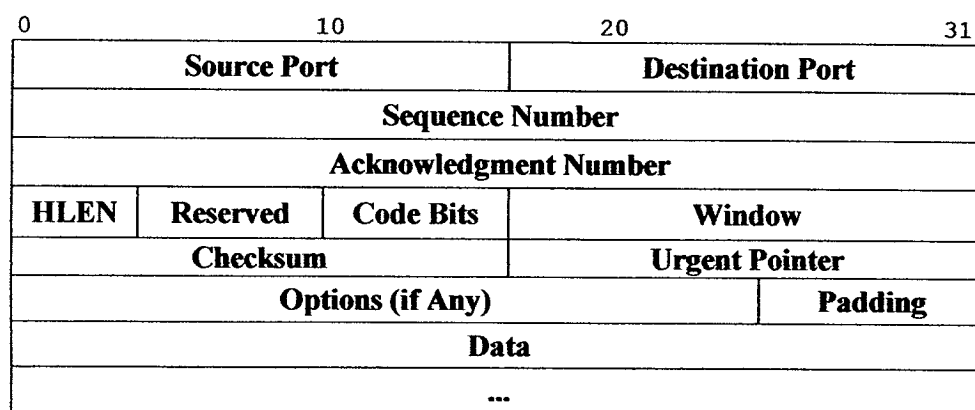


Figure 2.2: The format of a TCP segment

The fields *Source port* and *destination port* identify the points at which upper-layer source and destination processes receive TCP services. The *Sequence Number* field specifies the number assigned to the first byte of data in the current message. Regarding the *Acknowledgment Number*, it contains the sequence number of the next byte of data the sender of the packet expects to receive. The *Reserved field* is for future use. The *Code Bits* field carries a variety of control information whereas the *Window* field specifies the size of the sender's receive window. *Checksum* in TCP segment indicates whether the header was damaged in transit. The *Urgent pointer* points to the first urgent data byte in the packet. The *Options* fields specifies various TCP Options which makes the Header Length Field (*HLEN*) field a must to identify the length of the fragment since *Options* could vary in length. Finally, *Data* field contains upper-layer information.

The operational aspect of both TCP and IP protocols are represented by certain message delivery services such as Internet routing, queuing processes, fragmentation and reassembly as well as disordering and resequencing. Those services will be elaborated further in section 2.3 below.

2.3 Packet Delivery Services

2.3.1 Internet Routing

An Internet is composed of multiple physical networks interconnected by computers called *routers or gateways*. Each router has direct connections to two or more networks. (see Fig. 2.3)

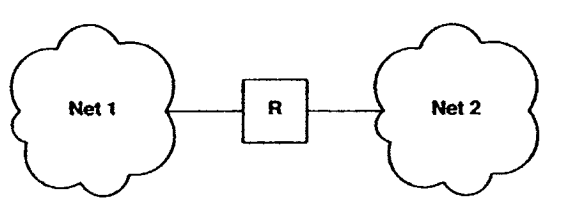


Fig 2.3: Two Physical Networks interconnected by a router R.

Some of the main Internet routers' functions are:

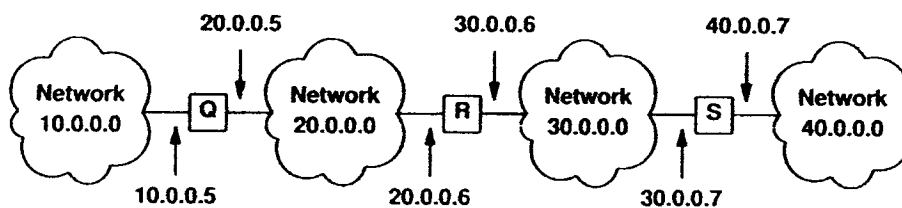
- 1- They conform to most of the Internet protocols.
- 2- Encapsulating and decapsulating IP datagrams [1].
- 3- Sending and receiving IP datagrams up to the maximum size supported by those networks, this size is the network's Maximum Transmission Unit or MTU.
- 4- Routers receive and forward Internet datagrams taking into consideration important issues in this process such as buffer management, congestion control, and fairness.
- 5- Routers choose a next-hop destination for each IP datagram, based on the information in its routing database.

Forwarding an IP datagram generally requires the router to choose the address and relevant interface of the next-hop router or (for the final hop) the destination host. This choice, called relaying or forwarding, depends upon a route database within the router. The forwarding algorithm consists of the following: The router receives the IP packet from the

Link Layer. It validates the IP header. After that, it performs most of the processing of any IP options. Some IP options require additional processing after the routing decision has been made. Following this step, the router examines the destination IP address of the IP datagram to determine how it should continue to process the IP datagram. In this case, there are three possibilities:

- Either, the IP datagram is destined for the router. In this case, it should be queued for local delivery, doing reassembly if needed.
- Alternatively, The IP datagram is not destined for the router. It should be queued for forwarding.
- Otherwise, the IP datagram should be queued for forwarding, but (a copy) must also be queued for local delivery.

The usual IP routing algorithm employs an Internet routing table called *IP routing table* on each machine that stores information about the possible destination and how to reach them [8]. Whenever the IP routing software in a host or router needs to transmit a datagram, it consults the routing table to decide where to send the datagram. This *IP Routing Table* or *Forwarding Table* contains network prefix and not full IP addresses. The size of this table depends on the number of the networks in the Internet. IP routing protocols are dynamic. Dynamic routing calls for routes to be calculated at regular intervals by software in the routing devices. In fact, IP routing specifies that IP datagrams travel through internetworking one hop at a time. The entire route is not known at the beginning of the journey. Instead, at each stop, the next destination is calculated by matching the destination address within the datagram with an entry in the current node's routing table. (See Fig 2.4)



(a)

TO REACH HOSTS ON NETWORK	ROUTE TO THIS ADDRESS
20.0.0.0	DELIVER DIRECTLY
30.0.0.0	DELIVER DIRECTLY
10.0.0.0	20.0.0.5
40.0.0.0	30.0.0.7

(b)

Fig 2.4: (a) Example of Internet with 4 networks and 3 routers, and (b) the routing table in R.

2.3.2 Queuing Management

Typical router queues use a First-In-First-Out queue management policy. The traditional technique for managing router queue lengths is to set a maximum length (in terms of packets) for each queue, accept packets for the queue until the maximum length is reached, then reject (drop) subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted. This technique is known as "tail drop", since the packet that arrived most recently (i.e., the one on the tail of the queue) is dropped when the queue is full.

This method has served the Internet well for years, but it has two important drawbacks: Lockout and Full Queues. In some situations, tail drop allows a single connection or little flows to monopolize queue space, preventing other connections from getting room in the queue. This "lockout" phenomenon is often the result of synchronization or other timing effects. The tail drop discipline allows queues to maintain a full (or, almost full) status for long periods, since tail drop signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the steady-state queue size, and this is perhaps queue management's most important goal. We should note that the drop of packets occurs when the

arrival rate (λ) of packets becomes lower than the service rate (μ) of a packet in a given queue.

Random Early Detection, or RED, is another active queue management algorithm for routers that will provide the Internet performance advantages cited in the previous section [3]. In contrast to traditional queue management algorithms, which drop packets only when the buffer is full, the RED algorithm drops arriving packets probabilistically. The probability of drop increases as the estimated average queue size grows. Note that RED responds to a time-averaged queue length, not an instantaneous one. Thus, if the queue has been mostly empty in the "recent past", RED would not tend to drop packets (unless the queue overflows). On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped.

2.3.3 Fragmentation and Reassembly

The Internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment-offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag (MF) indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams. There are two different fragmentation methods: Intranet fragmentation and Internet Fragmentation. With intranet fragmentation, the fragmentation/reassembly is performed at every subnetwork. With Internet fragmentation, the gateways do not perform any reassembly, only fragmentation procedures. The reassembly job is left to end-systems.

Fragmentation of an Internet datagram is necessary when a packet, in a local network allowing a large packet size, must traverse a local net that limits packets to a smaller size in order to reach its destination. To fragment a long Internet datagram, an Internet protocol module, creates two new Internet datagrams and copies the contents of the Internet header fields from the long datagram into both new Internet headers. The data of the long datagram is divided into two portions on an 8-octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8-octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is

placed in the first new Internet datagram, and the total length field is set to the length of the first datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new Internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new Internet datagram is set to the value of that field in the long datagram plus NFB. An example of fragmentation is given in Figure 2.5:

0	10	20	31
Vers = 4	IHL	Type of Service	Total Length = 472
Identification = 111		MF=0	Fragment Offset = 0
Time to Live=123	Protocol = 6	Header Checksum	
Source Address			
Destination Address			
Options			Padding
Data			
Data			
Data			

Figure 2.5: Format of an Internet datagram

Now the first fragment that results from splitting the datagram after 256 data octets

0	10	20	31
Ver = 4	IHL=5	Type of Service	Total Length = 276
Identification = 111		MF=1	Fragment Offset = 0
TTL = 119	Protocol = 6	Header Checksum	
Source Address			
Destination Address			
Options			Padding
Data			
Data			
Data			

Figure 2.6: Fragment 1

Moreover, the second fragment has the following:

0		10		20		31
Ver = 4	IHL=5	Type of Service		Total Length = 216		
Identification = 111			MF=0	Fragment Offset = 32		
TTL = 119		Protocol = 6		Header Checksum		
Source Address						
Destination Address						
Options					Padding	
Data						
Data						
Data						

Figure 2.7: Fragment 2

To assemble the fragments of an Internet datagram, an Internet protocol module combines Internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's Internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero [23]. For each datagram, the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing. If no other fragment with this buffer identifier is on hand then reassembly resources are allocated.

The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length and bits are set in the fragment block bit table corresponding to the fragment blocks received. If this is the first fragment, (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment, (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram, tested by checking the bits set in the fragment block table, then the datagram is sent to the next step in datagram processing. Otherwise, the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control. If the timer runs out, the all reassembly resources for this buffer identifier are released. Note that the choice of the timer

value is related to the buffer capacity available and the data rate of the transmission medium; that is,

$$\text{buffer size} = \text{data rate} \times \text{timer value} \quad \text{Eq 1.2}$$

(e.g., $10\text{Kb/s} * 15\text{s} = 150\text{Kb}$) [23].

2.3.4 Disordering and Resequencing

In a store and forward communication network, such as the Internet, packets may go out of order when multiple links are used between intermediate nodes or routers [31]. The presence of this disordering process will affect the input arrival stream of datagrams and hence delays each datagram by a random amount, making them leave the network in a different order than the one in which they enter it. However, since the constraint that the datagrams should leave the Internet in the same order in which they entered it, than a datagram may have to undergo an additional delay, which is known as resequencing delay in addition to transmission and queuing delays.

2.4 Research Motivation

The TCP/IP Internet protocol architecture was designed in the early 1980s, at a time when there were many fewer hosts connected to it and typical long-haul links carried only 56 Kbps. Over the past three decades, the Internet has added support for automatic name translation, hierarchical routing, congestion avoidance, dynamic address assignment, multicast, mobility, and most recently, attempts at real-time support [29]. The Internet's scalability is the result of the single-minded focus of its designers on robustness and adaptability. Unfortunately, although its design has focused overwhelmingly on robustness, for all practical purposes, the Internet's largest performance issue is its availability and speed for real-time applications. The Internet's scale, heterogeneity, and dynamic nature make it difficult to determine the exact causes of Internet performance problems. Our focus will be mainly on the communication delay problem. This communication delay problem results from certain policies adopted in the Internet as well as it is the sum of several delays occurring at different parts of the Internet during packet delivery.

The packet size policy adopted nowadays in Internet is one of the causes behind the problem of high latency and low throughput. As we have seen above, the TCP/IP protocol

fragments the packets into fragments of 576-bytes. Those fragments are routed through networks with high MTU preventing the full utilization of those networks.

Routing inefficiency is another problem causing communication delay. The Internet was originally designed to provide universal reachability between networks; all network links were available to carry traffic for any host. Today's Internet restricts the exchange of routing information according to business agreements between service providers. This results in situations where A can reach B and B can reach C, but A cannot reach C [29]. Furthermore, because current Internet routing ignores performance information, two hosts may be forced to communicate over excessively long or overloaded links. Adding a slow link can actually hurt performance; because packets can be routed over it in preference to faster links, in this case the router is called inefficient.

A third cause behind communication delay on the Internet is the propagation and queueing delay. As anyone who has used the Internet knows, the path to a server can be very slow or often completely unavailable. The result of the extreme delay is lost of productivity while users wait for documents to be transmitted over the Internet. Moreover, queuing delay is also one of the main causes to communication delay. Recent publications strongly recommended the widespread deployment of active queue management technology in routers to improve the performance of today's Internet [7]. Active queue management refers to the manipulation of the queue in a router as prejudice to the performance of flows that transit the router. The goals of active queue management are: first, to reduce the average length of queues in routers and thereby decrease the end-to-end delay experienced by packets. Second, ensure that network resources are used more efficiently by reducing the packet loss that occurs when queues overflow.

All the above problems consisted a main motivation for us to consider a new technique for packet delivery over the Internet since packet delay greatly influences the overall performance of the network applications [24]. Therefore, it is a necessary to find the causes of delay performance degradation on the Internet and try to suggest solutions for them. In this thesis, we adapt a new technique called "Hop-By-Hop Resequencing and Reassembly". This technique takes into consideration the different types of problems mentioned above and tries to suggest a new method of improvement. This method consists on:

- 1 Avoiding as much as possible the fragmentation of packets by selecting the path with the highest bandwidth and MTU, which reduced the propagation delay and processing delay.
- 2 The reassembly and resequencing of the packets at each router improves the routing phenomena, reduces the queuing delay, the resequencing delay and hence minimizes the overall communication delay.

Chapter 3

Hop-by-Hop Resequencing and Reassembly

3.1 Introduction

One of the most important performance measures of a data network is the average delay required to deliver a packet from origin to destination known as communication delay. The packet delay will affect the choice and performance of several network algorithms such as routing and flow control. This communication delay is the sum of the delays experienced by the packet at each hop while being forwarded to the destination. Each communication delay consists of the transmission delay at a node and the propagation delay at the links. In addition, it contains a variant component including the processing and queuing delay at a node. To understand and minimize the communication delay we have to introduce better designs and algorithms for the routing and flow control as well as new studies concerning buffer size, link capacity and parameter selection. In this chapter, we are advancing a comparison between the normal TCP/IP protocol and a new method we are introducing: "Hop-by-Hop Resequencing and Reassembly". Our study starts by introducing in section 3.2 the packet handling routine at the source node. We elaborate on the size of a packet as well as we discuss the Poisson process based on which packets are routed from source to destination through the Internet. In section 3.3, we will discuss the queueing management. We start by elaborating on queueing delay, which is the delay between the time the packet is assigned to a queue for transmission and the time it is being transmitted. During this time, the packet waits in the queue while other packets are being transmitted; then, we talk about the queueing theory focusing mainly on the M/M/1 model adopted in this thesis. In section 3.4, we describe the packet routing mechanism. We introduce the transmission delay that corresponds to the delay from the time the packet is transmitted at one end of the link until the time it is received at the other end of the link. We explain the Path MTU discovery and introduce the Fastest Path Routing method used in this thesis to select a path. In section 3.5, we introduce Internet Gateways and their functionalities. Finally, in section 3.6, we describe

the Hop-By-Hop resequencing and reassembly method newly advanced; elaborating on its fragmentation technique, buffer management and resequencing and reassembly procedure.

3.2 Packet Handling

3.2.1 Packet Size

The ideal case in networking is to let a datagram fit in one frame that will be transmitted from source to destination making transmission across the network efficient. Many Link Layer protocols define a maximum frame size according to which no frame with higher size is allowed to be transmitted. We refer to this limit as the MTU; The MTU of a network element is defined to be the maximum transmission unit the network element can accommodate without fragmentation, including IP and upper-layer protocol headers but not including link level headers [8]. MTU could vary from network to another, some networks like Ethernet has an MTU of 1500 bytes for a bandwidth of 10Mbps; whereas, the FDDI network has an MTU equal to 4470 bytes for a bandwidth of 100Mbps. Some hardware technologies limit MTU of certain networks to 128 bytes. Usually, the TCP protocol computes a maximum segment size such that the resulting IP datagrams will match the network MTU. If the end-point does not lie on the same physical network, they can attempt to discover the minimum MTU along the path between them or choose a maximum size of 576 bytes.

Studies have been made to find the link between the packet size and the network delay and throughput. A set of experiments was designed to determine whether delays across the Internet are significantly influenced by packet length [21]. In cases where the usual propagation delays are high relative to the time of transmission for an individual packet, one would expect that delays would not be strongly affected by packet length. However, the data resulting from these experiments shows a strong correlation between delay and length, with the longest packets showing delays two to three times the shortest. Also, other studies showed that limiting datagrams to fit the smallest possible MTU in the Internet makes transfers inefficient when those datagrams pass across a network that can carry large size frames [8]. So, as we can see, in a general Internet environment, choosing a good maximum segment size can be difficult because performance can be poor for either extremely large segment sizes or extremely small sizes. When the segment size is small, the network

utilization remains low. On the other hand, extremely large fragments can also produce low performance if they are sent across small MTU networks where they will be fragmented leading to a larger delay.

In order, to maintain the lowest delay possible with the highest throughput we have to find an optimal segment size that will be keeping the packets in the maximum size possible adaptable to the highest MTU of the network. In this thesis, we adapted this approach. In fact, before sending each packet, we select for each packet the available path having the MTU greater or equal to the size of the packet. If no such path exists then we select the largest path's MTU available and fragment the packet according to this MTU. On the next hop, the host compares the packet maximum size, to which the fragments belong, to the MTU available; if a path with MTU greater or equal to the packet size exists, the gateway will reassemble the fragments of this packet. The fragmentation and reassembly process will be discussed later in this chapter.

