HOP-BY-HOP FLOW CONTROL WITH PACKET AGGREGATION IN TCP/IP NETWORKS

By

WALID A. KAMOUH

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Natural and Applied Sciences

Notre Dame University

July 2000

HOP-BY-HOP FLOW CONTROL WITH PACKET AGGREGATION IN TCP/IP NETWORK

BY

WALID A. KAMOUH

Approved by

[illegible]

Hoda Maalouf: Assistant Professor of Computer Science.
Advisor.

[illegible]

Fouad Chedid: Associate Professor of Computer science. Chairperson.
Member of committee.

[illegible]

Khaldoun el-Khalidi: Assistant Professor of Computer Science.
Member of Committee.

[illegible]

Roger Hajjar: Assistant Professor of Physics.
Member of Committee.

ABSTRACT

The Internet is once again suffering from its own success. Since 1995, there has been a massive increase in demand for Internet services, resulting in an exponential growth of the Internet. We have entered now an era where the users of the Internet are unable to obtain the bandwidth needed to support their applications, and they are experiencing high packet loss.

Packet loss problem arises whenever the number of packets arriving at a given router is much higher than its buffering space. The main goal of this thesis is to find adaptive schemes capable of optimizing the communication network performance of systems with buffering constraints such as TCP/IP networks. A new strategy, called hop-by-hop flow-control with packet aggregation (HFCPA), is devised for optimizing the network performance.

HFCPA is a variation of the current TCP/IP protocol. Flow control with aggregation of packets is implemented in the main routers of the network, in contrary to the current TCP/IP protocol where the flow control occurs at the edge of the network (i.e. at the end-users). In fact, the Internet will be subdivided into tunnels where the edge routers of each tunnel are responsible for the packet management inside the tunnel (aggregation, fragmentation, bandwidth allocation, path selection...).

With this newly proposed technique, we have shown that TCP connections experience lower packet loss and higher throughput compared with normal TCP/IP implementation, especially during network congestion.

TABLE OF CONTENTS

List of figures .... page

List of tables .... page

ACKNOWLEDGMENTS

This thesis is the culmination of one long year of work on my part. I accept full responsibility for any shortcomings, but I would like to share whatever credit may be due with all those individuals who helped me along the way.

I would like to express my sincerest gratitude and appreciation for my thesis advisor, Dr. Hoda Maalouf, for her guidance, support and patience. The successful completion of this thesis is in a large part due to her encouragement, probing criticisms and judicious advice.

I am also grateful to the faculty and staff of the faculty of Natural and Applied Sciences for making my years of graduate study at Notre Dame University very enjoyable.

"A friend in need is a friend indeed." I deeply appreciate the help and encouragement of Nicolas Smayra and his offer of friendship during what was a difficult time for me.

I am eternally grateful to my parents, my uncle Fawzi, and my brothers for their priceless gift of love throughout my whole life. My strength and determination is but a product of their efforts.

My debt to my loving girlfriend for her selfless support. There is no doubt that her friendship made it possible for me to fulfill my dream of acquiring a graduate degree.

CHAPTER 1
INTRODUCTION AND PROBLEM DEFINITION

1.1 Introduction

The Internet is a global data network connecting millions of computers, and is rapidly growing. Whenever Internet computers wish to communicate with one another, they divide the data they wish to exchange into a sequence of packets (or datagrams) that they inject into the network.

The Internet's infrastructure consists of a series of "routers" inter-connected by "links". The routers examine each packet they receive in order to determine the next "hop" (either another router or the destination computer) to which they should forward the packet so that it will ultimately reach its destination. Sometimes routers receive more packets than they can immediately forward. In this case they momentarily queue the data in "buffers", increasing the delay of the packets through the network; or sometimes they must drop incoming packets, not forwarding them at all.

The specifics of how to format individual packets for transmission through the network form one of the Internet's underlying "protocols". This fundamental one is called the Internet Protocol, or simply IP [RFC791]. Another important protocol, Transmission Control Protocol (TCP), regulates other facets of Internet communication, such as how to divide streams of data into individual packets such that the original data can be delivered to the receiving computer intact, even if some of the individual packets are lost due to drops or damage.

Since its launch two decades ago, TCP [RFC793] remains the most dominant end-to-end transport protocol in the Internet and is likely to be a widely used protocol in the next century. Most popular Internet applications, such as world wide web, file transfer, e-mail and so on, use the reliable services provided by TCP. The performance perceived by the users of these applications depends largely on the performance of TCP. Some of the questions that still arise in the analysis of TCP/IP network performances are: how TCP congestion control algorithms interact with the underlying network protocols? Will TCP protocol processing overhead become the bottleneck over high-speed networks? Can we continue to rely on packet drop at the routers as congestion indication for the TCP for high-speed access links? What role the next generation protocols can play to help TCP making the best use of the network bandwidth?

The goal of this research is to devise some new techniques to be used in routers. These techniques aim to improve TCP performance and to optimize end-to-end traffic management in TCP/IP Network.

1.2. Problem Definition

Internet Protocols suite, or TCP/IP, was designed for use with many different kinds of network. Unfortunately, network designers do not agree about how big packets can be. Wide Area Networks tend to use small packet sizes, while Local Area Networks (which are much faster networks) have much larger packet sizes. At first, one might think that IP should simply settle on the smallest possible size. Unfortunately, this would cause serious performance problems. When transferring large files, big packets are far more efficient than small ones. So we want to be able to use the largest packet size possible. But we also want to be able to handle networks with small packet size constraints.

One of the greatest difficulties involved in communication networks is the need to interconnect different networks to form a unique inter-network. A major problem is the differing maximum packet sizes allowed by each network. The Internet model requires a router to be able to fragment packets as necessary to match the Maximum Packet Size

(MPS) of the network to which they are being forwarded. Reassembly of fragmented packets is generally left to the destination host [RFC815].

In the Internet, fragmentation is usually done at IP-level protocol at the sending host node. Although IP-level fragmentation often makes it possible to interconnect two dissimilar networks, but it is usually avoided. One reason is that when a single IP fragment is lost, all other fragments belonging to the same datagram (IP packet) are effectively also lost and the entire datagram must be retransmitted by the source. Even without loss, fragments require the allocation of temporary buffer memory at the destination, and it is never easy to decide how long to wait for missing fragments before giving up and discarding those that have already arrived.

A common method used by some implementations of TCP/IP is to fragment an IP datagram into IP fragments that are no larger than 576 bytes when the IP datagram is to travel through a router. This is intended to allow the resulting IP fragments to pass the rest of the path without further fragmentation. This would, though, create more of a load on the destination host, since it would have a larger number of IP fragments to reassemble into one IP datagram. It would also not be efficient on networks where the maximum transmission unit (MTU) only changes once and stays much larger than 576 bytes. Examples include networks such as an IEEE 802.5 network with an MTU of 2048 or an Ethernet network with an MTU of 1500.

Also, the growth of the global Internet has brought with it an increase in "undesirable elements" manifested in antisocial behavior. Recent months have seen the use of novel attacks on Internet hosts, which have in some cases led to the compromise of sensitive data. Increasingly sophisticated attackers have begun to exploit the more subtle aspects of the Internet Protocol; fragmentation of IP packets, an important feature in heterogeneous internetworks, poses several potential problems, which we will explore in this thesis.

In this thesis the problem of fragmentation and reassembly of packets in the Internet is analyzed in details. In fact, our main concern is to modify the reassembly process so that the Internet available capacity is used optimally.

1.3. Research Objectives

This research concentrates on the analysis of Internet fragmentation /reassembly process and its optimization. In particular, we will do the following:

- Optimize packet size and maximize the throughput. A host sending datagrams much smaller than the Path MTU allows is wasting Internet resources and throughput.
- Avoid network congestion and minimize end-to-end delay.
- Minimize random packet loss. In fact, packets often arrive at routers in bursts, where they are queued. Periodically a burst of packets fills up a queue and the router drops packets that arrive while this queue is full.
- Improve network security.

1.4. Approach

The current Internet does not provide any form of guarantee on packet delay or loss. Packets may be dropped or delayed at routers along a source to destination path as a result of network congestion. The approach considered in this thesis for handling such impairments is to adapt queues behavior in response to changing network conditions (path and bandwidth). In fact, our main task is to establish and maintain high throughput over a path or a distribution tree, independent of how the path or tree was created.

In fact, we are tackling problems in the existing Internet by using intelligent routers spread at key access and interchange points to "tunnel" traffic through the Internet. These intelligent routers can improve performance and availability by aggregating traffic information, shaping bursty traffic flows, and using more efficient routes.

First, we suggest (whenever possible) the usage of large packets instead of many small packets to carry a given amount of data, because much of the cost of communication is per packet rather than per byte. Throughput can increase almost linearly with packet size over a wide range of sizes.

Second, we suggest the aggregation of packets at the intelligent routers. Hop-by-hop aggregation will be used since the packet scheduling state will be significantly reduced with this method. In fact, only a few aggregates exist for each next hop. The solution proposed involves the aggregation of several end-to-end packets that cross an "aggregation region" and share common routers into one larger datagram. We define an "aggregation region" as a contiguous set of systems capable of performing aggregation along any possible route through this contiguous set. Routers that have at least one interface in the region can aggregate, deaggregate, or they are between an aggregator and a deaggregator.

This method would be of particular benefit for high-bandwidth large-propagation-delay TCP connections, such as those over satellite links. In fact, when a normal packet arrives at its destination, a receiver employing delayed acknowledgements (ACK) [Bra89] is forced to wait for a timeout before generating an ACK. With an aggregated packet of at least two packets, the receiver will generate an ACK after the data segment arrives. This eliminates the wait on the timeout (often up to 200 ms). Another benefit of the aggregation technique is the reduction in the level of packet loss rate at routers since it reduces the overall number of packets in the network in general and in the queues in particular.

Third, we will assume that Intranet fragmentation is used in the Internet. Intranet fragmentation provides a mechanism for deciding the actual packet size as late as possible. It especially allows protocols to avoid choosing to send small datagrams until absolutely necessary. Protocols can choose large segment sizes to take advantage of the large MTU in a local network, and rely on fragmentation at routers to send the segments through networks with small MTUs when needed. If datagrams must traverse a route consisting of several high MTU links followed by a low MTU link, by delaying the use of small packets in the queues for aggregation, intranet fragmentation allows the use of large packets on the initial high MTU links, and thus uses those links more efficiently.

1.5 Main Results

Due to the network's complexity, simulation models are e used to analyze the performance of our suggested techniques. These models characterize how different facets of the network behave, and how proposed changes might affect the network's different properties. Our simulation shows that Hop-by-Hop Flow Control with Packet Aggregation HFCPA ouperforms the current TCP/IP in throughput, packet loss and deals better with the Internet traffic in case of congestion. (See chapter 5 for more details).

1.6 Thesis Organization

The thesis is made of six chapters. In Chapter II, we examine the behavior of the TCP and IP protocols concerning fragmentation, reassembly, routing, security, and congestion. In chapter III, we introduce our new HFCPA protocol by examining the encapsulation process, optimal path, security, the aggregation and disaggregation concept. Chapter IV highlights the queuing management and describes the HFCPA queue. Chapter V discusses the results of the simulation implementing the new protocol. The thesis is finally concluded with a discussion of our results and provides some open questions for further research.

CHAPTER 2

Background and Motivation

2.1 INTRODUCTION

The problems dealt with in this thesis arise in TCP/IP networks. In particular, we are focusing on the problems of fragmentation, reassembly and flow control in the Internet.

In section 2.2 of this introductory chapter, we provide some necessary definitions. We then provide (in sections 2.3, 2.4), the necessary background on TCP/IP protocols, where an example that describes the fragmentation procces in the Internet is also given.

In section 2.5 the main performance evaluation results of TCP protocol are described briefly. Finally, in section 2.6, a brief discussion on what motivated our research is given.

2.2 DEFINITIONS

2.2.1 Maximum Transmission Unit (MTU)

The MTU is a parameter that limits the size of the largest IP datagram that may be handled. IP datagrams arriving at a router that are larger than its MTU are each split into two or more fragments. The minimum acceptable interface MTU is 28 bytes: 20 bytes for the IP (fragment) header, plus 8 bytes of data.

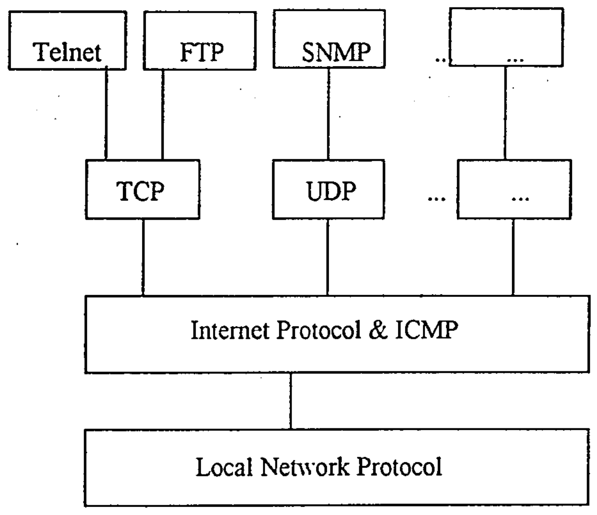
2.2.2 Tunnel

We define a tunnel as an alternate route of the original routing behavior. Tunnels are needed to bypass routing failures or to avoid congested routes. The tunnel "entry" router put an outer IP header in front of the original IP header of each packet, giving all intermediate routers an illusion that packets are delivered between the router source and destination addresses. When the encapsulated packet reaches the tunnel exit point, the outer header is thrown away and the packet is further routed towards its final destination.

2.3 TCP / IP

The Internet protocols are very popular open-system (nonproprietary) protocol suite because they can be used to communicate across any set of interconnected networks and are equally well suited for LAN and WAN communications. The Internet protocols consist of a suite of communication protocols, of which the two best known are Transmission Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite not includes lower-layer protocols (such as TCP and IP), but it also specifies common applications such as electronic mail, terminal emulation and file transfer.

The following diagram illustrates the place of the Internet Protocol (IP) in the protocol hierarchy:



Protocol Relationships

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This process is done by passing the datagrams from one network to another until the destination is reached. The TCP/IP layer architecture does not mean that higher layers must be kept ignorant of the IP issues. In fact, optimal performance depends on the cooperation between layers: for example, layers above IP should be involved in routing and fragmentation scheme, in addition that the TCP layer should not send datagrams in an aggressive way if the there is a congested route along the path.

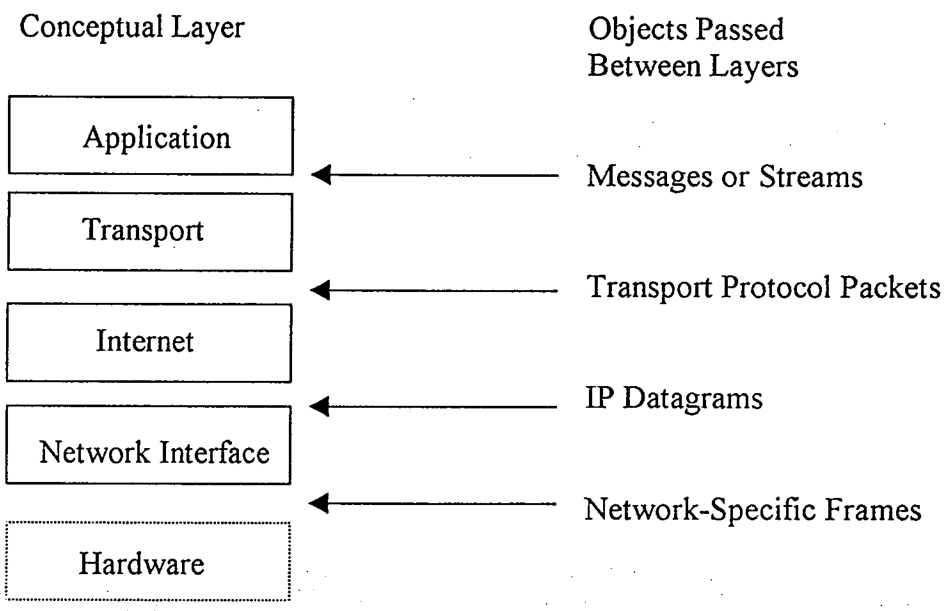


Figure1: The 4 conceptual layers of TCP/IP software above the hardware layer, and the form of objects passed between layers.

Next, we will give an overview of the IP and TCP protocols, in addition to their main functions in the current Internet architecture.

2.4 INTERNET PROTOCOL (IP)

The Internet protocol (IP) implements two basic functions: routing and fragmentation. In fact, IP is responsible for transmitting blocks of data called datagrams from a source to a destination identified by an IP address in the IP header, without a prior notification of the routed path. It is also the responsibility of IP to fragment and reassemble datagrams when necessary. However, IP is not highly reliable since there is no sequencing, flow control or other services found in other host-to-host protocols.

2.4.1 IP Header Fields

Next, we will give a brief description of some of the IP header fields that will be used in this thesis.

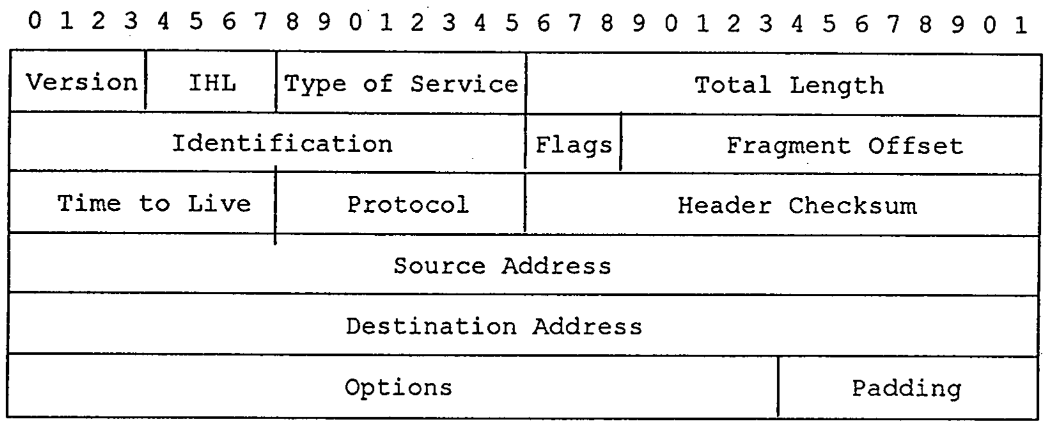


Figure2: Internet Datagram Header

The main fields are:

- *Time to Live Field*: Every packet in the Internet is given an amount of time to reach its destination, which is set by the sender and reduced on every hop. If the time reaches zero before reaching the destination, the datagram is destroyed, and is resent again.