3.2.2 Packet Arrival Process

The generation of packets at the source occurs on the Internet according to a Poisson Process. As we know a Poisson Process is a sequence of events randomly spaced in time. In our case, it is the packets generation from source according to λ . (See Figure 3.1)

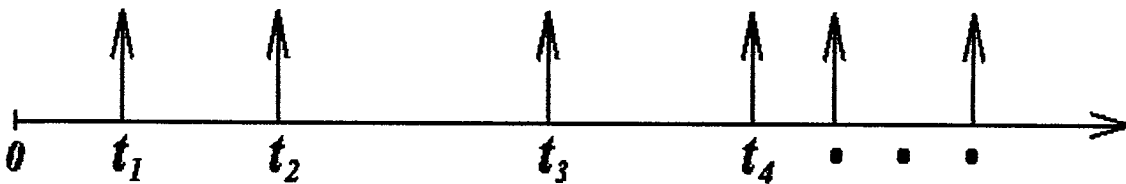


Figure 3.1: Poisson Process [32]

The rate λ of a Poisson process is the average number of packet arrival per unit time.

The Interarrival Times of a Poisson Process is set as follow. First, we pick an arbitrary starting point in time (call it $t = 0$). Let $T_1 =$ the time until the next arrival.

$$P(T_1 > t) = P_0(t) = e^{-\lambda t} \quad \text{Eq. 1.3}$$

So

Let $T_2 =$ the time between the first and second arrival. We can show that

$$P(T_2 > T_1 + t \mid T_1 = s) = e^{-\lambda t} \quad \text{for } s, t > 0 \quad \text{Eq. 1.4}$$

independently of T_1 ! Similarly define T_3 as the time between the second and third arrival; T_4 as the time between the third and fourth arrival; . . . The random variables $T_1, T_2, T_3, \dots, T_n, \dots$ are called the interarrival times of the Poisson process. Those interarrival times are independent of each other and each have an exponential distribution with mean $1/\lambda$. In our thesis, we are adopting this Poisson Process in the generation of packets with arrival rate λ ; having arrival rate λ greater than service rate μ . We start our process at time $T_0 = t = 0$ for the first packet. The second packet generated will have time $T_1 = T_0 + t$ where t is randomly generated and so on for n packets.

3.3 Queueing Management

3.3.1 Queueing Delay

Queueing delay is defined as the time between when a packet first arrives at the node output buffer until when it reaches the head of the output buffer. After processing the packet from source to the router, the packet reaches the queue. At the queue, the packet experiences a queueing delay as it waits to be transmitted onto the link. The queueing delay of a specific packet will depend on the number of other earlier-arriving packets that are queued and waiting for transmission across the link; the delay of a given packet can vary significantly from packet to packet. If the queue is empty and no other packet is currently being transmitted, then our packet's queueing delay is zero. On the other hand, if the traffic is heavy and many other packets are waiting to be transmitted, the queueing delay will be long. In our thesis, the number of packets that an arriving packet might expect to find on arrival is a function of the intensity and nature of the traffic arriving to the queue. The queueing delay is big when the average rate at which the traffic arrives to the queue is greater than the average rate of service in the queue [15]. Let λ denotes the average rate at which packets arrive to the queue (λ is units of packets). Recall that μ is the service rate. It is the rate at which bits are served in the queue. Assume that the queue is very big, so that it can hold essentially an infinite number of bits. The ratio λ/μ , called the traffic intensity, often plays an important role in estimating the extent of the queueing delay. If $\lambda/\mu > 1$, then the average rate at which bits

arrive to the queue exceeds the rate at which the bits can be transmitted from the queue. In this situation, the queue will tend to increase without bound and the queuing delay will approach infinity. Therefore, one of the main rules in traffic engineering is to design your system so that the traffic intensity is no greater than one. When the traffic intensity approaches 1, the average queueing delay increases rapidly. A small percentage of increase in the intensity will result in a much larger percentage wise increase in delay [28]. Therefore, in our system we assumed that the traffic intensity remains < 1 , by using infinite buffers and having service rate greater than arrival rate.

3.3.2 Queuing Theory

The queuing theory is the primary methodology for analyzing network delay. The classical queuing systems used are the M/M/1, M/G/1 and G/M/1. According to Kendall notation $X/Y/m/k$ refers to the following: [32]

X is a symbol representing the inter-arrival process:

- M = Poisson Process (exponential inter-arrival times t)
- D = deterministic (constant t).

Y is a symbol representing the service distribution

- M = exponential
- D = deterministic.
- G = General distribution for inter-arrival time.

m is the number of servers.

k is the number of buffer slots (omitted when $k = \infty$)

The M/M/1 system is based on the theory of Markov Chain. In particular, it is based on a special case of Markov process named the birth-death process in which two consecutive states can only differ by a unit. (See figure 3.2)

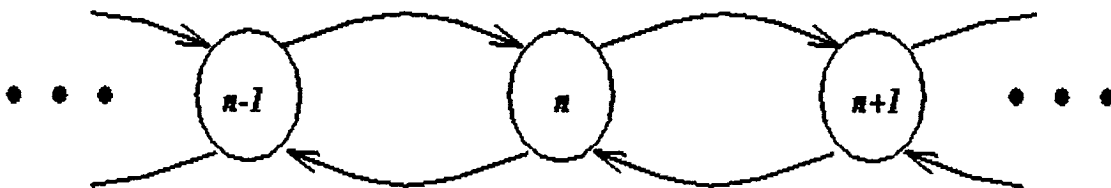


Figure 3.2: Birth-Death Process [32]

The M/M/1 queue is the most basic and important queueing model. It has a Poisson Arrival with rate λ , an exponential service time (with mean $1/\mu$, so μ is the service rate), one server and an infinite length buffer [26].

3.3.3 M/M/1 Queueing Model

The system advanced in this thesis adopts the M/M/1 queueing model. It has a Poisson arrival (Interarrival time = $1/\lambda$), an exponential service time (with mean $1/\mu$, so μ is the service rate) as well as 1 server and an infinite length buffer; hence, packets will never be dropped. In this system the arrival rate is less than the service rate ($\lambda < \mu$). In the simulation, we took $0.1 < \lambda < 0.9$ and $1 < \mu < 3$. We will elaborate deeply on this issue in the Chapter 4 *The Simulation*. This selection infers that $\lambda < \mu$ that means the queueing delay is not diverging to infinity. Furthermore, we do not suffer from the problem of lockout in the M/M/1 system since we have an infinite size of buffer so at no time a packet will find no room in the queue. Regarding the “Full Queues” problem, the M/M/1 system adopted in this thesis as well as the selection of $\lambda < \mu$ avoid the problem of “Full Queues” since the delay does not diverge and hence at no time the queue will be full if the arrival rate is less than the service rate.

3.4 Packet Routing

3.4.1 Transmission Delay

The *transmission delay* is the amount of time required for the router to push out the packet; it is a function of the packet's length and the transmission rate of the link. It has nothing to do with the distance between the two routers. (See Figure 3.3)

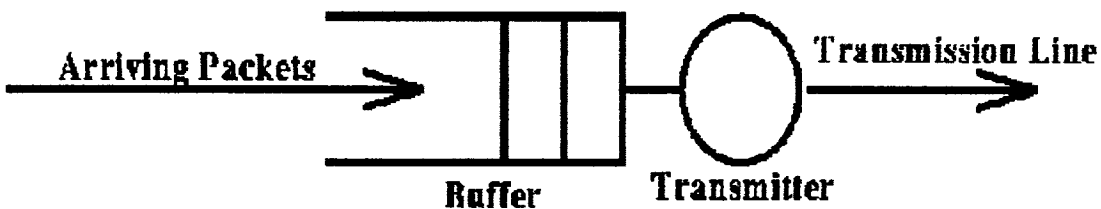


Figure 3.3: Packet Transmission

The transmission delay is calculated as: (see Eq.1.5)

$$\text{Transmission Delay} = \text{Packet Size (bits)} \div \text{Bandwidth (bits/sec)} \quad \text{Eq.1.5}$$

It is sometimes referred to as serialization delay for a single packet. The major issue that affects transmission time is the maximum transmission unit (MTU). For example, it takes almost .25 seconds to transmit 1500 bytes on a 64-kbps link. A queue build-up of ten or twenty 1500-byte packets would take two or three seconds to transmit. Long-haul lines in the Internet today are most frequently full duplex; For example, there are the 56 KBPS, the DS1 lines with 1.544 Mbps, or DS3 lines with 45 Mbps speeds [1].

3.4.2 Path MTU Discovery

To eliminate fragmentation or minimize it, it is desirable to know what is the path MTU along the path from the source to destination. The path MTU is the minimum of the MTUs of each hop in the path. This single path routing technique adopted in the usual Internet sets MTU less or equal to 576 bytes and the first-hop MTU as the Path MTU (PMTU) for any destination that is not connected to the same network or subnet as the source. In many cases, this results in the use of smaller datagrams than necessary, because many paths have a PMTU greater than 576 bytes. A host sending datagrams much smaller than the PMTU allows is wasting Internet resources and probably getting low throughput. Furthermore, current practice does not prevent fragmentation in all cases; since there are some paths whose PMTU are less than 576 bytes.

Another technique is suggested to discover the PMTU based on the Don't Fragment (DF) bit in the IP Header [14]. This method is based on the following: the source host assumes the PMTU is the MTU of the first hop, sends all datagrams on that path with the DF bit set. If any of the datagrams are too large to be forwarded without fragmentation by some router along the path, that router will discard them and return a message. Upon receipt of such a message, the source host reduces its assumed PMTU for the path [20]. Although, this method maximizes the throughput; however, it does not improve latency in the Internet [14].

Studies made by Vern Paxson examined several routing algorithm and found out that:

- For 30% to 55% of the path measured there exist an alternate path with a shorter round-trip time.
- The best alternate path has 50% better latency

- 70% to 80% of the paths have alternates with enhanced bandwidth.

As a result of this study, we proposed in our thesis a new technique consisting on each router should find the shortest path in term of higher bandwidth and higher MTU to forward a packet through it. This technique is elaborated in the next section.

3.4.3 Fastest Path Routing

As we have seen in the previous section, some techniques used to find the PMTU can forward packets along non-optimal routes, or it can spread load unequally, over-utilizing some links while leaving others idle, resulting an increase in delay and sometime decrease in throughout [29]. To find solution to those problems, we advanced in our thesis a method that finds the fastest path. In this method, the selection of the path is done base on the Shortest Path First (SPF) protocols. Those based routing protocols are a class of link-state algorithms that are based on the shortest-path algorithm of Dijkstra. In an SPF based system, each router obtains the entire topology database through a process known as flooding. Flooding insures a reliable transfer of the information. Each router then runs the SPF algorithm on its database to build the IP routing table [30].

We adjusted the IP routing table to include in addition to the path address between two hosts, the path's MTU and bandwidth. Using this technique, each router in our network will be able to consult its IP routing table and select the fastest path available. Using the IP routing table, we split the traffic of packets over several paths between two single pair nodes. We usually refer to this technique as "multi-path routing". The advantage of this method is the possibility of sending different packets over different paths depending on the size of the packet. Not only does this method transfer the packets through a high bandwidth path as they need but also prevents short packets from being delayed behind a queue of long packets. As a result, our method has proven a decrease in the transmission delay and a better throughput. Results will be elaborated later in the *Simulation* chapter.

3.5 Gateways

3.5.1 Internet Gateways

As explained earlier, the term routing refers to the process of choosing a path over which to send packets, and router refers to any computer making such a choice. Routing devices in the Internet have traditionally been called gateways. Local networks are connected together in the Internet model by means of Internet gateways. These gateways provide datagram transport only and normally seek to minimize the state information necessary to sustain this service in the interest of routing flexibility and robustness. (See Figure 3.4)

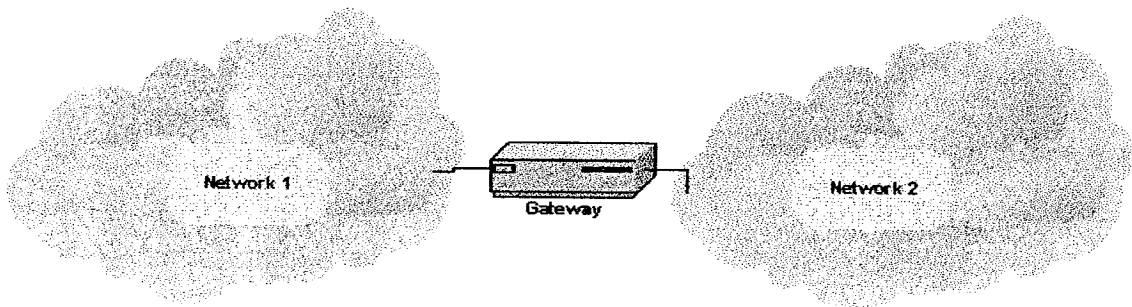


Figure 3.4: Gateways

In the conventional model, the gateway has a physical interface and address on each of the local nets between which it provides forwarding services [22]. Furthermore, The gateways provide Internet protocol translation, in order to establish a connection between networks, and to perform number of other functions that permit computers to communicate independent of hardware differences. The major problem with routing is how the host and gateways in the Internet obtain and maintain their routing information.

Gateways within the Internet are organized hierarchically. Some gateways are used to move information through one particular group of networks under the same administrative authority and control (such an entity is called autonomous system) [27]. Gateways used for information exchange within autonomous systems are called interior gateways, and they use a variety of interior gateway protocols (IGPs) to accomplish this purpose. Gateways that move information between autonomous systems are called Exterior Gateways and uses exterior gateways protocol (EGP) for this purpose [17].

3.5.2 Gateway Functionality

In normal TCP/IP implementation, two methods are used to define the job of the Exterior Gateway. The first method consists of letting the gateway do a transparent fragmentation. In this way, subsequent networks are not aware of the fragmentation. However, this method showed some problems such as each packet should include the fields end-of-packet so that the exit gateway will be able to know when it received the last packet. Furthermore, all packets should be routed in the same path otherwise we will witness packet loss. The second method consists of not allowing the reassembly at the gateway but to send all packets and fragments as original packets and keep the reassembly process to the destination host. This method was adopted as a way to solve the problem occurring from large size packets trying to access a network with MTU very small [30].

In our study, we gave the exterior gateway a new facility, that of resequencing and reassembly of packets. We are implementing our proposed hop-by-hop resequencing and reassembly algorithm at the exterior gateway. Previous studies made on multi-stage networks, showed that providing gateways with resequencing technique minimizes the latency and improves throughput [16]. Our study on the Internet domain showed the same performance improvement. Results are elaborated in the *Simulation* chapter. This gateway will check the MTU of the path through which the packets will be forwarded. Then according to this path the exterior gateway will fragment the packets or reassemble them. The fragmentation and reassembly method is done based on a numbering technique; this technique will be elaborated thoroughly in the next section. This numbering method will allow this gateway or any gateway to reassemble the fragments received according to their original packet number, their fragment number and their sequence number.

3.6 Hop-By-Hop Resequencing and Reassembly Method

In this thesis, we adopted the fact that the Fragmentation/Reassembly process is done at each router or gateway and not at the destination host. Therefore at each gateway of the Internet the Hop-By-Hop Resequencing and Reassembly” algorithm is applied.

3.6.1 Fragmentation

The fragmentation process, in the “Hop-By-Hop Resequencing and Reassembly” algorithm, consists on the following. First, we select the path with the largest MTU and Bandwidth. After that, we check for each datagram in the buffer, if its size is less than or equal the MTU of the path then submit this datagram to the next hop using selected path; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next hop at the end of the path, while the second fragment is fragmented again into new fragments in case it is still too large and so on until all the datagrams can be sent. In the fragmentation procedure, each fragment (except the last which usually less in size) was made the maximum allowable size.

We also suggest the following numbering technique while fragmenting packet. The numbering of packets is done according to a system of numbering giving each packet three numbers:

- First number refers to the original packet.
- Second number refers to the fragment number.
- Third number refers to a sequence number.

Of course, the sequence number depends on the sub-fragment position in the packet. Each fragment will be identified by those three numbers; in addition to its source, destination as well as network level Id. We meant by network level the number of the each phase between two hosts or a host and a gateway. (See Fig 3.5)

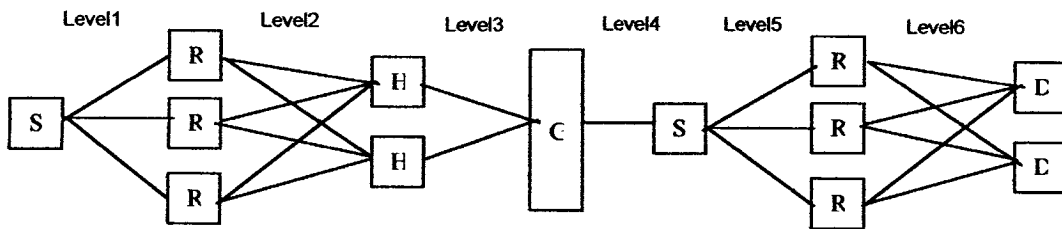


Figure 3.5: Level Partitioning of our simulation network

During fragmentation, the packet number remains the same. Whereas the fragment number will be equal to 1 if packet is not fragmented otherwise it will refer to the number of the fragment in this packet. Moreover, the last number refers to the sequence number of the

fragments. This number will be equal to the sequence number of the sub fragments of the main fragment. In case we had a third level of fragmentation, a resequencing will take place to reassign sequence numbers for the packets. Check flowchart below to visualize this procedure. (See Fig 3.6)

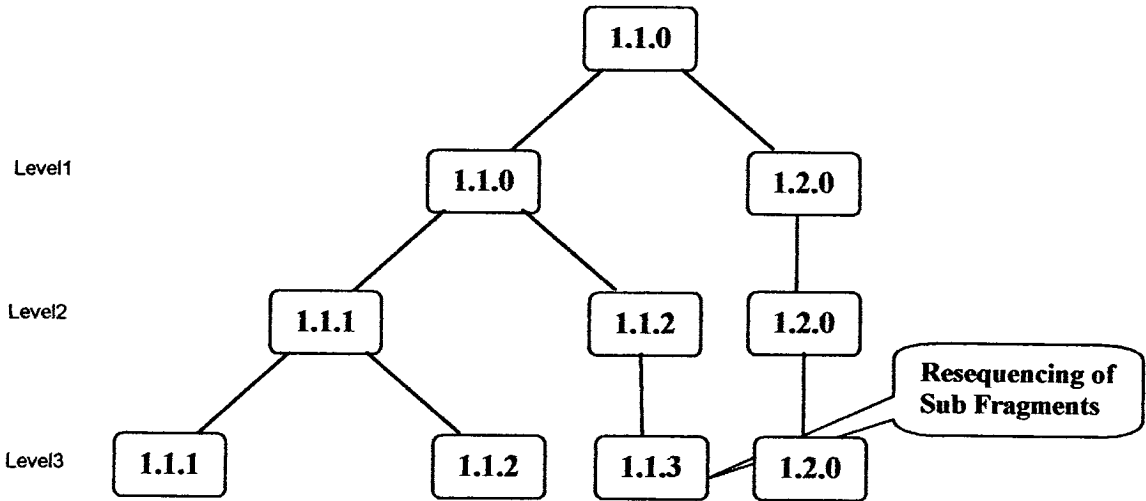


Figure 3.6: Sequence number settings in Hop-By-Hop Resequencing and Reassembly

The fragmentation process consists on assigning to each packet fragmented two identifiers the identifier of the original packet and the number of the fragment. In addition, we identify a sequence number for sub levels fragmentation. Example, a packet is send from source it has the following identifiers 1.1.0 is fragmented at level 1 into 2 fragments: 1.1.0 and 1.2.0. The fragment 1.2.0 remains intact. At level 2, the fragment 1.1.0 is fragmented into 1.1.1 and 1.1.2; At level 3, the fragment 1.1.1 in divided into 1.1.1 and 1.1.2. Since, there is a fragment 1.1.2 then all the fragments that comes after will be reassigned a different sequence number according to the reordering; hence, 1.1.2 of level 2 becomes 1.1.3. At the reassembly stage, the system will reorder the fragments in the packets according to those three identifiers and reassemble the available packets. For example, if packets 1.1.1, 1.1.2 and 1.1.3 reach a certain router where it is possible to reassemble them, they will be regrouped into a packet having the number 1.1.1. Reassembly and resequencing of the fragments will be explained in the section 3.6.3.

3.6.2 Buffer Management

In our thesis, we added to the gateways a resequencing buffer. The incoming packets at any of the routers or gateway are placed in memory for processing. Ideally, a system could make memory allocation efficient by dividing memory into fixed-size buffers, where each buffer is sufficient to hold a packet. In practice, choosing an optimum buffer size is complex for several reasons. First, each network connected to the Internet could have different packet size. Second, IP may need to store large datagrams especially for reassembly. One of the suggested solutions for these problems is to use large buffers. However, in the case of 576-byte we will be having a lot of waste space in memory [9].

In our thesis we introduced resequencing and reassembly buffer based on the dynamic allocation concept. We adopt a large size buffer. This large buffering is no more a constraint now with 512Mbyte memory available. The allocation of coming packets is done as follow: We check the packet original size of the first fragment received and we reserve a space in the buffer equal to this size. When the second fragment arrives, if it belongs to the same packet already allocated it will be set after the first fragment otherwise if it belongs to a new packet not allocated yet then it will be allocated a space in the buffer after the space reserved for the first packet. When all the fragments of the same packet arrive, they are resequenced in the buffer then reassembled depending on the available paths from the source.

(See Figure 3.7)

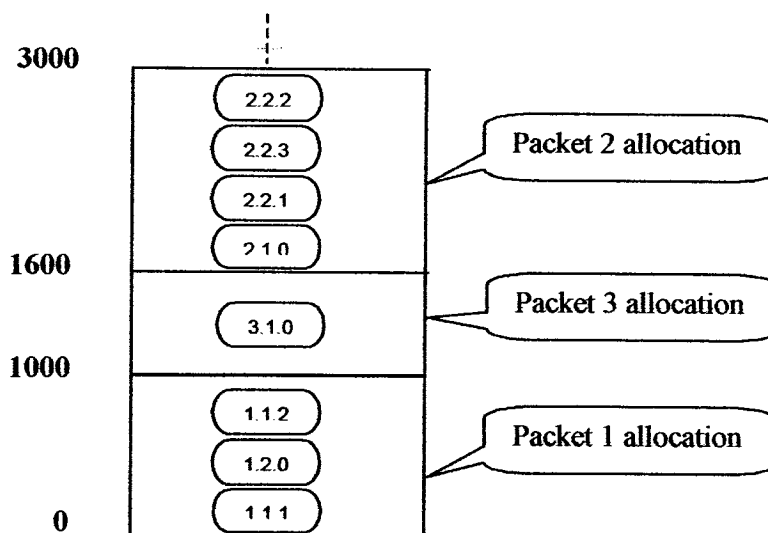


Figure 3.7: Resequencing Buffer Allocation

3.6.3 Resequencing and Reassembly

In our methodology, the reassembly and resequencing occur at each Internet gateway. The algorithm works as follow: at each router, we group fragments in the Resequencing Buffer according to their packet ID using the method explained in the previous section. Then, we start for each packet whose complete fragments are received by resequencing those fragments according to their three numbers references. The router selects from its routing table the path with the highest MTU and largest bandwidth. Then, the router compares the original size of the packet with that of the Path MTU selected. If the Packet Size is less than the MTU the packets fragments will be reassembled to the original size of the packet; after that, the reassembled packet will be routed. Otherwise, we will apply a partial reassembly to the fragments into new fragments of size equal to the MTU of the path. And so on until the last packet, leave the gateway. Note that, the packets will be treated in the buffer according to FIFO method. The packet arrival time plus the resequencing delay will be calculated to specify which packet will be routed first. (See Figure 3.8)

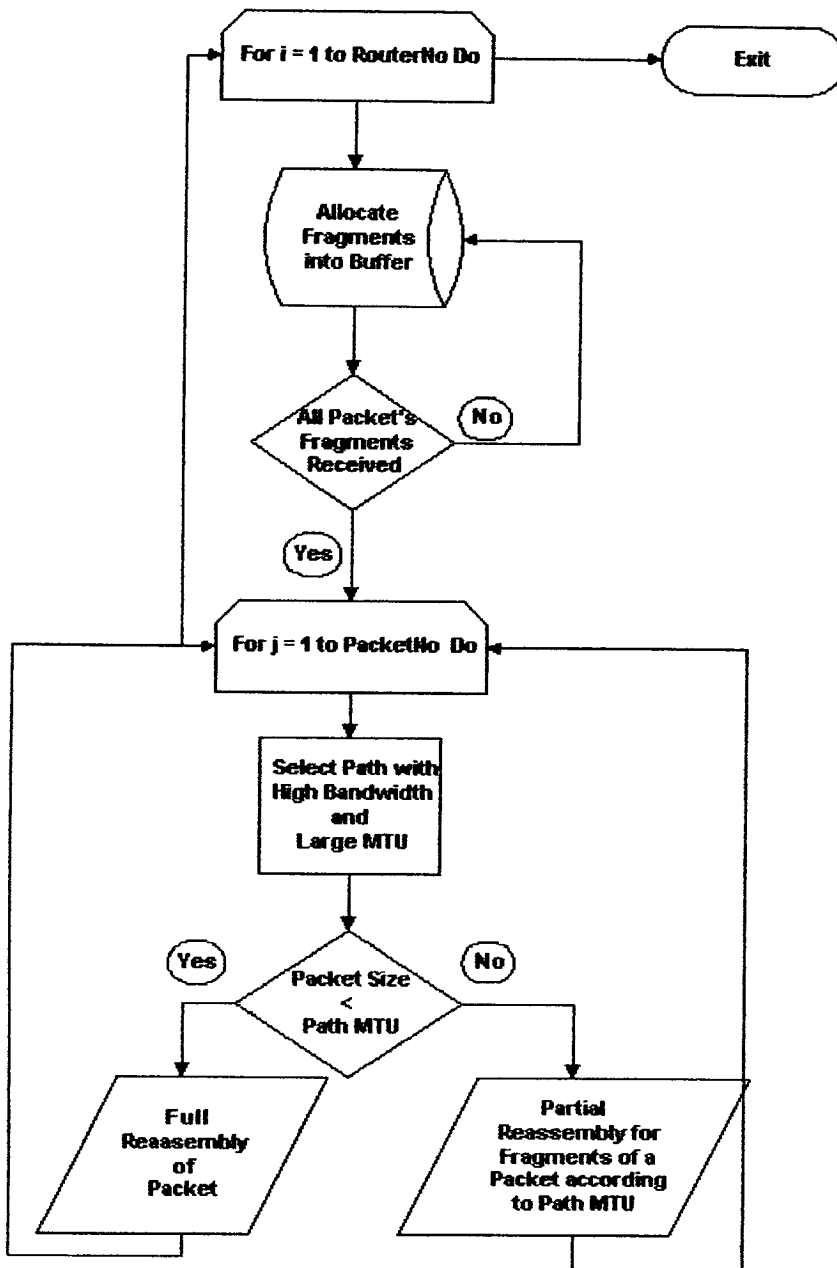


Figure 3.8: Hop-By-Hop Resequencing and Reassembly Algorithm

Chapter 4

Simulation Analysis

4.1 Introduction

We have discussed in the two previous chapters the main types of delays that occurs in TCP/IP network as well as how the problems faced in those delays affected the overall packet delivery delay or communication delay. Furthermore, we proposed solutions and suggestions to minimize the delay at each of those levels leading to the minimization of the whole communication delay. In this chapter, we will prove the ideas and techniques advanced using a simulation program. In this simulation, we will compare the results of the end-user resequencing technique normally used in TCP/IP networks with the new hop-by-hop resequencing and reassembly technique. The results that will be displayed in this chapter show clearly that our techniques gives better throughput and lower delay.

Our simulation is generated using Visual Basic software. We will start by describing the network topology we are studying in section 4.2. Then we describe the packet generation and size handling in the section 4.3. In the section 4.4, we elaborate on the transmission delay and the link selection. Finally, in section 4.5, we describe the queueing technique and its results.

4.2 Network Topology

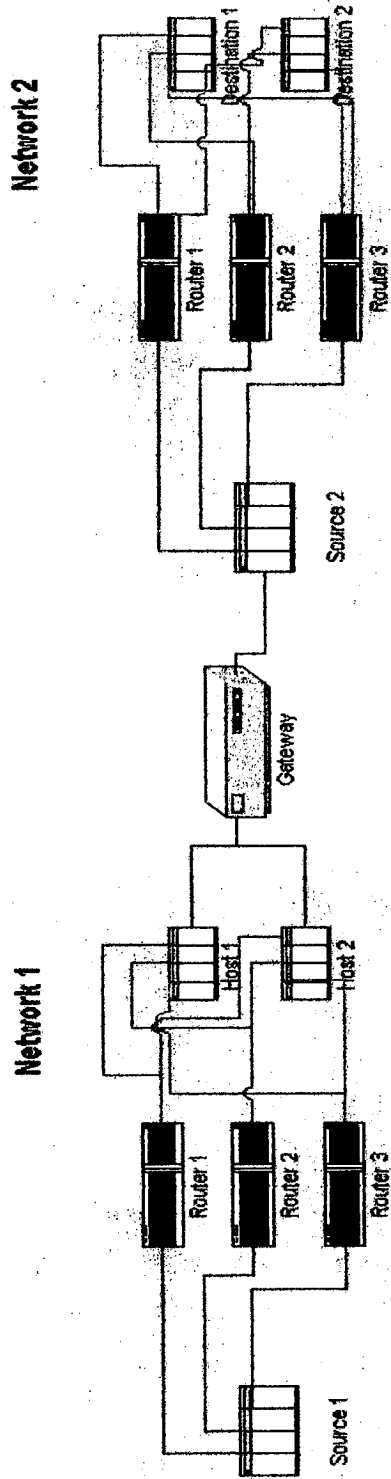
This topology consisted of two networks linked together by an exterior gateway. Each of those two networks contains a source host linked by three different links to three routers having the same service rate (μ). Those links have different MTUs and Bandwidths. Than, each of those routers is connect to two hosts; in the second network those hosts consisted the destination hosts. The 32kbps gateway links both networks. This gateway is responsible for fragmenting and reassembling packets before transmitting them to the next network. (See figure 4.1) In this network, we are assuming that no packet-drop will occur and that all packets will reach their destination. Furthermore, the source host is in the first network whereas the destination host is in the second network. Therefore, the packets will

have to travel through both networks facing different level of disordering and delay before reaching its destination. On the other hand, since no packets will be dropped we are taking the arrival rate less than the service rate such that the traffic intensity is: (Eq.1.6)

$$\lambda \div \mu < 1 \quad (\text{having } \lambda < \mu) \quad \text{Eq 1.6}$$

The propagation delay is neglected and the buffer size is infinite. Below is a figure representing the network topology we are studying. (See Fig 4.1)

Network Topology



4.3 Packet Generation

First we generate a random number of packets with sizes varying from 600 bytes to 3000 bytes. The source at the first network generates packets at different size in a random way for a range between 600 and 3000 bytes. The generation of those packets occurs according to a Poisson process with an arrival rate λ varying between 0.1 and 0.9. In the end-user method, the first source will start by fragmenting the packets into 576-bytes fragments. All the fragments will be of the same size except for the last packet. Then the source will select a random path out of the three paths to which it is connected and forward the packets. In our methodology, the source selects the path with the largest bandwidth and checks its MTU. If the MTU of this path is greater than that of the packet will be forwarded without fragmentation. Otherwise, if the highest bandwidth path has an MTU smaller than the packet size, this packet will be fragmented into packets having the paths' MTU. This method reduces the number of fragments that will be routed through the network.

At the source of the second network, the packets will arrive with distributed time after traversing the first network. In the end-user resequencing technique, the source 2 will select a random path for those fragments and start sending each packet according to the first in first out concept. Whereas, in our hop-by-hop resequencing, the source 2 will check for the highest bandwidth path and compare its MTU with that of the packets arriving if some of the packets' fragments could be reassembled it will do so else if other packets have higher size than the path's MTU it will fragmented them according to the MTU of the path.

Our studies for the effect of the variation of arrival rate on the mean delay showed that as the arrival rate increase the mean delay increases with it. This assumption was studied for a fixed service rate $\mu = 1.5$. However, our simulation showed a lower mean delay with our method than that of end-user resequencing method. The figure 4.2 shows the results.

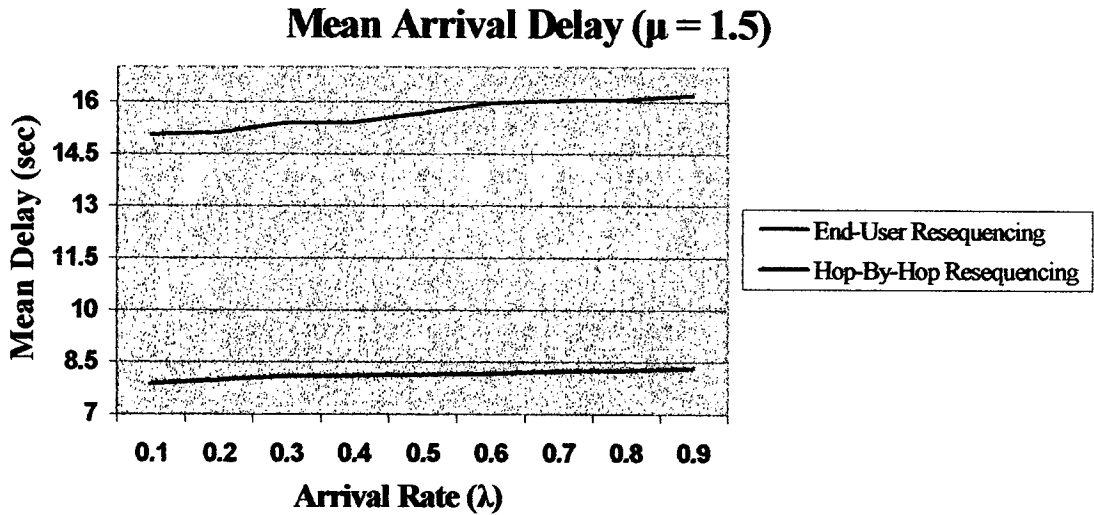


Figure 4.2: Mean Arrival Delay for $\mu = 1.5$

Furthermore, our methodology showed a better throughput for a variation of arrival rate λ . Since, the selection of large packets for small MTU paths lead the inefficient use of network resources and drop the throughput as well as the use of small packets in large MTU paths will lead to the same situation. Figure 4.3 shows the optimization.

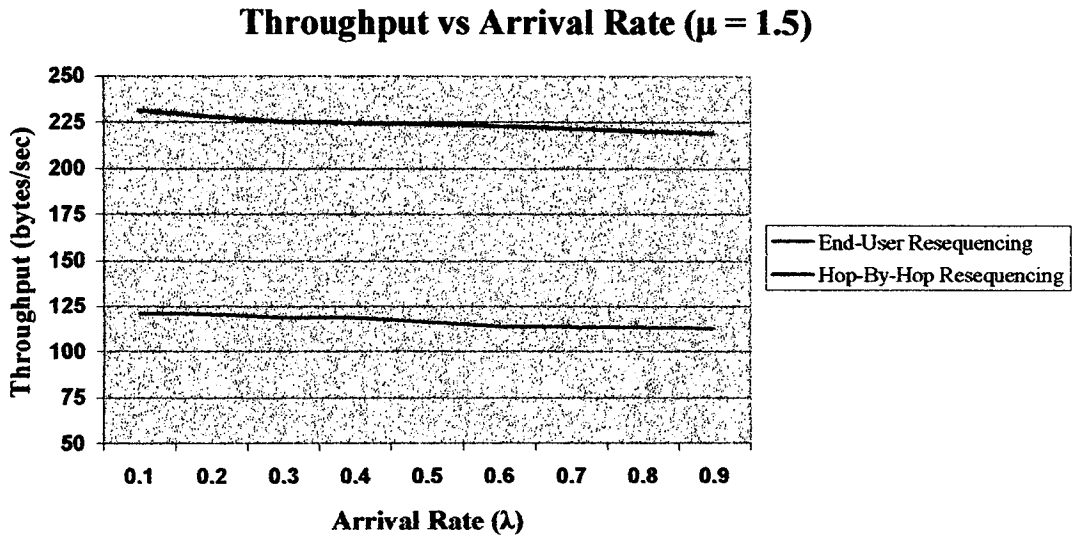


Figure 4.3: Throughput for different Arrival Rates λ ($\mu = 1.5$)

4.4 Queueing Delay

These networks, queueing delay can be seen in the routers at each of the networks as well as in the gateway connecting both networks. As we have mentioned previously each network contains three routers. The service rate adopted in those routers is identical for all of them; it is equal to $\mu=1.5$. Whereas, the service rate at the gateway is fixed to $\mu = 3$. The service time in the routers is calculated as:

$$\text{Service Time} = (1 \div \mu) \times \log(1 - \text{rand}) \quad \text{Eq 1.7}$$

As we see, the service time is generated according to an exponential distribution process. Whereas the queueing delay for packet i is calculated as follow: (in Eq 1.8)

$$\text{QDelay}[i] = (\text{ArvlTime}[i-1] + \text{SrvcTime}[i-1] + \text{QDelay}[i-1]) - (\text{ArvlTime}[i]) \quad \text{Eq 1.8}$$

We are adapting the M/M/1 model. This is the easiest Markovian model. It is considered as the single server with infinite capacity. We are assuming that the packet arrival occurs in a Poisson process and the service time is exponentially distributed with service rate μ . The assumption that customers arrival and service is done separately is known as birth-death structure. In our simulation, we are adopting the condition: (see Eq 1.9)

$$\rho = \lambda / \mu < 1 \Leftrightarrow \lambda < \mu \quad \text{Eq 1.9}$$

The ρ calculated in Eq 1.9 consists the traffic intensity. We are assuming that $\rho < 1$ which means that the number of packets arriving is less than the number of packets leaving. Therefore, the arrival rate λ is between 0.1 and 0.9 whereas the service rate μ is between 1 and 3.