- *Identification Field*: this field contains a unique integer that identified the datagram. It allows the destination to know which arriving fragments belong to which datagrams since each fragment has the same identification number as the other fragment belonging to the same datagram. In other words, it is used to distinguish the fragments of one datagram from those of another.

- *The fragment-offset field* tells the receiver the position of a fragment in the original datagram.

- *The More Fragments flag bit* (MF) is set if the datagram is not the last fragment.

We should notice that an unfragmented datagram has zero values in its all fragmentation information fields (i.e. MF = 0, fragment offset = 0).

2.4.2 Internet Routing

The Internet is partitioned into a number of autonomous systems (AS's), where routers within an AS route packets according to an interior gateway protocol (IGP); IGP is used for selecting paths within an AS, and each AS is free to use its own metrics for selecting these internal routes. When leaving one AS, a packet is managed by a separate Exterior Gateway Protocol (EGP) that is responsible for the communication to all other AS. A newer version of EGP, called Border Gateway Protocol (BGP) [CHS99] does not necessarily select routes by minimizing some global metric such as hop count or delay. Instead, the network administrators at each AS define a "routing policy" that selects routes to favor certain AS.

2.4.3 Fragmentation and Reassembly Process

In this paragraph, we explain why Internet fragmentation is needed. In fact, when a TCP connection is established between two hosts, these two hosts will negotiate about the maximum datagram size they can handle. The smaller of these numbers is used for the rest of the connection. The most serious problem is that the two ends don't necessarily know about all of the steps in between. For example, when sending data between site A and site B, it is likely that both computers will be on Ethernets. Thus they will both be prepared to handle 1500-octet datagrams. However the connection will at some point end up going over a smaller MTU network, that cannot handle packets of that size. For this reason, fragmentation is needed. In this case, the Ethernet packets are usually split into pieces that fit the Internet.

To be on the safe side and to minimize the fragmentation cost, TCP/IP implementors suggested a 576-byte datagram size to be used along the Internet. This datagram size will be divided for 20 bytes for a TCP header (on average), 20 bytes for an IP header (on average) and 536 bytes for data. This conservative strategy is used to avoid any further fragmentation at intermediary nodes, and also to avoid the bugs of the new implementation of reassembly code. Thus, 576 bytes is a "safe" size, which every implementation must support [RFC-791].

2.4.3.1 *Fragmentation*

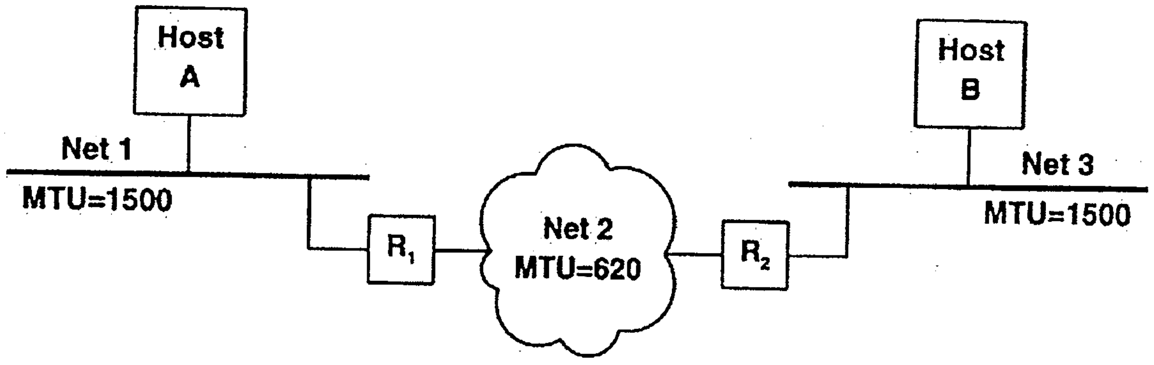


Figure3: Case of occuring fragmentation

As mentioned earlier, 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.

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). If the second fragment might not be multiple of 8, the first must be. We refer to the number of 8 octet blocks in the first portion as the Number of Fragment Blocks (*NFB*). 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.This procedure can be generalized for an n-way split, rather than the two-way split described above.

It is important to note that an IP datagram can be marked "don't fragment". So, if this bit is set, the datagram is not to be fragmented under any circumstances. If a datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is discarded, and is resent again.

2.4.3.2 *Reassembly*

The reassembly process is responsible to group the fragment packets at the destination host. It requires four basic fields in the Internet header to be able to resequence the fragments: Identification, source, destination, and protocol fields. The fragments having the same identification correspond to the same packet; Fragment with offset zero will be the first fragment, and the last fragment will have the more-fragments flag set to zero.

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; If the timer runs out, the all reassembly resources for this buffer identifier are released. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds.

In the case that two or more fragments contain the same data either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

2.4.4 Example on Fragmentation and Reassembly

In this example, we show first an average size Internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum transmission size allowed is 280 octets.

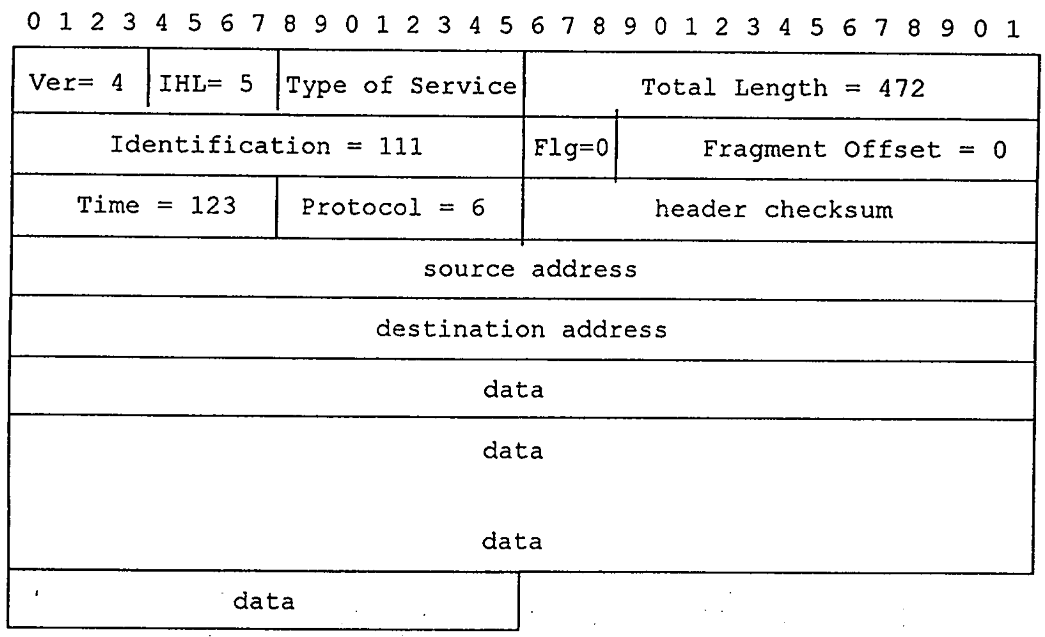


Figure4: Internet Datagram

Now the first fragment that results from splitting the diagram after 256 data octets has the following format:

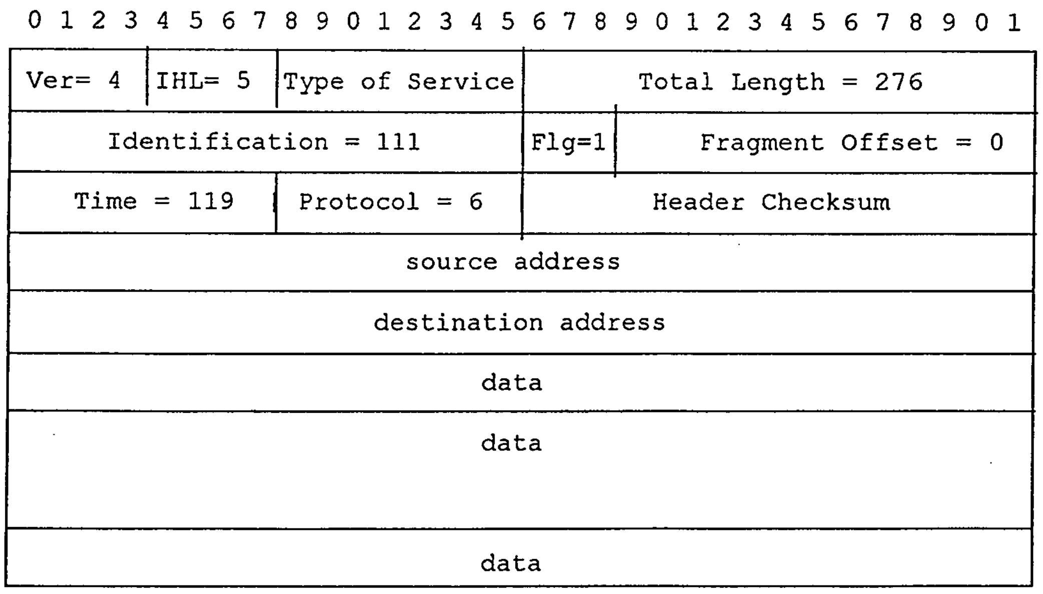


Figure5: Internet Fragment No. 1

And the second fragment has the following format:

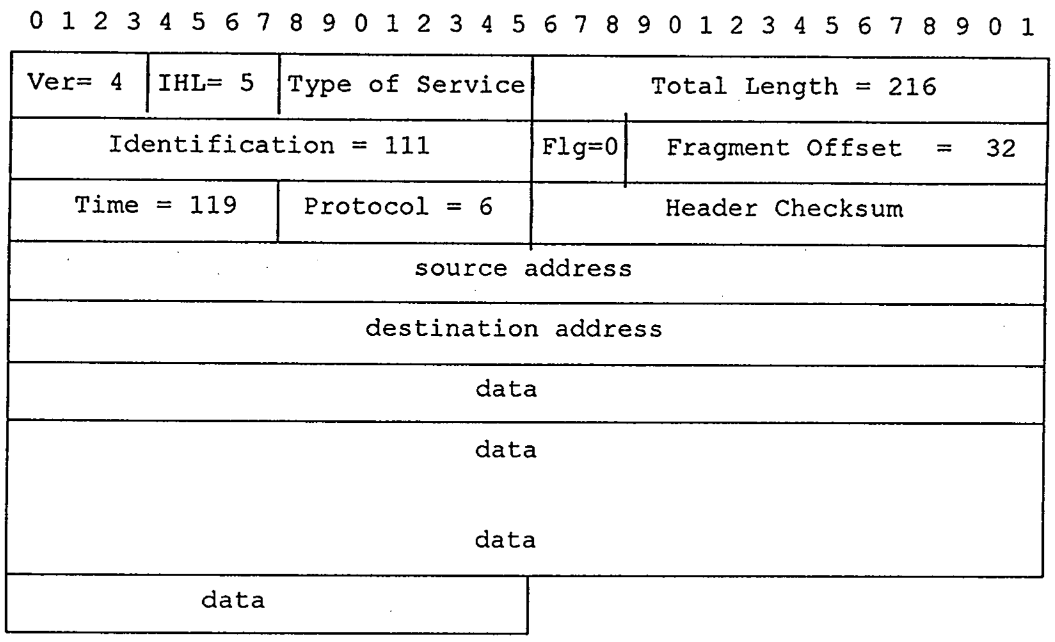


Figure6: Internet Fragment No.2

2.4.5 Connectionless Unreliable IP

It is important to note that the Internet protocol is defined as unreliable, connectionless packet delivery system. It is connectionless since each packet in the Internet is treated independently, unrelated to any other Internet datagram. And, it is unreliable since delivery is not guaranteed; i.e the packets may be lost, duplicated, arrived out of order, without the notice of the sender nor the receiver.

Data is injected in the Internet as a sequence of datagrams where each datagram is sent to its destination independent of the others. For example, suppose you want to transfer a file of size 45000 bytes. Most networks cannot handle a 45000 bytes datagram. So the protocols will break this up into something like 90 500-bytes datagrams. Each of these datagrams will be sent to the other end. At that point, they will be put back together into the file of 45000-bytes.

However, while those datagrams are in transit, the network doesn't know that there is any connection between them. It is perfectly possible that datagram 14 will actually arrive before datagram 13. It is also possible that somewhere in the network, an error will occur, and some datagram won't get through at all. In that case, that datagram has to be sent again.

2.5 Transmission Control Protocol (TCP)

If IP protocol is responsible only for routing and fragmentation, TCP is the core of the Internet. In fact, in addition to ensuring that all the data sent from the source reaches its destination, TCP is responsible for deciding when and how fast to send data and this is a hard task to perform since it should adapt to the available bandwidth in different subnetworks across the Internet. Slowing down the sending rate when a congestion occurs and speeding up when the bandwidth increases are the major issues of the TCP protocol. As a result, over the last fifteen years TCP has grown increasingly complex.

2.5.1 The Mechanics of TCP

To implement its service, TCP gives each byte in the stream a unique 32-bit sequence number. Then it divides the sequence of bytes into segments of some maximum segment size (MSS) which is typically 536 bytes [TMW97]. Each TCP segment is placed in an IP packet and then sent to the receiver.

Before sending any data, the two end-points (or hosts) must establish a connection between themselves by a three-way hand-shake (see figure 7 below). First, the connection initiator sends a "synchronize" sequence number segment (SYN) with its initial sequence number to the other side. Then the other side responds by sending a segment with an acknowledgment ACK for this sequence number. Finally, the initiator sends an ACK segment acknowledging the other side's initial sequence number.

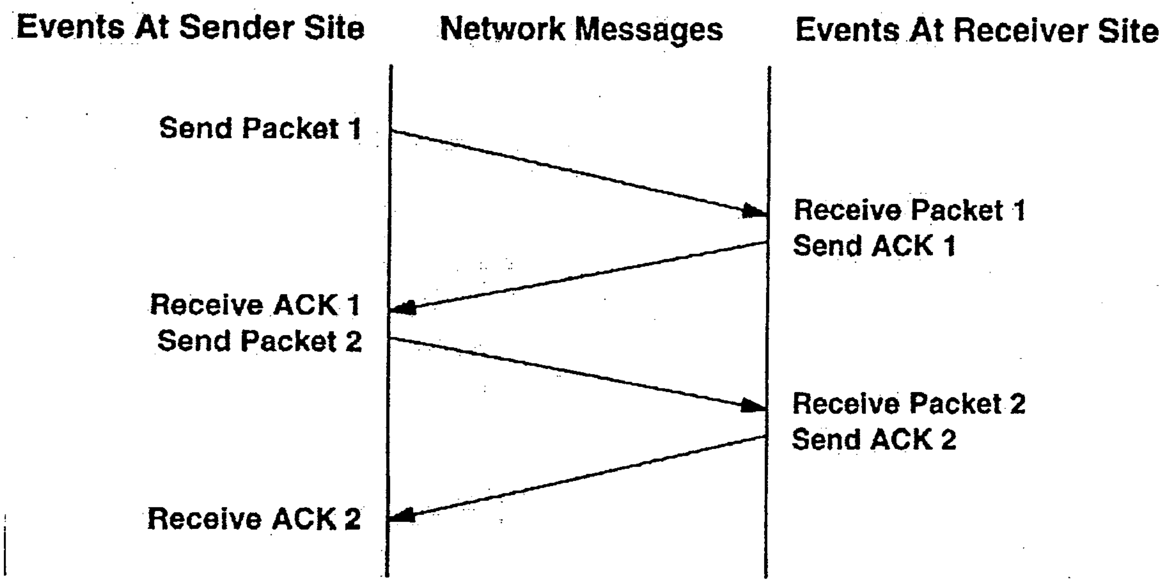


Figure7: TCP model 3-way Hand-Shake

Once ,the connection has been established, the sender begins sending data segments. When the receiver receives a data segment, it sends back an acknowledgment that specifies the highest in-sequence number that it has received. The sender then knows that every byte up to that sequence number has been received. For example, if a receiver receives segments 1 and 2 and 4, it will acknowledge segment 2, since there is a hole where segment 3 should be. In general, not every packet is acknowledged upon its receipt. However, a "delayed ACK" is usually used upon receipt of the second segment and sometimes on the fourth segment.

As the sender is sending data segments, it must decide when and how fast to send them. For this, TCP uses a window based scheme, The TCP sender always sends as much data as its window allows, and then waits for an acknowledgment.

Moreover, after an occurrence of a congestion, TCP uses a "Slow Start" mode where TCP sets its windows to one MSS and increases by one MSS for each acknowledgment segment received. To avoid increasing the window size too quickly, TCP enters a "congestion avoidance" phase where it increases its window size by one only if all segments in the window have been acknowledged [Ste97, RFC2581].

Finally, during a normal TCP connection, we can observe the following TCP aspects:

- *The detection of lost packets*: In fact, a lost packet is usually a dropped packet since it arrives at a router faster than it can leave. Since TCP assumes that every loss is a congestion loss and reduces its transmission rate in a direct way, TCP throughput is very able to drop. For a fixed loss rate, the throughput goes down as the inverse square of the bandwidth-delay product [MKS98].
- *The creation of burst traffic*: This is because data transmission in TCP is normally acknowledged with every packet. If the packets have small sizes this will create burstiness of traffic. Also, TCP slow start is very bursty due to the doubling of the window sizes every round-trip time.

2.5.2 TCP Model

We can easily estimate TCP's performance by making some major simplifications [MSM97].

Let p be the constant probability of a random packet loss; therefore, the link delivers approximately $1/p$ consecutive packets followed by one drop.

Let the maximum value of the window be W packets. Then by the definition of Congestion Avoidance, we know that the minimum window must be

$W/2$ packets.

If the receiver is acknowledging every segment, then the window opens by one segment per round trip, so each cycle must be W/2 round trips, or

RTT * W/2 seconds.