Concerning the routing algorithm, the routers adopt the SPF (shortest path first) algorithm to find the fastest path in our method whereas in the normal TCP/IP protocol the router selects a path randomly. When we talk about shortest path, we refer to the Dijkstra's algorithm. However, here since we are ignoring the processing delay we are not interested in searching the shortest distance rather than the fastest path. The method consists on the following: each router will search for the path with the largest bandwidth. When this path is

found the packet size is checked with the size of this path then it will be either fragmented accordingly to the size of the path, or reassembled if path larger than packet fragments when reassembled; otherwise the packet will be routed as one piece. If two paths have the same bandwidth then we select one of them randomly. We repeat this step at each router until the packets reaches their destination. The selection of the path is done by referring to a routing table in which each router set all the paths available from its location to all other location of the next hop. This method is adopted in the hop-by-hop resequencing method whereas in the usual TCP/IP protocol, the router will select one path randomly and route the packet of 576-byte size. However, if the MTU of this path is small the fragment has to be divided into several fragments. This will increase the delay and performance of the Internet will decrease.

According to our assumptions, the simulation showed that our method of routing outcome the end-user resequencing method. In fact, when we simulate for different λ the variation of the service rate, we get a mean delay for the end-user resequencing method bigger than the mean delay of the hop-by-hop resequencing. We can check figure 4.4

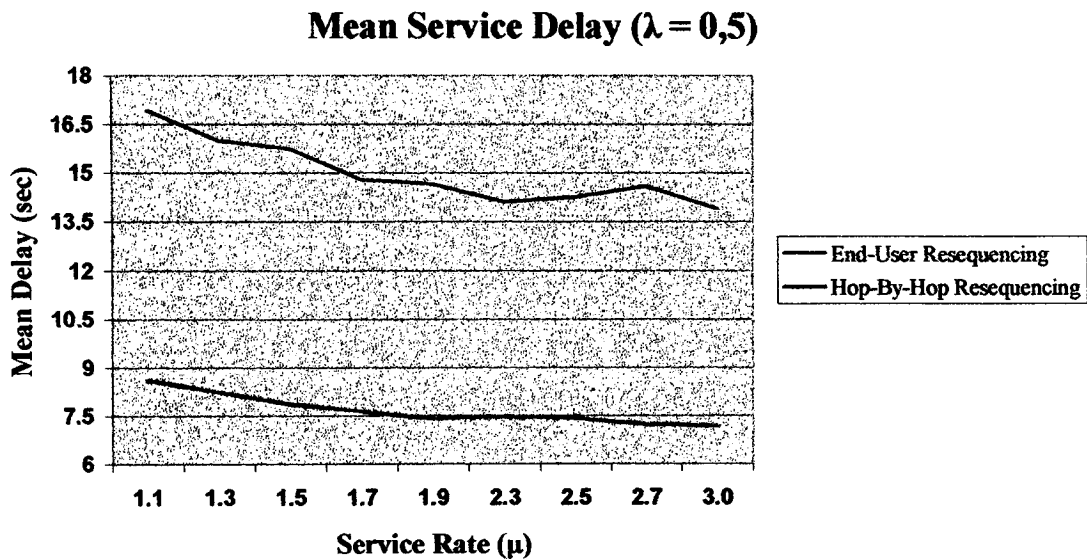


Figure 4.4: Mean Service Delay for $\lambda = 0.5$

4.5 Transmission Delay

As we have seen in the previous section, the routers adopt the SPF (shortest path first) algorithm to find the fastest path for the hop-by-hop resequencing method whereas in the normal TCP/IP protocol the router selects a path randomly. When the packet is routed in a certain path, the transmission time is calculated according to Eq 2.0

$$\text{TransTime} = \text{FragmentSize} / \text{Bandwidth (in sec)} \quad \text{Eq 2.0}$$

The transmission delay is calculated according to Eq 2.1

$$\text{TransDelay}[i] = (\text{InitTime}[i-1] + \text{TransTime}[i-1] + \text{TransDelay}[i-1]) - \text{InitTime}[j] \quad \text{Eq 2.1}$$

The “InitTime” consists the start the first time at which the fragment start its routing in the link, whereas the “Transtime” is the time taken by the packet to path all along the link. The delay is calculated as being the total time of the previous packet (–) the time the current packet for the first time enters the link. The transmission delay varies according to the MTU and bandwidth of the path. The selection of the path in our simulation is done according to a routing table: (see Figure 4.5)

Network	Level	Path	From Node	To Node	MTU (bytes)	Bandwidth (bytes/sec)
1	1	1	S1	R1	600	250
1	1	2	S1	R2	1500	800
1	1	3	S1	R3	2500	4250
1	2	1	R1	H1	300	750
1	2	1	R2	H1	2000	4000
1	2	1	R3	H1	750	500
1	2	2	R1	H2	850	875
1	2	2	R2	H2	950	900
1	2	2	R3	H2	2500	4250
1	3	1	H1	G1	3000	5000
1	3	1	H2	G1	3500	5250
2	4	1	G1	S2	3750	6000
2	5	1	S2	R1	3200	900
2	5	2	S2	R2	3500	5250
2	5	3	S2	R3	3000	750
2	6	1	R1	D1	2600	900
2	6	1	R2	D1	3400	5150
2	6	1	R3	D1	2300	850
2	6	2	R1	D2	1000	1300

Network	Level	Path	From Node	To Node	MTU (bytes)	Bandwidth (bytes/sec)
2	6	2	R2	D2	3500	5250
2	6	2	R3	D2	850	900

Figure 4.5: Routing Table

The application of our method on traffic of 1000 packets showed better results for the throughput comparing to the mean delay. (See Figure 4.6)

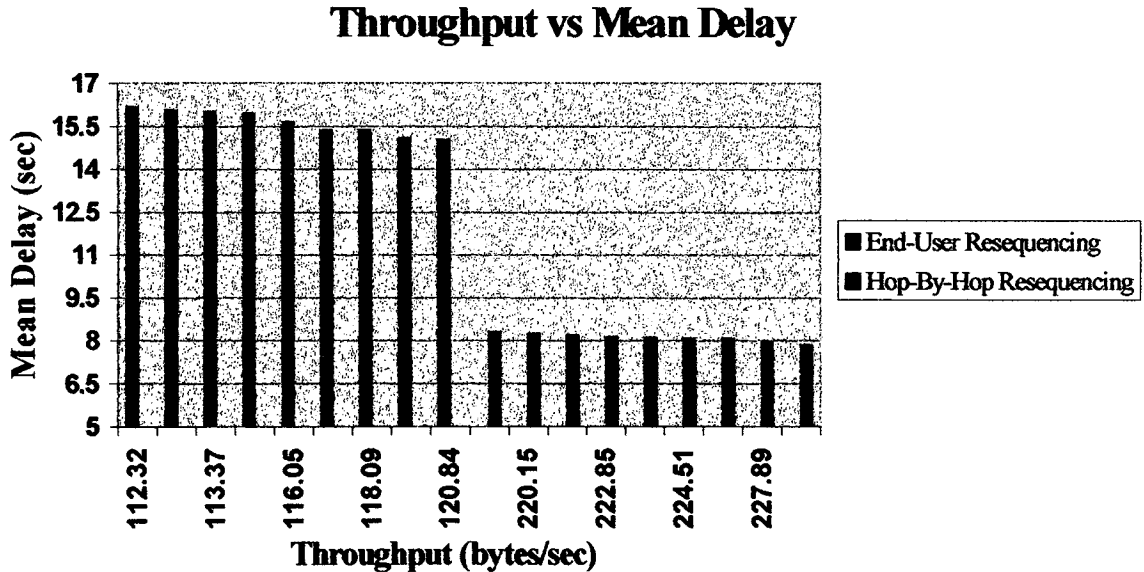


Figure 4.6: Throughput vs Mean Delay for $\lambda = 0.5$ and $\mu = 1.5$

4.6 Fragments Tracing

In this section we are tracing the fragmentation and reassembly of the fragments belonging to a packet of size = 3000 bytes using both End-user reassembly method (See Figure 4.7) and Hop-By-Hop Resequencing and Reassembly method (See Figure 4.8)

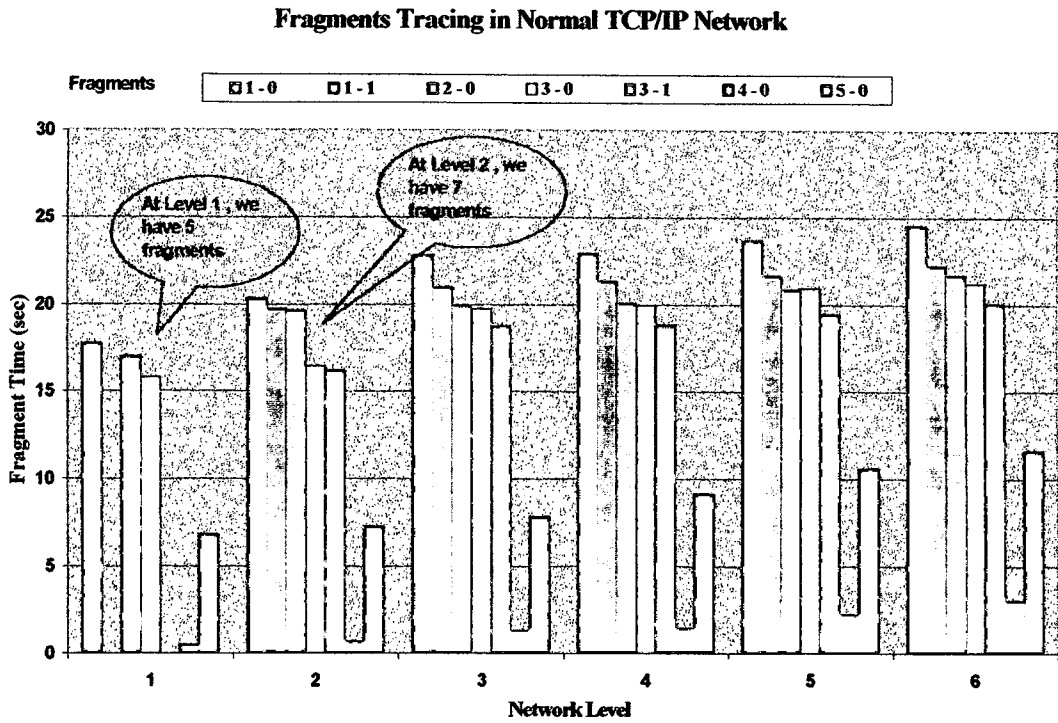


Figure 4.7: Fragments Tracing in Normal TCP/IP Networks

In our thesis, fragments tracing is as follow:

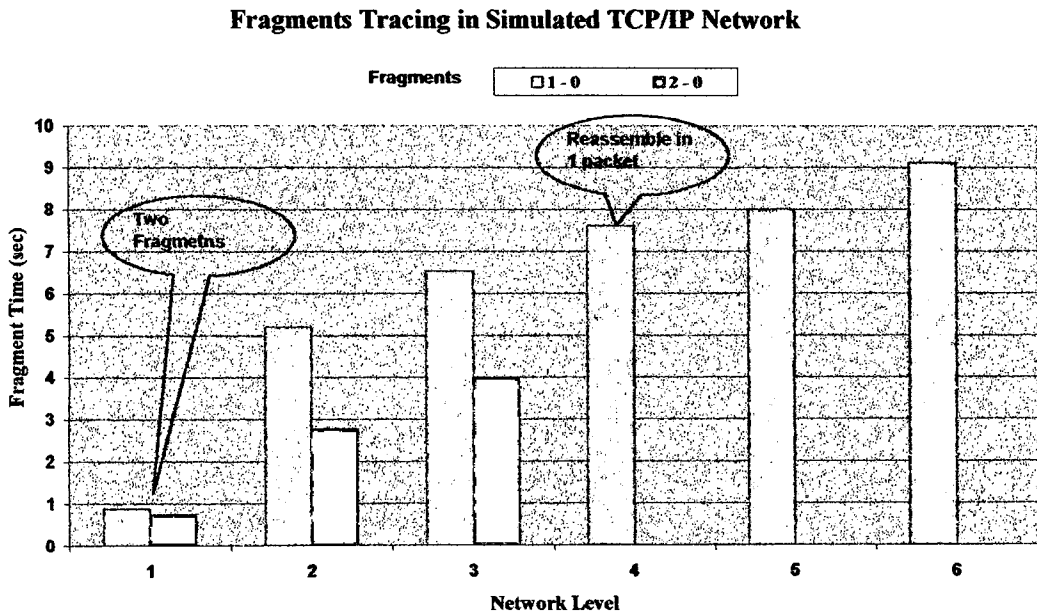


Figure 4.8: Fragments tracing in our simulated TCP/IP Networks

Chapter 5

Conclusion

The Internet was built by a small community of researchers. In that environment, it was reasonable to assume that end hosts would cooperate in the management of network resources. As the Internet has evolved from a research project into a popular consumer technology, this assumption has lost some of its validity. Everybody has experienced delays in accessing their favorite web servers, and if you use TELNET or rlogin to log in on a remote server across a long distance, you've almost certainly experienced brief or lengthy interruptions in service in at least one of the wide area Internet carriers or in one of their interconnection points [25].

However, many studies have been made to optimize this delay on the Internet and they lead to remarkable improvement. Some of them, worked on improving the Queuing Delay by enhancing queueing techniques and algorithms. Others, worked on the improvement of Transmission Delay by elaborating different algorithms to find the shortest path. Some other studies, introduces new techniques in packet fragmentation depending on the network paths and their bandwidth.

In our thesis, we worked on improving the performance of the network by suggesting techniques that minimize each of the delays that a packet might face during its delivery from host to destination such as: queueing delay, transmission delay and resequencing delay. We introduces suggestions to avoid fragmentation as much as possible by selecting the fastest path with the best MTU. Furthermore, the introduction of resequencing and reassembly procedure at the routers level showed an improvement in the over all delay. Finally, the resequencing technique elaborated showed a good management for packets delivery. With the latest developments in hardware technology, such as the production of 1Gbytes memory and 1Ghz processor, it is no longer difficult to apply or add new function on routing and queueing. We think we can benefit from these technique to adjust the current methodologies in the aim of improving the performance of the Internet.

BIBLIOGRAPHY

- [1] Baker, F. “*Requirements For IP Version 4 Routers*”, RFC 1812, Network Working Group Editor, Cisco Systems, 1995.
- [2] Bolot, Jean-Chrysostome “*End-to-End Packet Delay and Loss Behavior in the Internet*”, INRIA, France, 1993.
- [3] Braden, B. “*Recommendations on Queue Management and Congestion Avoidance in the Internet*”, RFC 2309, Network Working Group Editor, UCLA, 1998.
- [4] Braden, R.; Postel, J. “*Requirements for Internet Gateways*”, RFC 1009, Network Working Group, 1987.
- [5] “*Routing Basics*”, CISCO System Incorporation, 1995.
- [6] Chinoy, Bilal “*Dynamics of Internet Routing Information*”, bac@sdsc.edu, 1993.
- [7] Christiansen, Mikkel; Jeffoy, Kevin; Ott, David; Smith, Donelson F. “*Turning RED for Web Traffic*”, University of North Carolina, Department of Computer Science, 2000.
- [8] Comer, Douglas E. “*Internetworking with TCP/IP: Principles, Protocols and Architecture*”, Volume I, 3rd Edition, Prentice Hall, 1995.
- [9] Comer, Douglas; Stevens, David “*Internetworking with TCP/IP: Design, Implementation and Internals*”, Volume II, 2nd Edition, Prentice Hall, 1994.
- [10] De Cnodder, Stefaan; Elloumi, Omar; Pauwels, Kenny “*Effect of different packet sizes on RED performance*”, Traffic and Routing Technologies project, Alcatel Corporate Research Center.
- [11] Firoiu, Victor; Borden, Marty “*A Study of Active Queue Management for Congestion Control*”, Nortel Networks.
- [12] Jacobson, V. “*TCP Extensions for Long-Delay Paths*”, RFC 1072, Network Working Group Editor, 1988.
- [13] Jacobson, V. “*TCP Extensions for High Performance*”, RFC 1323, Network Working group, 1992.
- [14] Kamouh, Walid “*Hop-By-Hop Flow Control with packet aggregation in TCP/IP Networks*”, Thesis Study, Notre Dame University, 2000
- [15] Kleinrock, L. “*Queuing Systems, Volume 2*”, John Wiley, New York, 1976.