The total data delivered is

$$\left(\frac{w}{2}\right)^2 + \frac{1}{2}\left(\frac{w}{2}\right)^2 = \frac{3}{8}w^2$$ packets per cycle.

By assumption, each cycle also delivers 1/p packets. Solving for W we get:

$$W = \sqrt{\frac{8}{3p}}$$

Substitute W into the bandwidth equation below:

$$BW = \frac{Data\ per\ cycle}{Time\ per\ cycle} = \frac{MSS \times \frac{3}{8}w^2}{RTT \times \frac{w}{2}} = \frac{MSS/p}{RTT\sqrt{\frac{2}{3p}}}$$

Collect the constants in one term, $C = \sqrt{3/2}$, then we have:

$$BW = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \qquad \textbf{Eq (1)}$$

where:

MSS is the maximum segment size, *RTT* is the round trip time or delay, p the constant probability of a random packet loss and C a constant depending on the window mechanism.

Other forms of this derivation have been published in [Flo91,LM94]

2.5.2.1 Example

Suppose a user needs to move 1 Gigabyte of data in 2 hours. This requires a sustained transfer rate of about 1 Mb/s. The question will arise here is that what loss rate does the user need to meet this requirement?

Assuming C < 1:

$$p < \left(\frac{MSS}{BW * RTT}\right)^2 \qquad \textbf{Eq (2)}$$

The model predicts that the user needs a loss rate better than 0.18 % (i.e. $p = 0.0018$) with 536 byte packets and a *RTT* equal to 100 ms. At 1460 bytes, the maximum loss rate rises to 1.4%. If the user upgrades to FDDI and uses 4312 byte packets, Equation 2 suggests that the network only needs to have less than 11% loss.

2.6 RESEARCH MOTIVATION

Congestion collapse observed in the mid 1980's is the primarily result of lack of attention of packet forwarding. Van Jacobson was the first to treat this problem in 1988. His work on causing TCP connections to back off during congestion i.e. dropping packets from the network, was a revolution on TCP congestion avoidance algorithm. However, that is not the end of the story. Considerable research has been done on Internet dynamics since 1988, and the Internet has grown. It has become clear that the TCP congestion avoidance mechanisms [RFC2581], while necessary and powerful, are not sufficient to provide good service in all circumstances.

Another form of congestion collapse occurs due to undelivered packets. Dropping packets before reaching their destination is probably the largest unresolved danger in the Internet today. Therefore, these packets waste network bandwidth and make other packets also to be delayed or dropped [FKS98]. Even more destructive would be best-effort applications that increase their sending rate in response to an increased packet drop rate .

This has motivated us to the development of a new congestion control mechanism which is based on a combination of aggregation mechanism together with intranet fragmentation. The aggregation of several small packets into one larger packet will decrease the number of data and acknowledgement packets in the network and will have the following effects:

- Decrease the congestion in the network and in the routers,
- Minimise the packet loss rate,
- Improve the throughput of the system.

CHAPTER 3

HFCPA Protocol

3.1 INTRODUCTION

The basic architecture of the Internet consists of communication nodes connected to each other via a wide area communication network (figure 8).

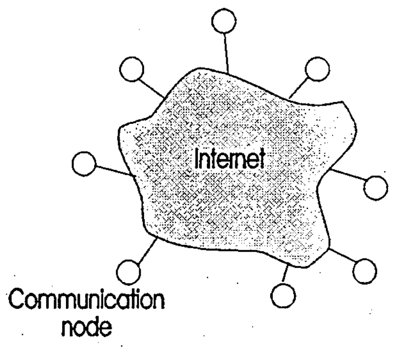


Figure8: Internet

In the Internet, nodes from different networks are usually connected by routers (or gateways). The function of a router is to provide Internet protocol translation, to establish a connection between networks, and to perform a number of other functions that permit computers to communicate, independent of hardware differences.

It has not been clear whether the idea of providing routers with some additional functions (such as aggregation) could be useful for optimizing the throughput and improving the delay performance. *Therefore, the problem of whether routers should have some intermediary functions in the flow control issue (which is normally implemented at the end-nodes) will be addressed in this thesis.*

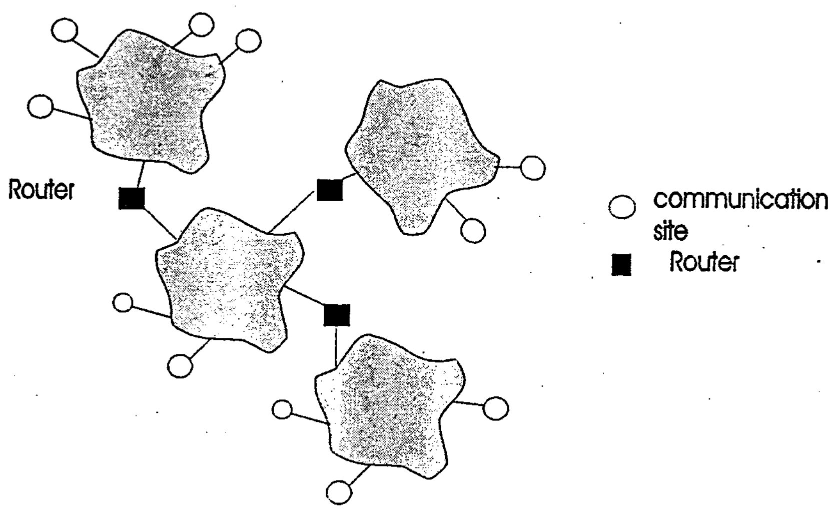


Figure9: Utilisation of Routers

In this thesis we introduce a novel flow control strategy that *is hop-by-hop flow control with packet aggregation.* The aggregation procedure is performed at the routers where every batch of several packets could be treated as a single new packet and be sent to another router site, instead of sending the constituent smaller packets separately. This can lead to a lower traffic in the following stage of the network and therefore to a lower congestion, less packet drop at routers and finally higher throughput in the network.

The proposed technique at routers is adaptive by nature since with any changes in network parameters (such as the packets' arrival rate, the network service rate and the destination node service rate) the aggregation rate and the size of the aggregated packet can be changed to minimize the system response time and to optimize the throughput.

Some other functions will be required at the routers to enable the successful implementation of our aggregation techniques, namely encapsulation-decapsulation and intranet fragmentation when neccesary. Also, in this thesis we will suggest the use of four HFCPA messages to help implementating the flow control mechanism.

3.2 REQUIREMENTS

As the Internet evolves, there is a growing need to support more sophisticated services (e.g., traffic management, QoS) than the traditional best effort service. Therefore, two classes of solutions appeared: those maintaining the stateless property of the original IP architecture, and those requiring a new stateful architecture. Examples of stateless solutions are RED for congestion control [FJ93] and Differentiated Service (Diffserv) for QoS. The corresponding examples of stateful solutions are Fair Queuing [DKS89] for congestion control and Integrated Service (Intserv) for QoS. In general, stateful solutions can provide more powerful and flexible services, while stateless solutions have higher flexibility and utilization, and are more scalable and robust than their stateful counterparts.

The question we want to answer is: is it possible to have the best of the two solutions: Powerful as stateful networks, while utilizing algorithms as scalable and robust as those used in stateless networks?

While we cannot answer directly this question, we can answer it from different aspects. Remember that path information is not revealed in the current Internet; it is up to the TCP protocol to investigate about network conditions and the presence of congestion. In our HFCPA protocol, this problem is alleviated since we assume that messages are delivered along the Internet path to reveal the specific information about the link.

In fact, the HFCPA protocol considers the Internet as subsequent tunnels where edge routers are responsible for the intra-domain of the tunnel. Edge routers communicate with each other through *Path* and *ACK* messages (which will be explained later on in this chapter) and perform the corresponding aggregation, encapsulation and decapsulation of the packets. However, core routers do not; they are intermediate routers to pass path information messages and data packets.

Using this hop-by-hop architecture will prevent from congestion collapse and will optimize network performance regarding throughput, delay and loss. The process of aggregation and reassembly of end-to-end packets will be discussed in details. Next, we will introduce the encapsulation process, the choice of the optimal path in addition to the need of reassembly process in routers for security reasons.

3.3 ENCAPSULATION

Encapsulation is a process of changing the normal route of an IP datagrams by delivering them to an intermediate destination that is not selected on the IP Destination Address field in the original IP header [RFC1241]. Once the encapsulated datagram arrives at this intermediate destination node, it is decapsulated, yielding the original IP datagram, which is then delivered to the destination indicated by the original Destination Address field. This use of encapsulation and decapsulation of a datagram is frequently referred to as "tunneling" the datagram, and the encapsulator and decapsulator are then considered to be the "endpoints" of the tunnel.

In the most general tunneling case we have the following flow:

Source ⟶ Encapsulator ⟶ Decapsulator ⟶ Destination

The encapsulator node is considered the "entry point" of the tunnel, and the decapsulator node is considered the "exit point" of the tunnel. There, in general, may be multiple source-destination pairs using the same tunnel between the encapsulator and decapsulator.

3.3.1 Encapsulation Technique: IP-in-IP Encapsulation

IP-in-IP encapsulation technique is fairly simple [RFC1853]. An outer IP header is added before the original IP header. Other headers for the path, such as security headers specific to the tunnel configuration are inserted between them[RFC2003].

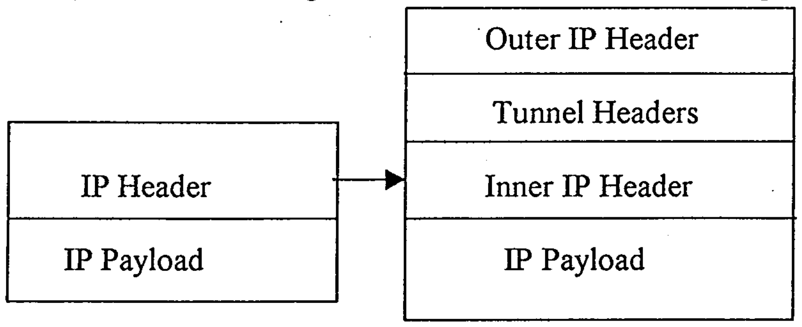


Figure10: Encapsulation

The format of IP headers is described in [RFC-791] and in chapter 2 of this thesis. Some of the original IP header fields are copied to the new outer IP header and some are not, as described next:

- *Type Of Service* field is copied from the inner IP header, so that if the user was expecting a given level of service, then the tunnel should provide the same service. However, some tunnels may be constructed specifically to provide a different level of service.
- *Don't Fragment* field is not copied from the inner IP header. Since no internal router is allowed to fragment packets. It is the encapsulator that has permission to fragment packets.
- *More Fragments* flag is usually set as required when fragmenting. Since fragmentation is not allowed inside the tunnel, the flag is not copied to the outer IP header.
- *Time To Live* field ensures that long unanticipated tunnels do not interrupt the flow of datagrams between endpoints. When encapsulating a datagram, the TTL in the inner IP header is decremented by one if the tunneling process will forward the datagram. The TTL in the inner IP header is not changed when decapsulating. If, after decapsulation, the inner datagram has TTL = 0, the decapsulator must discard the datagram. If, after decapsulation, the decapsulator forwards the datagram to one of its network interfaces, it will decrement the TTL as a result of doing normal IP forwarding.
- *Source* field contains an IP address associated with the sender of the datagram.
- *Destination* field contains an IP address of the tunnel decapsulator.
- *Options* field is not copied from the inner IP header. However, new options particular to the path may be added. Other options are hidden within the tunnel.

Field	**Mapping**
Version	Ignore
Header Length	Ignore
Precedence	Ignore
QoS bits	Copy
Total Length	Copy
Identification	Ignore
Don't Fragment Bit	Ignore
More Fragments Bit	Ignore
Fragment Offset	Ignore
Time to Live	Ignore
Protocol	Ignore
Header Checksum	Ignore
Source Address	Ignore
Destination Address	Ignore
Security Option	Copy

Table 1: Summary of IP Header Mappings

We should note the following:

- The protocol field in the new IP header should be filled with the protocol number of the encapsulation protocol.
- The packet offset for each packet encapsulated is usually required along with the source address of the Encapsulator.
- The destination address becomes the IP address of the Decapsulator as found in the encapsulation table.

3.3.2 Implementation

We treat the two tunnel end-points as a source and destination host with R_{entry} as the source address and R_{exit} as the destination address and R_{port} as source and destination ports.

However we could encapsulate the qualified packets not only with an IP header but also with a TCP header. This allows intermediate routers to use standard filtering without knowing the existence of tunnels.

To simplify the implementation, we decided that all data packets using aggregation are encapsulated in (IP+TCP) header. The source port for the TCP header is chosen by the tunnel entry point Rentry when it establishes the initial Path message for the new tunnel session. The destination TCP port used in tunnel sessions is a well known port, established by the Path message.

We should note that all the packets that reach one of the tunnel end-points are encapsulated before being sent to the other side. Hence, when this modified IP-in-(IP+TCP) encapsulation model is used, two main issues are to be considered:

- First, the end-to-end packets become invisible to intermediate routers residing between the tunnel end-points.
- Second, the usual filters can be used, since data packets are also encapsulated with an outer (IP+TCP) header, making the original IP (and UDP or TCP) header(s) invisible to intermediate routes between the two tunnel end points.

3.4 OPTIMAL PATH

In this paragraph, we will give a brief description on the routing techniques used in the Internet, and propose an optimal path within the tunnel to optimize our objective that is to maximize throughput and to minimize packet loss.

3.4.1 Optimal Path vs Shortest Path

The relationship between path selection and end-to-end performance on the Internet has not been the subject of much study. In fact, Internet routing concerns about providing good (not optimum) end-to-end performance, and minimizing cost. Recent studies show that, at any time *t*, there exists an alternate path that performs better than the Internet shortest path found using OSPF "shortest path" routing algorithm.

One optimal path method was proposed by Bonuccelli [GP98]. Bonuccelli aimed to find a path that minimizes the number of fragments C; and, this path should be no longer than a given length. He studied the relation between C and the bottleneck value along a path and proposed a fast routing algorithm, called MAXBOTTLENECK, to find a maximum bottleneck path. This path turns out to be optimum with respect to the objective function C only if some conditions on the network MPS's are met.

Seeking sub-optimal solutions in the network with respect to C, means that the first path is the path along which the minimum number of fragments is produced and each subsequent path gives an equal or greater value of C than the previous one. When a new path is found, its length is compared to the length of the paths already generated in order to test its possible membership to be the best path with respect to the objective function.

At any time there are many paths through the Internet connecting any two hosts. Some of these paths have higher bandwidth than others, some have lower propagation delay, and others see less congestion. So, end-to-end performance is limited by the path quality along a given path. In other words, if packets sent over a path are delayed or lost this directly reduces the throughput that a host can expect to obtain. In this case, an optimal path is suggested, since the difference between the default path and the alternate paths is largely seen.

Also Vern Paxon [Pax97] examined several existing Internet routing algorithms and found that there are alternates with significantly improved measures of quality. In fact, he found the following:

- For 30 to 55% of the paths measured, there is an alternate path through one or more additional hosts resulting in a smaller round-trip time.
- The best alternate has 50% better latency.
- 75 to 85% of the paths have alternates with a lower loss rate.
- 70 to 80% of the paths have alternates with improved bandwidth.

Consequently, the presence of superior alternate paths is caused by avoiding parts of the Internet with particularly poor quality and more specifically avoiding congestion, rather than by minimizing propagation delay.

In reality, today's Internet routing is based on a number of factors that are correlated with performance. The original ARPANET, for instance, used a distributed adaptive routing algorithm based on measurements of queuing delay at each link. Others used a congestion avoidance route [AV99]. Our optimal path algorithm, uses an algorithm based on maximum flow in the shortest path.

3.4.2 Path MTU Discovery

Path Maximum Transmission Unit (PMTU) discovery was suggested primarily [RFC1191] to avoid fragmentation by estimating the *PMTU* between the source and the destination. The technique was in using the "don't fragment" *DF* bit in the *IP* header so that the source host sends all datagrams on the path with the *DF* bit set and with MTU equal to the known MTU of the first hop. 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 *ICMP* Destination Unreachable messages with a code meaning "fragmentation needed and DF set" [RFC1191]. Upon receipt of such a message ("Datagram Too Big" message), the source host reduces its assumed PMTU for the path.

The PMTU discovery process ends when the estimation of the PMTU is low enough that its datagrams can be delivered without fragmentation. Normally, the host continues to set DF in all datagrams, so that if the route changes and the new PMTU is lower, it will be discovered. After receiving a "Datagram Too Big" message, the source must avoid receiving such messages in the near future; so it reduces the size of the datagrams it is sending along the path. Hence, the host must force the PMTU discovery process to converge.

Unfortunately, the "Datagram Too Big" message, as currently specified, does not report the MTU of the hop for which the rejected datagram was too big, so the source host cannot tell exactly how much to reduce its assumed PMTU. To remedy this, we propose the "Path" message discussed in the section 3.10 that will return the corresponding path information within the tunnel.

3.4.3 Proposed Algorithm

Since the HFCPA protocol keeps the communication between the end tunnels, the aggregator can select the path inside the tunnel through different criteria:
One form of selection could be by implementing the algorithm of "the shortest path in the maximum flow" concerning the desired bandwidth, for instance.
Another form is by selecting the highest PMTU in the tunnel. Therefore, routers should adapt to the network conditions before forwarding packets inside the tunnel to reach the maximum possible benefit.

3.5 SECURITY

The Internet has long suffered from several attacks that usually bypass Internet firewalls. Most of these attacks were caused not by the way fragmentation is done but rather because of the way datagrams are reassembled.

As previously stated, when datagrams are fragmented into packets, the header portion of the packet remains intact except for the modification of the DF bit and the filling in of an *offset* field in the IP header that indicates at which byte in the whole datagram the current packet is supposed to start. In reassembly, the IP reassembler creates a temporary packet with the fragmented part of the datagram in place and adds incoming fragments by placing their data fields at the specified *offsets* within the datagram being reassembled. Once the whole datagram is reassembled, it is processed as if it came in as a single packet.