- [16] Maalouf, H.W.; "*Optimization of the communication network performance of distributed systems with resequencing constraints*", Communication and Signal Processing Group, Department of Electrical and Electronic Engineering, Imperial College of Science Technology and Medicine, England, 1998.
- [17] Maalouf, H. "*Internetworking and Internet Protocols*", Lecture Notes, CSC626, Notre Dame University, 1998.
- [18] Mills, D.L. "*Internet Delay Experiments*", RFC 889, Network Working Group, 1983.
- [19] Mills, D.L. "*Exterior Gateway Protocol Formal Specification*", RFC 904, Network Working Group, 1984.
- [20] Mogul, J. "*Path MTU Discovery*", RFC 1191, Network Working Group Editor, 1990.
- [21] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, FACC, January 1984.
- [22] Network Technical Advisory Group, "*Requirements for Internet Gateways*", RFC985, Network Working Group, 1986.
- [23] Postel, J. "*DARPA Internet Program: Protocol Specification*", RFC 791, Information Science Institute, 1981.
- [24] Presti, Lo F.; Duffield, N.G.; Horowitz, J.; Towsley, D. "*Multicast-Based Inference of Network-Internal Delay Distributions*", UMASS CMPSCI Technical Report 99-55, 1999.
- [25] Quaterman, John "*Imminent Death of the Internet?*", Matrix News, 1996.
- [26] Ravindran, A; Philips, D.; Solberg, J. "*Operations Research: Principles and Practice*", 2nd edition, Wiley Publication, 1987.
- [27] Rosen, Eric C. "*Exterior Gateway Protocol (EGP)*", RFC 827, Bolt Beranek and Newman Inc., 1982.
- [28] Ross, K.W. "*Multiservice Loss Models for Broadband Telecommunication Networks*", Springer, Berlin, 1995.
- [29] Savage, Stefan; Anderson, Thomas; Aggarwal, Amit; Becker, David; Cardwell, Neal; Collins, Andy; Hoffman, Eric; Snell, John; Vahdat, Amin; Voelker, Geoff; Zahorjan, John "*DETOUR: Informed Internet Routing And Transport*", University of Washington, 1999.
- [30] Tanenbaum, Andrew S. "*Computer Networks*", 2nd Edition, Prentice Hall of India, 1996.
- [31] Turgul, Dayar "*Resequencing of messages in a Queueing System with heterodenuous Servers under various scheduling policies*", North Carolina State University, 1991.

- [32] Vastola, Kenneth S. "*Performance Modeling and Analysis of Computer communication Networks*", Rensselaer Polytechnic Institute, 1996.

Appendix A

Simulation Code

1 - Normal TCP/IP code:

```
Option Compare Database
Dim OptId As Integer
Dim PackNo As Integer
Dim Network As Integer
Dim Level As Integer
Dim ProcessId As Integer
Dim ObjId As String
Dim r As Single
```

```
Function DivPacket576(PackId As Long, PoisTime As Single, Procid As Integer, LevelId As Integer, FragNo, psize As Double, Old As String)
```

```
On Error GoTo DivPacket_err
Dim frag As String
Dim frec As DAO.Recordset
Dim db As Database
Dim NbrOfPath As Integer
Dim i
```

```
Set db = DBEngine(0)(0)
```

```
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & Old & "and LevelId = " & LevelId & "")
```

```
frag = ""
```

```
frag = frag & "select tmpSiFragSources.* from tmpSiFragSources; "
```

```
Set frec = db.OpenRecordset(frag, dbOpenDynaset)
```

```
For i = 1 To FragNo
```

```
    frec.AddNew
```

```
    frec!ProcessId = Procid
```

```
    frec!PacketId = PackId
```

```
    frec!FragID = i
```

```
    frec!LevelId = LevelId
```

```
    frec!SeqNum = 0
```

```
    If psize - 576 >= 0 Then
```

```
        frec!fragSize = 576
```

```
        psize = psize - 576
```

```
    Else
```

```
        If psize < 48 Then
```

```
            t = 48 - psize
```

```
            frec!LastPSize = psize
```

```
            frec!fragSize = psize + t
```

```
        Else
```

```
            frec!fragSize = psize
```

```
        End If
```

```
    End If
```

```

frec!StartTime = 0
frec!ObjectId = OId
frec!ObjectTime = PoisTime
frec!ObjectDelay = 0
frec!ObjectITime = RoundNum(frec!ObjectTime + frec!ObjectDelay, 4)
frec!pathid = Int((NbrOfPath + 1 - 1) * Rnd + 1)
frec.Update
Next i
Exit Function

```

```

DivPacket_err:
MsgBox "DivPacket576"
MsgBox (Error(Err))
Exit Function
End Function

```

Function PPath(TableName As String, Net As Integer, Procid As Integer, LevelId As Integer, OId As String)

```

On Error GoTo PPath_err
Dim PrevTime
Dim PrevDelay
Dim db As Database
Dim sql As String
Dim rec As DAO.Recordset
Dim NbrOfPath As Integer
Dim PathMTU As Long
Dim PathBand As Long
Dim SourceTo As String
Dim initTime As Single
Dim i, j

```

```

Set db = DBEngine(0)(0)
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource=" & OId & "" and
LevelId = " & LevelId & """)
For i = 1 To NbrOfPath
    PathMTU = DLookup("MTU", "tblSiPaths", "PathId=" & i & "" and LevelId=" & LevelId & "" And
FromSource=" & OId & """)
    PathBand = DLookup("Bandwidth", "tblSiPaths", "PathId=" & i & "" and LevelId=" & LevelId & ""
And FromSource=" & OId & """)
    SourceTo = DLookup("ToSource", "tblSiPaths", "PathId=" & i & "" and LevelId=" & LevelId & ""
And FromSource=" & OId & """)

    sql = ""
    sql = sql & "SELECT " & TableName & ".ProcessId, " & TableName & ".LevelId, "
    sql = sql & TableName & ".PathId, " & TableName & ".ObjectId, " & TableName &
".ObjectITime, "
    sql = sql & TableName & ".PacketID, " & TableName & ".FragId, " & TableName & ".SeqNum, "
    sql = sql & TableName & ".FragSize, " & TableName & ".LastPSize, " & TableName &
".StartTime, " & TableName & ".ObjectTime, "
    sql = sql & TableName & ".ObjectDelay, " & TableName & ".PathTime, " & TableName &
".PathDelay, "
    sql = sql & TableName & ".PathITime, " & TableName & ".LevelTime, " & TableName &
".Destin "
    sql = sql & "FROM " & TableName & " WHERE ((((" & TableName & ".ProcessId)=" & Procid &

```

```

        ") AND "
sql = sql & "(" & TableName & ".LevelId)=" & LevelId & ") AND (" & TableName &
        ".PathId)=" & i & ") AND "
sql = sql & "(" & TableName & ".ObjectID)=" & Old & ") ORDER BY " & TableName &
        ".ProcessId, "
sql = sql & TableName & ".LevelId, " & TableName & ".PathId, " & TableName & ".ObjectID, " &
        TableName & ".ObjectTTime; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    initTime = rec!ObjectTTime
    PrevTime = rec!ObjectTTime
    rec.MoveLast
    For j = 1 To rec.RecordCount
        If j = 1 Then
            rec.MoveFirst
            rec.Edit
        Else
            rec.Edit
            initTime = rec!ObjectTTime
        End If
        r = Rnd
        While r = 0
            r = Rnd
        Wend
        rec!pathTime = RoundNum((-1) * (rec!fragSize / PathBand) * Log(1 - r), 4)

        If (PrevTime - initTime) <= 0 Then
            rec!PathDelay = 0
        Else
            rec!PathDelay = RoundNum(PrevTime - initTime, 4)
        End If
        rec!pathTTime = rec!pathTime + rec!PathDelay
        rec!LevelTime = RoundNum(rec!ObjectTTime + rec!pathTime + rec!PathDelay, 4)
        PrevTime = rec!ObjectTTime + rec!pathTTime
        rec!Destin = SourceTo
        rec.Update
        rec.MoveNext
    Next j
End If
Next i

Exit Function

PPath_err:
result = MsgBox("Paths")
MsgBox (Error(Err))
Exit Function
End Function

Function PDestination()
On Error GoTo PDestination_Err

DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiMeanDelayBylamda"

```

```
DoCmd.OpenQuery "qrySiMeanDelayByMiou"
DoCmd.SetWarnings True
```

Exit Function

```
PDestination_Err:
MsgBox (Error(Err))
Exit Function
End Function
```

Function PGateway(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)

On Error GoTo Pgateway_err

Dim Gate As Long

Dim gateway As String

Dim k

```
Gate = DLookup("Gateway", "tblSiOptions", "OptionId = " & OptId & "")
gateway = "G1"
GoSub AddGatewayFrag
result = PPath("tmpSiFragGateways", Net, PId, LvId, gateway)
DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppGateData"
DoCmd.OpenQuery "qrySiDelGateData"
DoCmd.SetWarnings True
```

Exit Function

AddGatewayFrag:

Set db = DBEngine(0)(0)

Set rec = db.OpenRecordset("tmpSiFragGateways", dbOpenDynaset)

k = 0

```
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & gateway & ""
and LevelId = " & LvId & """)
```

sql = ""

```
sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
```

```
sql = sql & "tblSiFragments.FragId, tblSiFragments.LevelId, tblSiFragments.SeqNum, "
```

```
sql = sql & "tblSiFragments.FragSize, tblSiFragments.LastPSize, tblSiFragments.StartTime, "
tblSiFragments.ObjectId, "
```

```
sql = sql & "tblSiFragments.ObjectTime, tblSiFragments.ObjectDelay, "
tblSiFragments.ObjectTTime, "
```

```
sql = sql & "tblSiFragments.PathId, tblSiFragments.PathTime, tblSiFragments.PathDelay, "
```

```
sql = sql & "tblSiFragments.PathTTime, tblSiFragments.Destin, tblSiFragments.LevelTime "
```

```
sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.ProcessId)=" & PId & ") AND "
```

```
sql = sql & "(((tblSiFragments.LevelId)=" & (LvId - 1) & ") AND ((tblSiFragments.Destin)=" &
gateway & "")) "
```

```
sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.LevelId, "
tblSiFragments.Destin, tblSiFragments.LevelTime; "
```

Set rec = db.OpenRecordset(sql, dbOpenDynaset)

If Not rec.EOF Then

k = 1

rec.MoveFirst

initTime = rec!LevelTime

PrevSTime = rec!LevelTime

PrevSDelay = 0

```

While Not rec.EOF
  If k <> 1 Then
    rec.Edit
    initTime = rec!LevelTime
  End If

  rrec.AddNew
  rrec!ProcessId = PId
  rrec!PacketId = rec!PacketId
  rrec!FragID = rec!FragID
  rrec!LevelId = LvId
  rrec!SeqNum = rec!SeqNum
  rrec!fragSize = rec!fragSize
  rrec!LastPSize = rec!LastPSize
  rrec!StartTime = initTime
  rrec!ObjectId = gateway
  r = Rnd
  While r = 0
    r = Rnd
  Wend
  ServTime = RoundNum((-1) * (1 / 3) * Log(1 - r), 4)
  rrec!ObjectTime = ServTime
  If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
    rrec!ObjectDelay = 0
  Else
    rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
  End If
  rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
  rrec!pathid = 1
  PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
  PrevSDelay = rrec!ObjectDelay
  rrec.Update
  k = 0
  rec.MoveNext
Wend
End If
rec.Close
rec.Close
Return

Pgateway_err:
result = MsgBox("Gateway")
MsgBox (Error(Err))
Exit Function
End Function

Function PHosts(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)
On Error GoTo PHosts_Error
Dim HostNo As Integer
Dim db As Database
Dim rrec As DAO.Recordset
Dim tmpRec As DAO.Recordset
Dim rec As DAO.Recordset
Dim sql As String
Dim Host As String

```

Dim i, j, k

```

GoSub HostDelay
For j = 1 To 2
  Host = "H" & j
  result = PPath("tmpSiFragHosts", Net, PId, LvId, Host)
Next j
DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppHostData"
DoCmd.OpenQuery "qrySiDelHostData"
DoCmd.SetWarnings True

```

Exit Function

HostDelay:

```

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragHosts", dbOpenDynaset)

For i = 1 To 2
  Host = "H" & i
  NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & Host & " and
    LevelId = " & LvId & ")

  sql = ""
  sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
  sql = sql & "tblSiFragments.FragId, tblSiFragments.LevelId, tblSiFragments.SeqNum, "
  sql = sql & "tblSiFragments.FragSize, tblSiFragments.LastPSize, tblSiFragments.StartTime, "
  sql = sql & "tblSiFragments.ObjectId, "
  sql = sql & "tblSiFragments.ObjectTime, tblSiFragments.ObjectDelay, "
  sql = sql & "tblSiFragments.ObjectTTime, "
  sql = sql & "tblSiFragments.PathId, tblSiFragments.PathTime, tblSiFragments.PathDelay, "
  sql = sql & "tblSiFragments.PathTTime, tblSiFragments.Destin, tblSiFragments.LevelTime "
  sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.ProcessId)=" & PId & ") AND "
  sql = sql & "(((tblSiFragments.LevelId)=" & (LvId - 1) & ") AND ((tblSiFragments.Destin)=" &
    Host & "))) "
  sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.LevelId, "
  sql = sql & "tblSiFragments.Destin, tblSiFragments.LevelTime; "
  Set rec = db.OpenRecordset(sql, dbOpenDynaset)
  If Not rec.EOF Then
    k = 1
    rec.MoveFirst
    initTime = rec!LevelTime
    PrevSTime = rec!LevelTime
    PrevSDelay = 0
    While Not rec.EOF
      If k > 1 Then
        rec.Edit
        initTime = rec!LevelTime
      End If
      rrec.AddNew
      rrec!ProcessId = PId
      rrec!PacketId = rec!PacketId
      rrec!FragID = rec!FragID
      rrec!LevelId = LvId
      rrec!SeqNum = rec!SeqNum
    
```



```

    rrec!fragSize = rec!fragSize
    rrec!LastPSize = rec!LastPSize
    rrec!StartTime = initTime
    rrec!ObjectId = Host
    r = Rnd
    While r = 0
        r = Rnd
    Wend
    ServTime = RoundNum((-1) * Log(1 - r), 4)
    rrec!ObjectTime = ServTime
    If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
        rrec!ObjectDelay = 0
    Else
        rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
    End If
    rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
    rrec!pathid = Int((NbrOfPath + 1 - 1) * Rnd + 1)
    PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
    PrevSDelay = rrec!ObjectDelay
    rrec.Update
    k = 0
    rec.MoveNext
Wend
End If
rec.Close
Next i
rec.Close

Return
PHosts_Error:
result = MsgBox("Host")
MsgBox (Error(Err))
Exit Function
End Function

```

Function PRouter(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)

```

On Error GoTo PRouter_Error
Dim sql As String
Dim rec As DAO.Recordset
Dim db As Database
Dim i, k
Dim RNo As Integer
Dim router As String
Dim mu As Single
Dim PrevSTime As Single
Dim PrevDTime As Single
Dim ServTime As Single
Dim initTime As Single
Dim psize As Long
Dim PathMTU As Long
Dim PathBand As Long
Dim SourceTo As String
Dim FragNo As Integer
Dim FragSeqCount As Integer

```

```

'path in level6
Dim dsql As String
Dim drec As DAO.Recordset
Dim pdest As String

'Resequencing 1
Dim sql1 As String
Dim recl As DAO.Recordset

    mu = DLookup("Myou", "tblSiOptions", "OptionId = " & OptId & "")
    RNo = DLookup("RoutNbr", "qrySiRoutersPerNet", "NetworkId = " & Net & "")
    GoSub AddRouterFrag
    GoSub ChkRouterFrag
    'Add to fragment table and Adjust sequence.
    GoSub CheckSeq
    For i = 1 To RNo
        router = "R" & i
        result = PPath("tmpSiFragRouters", Net, Pid, Lvid, router)
    Next i
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "qrySiAppRouterData"
    DoCmd.OpenQuery "qrySiDelRouterData"
    DoCmd.SetWarnings True

Exit Function

AddRouterFrag:

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragRouters", dbOpenDynaset)
FragSeqCount = 0

For i = 1 To RNo
    k = 0
    router = "R" & i
    NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & router & " and
        LevelId = " & Lvid & "")

    sql = ""
    sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
    sql = sql & "tblSiFragments.FragId, tblSiFragments.LevelId, tblSiFragments.SeqNum, "
    sql = sql & "tblSiFragments.FragSize, tblSiFragments.LastPSize, tblSiFragments.StartTime,
        tblSiFragments.ObjectId, "
    sql = sql & "tblSiFragments.ObjectTime, tblSiFragments.ObjectDelay,
        tblSiFragments.ObjectTTime, "
    sql = sql & "tblSiFragments.PathId, tblSiFragments.PathTime, tblSiFragments.PathDelay, "
    sql = sql & "tblSiFragments.PathTTime, tblSiFragments.Destin, tblSiFragments.LevelTime "
    sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.ProcessId)=" & Pid & ") AND "
    sql = sql & "((tblSiFragments.LevelId)=" & (Lvid - 1) & ") AND ((tblSiFragments.Destin)=" &
        router & "))) "
    sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.LevelId,
        tblSiFragments.Destin, tblSiFragments.LevelTime; "
    Set rec = db.OpenRecordset(sql, dbOpenDynaset)
    If Not rec.EOF Then
        rec.MoveLast

```

```

rec.MoveFirst
k = 1
initTime = rec!LevelTime
PrevSTime = 0
PrevSDelay = 0
While Not rec.EOF
  If k <> 1 Then
    rec.Edit
    initTime = rec!LevelTime
  End If
  rrec.AddNew
  rrec!ProcessId = PId
  rrec!PacketId = rec!PacketId
  rrec!FragID = rec!FragID
  rrec!LevelId = LvId
  rrec!SeqNum = rec!SeqNum
  rrec!SubSeqNum = rec!SeqNum
  rrec!fragSize = rec!fragSize
  rrec!LastPSize = rec!LastPSize
  rrec!ObjectId = router
  rrec!StartTime = rec!LevelTime
  r = Rnd
  While r = 0
    r = Rnd
  Wend
  ServTime = RoundNum((-1) * (1 / mu) * Log(1 - r), 4)
  rrec!ObjectTime = ServTime
  If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
    rrec!ObjectDelay = 0
  Else
    rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
  End If
  rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
  If LvId = 6 Then
    pdest = DLookup("DestinId", "tblSiPackets", "PacketId = " & rec!PacketId & " ")
    dsq1 = ""
    dsq1 = dsq1 & "SELECT tblSiPackets.PacketId, tblSiPaths.NetworkId, tblSiPaths.LevelId, "
    dsq1 = dsq1 & "tblSiPaths.PathId, tblSiPaths.FromSource, tblSiPackets.DestinId, "
    dsq1 = dsq1 & "tblSiPaths.MTU, tblSiPaths.Bandwidth FROM tblSiPackets INNER JOIN "
    dsq1 = dsq1 & "tblSiPaths ON tblSiPackets.DestinId = tblSiPaths.ToSource Where ( "
    dsq1 = dsq1 & "tblSiPaths.NetworkId = " & Net & " And tblSiPackets.PacketId = " &
      rec!PacketId & " "
    dsq1 = dsq1 & "and tblSiPaths.LevelId = " & LvId & " and tblSiPackets.DestinId = " & pdest
      & " and "
    dsq1 = dsq1 & "tblSiPaths.FromSource = " & router & " ); "
    Set drec = db.OpenRecordset(dsq1, dbOpenDynaset)
    If Not drec.EOF Then
      drec.MoveFirst
      rrec!pathid = drec!pathid
    Else
      rrec!pathid = Int((NbrOfPath + 1 - 1) * Rnd + 1)
    End If
    drec.Close
  Else
    rrec!pathid = Int((NbrOfPath + 1 - 1) * Rnd + 1)

```

```

    End If
    PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
    PrevSDelay = rrec!ObjectDelay
    rrec.Update
    k = 0
    rec.MoveNext
Wend
End If
rec.Close
Next i
    rrec.Close
Return

ChkRouterFrag:
Set tmpRec = db.OpenRecordset("tmpSiFragments", dbOpenDynaset)
Set rrec = db.OpenRecordset("tmpSiFragRouters", dbOpenDynaset)
If Not rrec.EOF Then
    rrec.MoveFirst
    While Not rrec.EOF
        PathMTU = DLookup("MTU", "tblSiPaths", "PathId= " & rrec!pathid & " and LevelId= " &
rrec!LevelId & " And FromSource= " & rrec!ObjectId & "")
        psize = rrec!fragSize
        If psize > PathMTU Then
            GoSub fragNum
            If FragNo <> 0 Then
                For i = 1 To FragNo
                    tmpRec.AddNew
                    tmpRec!ProcessId = Pid
                    tmpRec!PacketId = rrec!PacketId
                    tmpRec!FragID = rrec!FragID
                    tmpRec!LevelId = rrec!LevelId
                    tmpRec!SeqNum = rrec!SeqNum
                    tmpRec!SubSeqNum = i - 1
                    If psize - PathMTU >= 0 Then
                        tmpRec!fragSize = PathMTU
                        psize = psize - PathMTU
                    Else
                        If psize < 48 Then
                            t = 48 - psize
                            tmpRec!LastPSize = psize
                            tmpRec!fragSize = psize + t
                        Else
                            tmpRec!fragSize = psize
                        End If
                    End If
                    tmpRec!ObjectId = rrec!ObjectId
                    tmpRec!StartTime = rrec!StartTime
                    tmpRec!ObjectTime = rrec!ObjectTTime
                    tmpRec!ObjectDelay = 0
                    tmpRec!ObjectTTime = rrec!ObjectTTime
                    tmpRec!pathid = rrec!pathid
                    tmpRec.Update
                Next i
            rrec.Delete
            rrec.MoveNext

```

```

    End If
  End If
  rrec.MoveNext
Wend
End If

```

```

  DoCmd.SetWarnings False
  DoCmd.OpenQuery "qrySiAppRoutersFrag"
  DoCmd.OpenQuery "qrySiTmpFragDelete"
  DoCmd.SetWarnings True
Return

```

```

fragNum:
  FragNo = Round(psize / PathMTU, 0)
  If FragNo < 1 Then
    FragNo = 0
  End If

```

```

  If FragNo = 1 Then
    If (psize Mod PathMTU) <> 0 Then
      FragNo = FragNo + 1
    Else
      FragNo = 0
    End If
  End If
Return

```

CheckSeq:

```

Set db = DBEngine(0)(0)
sql = ""
sql = sql & "Select qrySiFragNumber.* from qrySiFragNumber; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
  rec.MoveFirst
  While Not rec.EOF
    FragSeqCount = rec!FragNoCount
    sql1 = ""
    sql1 = sql1 & "SELECT tmpSiFragRouters.ProcessId, tmpSiFragRouters.PacketID, "
    sql1 = sql1 & "tmpSiFragRouters.FragId, tmpSiFragRouters.LevelId, "
    sql1 = sql1 & "tmpSiFragRouters.SeqNum, tmpSiFragRouters.SubSeqNum FROM "
    sql1 = sql1 & "tmpSiFragRouters "
    sql1 = sql1 & "Where (tmpSiFragRouters.FragId = " & rec!FragID & " And "
    sql1 = sql1 & "tmpSiFragRouters.PacketId = " & rec!PacketId & ") "
    sql1 = sql1 & "ORDER BY tmpSiFragRouters.SeqNum, tmpSiFragRouters.SubSeqNum; "
    Set recl = db.OpenRecordset(sql1, dbOpenDynaset)
    If Not recl.EOF Then
      recl.MoveFirst
      For i = 0 To FragSeqCount - 1
        recl.Edit
        recl!SubSeqNum = i
        recl.Update
        recl.MoveNext
      Next i
    End If
  End While
End If

```

```

    rec1.Close
    rec.MoveNext
Wend
End If
rec.Close
Return

```

```

PRouter_Error:
result = MsgBox("Router")
MsgBox (Error(Err))
Exit Function
End Function

```

Function PSource(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)

On Error GoTo PSource_Error

```

Dim sql As String
Dim rec As DAO.Recordset
Dim db As Database
Dim ObjId As String
Dim i, k

```

```

Select Case PId
Case 1, 2
    Select Case Net
    Case 1
        GoSub Net1Source
        DoCmd.SetWarnings False
        DoCmd.OpenQuery "qrySiAppSourceData"
        DoCmd.OpenQuery "qrySiDelSourceData"
        DoCmd.SetWarnings True
    Case 2
        GoSub Net2Source
        result = PPath("tmpSiFragSources", Network, ProcessId, Level, ObjId)
        DoCmd.SetWarnings False
        DoCmd.OpenQuery "qrySiAppSourceData"
        DoCmd.OpenQuery "qrySiDelSourceData"
        DoCmd.SetWarnings True

```

End Select

```

Case 3
End Select

```

Exit Function

Net1Source:

```

Set db = DBEngine(0)(0)
sql = ""
sql = sql & "SELECT tblSiPackets.PacketId, tblSiPackets.Size, tblSiPackets.DestinId, "
sql = sql & "tblSiPackets.Poisson FROM tblSiPackets ORDER BY tblSiPackets.PacketId; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
        GoSub Frag576

```

```

ObjId = "S1"
result = DivPacket576(rec!PacketId, rec!Poisson, Pid, LvId, FragNo, rec!Size, ObjId)
rec.MoveNext
Wend
result = PPath("tmpSiFragSources", Network, ProcessId, Level, ObjId)
End If
rec.Close
Return

Net2Source:
Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragSources", dbOpenDynaset)
ObjId = "S2"
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & ObjId & " and
                    LevelId = " & LvId & "")

k = 0
sql = ""
sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "tblSiFragments.FragId, tblSiFragments.LevelId, tblSiFragments.SeqNum, "
sql = sql & "tblSiFragments.FragSize, tblSiFragments.LastPSize, tblSiFragments.StartTime, "
sql = sql & "tblSiFragments.ObjectID, "
sql = sql & "tblSiFragments.ObjectTime, tblSiFragments.ObjectDelay, "
sql = sql & "tblSiFragments.ObjectTTime, "
sql = sql & "tblSiFragments.PathId, tblSiFragments.PathTime, tblSiFragments.PathDelay, "
sql = sql & "tblSiFragments.PathTTime, tblSiFragments.Destin, tblSiFragments.LevelTime "
sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.ProcessId)=" & Pid & ") AND "
sql = sql & "((tblSiFragments.LevelId)=" & (LvId - 1) & ")) "
sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.LevelId, "
sql = sql & "tblSiFragments.Destin, tblSiFragments.LevelTime; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    k = 1
    initTime = rec!LevelTime
    PrevSTime = rec!LevelTime
    PrevSDelay = 0
    While Not rec.EOF
        If k > 1 Then
            rec.Edit
            initTime = rec!LevelTime
        End If
        rrec.AddNew
        rrec!ProcessId = Pid
        rrec!PacketId = rec!PacketId
        rrec!FragID = rec!FragID
        rrec!LevelId = LvId
        rrec!SeqNum = rec!SeqNum
        rrec!fragSize = rec!fragSize
        rrec!LastPSize = rec!LastPSize
        rrec!StartTime = initTime
        rrec!ObjectID = ObjId
        r = Rnd
        While r = 0
            r = Rnd
        Wend
    Wend

```

```

ServTime = RoundNum((-1) * Log(1 - r), 4)
rrec!ObjectTime = ServTime
If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
  rrec!ObjectDelay = 0
Else
  rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
End If
rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
rrec!pathid = Int((NbrOfPath + 1 - 1) * Rnd + 1)
PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
PrevSDelay = rrec!ObjectDelay
rrec.Update
k = 0
rec.MoveNext
Wend
End If
rec.Close
rrec.Close
Return

```

```

Frag576:
FragNo = Round(rec!Size / 576, 0)
If FragNo < 1 Then
  FragNo = 0
End If

```

```

If FragNo = 1 Then
  If (rec!Size Mod 576) <> 0 Then
    FragNo = FragNo + 1
  Else
    FragNo = 0
  End If
End If
Return

```

```

PSource_Error:
  result = MsgBox("Source")
  MsgBox (Error(Err))
Exit Function
End Function

```

```

Function Process1()
On Error GoTo Process1_Err

```

```

ProcessId = 1

```

```

Network = 1
  Level = 1
  result = PSource(Network, ProcessId, PackNo, Level)   'Source
  Level = 2
  result = PRouter(Network, ProcessId, PackNo, Level)  'Router
  Level = 3
  result = PHosts(Network, ProcessId, PackNo, Level)   'Host
  Level = 4
  result = PGateway(Network, ProcessId, PackNo, Level) 'Gateway

```



```

Network = 2
  Level = 5
  result = PSource(Network, ProcessId, PackNo, Level)   'Source
  Level = 6
  result = PRouter(Network, ProcessId, PackNo, Level)  'Router
  Level = 7
  result = PDestination()                               'Destination

```

Exit Function

```

Process1_Err:
  result = MsgBox("Process 1")
  MsgBox (Error(Err))
Exit Function
End Function

```

Private Sub Go_Click()

```

On Error GoTo Go_Click_err
Dim NbrOfProcess As Integer
Dim result, i

```

```

  result = GeneratePackets()
  DoCmd.SetWarnings False
  DoCmd.OpenQuery "qrySiEmptyFragTable"
  DoCmd.SetWarnings True
  OptId = 1
  result = Process1()
  MsgBox "Simulation Ended"
Exit Sub

```

Go_Click_err:

```

  result = MsgBox("Go Click")
  MsgBox (Error(Err))
Exit Sub
End Sub

```

'Fill table tblGnPackets

'Generate packet size, source and destination

'Size between 500 and 3500 bytes => Select Randomly

'Destinations between 1 and 3 => Select Randomly

'Genrate Poisson Timing for each Packet

Function GeneratePackets()

```

On Error GoTo GeneratePackets_Err

```

```

Dim sql As String
Dim rec As DAO.Recordset
Dim db As Database

```

'packet Number

Dim i As Integer

Dim OpId As Integer

```

'Poisson timing
Dim Gentime As Single
Dim SendTime As Single

OptId = 1
Gentime = DLookup("Lamda", "tblSiOptions", "OptionId= " & OptId & "")
PackNo = DLookup("NbrOfPackets", "tblSiOptions", "OptionId= " & OptId & "")

Set db = DBEngine(0)(0)

sql = ""
sql = sql & "Select tblSiPackets.* from tblSiPackets;"
Set rec = db.OpenRecordset(sql, dbOpenDynaset)

For i = 1 To PackNo
    rec.AddNew
    rec!PacketId = i
    rec!Size = Int((3000 - 600 + 1) * Rnd + 600)
    rec!DestinId = Int((2 - 1 + 1) * Rnd + 1)
    If i = 1 Then
        SendTime = 0
        rec!Poisson = SendTime
    Else
        r = Rnd
        SendTime = SendTime + (r * (1 / Gentime))
        rec!Poisson = RoundNum(SendTime, 2)
    End If
    rec.Update
Next i
rec.Close
Exit Function

GeneratePackets_Err:
result = MsgBox("Generate Packets")
MsgBox (Error(Err))
Exit Function
End Function

Function RoundNum(InVal, Places) As Double
Dim Factor As Double
Dim temp As Double

If IsNull(InVal) Then InVal = 0
If IsNull(Places) Then Exit Function

If Places < 0 Then Places = 4

Let Factor = 10 ^ Int(Places)
temp = Abs(InVal) * Factor + 0.5000000001
RoundNum = (Int(temp) / Factor) * Sgn(InVal)

End Function

```

2 – Simulated Code:

```

Option Compare Database
Dim OptId As Integer
Dim PackNo As Integer
Dim Network As Integer
Dim Level As Integer
Dim ProcessId As Integer
Dim ObjId As String
Dim r As Single

```

Function DivPacket(PackId As Long, PoisTime As Single, ProcId As Integer, LevelId As Integer, FragNo, psize As Double, Old As String, pathid As Integer, PathSize As Long)

```
On Error GoTo DivPacket_err
```

```

Dim frag As String
Dim frec As DAO.Recordset
Dim db As Database
Dim NbrOfPath As Integer
Dim i

```

```
Set db = DBEngine(0)(0)
```

```
frag = ""
```

```
frag = frag & "select tmpSiFragSources.* from tmpSiFragSources; "
```

```
Set frec = db.OpenRecordset(frag, dbOpenDynaset)
```

```
If FragNo = 0 Or IsNull(FragNo) Then
```

```
    frec.AddNew
```

```
    frec!ProcessId = ProcId
```

```
    frec!PacketId = PackId
```

```
    frec!FragID = 1
```

```
    frec!LevelId = LevelId
```

```
    frec!SeqNum = 0
```

```
    frec!fragSize = psize
```

```
    frec!LastPSize = 0
```

```
    frec!StartTime = 0
```

```
    frec!ObjectId = Old
```

```
    frec!ObjectTime = PoisTime
```

```
    frec!ObjectDelay = 0
```

```
    frec!ObjectTTime = RoundNum(frec!ObjectTime + frec!ObjectDelay, 4)
```

```
    frec!pathid = pathid
```

```
    frec.Update
```

```
Else
```

```
For i = 1 To FragNo
```

```
    frec.AddNew
```

```
    frec!ProcessId = ProcId
```

```
    frec!PacketId = PackId
```

```
    frec!FragID = i
```

```
    frec!LevelId = LevelId
```

```
    frec!SeqNum = 0
```

```
If psize - PathSize >= 0 Then
```

```
    frec!fragSize = PathSize
```

```
    psize = psize - PathSize
```

```
Else
```

```
    If psize < 48 Then
```

```

    t = 48 - psize
    frec!LastPSize = psize
    frec!fragSize = psize + t
Else
    frec!fragSize = psize
End If
End If
frec!StartTime = 0
frec!ObjectId = Old
frec!ObjectTime = PoisTime
frec!ObjectDelay = 0
frec!ObjectITime = RoundNum(frec!ObjectTime + frec!ObjectDelay, 4)
frec!pathid = pathid
frec.Update
Next i
End If
Exit Function

```

```

DivPacket_err:
MsgBox "DivPacket"
MsgBox (Error(Err))
Exit Function
End Function

```

Function PPath(TableName As String, Net As Integer, Procid As Integer, LevelId As Integer, Old As String)

```

On Error GoTo PPath_err
Dim PrevTime
Dim PrevDelay
Dim db As Database
Dim sql As String
Dim rec As DAO.Recordset
Dim NbrOfPath As Integer
Dim PathMTU As Long
Dim PathBand As Long
Dim SourceTo As String
Dim initTime As Single
Dim i, j

```

```

Set db = DBEngine(0)(0)
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & Old & " and
LevelId = " & LevelId & "")

```

```

For i = 1 To NbrOfPath
    PathMTU = DLookup("MTU", "tblSiPaths", "PathId= " & i & " and LevelId= " & LevelId & "
And FromSource= " & Old & "")
    PathBand = DLookup("Bandwidth", "tblSiPaths", "PathId= " & i & " and LevelId= " & LevelId
& " And FromSource= " & Old & "")
    SourceTo = DLookup("ToSource", "tblSiPaths", "PathId= " & i & " and LevelId= " & LevelId &
" And FromSource= " & Old & "")
    sql = ""
    sql = sql & "SELECT " & TableName & ".ProcessId, " & TableName & ".LevelId, "
    sql = sql & TableName & ".PathId, " & TableName & ".ObjectId, " & TableName &
".ObjectITime, "
    sql = sql & TableName & ".PacketID, " & TableName & ".FragId, " & TableName & ".SeqNum, "

```

```

sql = sql & TableName & ".FragSize, " & TableName & ".LastPSize, " & TableName &
      ".StartTime, " & TableName & ".ObjectTime, "
sql = sql & TableName & ".ObjectDelay, " & TableName & ".PathTime, " & TableName &
      ".PathDelay, "
sql = sql & TableName & ".PathTTime, " & TableName & ".LevelTime, " & TableName &
      ".Destin "
sql = sql & "FROM " & TableName & " WHERE ((((" & TableName & ".ProcessId)=" & Procid &
      ") AND "
sql = sql & "(" & TableName & ".LevelId)=" & LevelId & ") AND ((" & TableName &
      ".PathId)=" & i & ") AND "
sql = sql & "(" & TableName & ".ObjectId)=" & Old & ")") ORDER BY " & TableName &
      ".ProcessId, "
sql = sql & TableName & ".LevelId, " & TableName & ".PathId, " & TableName & ".ObjectId, " &
      TableName & ".ObjectTTime; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    initTime = rec!ObjectTTime
    PrevTime = rec!ObjectTTime
    rec.MoveLast
    For j = 1 To rec.RecordCount
        If j = 1 Then
            rec.MoveFirst
            rec.Edit
        Else
            rec.Edit
            initTime = rec!ObjectTTime
        End If
        r = Rnd
        While r = 0
            r = Rnd
        Wend
        rec!pathTime = RoundNum((-1) * (rec!fragSize / PathBand) * Log(1 - r), 4)

        If (PrevTime - initTime) <= 0 Then
            rec!PathDelay = 0
        Else
            rec!PathDelay = RoundNum(PrevTime - initTime, 4)
        End If
        rec!pathTTime = rec!pathTime + rec!PathDelay
        rec!LevelTime = RoundNum(rec!ObjectTTime + rec!pathTime + rec!PathDelay, 4)
        PrevTime = rec!ObjectTTime + rec!pathTTime
        rec!Destin = SourceTo
        rec.Update
        rec.MoveNext
    Next j
End If
Next i
Exit Function

PPath_err:
result = MsgBox("Paths")
MsgBox (Error(Err))
Exit Function
End Function