According to the IP protocol, fragmented packets are to be *reassembled at the destination host.* This means that they are not supposed to be reassembled at intermediate sites such as firewalls or routers. This decision was made to prevent repeated reassembly and refragmentation in intermediate networks; so, when routers and firewalls followed the rules, they found a strange problem[RFC1858].

In fact, the way firewalls and routers block specific services (such as *telnet*) while allowing other services (such as the world wide web *http* service) is by looking into the

IP packet to determine which *Transfer Control Protocol* (TCP) *port* is being used. If the port corresponds to 80, the datagram is destined for *http* service, while port 23 is used for *telnet*. In normal datagrams, this works fine. But suppose we didn't follow the rules for fragmentation and created improper fragmented packets. Here's what the result could be:

- Create an initial packet that claims to be the first fragment of a multi-packet datagram. Specify TCP port 80 in the TCP header so it looks like a datagram going to *http* service, which is allowed to pass the firewall.
- The firewall passes the packet to the host under attack and passes the other packet fragments in order to allow the destination host to reassemble the packet.
- One of the subsequent packets has an *offset* of *1* that causes the reassembler to overwrite the initial part of the IP packet. This is the part of the IP packet that specifies the TCP port. The attacker overwrites the IP port number that was originally 80 with a new port number such as 23, and is now granted telnet access to the host under attack despite the firewall that is supposed to block the service.

In this case, the attacker lies about the port this datagram is destined for, and the packet filters and firewalls believe the lie. Since a datagram can be made to pass where it is not supposed to go, it may be able to enter a trusted network and act as if it were a trusted datagram. This can be exploited to corrupt information. If the firewall can be fooled into allowing ICMP or similar packets to pass, this mechanism can also be used to deny services. Once an attacker has hacked a host, the same mechanism may be repeated in that host to send outbound packets past the firewall even though they are not supposed to be passed.

Due to the fact that we cannot refuse to allow fragmentation, we have to live with it. The question is how? This problem was being exercised and many vendors have resolved the problem by reassembling fragments in the router before making access control decisions [Coh96]; so, we can conclude that fragments should be reassembled in routers prior to their arrival to their destination.

3.6 ARCHITECTURE

The Internet nowadays is suffering from the low throughput of packets delivered from their corresponding source to their ultimate destination; because packets of 576-byte are routed in networks where the MTU is much higher and where the Bit Error Rate (BER) is very low.

So, our main concern is to benefit from these high MTU networks. In fact, we will aggregate packets on a per-hop basis rather than end-to-end in order to improve the effectiveness of the links and to avoid the scalability problems in core routers. Per-hop basis leads to a network size reduction. So, when reducing the size of the network (by using tunnels), the congestion or the bottleneck will be reduced and the throughput will be increased.

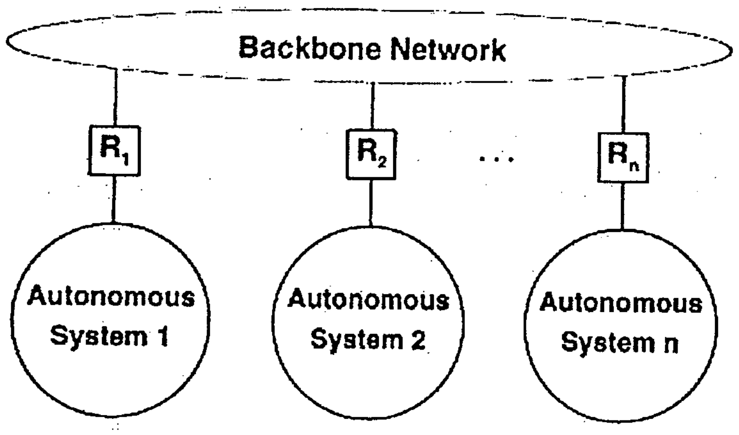


Figure11: Interconnection of autonomous systems to the backbone

The solution suggested in this thesis has the following design principles or features:

- Each router can operate based on its own aggregation policy. Routers aggregate on a next-hop basis (disaggregator) when sufficient classified packets with "similar" requirements are met.
- When a router decides to aggregate datagrams, it uses HFCPA protocol elaborated later to establish and maintain the aggregated packet (PATH and ACK messages are required within the tunnel.
- Packets are scheduled before being forwarded

- Classified packets are scheduled according to the aggregate's characteristics

The larger the degree of aggregation at the tunnel end-points the larger is the gain in the network backbone routers. At one end of the tunnel, we can achieve the largest amount of aggregation possible by mapping all end-to-end packets of the same class or criteria to a single aggregated datagram. At the opposite end of the tunnel, we have individual end-to-end packets getting mapped to the other tunnel, thus achieving no reduction in the intermediate routers. In this case intermediate routers in the tunnel only see one aggregated packet per tunnel per predifined class, and due to encapsulation they are invisible to intermediate routers in the tunnel and therefore require no processing.

This New architecture is to be discussed in the following paragraphs, introducing intranet fragmentation, encapsulation and aggregation within the tunnel.

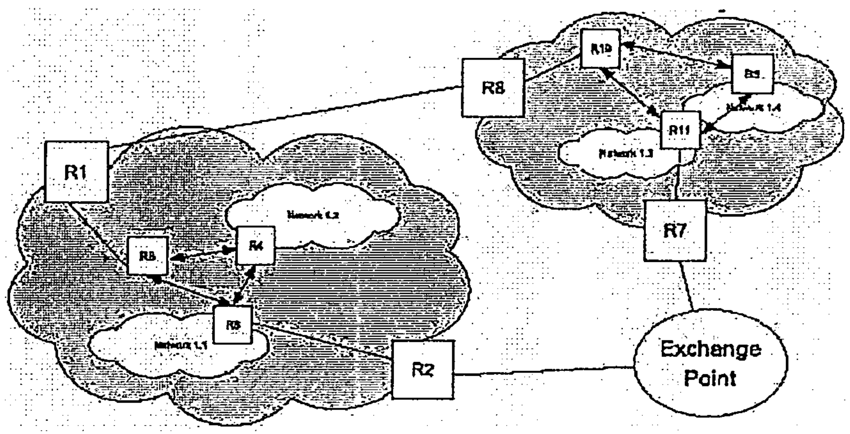


Figure12: how tunnels are connected to each other

3.7 INTRANET FRAGMENTATION

Since our HFCPA architecture subdivides the Internet into aggregating regions, we advise to use the intra-net fragmentation inside the tunnels. Since fragmentation cannot be avoided, careful design can make fragmentation normally unnecessary and can avert its most serious drawbacks. Therefore, intranet fragmentation or transparent fragmentation reduces and sometimes eliminates the dangers of Internet fragmentation.

Since all datagrams are tunneled and sent to a unique next hop router for reassembly or aggregation, two benefits are obtained:

1. Fragment loss is reduced since the protocol between the two edge routers supports acknowledgments of individual fragments, and the end-hosts are willing to accept occasional lost or mis-sequenced datagrams.
2. If the (reassembled) datagram subsequently traverses a network with a larger MTU, it makes more efficient use of that network than a collection of smaller fragments.

There is little value in the ability to send fragments of one datagram along different routes within the tunnel, and reassembly by routers should not be prohibitively expensive. Main memory sizes and costs are improving so rapidly that buffer space should no longer be considered the limiting resource; the ability to record path information ---- not only about MTU but also about congestion, bandwidth, etc.---- is so valuable that it will be presented in the Path Message section.

Aggregators must know the MTU of the tunnel. In particular, it is much better to fragment the original datagram when encapsulating, than to allow the encapsulated datagram to be fragmented. Fragmenting the original datagram can be done by the encapsulator without special buffer requirements and without the need to keep reassembly state in the decapsulator. By contrast, if the encapsulated datagram is fragmented, then the decapsulator must reassemble the fragmented (encapsulated) datagram before decapsulating it, requiring reassembly state and buffer space within the decapsulator.

However, no fragmentation is allowed within the tunnel. In case it will occur, an error message will be generated to the encapsulator.

Concerning the reassembly process, in the HFCPA protocol, a destination host may still have to perform reassembly, since the MTU of the last-hop link may be smaller than the datagram size; this means that most of the problems associated with inter-network fragmentation would still be present, although to a lesser degree.

Since transparent fragmentation cannot entirely neglect the use of inter-network fragmentation, there must be a way to recover from inappropriate fragmentation. The receiving host can detect the problem, and should notify the sending host via an ICMP message [RFC792].

3.8 AGGREGATION

We define an "aggregating region" as a set of routers for which end-to-end packets arriving to that router in the set would be aggregated and encapsulated before traversing one or more inside routers before finally traversing an outside interface. Such an end-to-end packet is said to have crossed the aggregating region, or what is previously defined as "tunnel".

The "aggregating" router is the first router that processes the end-to-end packet as it enters the aggregation region. The "disaggregating" router is the last router to process the end-to-end Path as it leaves the aggregation region.

3.8.1 Implementation

The following assumptions are made to enable the implementation of the aggregation method:

1. A region, called an aggregating or encapsulating region, will aggregate packets and will be responsible of all the mechanisms implemented in this region.
2. The choice to aggregate and the mechanism used to do so is an intra-domain issue, not an inter-domain issue and therefore there can be variability across different regions.
3. Data packets are classified, aggregated and encapsulated on entry to the aggregating region; and, the aggregator does not determine which packets can use the aggregation process, but merely specifies what amount of bandwidth is available for that quantum of time.

4. Each router in the Internet is not required to perform classification, scheduling, aggregation, disaggregation and buffer management on the data path, since performing aggregation inside the network affects both the network scalability and robustness.
5. This aggregation scheme can be a recursive aggregation, with aggregated packets being themselves aggregated. Multi-level aggregation can be accomplished in the tunnels.
6. Each sender is logically distinct from a receiver, but any router can act as a sender and receiver.

Finally, in the HFCPA protocol, the messages (Path, ACK) are a tunnel attribute that is used to determine the way in which data packets are handled by routers (sender, receiver or intermediate). To initiate a connection, a potential sender (aggregator) starts sending *Path* messages to the IP destination address (disaggregator). The receiver application receives a *Path* message and sends back appropriate *ACK (ACK)* messages for that specific route. After the sender application receives a *ACK* message, the sender starts sending data packets.

3.9 DISAGGREGATION AND DECAPSULATION

In the ideal situation, a Decapsulator receives an Encapsulated Datagram, cuts off the Encapsulation Header and sends the original datagram back into IP so that it is forwarded from that point. However, if the original datagram has not yet reached its original destination, it must again be encapsulated to forward it toward its original destination. In this latter case the Decapsulator becomes an encapsulator and hence will have to perform the same job to generate the Encapsulation Header as did the previous Encapsulator.

One good question arises here is how to determine the disaggregator. One method for Disaggregator determination is by manual configuration. With this method the network operator would configure the Aggregator and the Disaggregator with the necessary information. Another method allows automatic Disaggregator determination and corresponding Aggregator notification. For instance, if because of a topology change, another disaggregator is now on the optimal path, this method will automatically identify the new Disaggregator and swap to it.

3.10 PROTOCOL MESSAGES

We suggest that the state of the tunnel should be managed at each router by four different types of messages, namely the *Path* messages, the *ACK* messages, the *Error* messages and the *Refresh* messages.

Each aggregator or sender periodically sends a *Path* message that establishes or updates the route to the receiver, and each receiver periodically sends back a *ACK* message established along the path to the sender. *Path* messages are forwarded using the existing routing table, and *ACK* messages are forwarded back towards the sources by reversing the paths of *path* messages. In fact, the path information is maintained solely to do this reverse path forwarding of the *ACK* messages.

Like any normal datagram, *path* and *ACK* messages carry a timeout value that is used by intermediate routers to set corresponding timers; the timers get reset whenever new messages are received.

It is the responsibility of both senders (aggregators) and receivers (disaggregators) to maintain the proper information inside the network by periodically refreshing the *path* and *ACK* messages, so that when a route changes, routers will forward future *path* messages along the new route(s) and reach new members. As a result, the path will be updated, causing future *ACK* messages to traverse the new routes.

VALUE	MESSAGE TYPE
1	Path
2	ACK
3	Error
4	Refresh

Table 2: HFCPA Message Type Field Values

3.10.1 Path Messages

A path message is sent by each sender along the routes toward the receiver [DEZ] inside the tunnel provided by the routing protocol(s). A path message is used to store the path state in each node. The path state is used to route ACK messages in the reverse direction.

Path Messages are useful for discovering other path characteristics besides PMTU. As long as one is processing an investigation, it makes sense to collect a variety of information, since it comes at little additional cost. This information could include the following fields:

Type	
Identifier	Sequence Number
Minimum MTU Encountered	
Minimum Bandwidth Encountered	
Maximum Delay Encountered	
Maximum Queue length	
Maximum Error Rate Encountered	
Hop Count	

Table3: Path Message Fields

Minimum bandwidth

This field is useful for determining the appropriate transmission rates; if a host knows that a 9600 bps link is part of the path, it should behave differently than if the path is entirely via 100 Mbps fiber networks. Initially set to MAXINT. Each router compares this value to the bandwidth of the incoming and outgoing links for the message, and reduces the recorded value, if necessary.

Maximum delay

This field is useful for determining round-trip times; if a satellite channel is in use, with a delay of several hundred milliseconds, a host should not retransmit as quickly as if the end-to-end delay was several milliseconds. Initially set to zero. Each router compares this value to the time it will take the packet to traverse the incoming and outgoing links, and increases the recorded value, if necessary.

Maximum queue length

A high value implies congestion; a flag could be set if any router along the path is experiencing congestion. Initially set to the length of the output queue of the source host when the packet is placed on that queue, or to zero. Each router compares this value to the length of the queue in which the packet is placed, and increases the recorded value, if necessary.

Maximum error rate

When a link along the path is experiencing a high error rate, a host might choose to send shorter packets (so as to reduce the likelihood that an entire datagram is dropped because of a single error). Each router compares this value to the error rate of the incoming and outgoing links, and increases the recorded value, if necessary.

Hop Count

The total number of links traversed along the route may be of interest, for example, in choosing a value for the "Time To Live" field. Initially set to 1. Each router increments this value by one.

Minimum MTU encountered

Initially set to the MAXINT. Each router compares this value to the MTU of the incoming and outgoing links for the message, and reduces the recorded value, if necessary.

One of the main benefits of using Path messages, is to prevent multiple fragmentation of a single datagram inside the tunnel; and, to improve the processing efficiency at the decapsulator and the routers within the tunnel.

3.10.2 ACK Messages

An ACK message is sent by each receiver host toward the sender to acknowledge a given *path* message. This message is created at the disagregator after its corresponding *path* message has been processed. The necessary information is included in the *ACK message* before it is sent back following in reverse the routes that the data packets use, all the way to the sender hosts. Because ACK messages are initiated by each receiver, HFCPA must make sure that the ACK messages from a receiver follow exactly the reverse routes of the data packets from all the sources (that the receiver is interested in). In other words, HFCPA must establish a sink tree from each disaggregator to all the aggregators to forward ACK messages.

3.10.3 Error Messages

It is possible that one of the interior routers along the tunnel might encounter an error while processing the datagram, causing it to return an ICMP [RFC 792] error message to the encapsulator at the IP Source of the tunnel. Unfortunately, ICMP only requires IP routers to return 8 bytes (64 bits) of the datagram beyond the IP header. This is not enough to include the entire encapsulated header. Thus, it is not generally possible for an aggregating router to **immediately** reflect an ICMP message from the interior of a tunnel back to the originating host.

However, the encapsulator could return accurate ICMP messages if it maintains the following information:

- Reachability of the end of the tunnel (recording the Path msg).
- Congestion of the tunnel.
- MTU of the tunnel.

After an encapsulated datagram has been sent, the encapsulator may receive an ICMP message from any intermediate router within the tunnel other than the tunnel exit point. The action taken by the encapsulator depends on the type of ICMP message received. When the received message contains enough information, the encapsulator may use the incoming message to create a similar ICMP message, to be sent to the originator of the original unencapsulated IP datagram (the original sender).

3.10.4 Refresh Messages

Path messages and *ACK* messages are refreshed periodically in the tunnel using *refresh* messages. The frequency of generation of these refresh messages depends on several issues in the network. Our current implementation uses static timer values for generating refresh messages. As a future work, one can investigate adaptive algorithms to optimally adjust the timer values by taking into consideration the dynamics of route change as well as the loss probability of the messages.

When a route changes, no *ACK* messages will be sent along the new route until a *Path* message will pass by it. Therefore, a new *Path* message will be initialized and a ACK is established along the new route. So, the HFCPA protocol is responsible in propagating those changes from end to end within the tunnel network without delay, and in refreshing the network status continuously by sending refresh messages onward and backward.

3.11 ROUTERS

Advance in hardware capabilities during recent years have provided mechanisms to overcome the limitation of quality of service and differentiation features in routers [KLS]. The following operations can now be done at high speed:

1. Packet classification and filtering according to their different requirements (same destination, sequence numbers in sequence).
2. Buffer management that determines how much buffer space should be given to arriving burst of packets and which packets should be discarded.

3. Packet scheduling that decides which packets to group so as to meet the specific bandwidth and delay requirements of the outgoing link.
4. Determine the next hop of the packet, using the forwarding databases.

One important issue that we suggest is that aggregator routers should use caching methods for forwarding packets. The justification is that packet arrivals are temporally correlated and can be grouped into connections or flows, so that if a packet belonging to a new connection arrives then more packets belonging to that same connection can be expected to arrive in the near future.

3.11.1 Packet Lifetime

The idea behind our solution is that each aggregated packet carries in its header the TTL field that is initialized by the aggregator, and then updated by the intermediate routers along the packet's path within the tunnel. The aggregator aims to approximate the tunnel it belongs to from the delay point of view using a delay-jitter controlled virtual clock [HZ]. Remember that the deadline of the normal datagram before being aggregated depends on the option variables of the encapsulated packet it belongs to. Option can be stored only at the aggregator node into the packet header and retrieved later by core nodes, which then use it to determine the packet's deadline.