```

Function PDestination()

On Error GoTo PDestination_Err

```

DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiMeanDelayBylamda"
DoCmd.OpenQuery "qrySiMeanDelayByMiou"
DoCmd.SetWarnings True

```

Exit Function

```

PDestination_Err:
MsgBox (Error(Err))
Exit Function
End Function

```

Function PGateway(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)

On Error GoTo Pgateway_err

```

Dim Gate As Long
Dim gateway As String
Dim k

```

```

Gate = DLookup("Gateway", "tblSiOptions", "OptionId = " & OptId & "")
gateway = "G1"
GoSub GatewayResembPack
GoSub AddGatewayFrag
result = PPath("tmpSiFragGateways", Net, PId, LvId, gateway)
DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppGateData"
DoCmd.OpenQuery "qrySiDelGateData"
DoCmd.SetWarnings True

```

Exit Function

GatewayResembPack:

```

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpS2FragGatewayReasmb", dbOpenDynaset)
sql = ""
sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "Sum(tblSiFragments.FragSize) AS [Size], Max(tblSiFragments.LevelTime) AS "
sql = sql & "DestTime "
sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.LevelId) = " & (LvId - 1) & ") And "
sql = sql & "((tblSiFragments.Destin) = " & gateway & ")))"
sql = sql & "GROUP BY tblSiFragments.ProcessId, tblSiFragments.PacketID "
sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "Max(tblSiFragments.LevelTime); "

```

Set rec = db.OpenRecordset(sql, dbOpenDynaset)

If Not rec.EOF Then

```

While Not rec.EOF
rrec.AddNew
rrec!ProcessId = rec!ProcessId
rrec!PacketId = rec!PacketId
rrec!FragID = 1
rrec!LevelId = LvId
rrec!SeqNum = 0

```

```

    rrec!SubSeqNum = 0
    rrec!fragSize = rec!Size
    rrec!LastPSize = 0
    rrec!ObjectId = gateway
    rrec!StartTime = rec!DestTime
    rrec!ObjectTime = 0
    rrec!ObjectDelay = 0
    rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
    rrec!pathid = 0
    rrec.Update
    rec.MoveNext
Wend
End If
rec.Close
rrec.Close
Return

```

AddGatewayFrag:

```

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragGateways", dbOpenDynaset)
k = 0
NbrOfPath = DLookup("PathNo", "qrySiPathNbrfromSource", "FromSource = " & gateway & "
    and LevelId = " & LvId & ")
sql = ""
sql = sql & "SELECT tmpS2FragGatewayReasmb.ProcessId, tmpS2FragGatewayReasmb.PacketID, "
sql = sql & "tmpS2FragGatewayReasmb.FragId, tmpS2FragGatewayReasmb.LevelId,
    tmpS2FragGatewayReasmb.SeqNum, "
sql = sql & "tmpS2FragGatewayReasmb.FragSize, tmpS2FragGatewayReasmb.LastPSize,
    tmpS2FragGatewayReasmb.StartTime, tmpS2FragGatewayReasmb.ObjectId, "
sql = sql & "tmpS2FragGatewayReasmb.ObjectTime, tmpS2FragGatewayReasmb.ObjectDelay,
    tmpS2FragGatewayReasmb.ObjectTTime, "
sql = sql & "tmpS2FragGatewayReasmb.PathId, tmpS2FragGatewayReasmb.PathTime,
    tmpS2FragGatewayReasmb.PathDelay, "
sql = sql & "tmpS2FragGatewayReasmb.PathTTime, tmpS2FragGatewayReasmb.Destin,
    tmpS2FragGatewayReasmb.LevelTime "
sql = sql & "FROM tmpS2FragGatewayReasmb WHERE ((tmpS2FragGatewayReasmb.ProcessId)="
    & Pid & ") AND "
sql = sql & "((tmpS2FragGatewayReasmb.LevelId)=" & (LvId) & ") AND
    ((tmpS2FragGatewayReasmb.ObjectId) = " & gateway & ") "
sql = sql & "ORDER BY tmpS2FragGatewayReasmb.ProcessId,
    tmpS2FragGatewayReasmb.LevelId, tmpS2FragGatewayReasmb.Destin,
    tmpS2FragGatewayReasmb.LevelTime; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    k = 1
    rec.MoveFirst
    initTime = rec!LevelTime
    PrevSTime = rec!LevelTime
    PrevSDelay = 0
    While Not rec.EOF
        If k <> 1 Then
            rec.Edit
            initTime = rec!LevelTime

```

```

End If

rrec.AddNew
rrec!ProcessId = PId
rrec!PacketId = rec!PacketId
rrec!FragID = rec!FragID
rrec!LevelId = LvId
rrec!SeqNum = rec!SeqNum
rrec!fragSize = rec!fragSize
rrec!LastPSize = rec!LastPSize
rrec!StartTime = rec!StartTime
rrec!ObjectId = gateway
r = Rnd
While r = 0
    r = Rnd
Wend
ServTime = RoundNum((-1) * (1 / 3) * Log(1 - r), 4)
rrec!ObjectTime = ServTime
If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
    rrec!ObjectDelay = 0
Else
    rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
End If
rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
rrec!pathid = 1
PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
PrevSDelay = rrec!ObjectDelay
rrec.Update
k = 0
rec.MoveNext
Wend
End If
rec.Close
rrec.Close
Return

Pgateway_err:
result = MsgBox("Gateway")
MsgBox (Error(Err))
Exit Function
End Function

Function PHosts(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)
On Error GoTo PHosts_Error
Dim HostNo As Integer
Dim db As Database
Dim rrec As DAO.Recordset
Dim tmpRec As DAO.Recordset
Dim rec As DAO.Recordset
Dim sql As String
Dim Host As String
Dim i, j, k

GoSub HostReassembly
GoSub HostDelay

```



```

For j = 1 To 2
  Host = "H" & j
  result = PPath("tmpSiFragHosts", Net, PId, LvId, Host)
Next j
DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppHostData"
DoCmd.OpenQuery "qrySiDelHostData"
DoCmd.SetWarnings True

```

Exit Function

```

HostReassembly:
Set db = DBEngine(0)(0)
For i = 1 To 2
  Host = "H" & i
  Set rrec = db.OpenRecordset("tmpS2FragHostReasmb", dbOpenDynaset)
  sql = ""
  sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
  sql = sql & "tblSiFragments.FragId, Sum(tblSiFragments.FragSize) AS [Size], "
  sql = sql & "First(tblSiFragments.SeqNum) AS NumSeq, "
  sql = sql & "Max(tblSiFragments.LevelTime) AS DestTime FROM tblSiFragments "
  sql = sql & "WHERE (((tblSiFragments.LevelId) = " & (LvId - 1) & ") and tblSiFragments.Destin = "
  sql = sql & " & Host & ") "
  sql = sql & "GROUP BY tblSiFragments.ProcessId, tblSiFragments.PacketID, "
  sql = sql & "tblSiFragments.FragId ORDER BY tblSiFragments.ProcessId, "
  sql = sql & "tblSiFragments.PacketID, Max(tblSiFragments.LevelTime); "
  Set rec = db.OpenRecordset(sql, dbOpenDynaset)
  If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
      rrec.AddNew
      rrec!ProcessId = rec!ProcessId
      rrec!PacketId = rec!PacketId
      rrec!FragID = rec!FragID
      rrec!LevelId = LvId
      rrec!SeqNum = rec!NumSeq
      rrec!SubSeqNum = rec!NumSeq
      rrec!fragSize = rec!Size
      rrec!LastPSize = 0
      rrec!ObjectId = Host
      rrec!StartTime = rec!DestTime
      rrec!ObjectTime = 0
      rrec!ObjectDelay = 0
      rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
      rrec!pathid = 0
      rrec.Update
      rec.MoveNext
    Wend
  End If
  rec.Close
Next i
rrec.Close
Return

```

HostDelay:

```
Set db = DBEngine(0)(0)
```

```
Set rec = db.OpenRecordset("tmpSiFragHosts", dbOpenDynaset)
```

```
For i = 1 To 2
```

```
Host = "H" & i
```

```
sql = ""
```

```
sql = sql & "SELECT tmpS2FragHostReasmb.ProcessId, tmpS2FragHostReasmb.PacketID, "
```

```
sql = sql & "tmpS2FragHostReasmb.FragId, tmpS2FragHostReasmb.LevelId, "
      tmpS2FragHostReasmb.SeqNum, "
```

```
sql = sql & "tmpS2FragHostReasmb.FragSize, tmpS2FragHostReasmb.LastPSize, "
```

```
sql = sql & "tmpS2FragHostReasmb.ObjectId, tmpS2FragHostReasmb.ObjectTime, "
      tmpS2FragHostReasmb.ObjectDelay, "
```

```
sql = sql & "tmpS2FragHostReasmb.ObjectTTime, tmpS2FragHostReasmb.PathId, "
      tmpS2FragHostReasmb.PathTime, "
```

```
sql = sql & "tmpS2FragHostReasmb.PathDelay, tmpS2FragHostReasmb.PathTTime, "
      tmpS2FragHostReasmb.Destin, tmpS2FragHostReasmb.StartTime, "
```

```
sql = sql & "tmpS2FragHostReasmb.LevelTime FROM tmpS2FragHostReasmb WHERE "
      (((tmpS2FragHostReasmb.ProcessId)=" & Pid & ") AND "
```

```
sql = sql & "(((tmpS2FragHostReasmb.LevelId)=" & LvId & ") AND "
      ((tmpS2FragHostReasmb.ObjectId)=" & Host & "))) "
```

```
sql = sql & "ORDER BY tmpS2FragHostReasmb.ProcessId, tmpS2FragHostReasmb.LevelId, "
      tmpS2FragHostReasmb.Destin, tmpS2FragHostReasmb.StartTime; "
```

```
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
```

```
If Not rec.EOF Then
```

```
  k = 1
```

```
  rec.MoveFirst
```

```
  initTime = rec!StartTime
```

```
  PrevSTime = rec!StartTime
```

```
  PrevSDelay = 0
```

```
  While Not rec.EOF
```

```
    If k <> 1 Then
```

```
      rec.Edit
```

```
      initTime = rec!StartTime
```

```
    End If
```

```
    rrec.AddNew
```

```
    rrec!ProcessId = Pid
```

```
    rrec!PacketId = rec!PacketId
```

```
    rrec!FragID = rec!FragID
```

```
    rrec!LevelId = LvId
```

```
    rrec!SeqNum = rec!SeqNum
```

```
    rrec!fragSize = rec!fragSize
```

```
    rrec!LastPSize = rec!LastPSize
```

```
    rrec!StartTime = initTime
```

```
    rrec!ObjectId = Host
```

```
    r = Rnd
```

```
    While r = 0
```

```
      r = Rnd
```

```
    Wend
```

```
    ServTime = RoundNum((-1) * Log(1 - r), 4)
```

```
    rrec!ObjectTime = ServTime
```

```
    If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
```

```
      rrec!ObjectDelay = 0
```

```
    Else
```

```
      rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
```

```

    End If
    rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
    rrec!pathid = 1
    PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
    PrevSDelay = rrec!ObjectDelay
    rrec.Update
    k = 0
    rec.MoveNext
Wend
End If
rec.Close
Next i
rrec.Close
Return

```

PHosts_Error:

```

result = MsgBox("Host")
MsgBox (Error(Err))
Exit Function
End Function

```

Function PRouter(Net As Integer, PId As Integer, PackNo As Integer, LvlId As Integer)

On Error GoTo PRouter_Error

```

Dim sql As String
Dim rec As DAO.Recordset
Dim db As Database
Dim i, k
Dim RNo As Integer
Dim router As String
Dim mu As Single
Dim PrevSTime As Single
Dim PrevDTime As Single
Dim ServTime As Single
Dim initTime As Single
Dim psize As Long
Dim PathMTU As Long
Dim PathBand As Long
Dim SourceTo As String
Dim FragNo As Integer
Dim FragSeqCount As Integer

```

'path in level6

```

Dim dsql As String
Dim drec As DAO.Recordset
Dim pdest As String

```

'Resequencing 1

```

Dim sql1 As String
Dim rec1 As DAO.Recordset
Dim frag

```

```

frag = False
mu = DLookup("Myou", "tblSiOptions", "OptionId = " & OptId & "")
RNo = DLookup("RoutNbr", "qrySiRoutersPerNet", "NetworkId = " & Net & "")

```

```

GoSub RouteResembFrag
GoSub RouteResembPack
GoSub AddRouterFrag
If frag = True Then
  GoSub ChkRouterFrag
  GoSub CheckSeq
End If
For i = 1 To RNo
  router = "R" & i
  result = PPath("tmpSiFragRouters", Net, PId, LvId, router)
Next i
DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppRouterData"
DoCmd.OpenQuery "qrySiDelRouterData"
DoCmd.SetWarnings True

```

Exit Function

```

RouteResembFrag:
Set db = DBEngine(0)(0)
For i = 1 To RNo
  router = "R" & i
  Set rrec = db.OpenRecordset("tmpS2FragRoutersReasmb", dbOpenDynaset)
  sql = ""
  sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
  sql = sql & "tblSiFragments.FragId, Sum(tblSiFragments.FragSize) AS [Size], "
  sql = sql & "First(tblSiFragments.SeqNum) AS NumSeq, "
  sql = sql & "Max(tblSiFragments.LevelTime) AS DestTime FROM tblSiFragments "
  sql = sql & "WHERE (((tblSiFragments.LevelId) = " & (LvId - 1) & ") and tblSiFragments.Destin = "
  sql = sql & " & router & " and tblSiFragments.Seqnum < 0) "
  sql = sql & "GROUP BY tblSiFragments.ProcessId, tblSiFragments.PacketID, "
  sql = sql & "tblSiFragments.FragId ORDER BY tblSiFragments.ProcessId, "
  sql = sql & "tblSiFragments.PacketID, Max(tblSiFragments.LevelTime); "
  Set rec = db.OpenRecordset(sql, dbOpenDynaset)
  If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
      rrec.AddNew
      rrec!ProcessId = rec!ProcessId
      rrec!PacketId = rec!PacketId
      rrec!FragID = rec!FragID
      rrec!LevelId = LvId
      rrec!SeqNum = 0
      rrec!SubSeqNum = 0
      rrec!fragSize = rec!Size
      rrec!LastPSize = 0
      rrec!ObjectId = router
      rrec!StartTime = rec!DestTime
      rrec!ObjectTime = 0
      rrec!ObjectDelay = 0
      rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
      rrec!pathid = 0
      rrec.Update
      rec.MoveNext
    Wend
  End If
Next i

```

```

End If
rec.Close
Next i
rec.Close
Return

RouteResembPack:
Set db = DBEngine(0)(0)
For i = 1 To RNo
router = "R" & i
Set rrec = db.OpenRecordset("tmpS2FragRoutersReasmb", dbOpenDynaset)
sql = ""
sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "Sum(tblSiFragments.FragSize) AS [Size], Max(tblSiFragments.LevelTime) AS
DestTime "
sql = sql & "FROM tblSiFragments WHERE (((tblSiFragments.LevelId) = " & (LvId - 1) & ") And "
sql = sql & "((tblSiFragments.Destin) = " & router & ") And ((tblSiFragments.SeqNum) = 0))"
sql = sql & "GROUP BY tblSiFragments.ProcessId, tblSiFragments.PacketID "
sql = sql & "ORDER BY tblSiFragments.ProcessId, tblSiFragments.PacketID,
Max(tblSiFragments.LevelTime); "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
While Not rec.EOF
rec.AddNew
rec!ProcessId = rec!ProcessId
rec!PacketId = rec!PacketId
rec!FragID = 1
rec!LevelId = LvId
rec!SeqNum = 0
rec!SubSeqNum = 0
rec!fragSize = rec!Size
rec!LastPSize = 0
rec!ObjectId = router
rec!StartTime = rec!DestTime
rec!ObjectTime = 0
rec!ObjectDelay = 0
rec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
rec!pathid = 0
rec.Update
rec.MoveNext
Wend
End If
rec.Close
Next i
rec.Close
Return

AddRouterFrag:

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragRouters", dbOpenDynaset)
FragSeqCount = 0

For i = 1 To RNo
k = 0

```

```

router = "R" & i
sql = ""
sql = sql & "SELECT tmpS2FragRoutersReasmb.ProcessId, tmpS2FragRoutersReasmb.PacketID, "
sql = sql & "tmpS2FragRoutersReasmb.FragId, tmpS2FragRoutersReasmb.LevelId, "
      tmpS2FragRoutersReasmb.SeqNum, "
sql = sql & "tmpS2FragRoutersReasmb.FragSize, tmpS2FragRoutersReasmb.LastPSize, "
      tmpS2FragRoutersReasmb.StartTime, "
sql = sql & "tmpS2FragRoutersReasmb.ObjectId, tmpS2FragRoutersReasmb.ObjectTime, "
      tmpS2FragRoutersReasmb.ObjectDelay, "
sql = sql & "tmpS2FragRoutersReasmb.ObjectTTime, tmpS2FragRoutersReasmb.PathId, "
      tmpS2FragRoutersReasmb.PathTime, "
sql = sql & "tmpS2FragRoutersReasmb.PathDelay, tmpS2FragRoutersReasmb.PathTTime, "
      tmpS2FragRoutersReasmb.Destin, "
sql = sql & "tmpS2FragRoutersReasmb.LevelTime FROM tmpS2FragRoutersReasmb WHERE "
      (((tmpS2FragRoutersReasmb.ProcessId)=" & Pid & ") AND "
sql = sql & "(((tmpS2FragRoutersReasmb.LevelId)=" & LvId & ") AND "
      ((tmpS2FragRoutersReasmb.ObjectId)=" & router & ")) "
sql = sql & "ORDER BY tmpS2FragRoutersReasmb.ProcessId, tmpS2FragRoutersReasmb.LevelId, "
      tmpS2FragRoutersReasmb.Destin, tmpS2FragRoutersReasmb.StartTime;"
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
  rec.MoveFirst
  k = 1
  initTime = rec!StartTime
  PrevSTime = 0
  PrevSDelay = 0
  While Not rec.EOF
    If k <> 1 Then
      rec.Edit
      initTime = rec!StartTime
    End If
    rrec.AddNew
    rrec!ProcessId = Pid
    rrec!PacketId = rec!PacketId
    rrec!FragID = rec!FragID
    rrec!LevelId = LvId
    rrec!SeqNum = rec!SeqNum
    rrec!SubSeqNum = rec!SeqNum
    rrec!fragSize = rec!fragSize
    rrec!LastPSize = rec!LastPSize
    rrec!ObjectId = router
    rrec!StartTime = rec!StartTime
    r = Rnd
    While r = 0
      r = Rnd
    Wend
    ServTime = RoundNum((-1) * (1 / mu) * Log(1 - r), 4)
    rrec!ObjectTime = ServTime
    If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
      rrec!ObjectDelay = 0
    Else
      rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
    End If
    rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
  GoSub SelectPath