3.11.2 Retransmission Timer (RTO) Estimation

A fundamental question is how long should an aggregator wait before retransmitting? In fact, it is a very hard problem to decide whether waiting long enough for a packet to arrive, or being fast to retransmit. In fact, the aggregator really needs to estimate the bandwidth and the round-trip time from the aggregator to the disaggregator plus the amount of time required for the aggregation for received data. Thus, estimating a good value for the retransmission timer not only is an estimation of the network path, but also a property of the tunnel itself. Moreover, during congestion, it will take the sender to wait longer than the maximum round-trip time, in order to give congestion more time to drain from the network. If the sender retransmits as soon as the time elapses, the retransmission may also be lost, whereas sending it later would be successful. Therefore, the setting of the retransmission timer is a hard task and needs to be more studied since many factors contribute to it.

CHAPTER 4

QUEUE MANAGEMENT AT TCP/IP ROUTERS

4.1 INTRODUCTION

In the Internet, it is important to avoid high packet loss rates. When a packet is dropped before it reaches its destination, all of the resources it has consumed along its route have been wasted. For this reason, studies during the last few years have been elaborated to optimize the congestion control in the Internet [Fj93] [Flo00], and a number of TCP enhancements have been implemented. But, still TCP connections experience high loss rates. These loss rates are especially high during times of congestion, when a large number of connections compete for network bandwidth.

If routers are considered as an integral entity in the Internet, then queues are considered to be the key component of the routers, since they absorb arrivals of packets and hence reduce losses. In the queues, the packets could be dropped (TCP/IP approach), or could be aggregated and reassembled, and then forwarded (our suggested method).

This chapter is concerned with the queue management at routers. In fact, we propose a new queuing technique (based on packet aggregation) to be implemented at Internet routers. This new technique aims to manage the queue length, reduce the end-to-end latency, minimize the packet dropping rate, and avoid the lock-out phenomenon within the Internet.

Before explaining our new queuing technique we start by presenting some existing ones namely, Random Early Detection (RED) [FJ93,Flo00] and Tail Drop method [RFC2309,Com00]. Finally, we should mention that recent studies showed that RED does not outperform the tail drop especially when the traffic is formed of TCP and UDP connections [NST]. Also, we notice that it is difficult to parameterize RED queues to perform well under different congestion scenarios. So, in the next chapter, we will show that the performance of Tail Drop queues, is not as good as our newly proposed technique.

4.2 QUEUES IMPLEMENTATION

Before describing some existing queue management schemes, it is useful to distinguish between two classes of router algorithms related to congestion control: "queue management" versus "scheduling" algorithms. Shortly, queue management algorithms manage the length of packet queues by dropping packets when necessary or appropriate, while scheduling algorithms determine which packet to send next and are used primarily to manage the allocation of bandwidth among packets. While these two router mechanisms are closely related, they address rather different performance issues[RFC2309].

As previously stated, TCP congestion control is window based. The sender keeps a congestion window whose size limits the number of unacknowledged packets. Upon receiving acknowledgments for successfully transmitted data, the sender increases its transmission rate by incrementing its window, and hence at some point, the transmission rate eventually exceeds the network's capacity. Therefore, queues in the routers will overflow, causing packets to be dropped. TCP assumes that packet loss is due to congestion and reduces its congestion window when detecting the loss.

The problem with TCP congestion algorithm is that the sender will reduce its transmission rate only after detecting a packet loss. We should note that time may pass between the packet drop and the source detection. Hence, more packets will be dropped meanwhile, until the source enters its "slow phase".

Active queue management has been proposed as a solution for preventing losses due to buffer overflow. One form of active queue management being proposed by the IETF for deployment in the network is Random Early Detection (Red) described below in section 4.2.2.

4.2.1 Tail Drop Queue Management

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:

1. **Lock-Out.** In some situations tail drop allows a single connection or a few packets to fill up the queue space, preventing other connections from getting room in the queue. This is known as "lock-out" phenomenon.

2. **Full Queues.** The Tail Drop discipline allows queues to maintain a full (or, almost full) status for long periods of time, since tail drop signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the number of awaiting packets and this is perhaps queue management's most important goal.

The probability of a packet being dropped p is computed as follows:

Assume that a drop occurs at time t = 0 in a Tail Drop router. The next incoming packet is dropped if and only if its arrival time $(1/\lambda)$ is smaller than the service time $(1/\mu)$ of a packet in the queue. Thus when a packet is dropped, the next packet is dropped with probability p, where

$$p = \int_0^\infty \mu\left(1 - e^{-\lambda x}\right)e^{-\mu x}\,dx = \frac{\lambda}{\lambda + \mu}$$

As a result, the probability *Prob* of a number *n* of consecutive drops in a tail Drop router satisfies

$$\text{Prob} = p^n$$

4.2.2 Random Early Detection (RED)

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. In other words, 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 won't tend to drop packets unless the queue overflows. On the other hand, if the queue has recently been relatively full, indicating congestion, newly arriving packets are more likely to be dropped.

To do this, it uses two threshold values to mark positions in the queue. *Tmin* and *Tmax*:

- If the queue contains fewer than *Tmin* datagrams, the new datagram is inserted in the queue.
- If the queue contains more than *Tmax* datagrams, the new datagram is discarded.
- If the queue contains between *Tmin* and *Tmax* datagrams, the datagram is discarded according to a probability *p*.

The choice of *p*, *Tmin* and *Tmax* is very important to make RED work perfectly. *Tmin* must be large enough, but it must not exceed *Tmax*. *Tmax* should be greater than *Tmin* by more than the typical increase in queue size during one TCP round-trip time.

RED randomly drops packets based on the average queue size that is estimated as follows:

$$avg(i) \leftarrow (1 - w_q) \cdot avg(i-1) + w_q \cdot q$$

Where, w_q is a value between 0 and 1; an example value suggested is 0.002; and, q is the current queue size.

For each arriving packet if *avg* is between a *Tmin* and *Tmax* then the packet is dropped with a certain probability *p*.

Count = count + 1
$p_b \leftarrow \max p \dfrac{avg - T_{\min}}{T_{\max} - T_{\min}}$
$p_b \leftarrow p_b . L/M$
$p \leftarrow p_b / (1 - count . p_b)$

Table 4: Drop probability estimation steps for RED

In the above table, the significance of the used parameters and variables is as follows: p_b is a temporarily dropping probability, *max p* is an upper bound on the temporarily packet drop probability, T_{min} and T_{max} are the two thresholds limiting the region where packets are randomly dropped, *L* is the size of the incoming packet, *M* is the maximum packet size and *count* is the number of accepted packet since the last drop or since *avg* exceeded $T_{\min}$.

When *max p* is small, early detection is ineffective and the behavior of the RED queue approaches that of a drop tail queue. If *max p* increases, RED becomes more aggressive and leads to a drop in the average queue length. Consequently, the round-trip times seen by TCP connections also drop. Because packet loss rates increase with decreasing round-trip times (see Equation (2) in chapter 3), this causes the loss rates in the RED queue to eventually exceed that of the Tail Drop queue.

Finally, note that besides RED and Tail Drop, two alternative queue disciplines that can be applied when the queue becomes full are "random drop on full" or "drop front on full". Under the *random drop on full* discipline, a router drops a randomly selected packet from the queue (which can be an expensive operation, since it naively requires an O(N) walk through the packet queue) when the queue is full and a new packet arrives.

4.2.3 HFCPA Technique

HFCPA queue management technique use packet aggregation approach and can be implemented on any type of queue management. In fact, HFCPA can be used with Tail Drop or RED.

The aggregation process is done in the router as follows. When packets arrive at the queue, the queue management algorithm finds a free router in the tunnel, gets its PMTU, and tries to aggregate packets as high as the seeking PMTU using the *knapsack* algorithm. The benefit from this algorithm is to keep the links in the tunnel on their high load. When exiting the tunnel, the disaggregator can fragment packets if the following PMTU is lower than the aggregated packet size.

HFCPA provides the following advantages:

1. **HFCPA reduces the number of packets dropped in routers and avoid the full queue problem.** Packet bursts are an unavoidable aspect of packet networks [Willinger95]. If all the queue space in a router is already filled, then the router will have no ability to buffer bursts. By packet aggregation, HFCPA queue management will provide greater capacity to absorb bursts with the minimization of dropping packets.

Furthermore, without HFCPA queue management, more packets will be dropped when a queue does overflow, and this is undesirable for several reasons:

- It results in lowered average link utilization, and hence lowered network throughput.
- TCP recovers with more difficulty from a burst of packet drops than from a single packet drop.

- Unnecessary packet drops represent a possible waste of bandwidth on the way to the drop point.

Therefore, HFCPA is estimated to manage queue lengths and reduce end-to-end latency even in the absence of end-to-end congestion control. Though, it will be able to reduce packet dropping in an environment that continues to be dominated by end-to-end congestion control.

2. HFCPA Avoids lock-out behavior. In fact, it can prevent lock-out behavior by ensuring that there will almost always be a buffer available for an incoming packet. For the same reason, HFCPA active queue management can protect routers from failure due to low bandwidth and high packets bursts.

3. HFCPA Increases the Throughput. With HFCPA queuing technique, packets will be aggregated, so the queue empty buffer size will increase, and thus the throughput of the TCP connection increases. In fact, if the bottleneck router will have sufficient number of buffers, to absorb incoming packets, with throughput X% higher, then its link bandwidth will increase for duration of about a round-trip time. Then, if each source only increases its transmission rate by X% every round-trip time, the network can ensure a minimal amount of packet loss.

Bandwidth increase depends on the amount of buffering at the bottleneck link. If empty buffers at the bottleneck link increase, the TCP sources can be more aggressive in increasing their transmission rates. Therefore, increased buffering can also prevent this problem by increasing the round-trip time and thus the latency in delivering the packets; this will be a good advantage for the aggregation process.

4.3 CONCLUSION

HFCPA queue management is needed even for routers that use per-flow scheduling algorithms such as Fair Queuing. This is because scheduling algorithms by themselves do nothing to control the overall queue size or the size of individual queues.

HFCPA Queue management controls the overall average queue sizes, so that arriving bursts can be accommodated without dropping packets. In addition, HFCPA queue management controls the queue size for each individual flow, so that they do not experience unnecessarily high delays.

Therefore, HFCPA queue management should be applied in routers since it outperforms normal Tail Drop queue management, and normal RED, as we will prove in the next chapter.

CHAPTER 5
SIMULATION RESULTS

5.1 INTRODUCTION

This chapter concentrates on analyzing and comparing two queuing strategies at routers in the Internet using simulations. We have explained in the previous chapter that hop-by-hop flow control with aggregation could optimize the throughput and minimize the packet loss in TCP/IP networks. In this chapter we will be interested in proving these ideas ,using simulations. We, therefore, will compare our system which includes aggregation, with the end-to-end flow-control used in normal TCP/IP networks.

The results we develop in this chapter indicate that an end-to-end flow control system with n-stages always gives a greater packet loss than that of an n-stages hop-by-hop flow control with aggregation. We also show, that our newly developed technique provides better throughput than normal TCP/IP implementation.

5.2 NETWORK SIMULATOR

In this section, we present the network simulator that we have developed using Delphi software. This network simulator will enable us to compare the performance of different types of systems namely end-to-end flow-control systems (such as in normal TCP/IP networks) and hop-by-hop flow-control with aggregation systems.

Our simulation model covers one tunnel of the network where aggregation and fragmentation are taken into consideration. It includes:

- Source generating packets of different sizes (ranging from 48 bytes to 1500 bytes). It is not fixed for 576 bytes as the normal TCP/IP protocol packets since we will take into consideration that they might be fragmented or aggregated before they arrive to our simulated tunnel.
- The generation rate of packet λ could be variable with $0.1 \leq \lambda \leq 0.9$.
- One bottleneck router with a buffer of 15 packets.
- *Knapsack* algorithm is implemented in the queue.
- 3 different TCP destinations groups.
- Timeout granularity is set to 200 ms.
- Propagation delay of 15 ms.
- 4 different routers with different bandwidth and different PMTU as described in the following table.

Router	**PMTU**	**Bandwidth**
1	1600	100 Mbits/sec
2	2000	100 Mbits/sec
3	2500	10 Mbits/sec
4	3500	16 Mbits/sec

Note that the aggregation process is done in the bottleneck router as follows: Packets arrive at the queue at a distribution rate with a random size. The queue management algorithm finds the free router in the tunnel, gets its PMTU, and tries to aggregate packets as high as the seeking PMTU using the knapsack algorithm. The benefit from this algorithm is to keep the links in the tunnel on their high load. When exiting the tunnel, the disaggregator can fragment packets if the following PMTU is lower than the aggregated packet size.

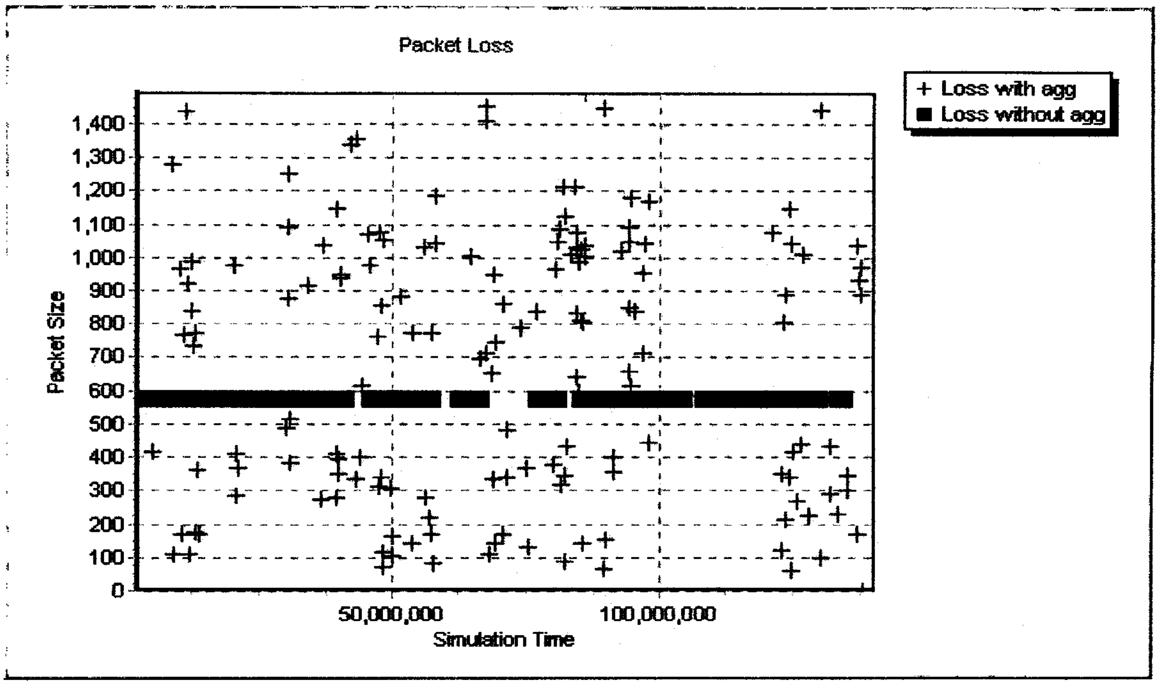


Figure14: Snapshot of simulation run-time

5.3.2 Dropped Packets Due to Congestion

The following diagram represents the results from three simulation tests, used to estimate the loss rate in the network. These tests have the following characteristics:

1) Aggregation with an MTU varying between 48 bytes and 1500 bytes.
2) Aggregation with a fixed MTU = 576
3) Without aggregation a with fixed MTU = 576 (i.e. normal TCP/ IP).

Figure 15 below shows that the current TCP/IP protocol has a much higher loss rate in comparison with our new model.

In the first test where the MTU is variable, we observe that the loss rate is low for large packets (1200-1500 bytes), in contrary to small packets (48-500 bytes). This is due to the aggregation process that is benefiting from the big packets in the queue.

Our second test, run with a fixed size packet of 576 bytes, has the smallest loss rate, because the queue rarely overflows due to the aggregation process that benefits from the high PMTU links in the tunnel.

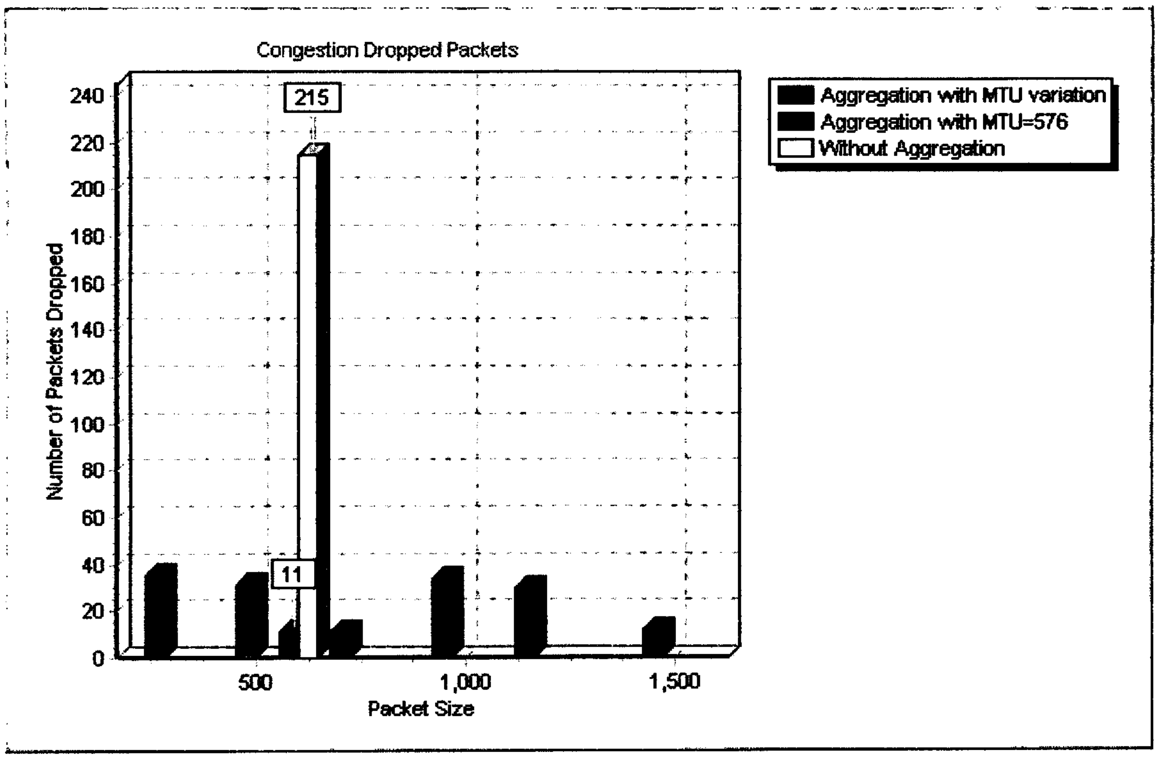


Figure15:Impact of Congestion on Packet Loss

5.3.3 Link Efficiency

One major benefit from the aggregation process, is that the links in the tunnel are run on their high load. Here, we observe that (PacketSize / PMTU) ratio is almost at its high capacity during the simulation of our new protocol, in contrast with the fixed 576-byte packet model which remains in a constant region (around 0.3).

We should note that due to the fact that the knapsack algorithm is implemented in the queue (or the aggregator), the outgoing links from the queue are always using their highest possible load.

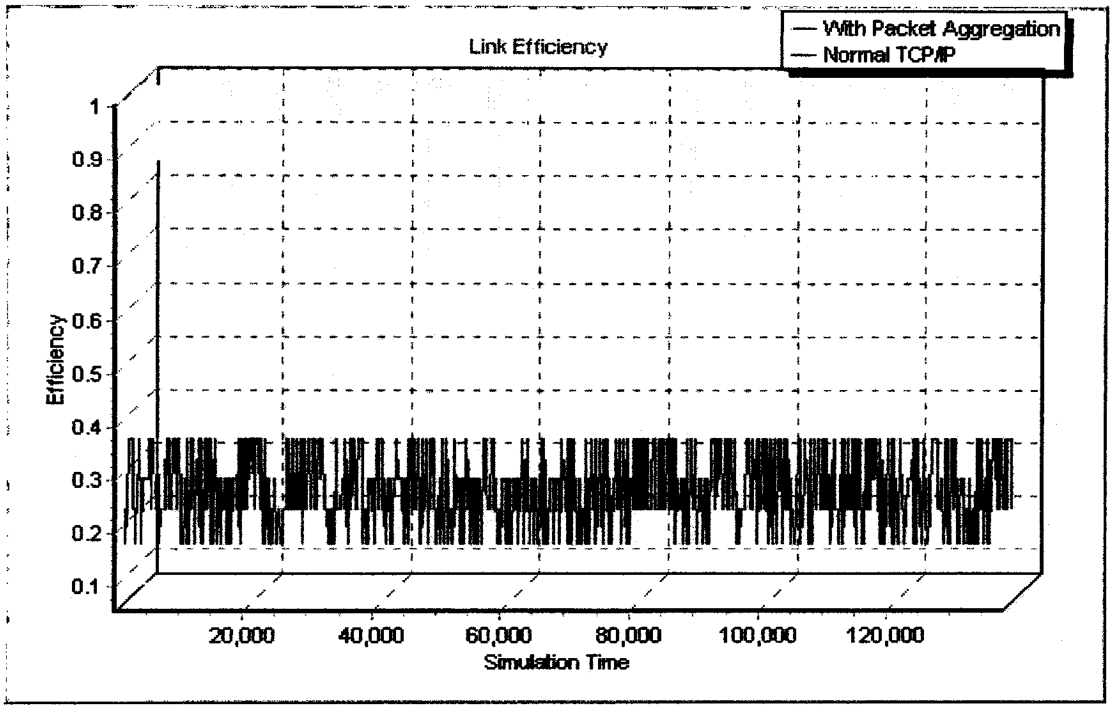


Figure16: Link Efficiency function of Time

5.3.4 Overhead

The overhead is calculated as the division of the size of the header over the size of data contained in a given packet. We see that the overhead is inversly proportional to the packet size.

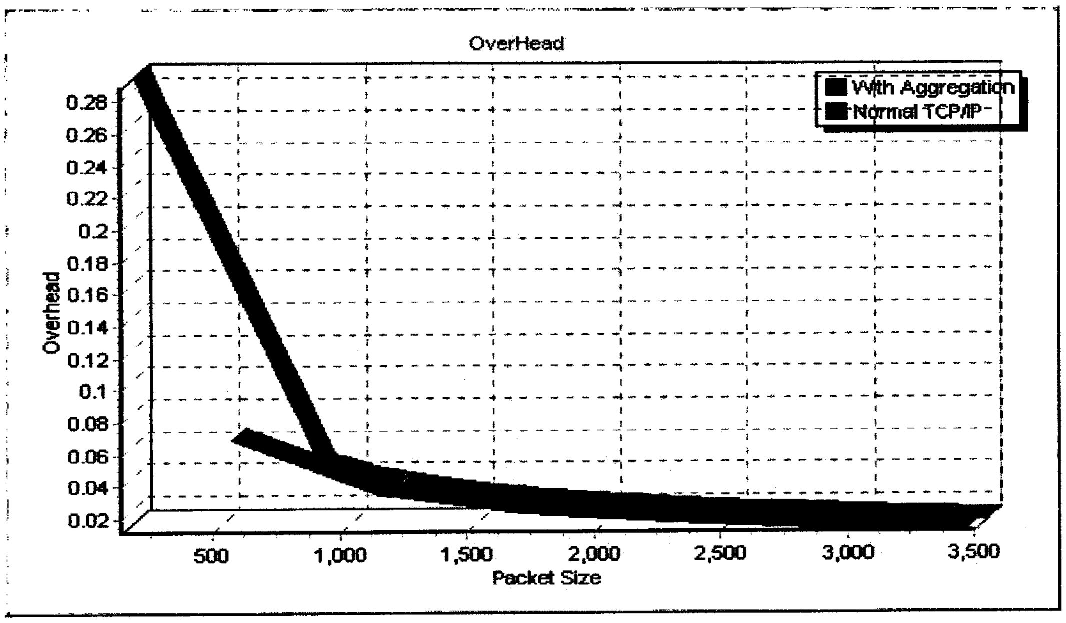


Figure 17: Impact of the MTU size on the Overhead

5.3.5 Throughput

Here, we could notice (Figure 18) that the throughput in the network increases when the packet size increases. In this chart, when the average packet size increases, the benefit of the aggregation will be much higher.

We also should notice that if all the packets in the queue are approximately of size equal to the links PMTU inside the tunnel, the aggregation process will be no more efficient and the queue will be overflowed rapidly.

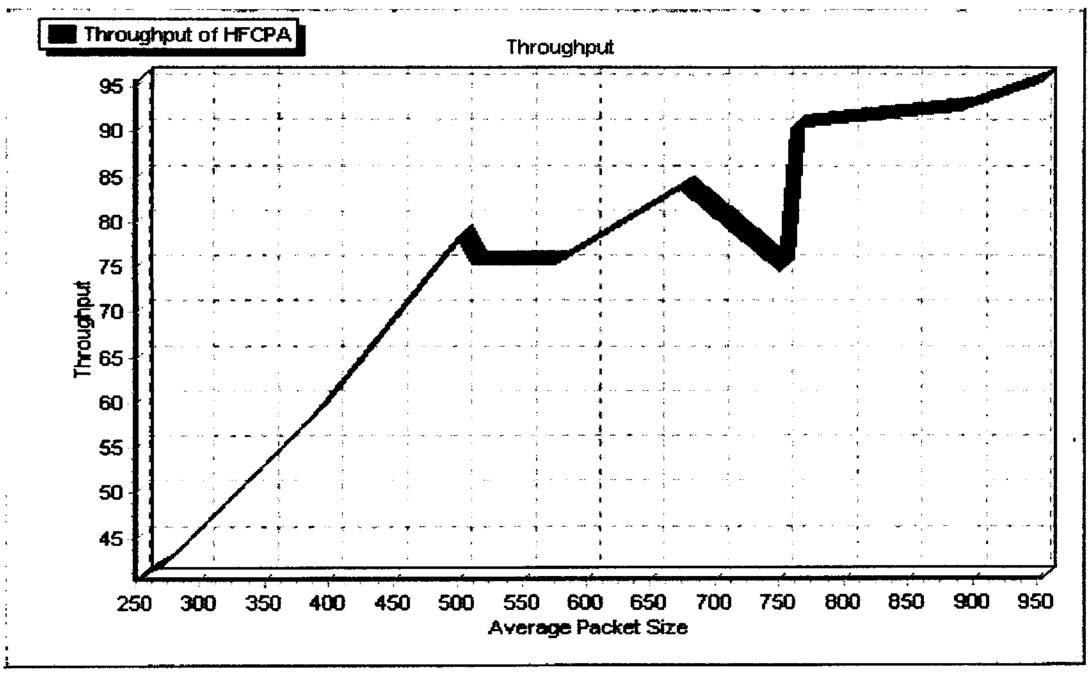


Figure18: Throughput (in packets/sec) as a function of the packet size

5.4 SIMULATION RESULTS FOR A VARIABLE PACKET GENERATION RATE

5.4.1 Packet Drop due to Congestion

As we notice from Figure 19 below, that the normal TCP/IP has a higher loss rate than our new protocol especially when the packet arrival rate is large. In fact, when there is a high network congestion, the new protocol outperforms the normal TCP/IP by means of 5.8 times.

In fact, as we have seen in the study, that the drop-tail queue (used in normal TCP/IP networks) during the congestion drops packets as long as the queue is full. However, newly developed technique, tries to locate a place for that coming packet in the queue, by aggregating before forwarding.

Moreover, the HFCPA protocol still outperforms the Normal TCP/IP in dropping packets, even when lambda is small (link is not congested) as the figure shows.

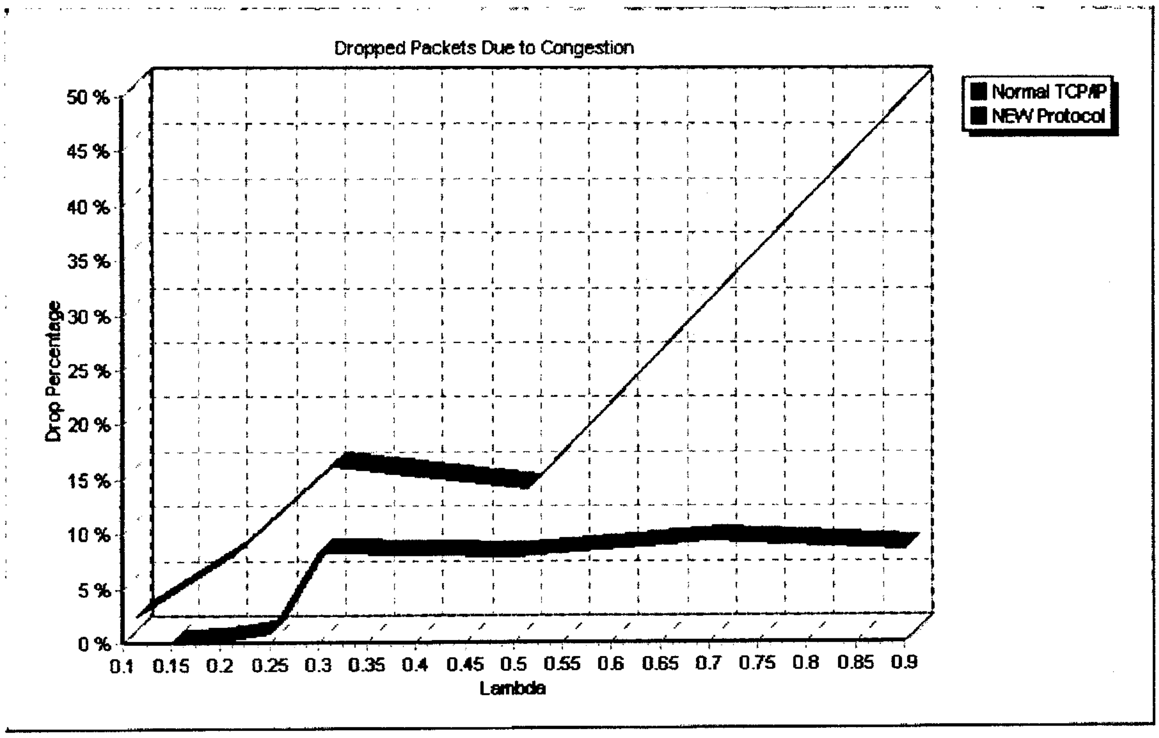


Figure19: The Impact of the arrival rate on the packet loss

5.4.2 Link Efficiency

We could notice here, that the new protocol is far more efficient than the normal TCP/IP protocol, since with the aggregation process, the links in the tunnel are run on their high load. This improvement is especially noticed when the packet arrival rate λ is high (i.e. when the network is congested).

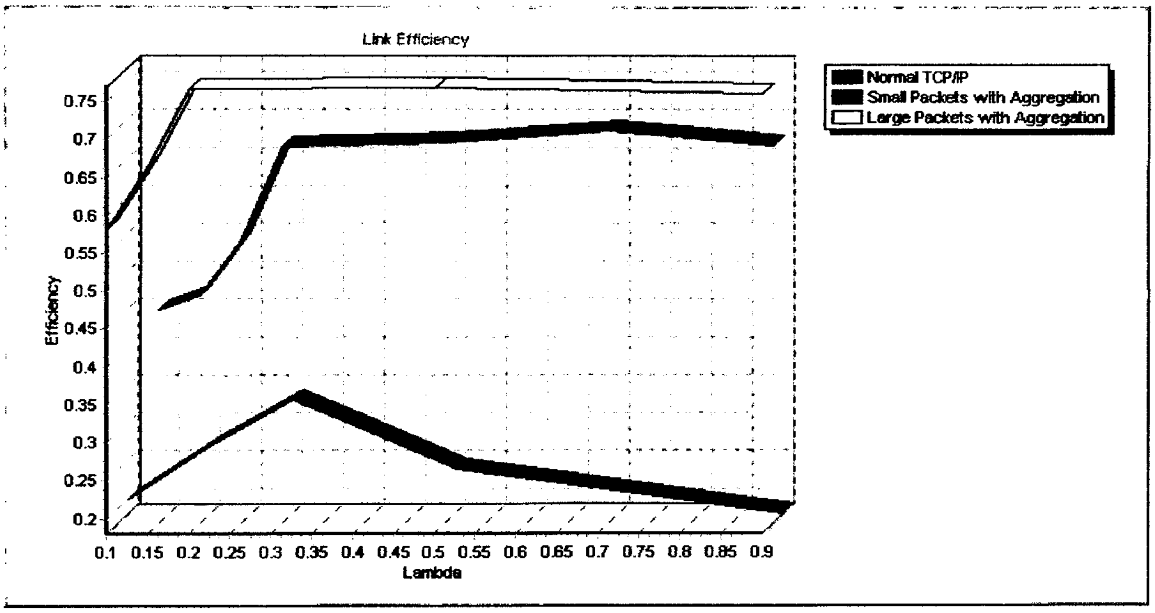


Figure20: Impact of the arrival on the efficiency

5.4.3 Throughput

Here, we can notice that the throughput of the network increases when the packet arrival rate increases. This improvement is even more obvious, when the average packet size increases, since in this case the benefit of the aggregation process will be much higher.

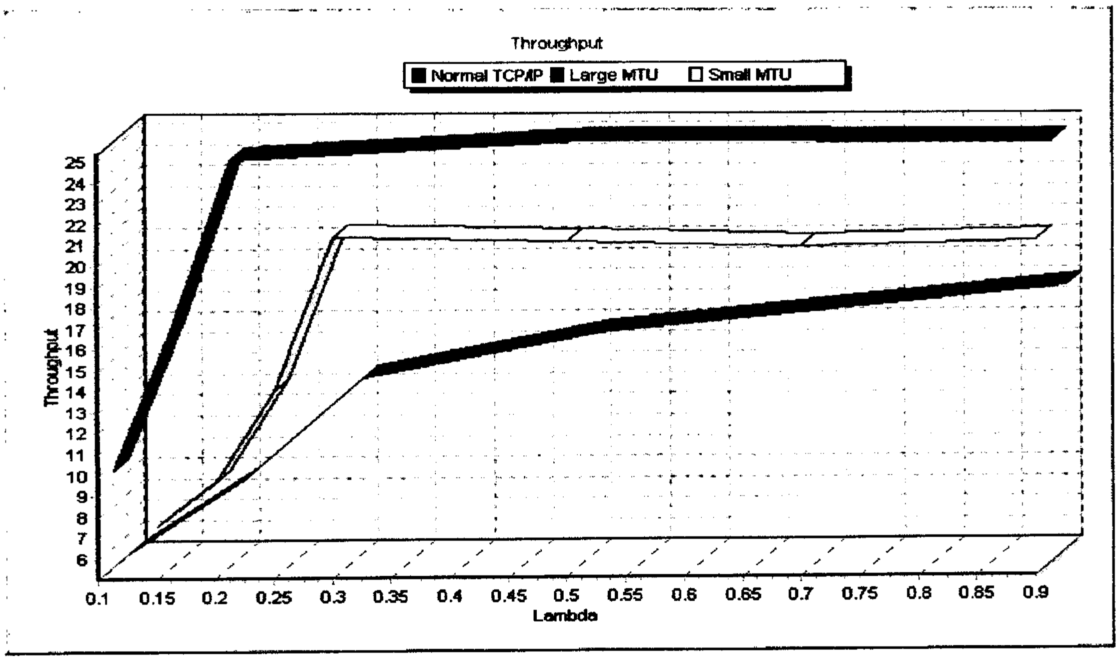


Figure21: Impact of the arrival rate on the Throughput

5.4.4 Aggregation

Our simulation has revealed that small packets are willing to be more aggregated in our new model than the large packets. The process of aggregating more packets has the following benefits:

1) Getting fewer packets at the destination; i.e. less time to process the data.
2) Returning back fewer acknowledgments to the sender, which makes the network more efficient.

Hence we can conclude that the end-to-end delay performance will be improved when using hop-by-hop flow control with aggregation.