```

```

    rrec!pathid = PathNo
    PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
    PrevSDelay = rrec!ObjectDelay
    rrec.Update
    k = 0
    rec.MoveNext
Wend
End If
rec.Close
Next i
rec.Close
Return

```

ChkRouterFrag:

```

Set tmpRec = db.OpenRecordset("tmpSiFragments", dbOpenDynaset)
Set rrec = db.OpenRecordset("tmpSiFragRouters", dbOpenDynaset)
If Not rrec.EOF Then
    rrec.MoveFirst
    While Not rrec.EOF
        psize = rrec!fragSize
        If psize > PathMTU Then
            GoSub fragNum
            If FragNo <> 0 Then
                For i = 1 To FragNo
                    tmpRec.AddNew
                    tmpRec!ProcessId = PId
                    tmpRec!PacketId = rrec!PacketId
                    tmpRec!FragID = i 'rrec!FragID
                    tmpRec!LevelId = rrec!LevelId
                    tmpRec!SeqNum = rrec!SeqNum
                    tmpRec!SubSeqNum = i
                    If psize - PathMTU >= 0 Then
                        tmpRec!fragSize = PathMTU
                        psize = psize - PathMTU
                    Else
                        If psize < 48 Then
                            t = 48 - psize
                            tmpRec!LastPSize = psize
                            tmpRec!fragSize = psize + t
                        Else
                            tmpRec!fragSize = psize
                        End If
                    End If
                    tmpRec!ObjectId = rrec!ObjectId
                    tmpRec!StartTime = rrec!StartTime
                    tmpRec!ObjectTime = rrec!ObjectTTime
                    tmpRec!ObjectDelay = 0
                    tmpRec!ObjectTTime = rrec!ObjectTTime
                    tmpRec!pathid = rrec!pathid
                    tmpRec.Update
                Next i
            rec.Delete
        End If
    End If

```

```

rrec.MoveNext
Wend
End If

```

```

DoCmd.SetWarnings False
DoCmd.OpenQuery "qrySiAppRoutersFrag"
DoCmd.OpenQuery "qrySiTmpFragDelete"
DoCmd.SetWarnings True
Return

```

SelectPath:

```

Set db = DBEngine(0)(0)
If LvId <> 6 Then
path = ""
path = path & "SELECT tblSiPaths.PathId, tblSiPaths.MTU, tblSiPaths.Bandwidth AS MaxBand "
path = path & "FROM tblSiPaths WHERE (((tblSiPaths.NetworkId) = " & Net & ") And "
path = path & "(((tblSiPaths.LevelId) = " & LvId & ") And ((tblSiPaths.FromSource) = "" & router &
"")) "
path = path & "ORDER BY tblSiPaths.Bandwidth DESC; "
Set prec = db.OpenRecordset(path, dbOpenDynaset)
If Not prec.EOF Then
'set path id equal to this path
prec.MoveFirst
PathNo = prec!pathid
PathMTU = prec!MTU
If rec!fragSize > PathMTU Then
frag = True
Else
frag = False
End If
prec.Close
Else
'set path equal to highest path available
PathNo = DLookup("PathId", "qryS2LargePathMTU", "NetworkId = " & Net & " and LevelId =
" & LvId & " and FromSource= "" & router & """)
PathMTU = DLookup("MaxSize", "qryS2LargePathMTU", "NetworkId = " & Net & " and
LevelId = " & LvId & " and FromSource= "" & router & """)
frag = True
prec.Close
End If

```

Else

```

pdest = DLookup("DestinId", "tblSiPackets", "PacketId = " & rec!PacketId & " ")
dsql = ""
dsql = dsql & "SELECT tblSiPackets.PacketId, tblSiPaths.NetworkId, tblSiPaths.LevelId, "
dsql = dsql & "tblSiPaths.PathId, tblSiPaths.FromSource, tblSiPackets.DestinId, "
dsql = dsql & "tblSiPaths.MTU, tblSiPaths.Bandwidth FROM tblSiPackets INNER JOIN "
dsql = dsql & "tblSiPaths ON tblSiPackets.DestinId = tblSiPaths.ToSource Where ( "
dsql = dsql & "tblSiPaths.NetworkId = " & Net & " And tblSiPackets.PacketId = " &
rec!PacketId & " "
dsql = dsql & "and tblSiPaths.LevelId = " & LvId & " and tblSiPackets.DestinId = "" & pdest & ""
and "
dsql = dsql & "tblSiPaths.FromSource = "" & router & ""), "
Set drec = db.OpenRecordset(dsql, dbOpenDynaset)

```



```

If Not drec.EOF Then
    drec.MoveFirst
    rrec!pathid = drec!pathid
Else
    PathNo = DLookup("PathId", "qryS2LargePathMTU", "NetworkId = " & Net & " and LevelId
        = " & LvId & " and FromSource= " & router & """)
    PathMTU = DLookup("MaxSize", "qryS2LargePathMTU", "NetworkId = " & Net & " and
        LevelId = " & LvId & " and FromSource= " & router & """)
End If
drec.Close
End If

```

Return

fragNum:

```

FragNo = Round(psize / PathMTU, 0)
If FragNo < 1 Then
    FragNo = 0
End If

```

```

If FragNo = 1 Then
    If (psize Mod PathMTU) <> 0 Then
        FragNo = FragNo + 1
    Else
        FragNo = 0
    End If
End If

```

Return

CheckSeq:

```

Set db = DBEngine(0)(0)
sql = ""
sql = sql & "Select qrySiFragNumber.* from qrySiFragNumber; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
        FragSeqCount = rec!FragNoCount
        sql1 = ""
        sql1 = sql1 & "SELECT tmpSiFragRouters.ProcessId, tmpSiFragRouters.PacketID, "
        sql1 = sql1 & "tmpSiFragRouters.FragId, tmpSiFragRouters.LevelId, "
        sql1 = sql1 & "tmpSiFragRouters.SeqNum, tmpSiFragRouters.SubSeqNum FROM "
        sql1 = sql1 & "tmpSiFragRouters "
        sql1 = sql1 & "Where (tmpSiFragRouters.FragId = " & rec!FragID & " And "
        sql1 = sql1 & "tmpSiFragRouters.PacketId = " & rec!PacketId & ") "
        sql1 = sql1 & "ORDER BY tmpSiFragRouters.SeqNum, tmpSiFragRouters.SubSeqNum; "
        Set rec1 = db.OpenRecordset(sql1, dbOpenDynaset)
        If Not rec1.EOF Then
            rec1.MoveFirst
            For i = 0 To FragSeqCount - 1
                rec1.Edit
                rec1!SubSeqNum = i
                rec1.Update
            Next i
        End If
    Wend
End If

```

```

        rec1.MoveNext
    Next i
End If
rec1.Close
rec.MoveNext
Wend
End If
rec.Close
Return

```

```

PRouter_Error:
result = MsgBox("Router")
MsgBox (Error(Err))
Exit Function
End Function

```

Function P2Source(Net As Integer, PId As Integer, PackNo As Integer, LvId As Integer)

```
On Error GoTo P2Source_Error
```

```

Dim sql As String
Dim rec As DAO.Recordset
Dim db As Database
Dim ObjId As String
Dim i, k
Dim PathNo As Integer
Dim PathSize As Long
Dim path As String
Dim prec As DAO.Recordset

```

```
Select Case Net
```

```
Case 1
```

```

    ObjId = "S1"
    GoSub Net1Source
    result = PPath("tmpSiFragSources", Network, ProcessId, Level, ObjId)
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "qrySiAppSourceData"
    DoCmd.OpenQuery "qrySiDelSourceData"
    DoCmd.SetWarnings True

```

```
Case 2
```

```

    ObjId = "S2"
    GoSub SourceReassembly
    GoSub Net2Source
    result = PPath("tmpSiFragSources", Network, ProcessId, Level, ObjId)
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "qrySiAppSourceData"
    DoCmd.OpenQuery "qrySiDelSourceData"
    DoCmd.SetWarnings True

```

```
End Select
```

```
Exit Function
```

```
Net1Source:
```

```

Set db = DBEngine(0)(0)
sql = ""
sql = sql & "SELECT tblSiPackets.PacketId, tblSiPackets.Size, tblSiPackets.DestinId, "
sql = sql & "tblSiPackets.Poisson FROM tblSiPackets ORDER BY tblSiPackets.PacketId; "

```

```

Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
        GoSub SelectPath
        GoSub FragPacket 'add paco 6/5/2001
        ObjId = "S1"
        result = DivPacket(rec!PacketId, rec!Poisson, PId, LvId, FragNo, rec!Size, ObjId, PathNo,
            PathSize)
        rec.MoveNext
    Wend
End If
rec.Close
Return

SourceReassembly:
Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpS2FragSourceReasmb", dbOpenDynaset)
sql = ""
sql = sql & "SELECT tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "tblSiFragments.FragId, Sum(tblSiFragments.FragSize) AS [Size], "
    First(tblSiFragments.SeqNum) AS NumSeq, "
sql = sql & "Max(tblSiFragments.LevelTime) AS DestTime FROM tblSiFragments "
sql = sql & "WHERE (((tblSiFragments.LevelId) = " & (LvId - 1) & ") and tblSiFragments.Destin = "
    & ObjId & ") "
sql = sql & "GROUP BY tblSiFragments.ProcessId, tblSiFragments.PacketID, "
sql = sql & "tblSiFragments.FragId ORDER BY tblSiFragments.ProcessId, "
sql = sql & "tblSiFragments.PacketID, Max(tblSiFragments.LevelTime); "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    While Not rec.EOF
        rrec.AddNew
        rrec!ProcessId = rec!ProcessId
        rrec!PacketId = rec!PacketId
        rrec!FragID = rec!FragID
        rrec!LevelId = LvId
        rrec!SeqNum = rec!NumSeq
        rrec!SubSeqNum = rec!NumSeq
        rrec!fragSize = rec!Size
        rrec!LastPSize = 0
        rrec!ObjectId = ObjId
        rrec!StartTime = rec!DestTime
        rrec!ObjectTime = 0
        rrec!ObjectDelay = 0
        rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
        rrec!pathid = 0
        rec.Update
        rec.MoveNext
    Wend
End If
rrec.Close
rec.Close
Return

```

Net2Source:

```

Set db = DBEngine(0)(0)
Set rrec = db.OpenRecordset("tmpSiFragSources", dbOpenDynaset)
sql = ""
sql = sql & "SELECT tmpS2FragSourceReasmb.ProcessId, tmpS2FragSourceReasmb.PacketID, "
sql = sql & "tmpS2FragSourceReasmb.FragId, tmpS2FragSourceReasmb.LevelId, "
      tmpS2FragSourceReasmb.SeqNum, "
sql = sql & "tmpS2FragSourceReasmb.FragSize, tmpS2FragSourceReasmb.LastPSize, "
      tmpS2FragSourceReasmb.StartTime, "
sql = sql & "tmpS2FragSourceReasmb.ObjectId, tmpS2FragSourceReasmb.ObjectTime, "
      tmpS2FragSourceReasmb.ObjectDelay, "
sql = sql & "tmpS2FragSourceReasmb.ObjectTTime, tmpS2FragSourceReasmb.PathId, "
      tmpS2FragSourceReasmb.PathTime, "
sql = sql & "tmpS2FragSourceReasmb.PathDelay, tmpS2FragSourceReasmb.PathTTime, "
      tmpS2FragSourceReasmb.Destin, "
sql = sql & "tmpS2FragSourceReasmb.LevelTime FROM tmpS2FragSourceReasmb WHERE "
      (((tmpS2FragSourceReasmb.ProcessId)=" & PId & ") AND "
sql = sql & "((tmpS2FragSourceReasmb.LevelId)=" & LvId & ") AND "
      ((tmpS2FragSourceReasmb.ObjectId)=" & ObjId & ") "
sql = sql & "ORDER BY tmpS2FragSourceReasmb.ProcessId, tmpS2FragSourceReasmb.LevelId, "
      tmpS2FragSourceReasmb.Destin, tmpS2FragSourceReasmb.StartTime; "
Set rec = db.OpenRecordset(sql, dbOpenDynaset)
If Not rec.EOF Then
    rec.MoveFirst
    k = 1
    initTime = rec!StartTime
    PrevSTime = rec!StartTime
    PrevSDelay = 0
    While Not rec.EOF
        If k > 1 Then
            rec.Edit
            initTime = rec!StartTime
        End If
        rrec.AddNew
        rrec!ProcessId = PId
        rrec!PacketId = rec!PacketId
        rrec!FragID = rec!FragID
        rrec!LevelId = LvId
        rrec!SeqNum = rec!SeqNum
        rrec!fragSize = rec!fragSize
        rrec!LastPSize = rec!LastPSize
        rrec!ObjectId = ObjId
        rrec!StartTime = rec!StartTime
        r = Rnd
        While r = 0
            r = Rnd
        Wend
        ServTime = RoundNum((-1) * Log(1 - r), 4)
        rrec!ObjectTime = ServTime
        If ((PrevSTime + PrevSDelay) - initTime) <= 0 Then
            rrec!ObjectDelay = 0
        Else
            rrec!ObjectDelay = RoundNum((PrevSTime + PrevSDelay) - initTime, 4)
        End If
    End While
End If

```

```

rrec!ObjectTTime = RoundNum(rrec!StartTime + rrec!ObjectTime + rrec!ObjectDelay, 4)
GoSub SelectPath
rrec!pathid = PathNo
PrevSTime = RoundNum(rrec!StartTime + rrec!ObjectTime, 4)
PrevSDelay = rrec!ObjectDelay
rrec.Update
k = 0
rec.MoveNext
Wend
End If
rec.Close
rec.Close

```

Return

SelectPath:

```

Set db = DBEngine(0)(0)
path = ""
path = path & "SELECT tblSiPaths.PathId, tblSiPaths.MTU, tblSiPaths.Bandwidth AS MaxBand "
path = path & "FROM tblSiPaths WHERE (((tblSiPaths.NetworkId) = " & Net & ") And "
path = path & "((tblSiPaths.LevelId) = " & LvId & ") And ((tblSiPaths.FromSource) = " & ObjId &
"")) "
path = path & "ORDER BY tblSiPaths.Bandwidth DESC; "
Set prec = db.OpenRecordset(path, dbOpenDynaset)
If Not prec.EOF Then
    'set path id equal to this path
    prec.MoveFirst
    PathNo = prec!pathid
    PathSize = prec!MTU
    prec.Close
Else
    'set path equal to highest path available
    PathNo = DLookup("PathId", "qryS2LargePathMTU", "NetworkId = " & Net & " and LevelId =
    " & LvId & """)
    PathSize = DLookup("MaxSize", "qryS2LargePathMTU", "NetworkId = " & Net & " and
    LevelId = " & LvId & """)
    GoSub FragPacket    'divide packet according to this path
    prec.Close
End If

```

Return

FragPacket:

```

FragNo = Round(rec!Size / PathSize, 0)
If FragNo < 1 Then
    FragNo = 0
End If

If FragNo = 1 Then
    If (rec!Size Mod PathSize) <> 0 Then
        FragNo = FragNo + 1
    Else
        FragNo = 0
    End If

```

```

    End If
Return

P2Source_Error:

    result = MsgBox("Source")
    MsgBox (Error(Err))
Exit Function
End Function

Function Process1()
On Error GoTo Process1_Err

ProcessId = 2

Network = 1
    Level = 1
        result = P2Source(Network, ProcessId, PackNo, Level)    'Source
    Level = 2
        result = PRouter(Network, ProcessId, PackNo, Level)    'Router
    Level = 3
        result = PHosts(Network, ProcessId, PackNo, Level)    'Host
    Level = 4
        result = PGateway(Network, ProcessId, PackNo, Level)    'Gateway

Network = 2
    Level = 5
        result = P2Source(Network, ProcessId, PackNo, Level)    'Source
    Level = 6
        result = PRouter(Network, ProcessId, PackNo, Level)    'Router
    Level = 7
        result = PDestination()    'Destination

Exit Function

Process1_Err:
    result = MsgBox("Process 1")
    MsgBox (Error(Err))
Exit Function
End Function

Private Sub Go_Click()
On Error GoTo Go_Click_err
Dim NbrOfProcess As Integer
Dim result, i

    result = GeneratePackets()
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "qryS2FragHostReasmbDel"
    DoCmd.OpenQuery "qryS2FragSourceReasmbDel"
    DoCmd.OpenQuery "qryS2FragRoutersReasmbDel"
    DoCmd.SetWarnings True
    OptId = 1
    result = Process1()

```

```
    MsgBox "Simulation Ended"
Exit Sub

Go_Click_err:
    result = MsgBox("Go Click")
    MsgBox (Error(Err))
Exit Sub
End Sub

Function RoundNum(InVal, Places) As Double
Dim Factor As Double
Dim temp As Double

If IsNull(InVal) Then InVal = 0
If IsNull(Places) Then Exit Function

If Places < 0 Then Places = 4

Let Factor = 10 ^ Int(Places)
temp = Abs(InVal) * Factor + 0.5000000001
RoundNum = (Int(temp) / Factor) * Sgn(InVal)

End Function
```