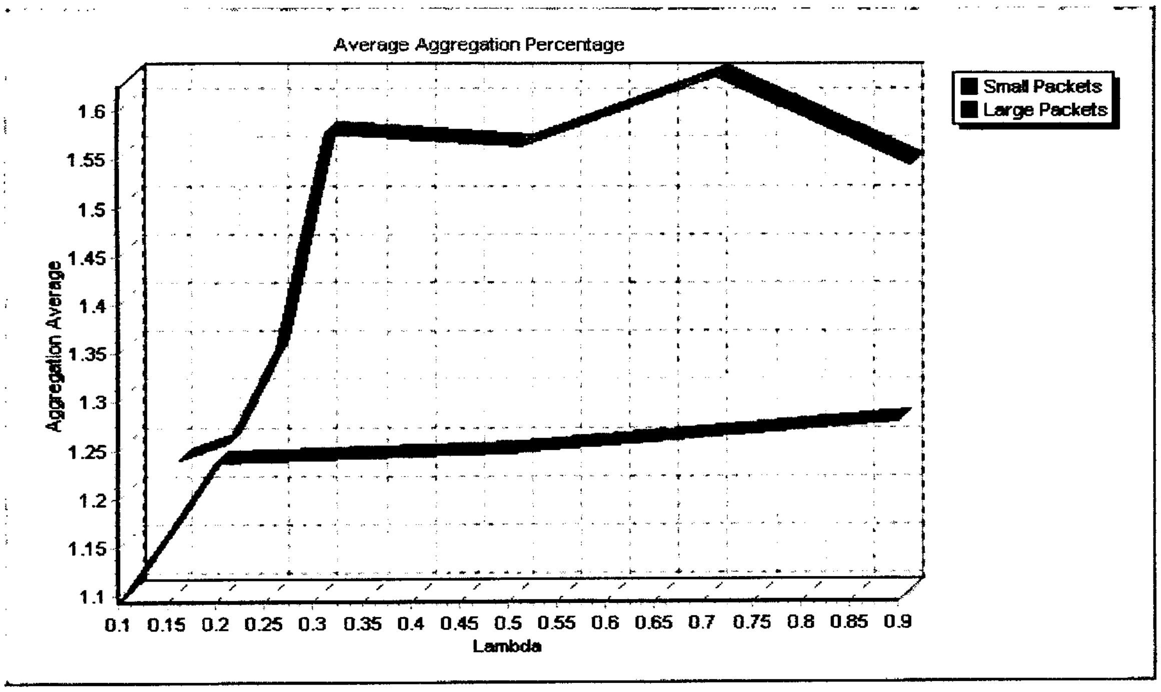


Figure22: Aggregation function of the traffic

5.5 CONCLUSION

In this chapter, we have presented a network simulator used to model TCP/IP networks. This simulator helped us comparing the performance of normal TCP/IP implementation with our proposed technique based on packet aggregation at routers. We have shown that for a large number of packets in the Internet, our new protocol exhibits superior quality as measured by loss rate, bandwidth and round-trip delay. We have proven that this finding is a robust one, and is largely independent of the precise set of hosts measured.

CHAPTER 6

CONCLUSIONS

This chapter summarizes the main results and contributions of the thesis and provides possible extensions to it. We start first by presenting the main results.

6.1 MAIN RESULTS

This thesis deals with flow control problems in general TCP/IP networks. In this thesis a *novel* strategy is introduced, namely *hop-by-hop flow control with packet aggregation.* The packet aggregation procedure is performed at the routers. The proposed technique at routers is adaptive by nature since, with any changes in network parameters, the aggregation rate can be changed to minimize the packet loss, optimize the throughput and minimize the system response time.

An approximate analysis (using Delphi programming tools) for the packet loss and the throughput as a function of the packet size and the packet arrival rate is obtained. And, the ranges of values for the packet size, which give the minimum packet loss (i.e. the optimum performance) for a given system are found.

The main contributions are:

1. We proved that our proposed strategy (i.e. hop-by-hop flow control with packet aggregation), gives a better performance (in term of packet loss, throughput, round-trip delay) than end-to-end flow control methods (as in normal TCP/IP networks).
2. The methodology developed in this thesis is a general one and is not limited to the present problem (i.e. TCP/IP networks) but covers any communication system with buffering constraints.

3. We found that we can minimize the packet loss in the system by changing the aggregation size. In fact, by making the aggregated packet large enough, the departing packets will not interfere with each other because in this case the batch-interdeparture time would be larger than the variance of the network delay. However, having very large packets will increase the end-to-end delay. Hence, a compromise is required to optimize the system performance.
4. We have argued on the need for the tunnels, and further on the need for the aggregation process in the Internet. Such mechanisms provide a support for the end-to-end congestion control, in addition to an important issue which allocates more buffering space to incoming packets which otherwise could have been dropped.

6.2 PROPOSED FUTURE WORK

We think that several aspects of the work conducted in this thesis could be extended. Possible directions to be explored are suggested below:

- For optimally efficient routing decisions, route selection should be integrated so that the choice of route can depend on the bandwidth requested, and so that the stability of the route can be maintained over the duration of the aggregation. Such an integration would lead to more coordination between the choice of which packets to aggregate and the mechanics of establishing the aggregation. This integration is something that requires further research.
- The reassembly of packets in the present models involves only packets that have the same destination, but we can work on different levels:
  1) Reassenbly of packets that are in increasing sequence number.
  2) Reassembly of packets that take the same path.
- Include the process of out-of-order delivery and resequencing in the simulation model.
- Analyze the impact of packet aggregation on the end-to-end delay performance, and find the optimum packet size that minimizes the delay and the packet loss together.

BIBLIOGRAPHY

[AV99] Mark Allman, Vern Paxon.
On estimating end-to-end network path properties. NASA Glem Research Center and AT & T Center for Interent Research. Mar 1999

[Bra89] R. Braden
Requirements for Internet hosts. Communication layers. RFC 1122. October 1989

[CHS99] Andy Collins, Eric Hoffman, Stefan Savage et al.
The end-to-end effects of Internet path selection. University of Washington, Seattle. 1999

[Coh96] Frederick Cohen and Associates.
Internet Holes. The Infosec Super Journal. Sep 1996

[COM2000] Douglas Comer
Internetworking with TCP/IP principles, protocols, and architectures. Volume 1 Fourth edition 2000

[CSA] Neal Cardwell, Stefan Savage, Tom Anderson.
Modeling TCP latency. Department of Computer Science and Engineering. University of Washington.October 1998

[CS94] Douglas Comer, David Stevens
Internetworking with TCP/IP design, implementation and internals. Volume 2. Second edition 1994

[DEZ] Steve Deering, Deborah Estrin, Lixia Zhang et al
RSVP: A new resource reservation protocol. Computer Science department. University of Southern California.IEEE Network, 7(5):8-18- Sep 93.

[DKS89] A.Demers, S. Keshav, S. Shenker
Analysis and simulation of a fair queuing algorithm. Proceedings of ACM SIGCOMM 89

[FJ93] Sally Floyd, Van Jackobson
Random early detection for congestion avoidance. IEEE/ACM Transactions on Networking. August 1993

[FKS98] Wu-chang Feng, Dilip Kandlur, Debanjan Saha, Kang Shin.
Techniques for eliminating packet loss in congested TCP/IP networks.
University of Michigan and Network Systems Department IBM Watson Research Center.1998

[Flo00] Sally Floyd.
Congestion Control Principles. Draft-floyd-cong-03.txt. ACIRI April 2000

[FP97] Sally Floyd, Vern Paxon
Why we don't know how to simulate the Internet. Network Research Group Lawrence Berkeley National Laboratory. University of Berkely. December 1997.

[GP98] Gallo Giorgio, Scaparra Maria Paola.
Routing with minimum fragmentation cost. Computer Department- University of Pisa. Technical Report TR-98-06. June 1998.

[HZ] Ion Stoica, Hui Zhang.
Providing Guaranteed services without per flow management. Carnegie Mellon University 1998.

[KLS] V.P Kumar, T.V Lakshman, D. Stiliadis.
Beyond best effort: Router architectures for the differentiated services of tomorrow's Internet.
Bell Laboratories Lucent technologies USA. 1997.

[LM97] T.V. Lakshman and U. Madhow.
The performance of TCP/IP for networks with high bandwidth-delay products and random-loss. IEEE/ACM Transactions on Networking, June 1997.

[MKS98] Sue Moon, Jim Kurose, Paul Skelly, Don Towsley
Correlation of packet delay and loss in the Internet. Technical report 98-11 University of Massachusetts. January 1998

[MSM97] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi.
The Macroscopic Behavior of the TCP congestion Avoidance Algorithm. Proceedings of ACM SIGCOMM'97 volume 27 number 3, July 1997.

[NS] UCB/LBNL/VINT Network Simulator- ns (version 2)
www-mash.cs.berkeley.edu/ns/

[Pax97] Vern Paxon.
End-to-end Internet packet dynamics. Lawrence Berkeley National Laboratory. June 1997. Proceedings of ACM SIGCOMM 97.

[RFC1191] J. Mogul, S. Deering
Request for Comments. Path MTU Discovery. Stanford University. November 1990.

[RFC1241] D. Mills, R. Woodburn
Request for Comments. A scheme for an Internet Encapsulation protocol: version 1. University of Delaware. July 1991

[RFC1853] W. Simpson
Request for Comments. IP in IP tunneling. Daydreamer. Category: Informational. October 1995

[RFC1858] G.Ziemba, D. Reed, P. Traina
Request for Comments. Security Considerations for IP fragment filtering. CISCO sytems. October 1995.

[RFC2003] C. Perkins.
Request for Comments. IP encapsulation within IP. IBM. October 1996

[RFC2309] Sally Floyd, Van Jackobson, L, Zhang and al
Request for Comments. Recommendations on queue management and congestion avoidance in the internet. April 1998.

[RFC2414] M. Allman, S. Floyd.
Request for Comments. Increasing TCP's initial window. BBN Technologies. Sep 1998.

[RFC2581] M. Allman, V. Paxon, W. Stevens
Request for Comments. TCP congestion Control. NASA Glenn/Sterling Software and ACIRI / ICSI. April 1999

[RFC791] J. Postel
Request for Comments. Internet Protocol. DARPA Internet Program. Protocol Specification. September 1981

[RFC792] J. Postel
Request for Comments. Internet Control Message Protocol. DARPA Internet Program. Protocol Specification. September 1981.

[RFC793] J. Postel
Request for Comments. Transmission Control Protocol. DARPA Internet Program. Protocol Specification. September 1981.

[RFC815] David Clark.
Request for Comments. IP datagram reassembly algorithms. MIT Laboratory for Computer Science. Computer Systems and communication group. July 1982.

[Ste97] Richard Stevens.
TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms. January 1997. RFC 2001

[TMW97] K. Thompson, G. Miller, R. Wilder
Wide Area Internet Traffic Patterns and Characteristics. IEEE Network, 11(6):10-23, Novemvber 1997.

[Wil95] Willinger
Statistical analysis of Ethernet LAN traffic at the source level. Proceedings of SIGCOMM 95, pp 100-113

APPENDIX

```
unit SimFormUnt;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  SimulationUtils, SimulationPanel, ExtCtrls, ComCtrls, ToolWin, Menus,
  ImgList, ImageRepository, MovableUnit;

const
  MINWIDTH = 404;
  MINHEIGHT = 240;
  MINZOOM = -4;
  MAXZOOM = 4;
  ZOOMINCREMENT = 50;
  STATUSPANELWIDTH = 200;

type
  TSimulationForm = class(TForm)
    MainMenu: TMainMenu;
    ActionMnu: TMenuItem;
    MainTBar: TToolBar;
    MainSBar: TStatusBar;
    SpacerPnl: TPanel;
    MainSBox: TScrollBox;
    HelpMnu: TMenuItem;
    LayoutMnu: TMenuItem;
    ViewMnu: TMenuItem;
    ContentsMnuItm: TMenuItem;
    IndexMnuItm: TMenuItem;
    SpacerMnuItm: TMenuItem;
    AboutMnuItm: TMenuItem;
    AlwaysOTMnuItm: TMenuItem;
    Spacer1MnuItm: TMenuItem;
    CloseMnuItm: TMenuItem;
    CentreMnuItm: TMenuItem;
    ModifyMnuItm: TMenuItem;
    ShowGridMnuItm: TMenuItem;
    SnapToGridMnuItm: TMenuItem;
    ConfigureMnuItm: TMenuItem;
    Spacer2MnuItm: TMenuItem;
    Spacer3MnuItm: TMenuItem;
    BackgroundMnuItm: TMenuItem;
    ColorMnuItm: TMenuItem;
    ObjectNameMnuItm: TMenuItem;
    ObjectLabelMnuItm: TMenuItem;
    SpriteMnuItm: TMenuItem;
    Spacer4MnuItm: TMenuItem;
    CloseTBtn: TToolButton;
    SpacerTBtn: TToolButton;    HelpTBtn: TToolButton;
    Spacer5TBtn: TToolButton;    CentreTBtn: TToolButton;
    GridTBtn: TToolButton;    ModifyTBtn: TToolButton;
    Spacer1TBtn: TToolButton;    BackgroundTBtn: TToolButton;
```

```
NamesTBtn: TToolButton;
LabelsTBtn: TToolButton;
AnimationTBtn: TToolButton;
Spacer2TBtn: TToolButton;
ColourTBtn: TToolButton;
Spacer4TBtn: TToolButton;
ZoomoutTBtn: TToolButton;
ZoominTBtn: TToolButton;
Spacer3TBtn: TToolButton;
SnapTBtn: TToolButton;
AutoCentreMnuItm: TMenuItem;
MainSimPnl: TSimulationPanel;
RefreshMnuItm: TMenuItem;
RefreshTBtn: TToolButton;
ConnectorTBtn: TToolButton;
ConnectorLineMnuItm: TMenuItem;
ModelBrowseTBtn: TToolButton;
Spacer5Mnu: TMenuItem;
ModelBrowserMnuItm: TMenuItem;
ZoomoutMnuItm: TMenuItem;
ZoominMnuItm: TMenuItem;
Spacer6Mnu: TMenuItem;
ToolBarMnuItm: TMenuItem;
StatusBarMnuItm: TMenuItem;
Spacer7Mnu: TMenuItem;
procedure AlwaysOTMnuItmClick(Sender: TObject);
procedure AnimationTBtnClick(Sender: TObject);
procedure AutoCentreMnuItmClick(Sender: TObject);
procedure BackgroundTBtnClick(Sender: TObject);
procedure CentreTBtnClick(Sender: TObject);
procedure CloseTBtnClick(Sender: TObject);
procedure ColourTBtnClick(Sender: TObject);
procedure ConfigureMnuItmClick(Sender: TObject);
procedure ConnectorTBtnClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure GridTBtnClick(Sender: TObject);
procedure LabelsTBtnClick(Sender: TObject);
procedure MainSBoxResize(Sender: TObject);
procedure ModelBrowseTBtnClick(Sender: TObject);
procedure ModifyTBtnClick(Sender: TObject);
procedure NamesTBtnClick(Sender: TObject);
procedure RefreshTBtnClick(Sender: TObject);
procedure SnapTBtnClick(Sender: TObject);
procedure ZoominTBtnClick(Sender: TObject);
procedure AboutMnuItmClick(Sender: TObject);
procedure ContentsMnuItmClick(Sender: TObject);
procedure IndexMnuItmClick(Sender: TObject);
procedure FormMouseWheelDown(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
procedure FormMouseWheelUp(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
procedure FormResize(Sender: TObject);
procedure ZoomoutMnuItmClick(Sender: TObject);
procedure ZoominMnuItmClick(Sender: TObject);
procedure ToolBarMnuItmClick(Sender: TObject);
procedure StatusBarMnuItmClick(Sender: TObject);
procedure ApplicationHint(Sender: TObject);
```

```
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
    ModelBrowser: TForm;
    //procedure WMOpenDialogue(var Message: TMessage); message WM_OPENDIALOGUE;
    procedure WMSizing(var Message: TMessage); message WM_SIZING;
  protected
    { Protected declarations }
    ZoomLevel: integer;
  public
    { Public declarations }
    procedure BuildFormCaption;
    procedure OpenDialogue(XPos, YPos: integer); virtual;
    procedure ReleaseDialoguePtr;
    procedure SetSpriteAniBtn(Value: Boolean);
  end;

var
  SimulationForm: TSimulationForm;

implementation

{$R *.DFM}

uses
  ConfigGrid, ModelBrowserDlg;

{Private Procedures -----------------------------------------------------------}

{procedure TSimulationForm.WMOpenDialogue(var Message: TMessage);
begin
  OpenDialogue;
  Message.Result := LRESULT(True);
end;}

procedure TSimulationForm.WMSizing(var Message: TMessage);
var
  pr: PRect;
begin
  pr := PRect(Message.lparam);
  if ((pr.Right - pr.Left) < MINWIDTH) then pr.Right := pr.Left + MINWIDTH;
  if ((pr.Bottom - pr.Top) < MINHEIGHT) then pr.Bottom := pr.Top + MINHEIGHT;
  Message.Result := LRESULT(False);
end;

{Public Procedures ------------------------------------------------------------}

procedure TSimulationForm.BuildFormCaption;
begin
  Caption := '''' + BuildLocationString(Self) + ''' - Simulation Form';
end;

procedure TSimulationForm.OpenDialogue(XPos, YPos: integer);
var
  TopLeft: TPoint;
begin
  if (Caption = '') then BuildFormCaption;
  if Visible then SetFocus
    else
      begin

        //Set initial location of Form
        if (XPos > -1) and (YPos > -1) then
        begin
          TopLeft := Point(XPos, YPos);
          if (TopLeft.x + Width) > Screen.Width then
            TopLeft.x := Screen.Width - Width;
          if (TopLeft.y + Height) > Screen.Height then
            TopLeft.y := Screen.Height - Height;
          SetBounds(TopLeft.x, TopLeft.y, Width, Height);
        end;
        Show;
      end;
end;

procedure TSimulationForm.ReleaseDialoguePtr;
begin
  ModelBrowser := nil;
end;

procedure TSimulationForm.SetSpriteAniBtn(Value: Boolean);
begin
  SpriteMnuItm.Checked := Value;
  AnimationTBtn.Down := Value;
end;

{Delphi Procedures -------------------------------------------------------}

procedure TSimulationForm.AboutMnuItmClick(Sender: TObject);
begin
  DisplayAboutBox(Self);
end;

procedure TSimulationForm.AlwaysOTMnuItmClick(Sender: TObject);
begin
  if (Sender is TMenuItem) then
  begin
    TMenuItem(Sender).Checked := not TMenuItem(Sender).Checked;
    if TMenuItem(Sender).Checked then
      //Self.FormStyle := fsStayOnTop
      SetWindowPos(Handle, HWND_TOPMOST, Left, Top, Width, Height, 0)
      else
        //Self.FormStyle := fsNormal;
        SetWindowPos(Handle, HWND_NOTOPMOST, Left, Top, Width, Height, 0);
  end;
end;

procedure TSimulationForm.AnimationTBtnClick(Sender: TObject);
begin
  MainSimPnl.SpriteAnimation := not MainSimPnl.SpriteAnimation;
  SetSpriteAniBtn(MainSimPnl.SpriteAnimation);
end;

procedure TSimulationForm.ApplicationHint(Sender: TObject);
```

```
begin
  if ((Sender is TApplication) and
    ((Sender as TApplication).Hint <> '')) and
      Windows.PtInRect(BoundsRect, Mouse.CursorPos) then
    MainSBar.Panels[0].Text := GetLongHint((Sender as TApplication).Hint);
end;

procedure TSimulationForm.AutoCentreMnuItmClick(Sender: TObject);
begin
  if (Sender is TMenuItem) then
  begin
    (Sender as TMenuItem).Checked := not (Sender as TMenuItem).Checked;
    CentreTBtn.Enabled := (not (Sender as TMenuItem).Checked and
MainSimPnl.Modify);
    CentreMnuItm.Enabled := (not (Sender as TMenuItem).Checked and
MainSimPnl.Modify);
    if (Sender as TMenuItem).Checked then CentreTBtn.Click;
  end;
end;

procedure TSimulationForm.BackgroundTBtnClick(Sender: TObject);
begin
  MainSimPnl.ShowBackground := not MainSimPnl.ShowBackground;
  BackgroundMnuItm.Checked := MainSimPnl.ShowBackground;
  BackgroundTBtn.Down := MainSimPnl.ShowBackground;
end;

procedure TSimulationForm.CentreTBtnClick(Sender: TObject);
var
  cr: TRect;
  TopLeft: TPoint;
begin
  if (MainSimPnl.Align = alNone) then
  with MainSimPnl do
  begin
    cr := MainSBox.ClientRect;
    TopLeft.x := 0; TopLeft.y := 0;
    if (Width < cr.Right) then TopLeft.x := (cr.Right - Width) shr 1;
    if (Height < cr.Bottom) then TopLeft.y := (cr.Bottom - Height) shr 1;
    SetBounds(TopLeft.x - MainSBox.HorzScrollBar.Position, TopLeft.y  -
MainSBox.VertScrollBar.Position,
      Width, Height);
  end;
end;

procedure TSimulationForm.CloseTBtnClick(Sender: TObject);
begin
  Close;
end;

procedure TSimulationForm.ColourTBtnClick(Sender: TObject);
begin
  ImageRepositoryForm.ColorDialog.Color := MainSimPnl.Color;
  if ImageRepositoryForm.ColorDialog.Execute then
    MainSimPnl.Color := ImageRepositoryForm.ColorDialog.Color;
end;
```

```
procedure TSimulationForm.ConfigureMnuItmClick(Sender: TObject);
begin
  ConfigGridFrm := TConfigGridFrm.Create(Application);
  ConfigGridFrm.SimulationPanel := MainSimPnl;
  ConfigGridFrm.InitialiseForm;
  ConfigGridFrm.ShowModal;
  ConfigGridFrm.Release;
end;

procedure TSimulationForm.ConnectorTBtnClick(Sender: TObject);
begin
  MainSimPnl.ShowConnectingLines := not MainSimPnl.ShowConnectingLines;
  ConnectorLineMnuItm.Checked := MainSimPnl.ShowConnectingLines;
  ConnectorTBtn.Down := MainSimPnl.ShowConnectingLines;
end;

procedure TSimulationForm.ContentsMnuItmClick(Sender: TObject);
begin
  DisplayHelpFile(hmContents, HelpContext);
end;

procedure TSimulationForm.FormActivate(Sender: TObject);
begin
  Application.OnHint := ApplicationHint;
end;

procedure TSimulationForm.FormCreate(Sender: TObject);
begin
  ZoomLevel := 0;

  MainSimPnl.Modify := not MainSimPnl.Modify;
  ModifyTBtn.Click;
  ShowGridMnuItm.Checked := MainSimPnl.ShowGrid;
  SnapToGridMnuItm.Checked := MainSimPnl.SnapGrid;
  SnapTBtn.Down := MainSimPnl.SnapGrid;
  GridTBtn.Down := MainSimPnl.ShowGrid;
  BackgroundMnuItm.Checked := MainSimPnl.ShowBackground;
  BackgroundTBtn.Down := MainSimPnl.ShowBackground;
  ConnectorLineMnuItm.Checked := MainSimPnl.ShowConnectingLines;
  ConnectorTBtn.Down := MainSimPnl.ShowConnectingLines;
  ObjectNameMnuItm.Checked := MainSimPnl.ShowObjectName;
  NamesTBtn.Down := MainSimPnl.ShowObjectName;
  ObjectLabelMnuItm.Checked := MainSimPnl.ShowObjectLabel;
  LabelsTBtn.Down := MainSimPnl.ShowObjectLabel;
  SpriteMnuItm.Checked := MainSimPnl.SpriteAnimation;
  AnimationTBtn.Down := MainSimPnl.SpriteAnimation;
  StatusBarMnuItm.Checked := MainSBar.Visible;
  ToolBarMnuItm.Checked := MainTBar.Visible;
end;

procedure TSimulationForm.FormMouseWheelDown(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);

  procedure SetScrollBar(SBar: TControlScrollBar);
  begin
    with SBar do
      if IsScrollBarVisible and (Position < Range) then
      begin
        Position := Position + (Increment * Trunc(Mouse.WheelScrollLines *
          MainSimPnl.ScaleFactor));
        Handled := True;
    end;
  end;

begin
  if (ssCtrl in Shift) then
    SetScrollBar(MainSBox.HorzScrollBar)
    else
      SetScrollBar(MainSBox.VertScrollBar);
end;

procedure TSimulationForm.FormMouseWheelUp(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);

  procedure SetScrollBar(SBar: TControlScrollBar);
  begin
    with SBar do
      if IsScrollBarVisible and (Position > 0) then
      begin
        Position := Position - (Increment * Trunc(Mouse.WheelScrollLines *
          MainSimPnl.ScaleFactor));
        Handled := True;
    end;
  end;

begin
  if (ssCtrl in Shift) then
    SetScrollBar(MainSBox.HorzScrollBar)
    else
      SetScrollBar(MainSBox.VertScrollBar);
end;

procedure TSimulationForm.FormResize(Sender: TObject);
begin
  MainSBar.Panels[0].Width := MainSBar.Width - STATUSPANELWIDTH;
end;

procedure TSimulationForm.GridTBtnClick(Sender: TObject);
begin
  MainSimPnl.ShowGrid := not MainSimPnl.ShowGrid;
  ShowGridMnuItm.Checked := MainSimPnl.ShowGrid;
  GridTBtn.Down := MainSimPnl.ShowGrid;
end;

procedure TSimulationForm.IndexMnuItmClick(Sender: TObject);
begin
  DisplayHelpFile(hmIndex, HelpContext);
end;

procedure TSimulationForm.LabelsTBtnClick(Sender: TObject);
begin
  MainSimPnl.ShowObjectLabel := not MainSimPnl.ShowObjectLabel;
  ObjectLabelMnuItm.Checked := MainSimPnl.ShowObjectLabel;
  LabelsTBtn.Down := MainSimPnl.ShowObjectLabel;
```

```
end;

procedure TSimulationForm.MainSBoxResize(Sender: TObject);
begin
  if (Showing and AutoCentreMnuItm.Checked) then
    CentreTBtn.Click;
end;

procedure TSimulationForm.ModelBrowseTBtnClick(Sender: TObject);
begin
  if (ModelBrowser <> nil)
  then
    ModelBrowser.Show
  else
    begin
      ModelBrowser := TModelBrowserForm.Create(Application);
      TModelBrowserForm(ModelBrowser).SimulationForm := self;
      TModelBrowserForm(ModelBrowser).InitialiseForm;
      ModelBrowser.Show;
    end;
end;

procedure TSimulationForm.ModifyTBtnClick(Sender: TObject);
begin
  MainSimPnl.Modify := not MainSimPnl.Modify;
  ModifyTBtn.Down := MainSimPnl.Modify;
  ModifyMnuItm.Checked := MainSimPnl.Modify;
  CentreTBtn.Enabled := (MainSimPnl.Modify and not AutoCentreMnuItm.Checked);
  CentreMnuItm.Enabled := (MainSimPnl.Modify and not AutoCentreMnuItm.Checked);
  GridTBtn.Enabled := MainSimPnl.Modify;
  ShowGridMnuItm.Enabled := MainSimPnl.Modify;
  SnapToGridMnuItm.Enabled := MainSimPnl.Modify;
  SnapTBtn.Enabled := MainSimPnl.Modify;
  ConfigureMnuItm.Enabled := MainSimPnl.Modify;
  BackgroundTBtn.Enabled := MainSimPnl.Modify;
  BackgroundMnuItm.Enabled := MainSimPnl.Modify;
  ConnectorLineMnuItm.Enabled := MainSimPnl.Modify;
  ConnectorTBtn.Enabled := MainSimPnl.Modify;
  NamesTBtn.Enabled := MainSimPnl.Modify;
  ObjectNameMnuItm.Enabled := MainSimPnl.Modify;
  LabelsTBtn.Enabled := MainSimPnl.Modify;
  ObjectLabelMnuItm.Enabled := MainSimPnl.Modify;
  AnimationTBtn.Enabled := MainSimPnl.Modify;
  SpriteMnuItm.Enabled := MainSimPnl.Modify;
  ColourTBtn.Enabled := MainSimPnl.Modify;
  ColorMnuItm.Enabled := MainSimPnl.Modify;
  ZoominTBtn.Enabled := (MainSimPnl.Modify and not (ZoomLevel = MAXZOOM));
  ZoomoutTBtn.Enabled := (MainSimPnl.Modify and not (ZoomLevel = MINZOOM));
  ZoominMnuItm.Enabled := ZoominTBtn.Enabled;
  ZoomoutMnuItm.Enabled := ZoomoutTBtn.Enabled;
end;

procedure TSimulationForm.NamesTBtnClick(Sender: TObject);
begin
  MainSimPnl.ShowObjectName := not MainSimPnl.ShowObjectName;
  ObjectNameMnuItm.Checked := MainSimPnl.ShowObjectName;
  NamesTBtn.Down := MainSimPnl.ShowObjectName;
```

```
end;

procedure TSimulationForm.RefreshTBtnClick(Sender: TObject);
begin
  MainSimPnl.Repaint;
end;

procedure TSimulationForm.SnapTBtnClick(Sender: TObject);
begin
  MainSimPnl.SnapGrid := not MainSimPnl.SnapGrid;
  SnapToGridMnuItm.Checked := MainSimPnl.SnapGrid;
  SnapTBtn.Down := MainSimPnl.SnapGrid;
end;

procedure TSimulationForm.StatusBarMnuItmClick(Sender: TObject);
begin
  MainSBar.Visible := not MainSBar.Visible;
  StatusBarMnuItm.Checked := MainSBar.Visible;
end;

procedure TSimulationForm.ToolBarMnuItmClick(Sender: TObject);
begin
  MainTBar.Visible := not MainTBar.Visible;
  ToolBarMnuItm.Checked := MainTBar.Visible;
end;

procedure TSimulationForm.ZoominMnuItmClick(Sender: TObject);
begin
  ZoominTBtnClick(ZoominTBtn);
end;

procedure TSimulationForm.ZoominTBtnClick(Sender: TObject);
var
  ClipBox: TRect;
  MidValX, MidValY: single;
  Multiplier: integer;
begin
  if (Sender is TToolButton) then
  begin

    {Calculate middle of current viewport}
    ClipBox := MainSimPnl.Canvas.ClipRect;
    MidValX := (ClipBox.Left + ((ClipBox.Right - ClipBox.Left) shr 1)) /
MainSimPnl.Width;
    MidValY := (ClipBox.Top + ((ClipBox.Bottom - ClipBox.Top) shr 1)) /
MainSimPnl.Height;

    if ((Sender as TToolButton) = ZoominTBtn) then
    begin
      inc(ZoomLevel);
      ZoominTBtn.Enabled := not (ZoomLevel = MAXZOOM);
      ZoomoutTBtn.Enabled := True;
    end
    else
      if ((Sender as TToolButton) = ZoomoutTBtn) then
      begin
       dec(ZoomLevel);
       ZoomoutTBtn.Enabled := not (ZoomLevel = MINZOOM);
       ZoominTBtn.Enabled := True;
      end;
    ZoominMnuItm.Enabled := ZoominTBtn.Enabled;
    ZoomoutMnuItm.Enabled := ZoomoutTBtn.Enabled;
    if (ZoomLevel < 0) then
      Multiplier := 10000 div (100 + (-ZoomLevel * ZOOMINCREMENT))
      else
        Multiplier := 100 + (ZoomLevel * ZOOMINCREMENT);

    MainSimPnl.Visible := False;
    MainSimPnl.ScalePanelBy(Multiplier, 100);
    if AutoCentreMnuItm.Checked then
      CentreTBtn.Click
      else
        with MainSimPnl do
          SetBounds(-MainSBox.HorzScrollBar.Position, -
MainSBox.VertScrollBar.Position, Width, Height);
    MainSimPnl.Visible := True;

    {Scroll so same area is in middle of viewport}
    MainSBox.HorzScrollBar.Position := Round(MainSBox.HorzScrollBar.Range *
MidValX) - MainSBox.ClientWidth shr 1;
    MainSBox.VertScrollBar.Position := Round(MainSBox.VertScrollBar.Range *
MidValY) - MainSBox.ClientHeight shr 1;
  end;
end;

procedure TSimulationForm.ZoomoutMnuItmClick(Sender: TObject);
begin
  ZoominTBtnClick(ZoomoutTBtn);
end;

end.
```

```
unit MainSimFormUnt;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  SimulationUtils, SimFormUnt, Menus, SimulationPanel, ExtCtrls, ComCtrls,
ToolWin,
  ObjectBasement, ObjectBase, EventController, MovableUnit, SimSink,
  SimStore, SimSingleProcess, SpriteControl, MovableElement, MovableObject,
  EventProcessor, EventDisruptor, EventBlock, MaterialFlow,
  MaterialFlowAni, MaterialFlowStats, MaterialBlock, MaterialDisruptor,
  MaterialFlowMU, MaterialFlowCap, SimSource, StdCtrls, SimDistribution,
  SimGenerator, SimFlowControl, SimMultiProcess, Db, ADODB, SimStatistics;

const INF    =    1000;
        maxobject =    50;
        CONSTANT  =    0.33333;

type
        ARRN      = array[1..maxobject] of integer;

  TSynchronizeEvents = (seNone, seClose, seZoomIn, seZoomOut);

  TMainSimForm = class(TSimulationForm)
    EventController: TEventController;
    SimSource1: TSimSource;
    packet: TMovableObject;
    SpriteControl1: TSpriteControl;
    SimSingleProcess1: TSimSingleProcess;
    SimStore1: TSimStore;
    SimSingleProcess2: TSimSingleProcess;
    SimSink1: TSimSink;
    ListBox1: TListBox;
    ListBox2: TListBox;
    ListBox3: TListBox;
    Edit3: TEdit;
    SimDistribution1: TSimDistribution;
    SimDistribution2: TSimDistribution;
    Label2: TLabel;
    Label3: TLabel;
    SimGenerator1: TSimGenerator;
    ZeroDist: TSimDistribution;
    Label4: TLabel;
    ListBox4: TListBox;
    ListBox5: TListBox;
    SimFlowControl1: TSimFlowControl;
    SimSingleProcess3: TSimSingleProcess;
    recycle: TSimSink;
    SimDistribution3: TSimDistribution;
    SimDistribution4: TSimDistribution;
    Aggregate: TMovableObject;
    SpriteControl2: TSpriteControl;
    Label1: TLabel;
    Label5: TLabel;
    dropped: TSimSink;
    ListBox6: TListBox;
```

```
Label6: TLabel;
SimSink2: TSimSink;
SimSingleProcess4: TSimSingleProcess;
SimSingleProcess5: TSimSingleProcess;
SimFlowControl2: TSimFlowControl;
SimSink3: TSimSink;
SimDistribution5: TSimDistribution;
ListBox7: TListBox;
SimMultiProcess1: TSimMultiProcess;
SimMultiProcess2: TSimMultiProcess;
SimMultiProcess3: TSimMultiProcess;
SpriteControl3: TSpriteControl;
fragment: TMovableObject;
Label7: TLabel;
ListBox8: TListBox;
Label8: TLabel;
Label9: TLabel;
ADOConnection1: TADOConnection;
Source: TADOTable;
SourceSequenceNb: TAutoIncField;
SourceSimTime: TIntegerField;
SourceSize: TSmallintField;
SourceTTL: TWordField;
SourceAgg: TWordField;
Queue: TADOTable;
QueueSequenceNb: TAutoIncField;
QueueSimTime: TIntegerField;
QueueAggSize: TSmallintField;
QueuePMTU: TSmallintField;
QueueTTL: TWordField;
QueueAgg: TWordField;
QueuePacketsAggregated: TWideStringField;
SimDistribution6: TSimDistribution;
Destination: TADOTable;
DestinationSequenceNb: TAutoIncField;
DestinationSimTime: TIntegerField;
DestinationSize: TSmallintField;
DestinationTTL: TWordField;
DestinationAgg: TWordField;
TTLDrop: TADOTable;
TTLDropSequenceNb: TAutoIncField;
TTLDropSimTime: TIntegerField;
TTLDropSize: TSmallintField;
TTLDropTTL: TWordField;
TTLDropAgg: TWordField;
SimStatistics1: TSimStatistics;
SimStatistics2: TSimStatistics;
SimDistribution7: TSimDistribution;
SimDistribution8: TSimDistribution;
SimDistribution9: TSimDistribution;
CongestionDropped: TADOTable;
CongestionDroppedSequenceNb: TIntegerField;
CongestionDroppedSimTime: TIntegerField;
CongestionDroppedSize: TSmallintField;
CongestionDroppedAgg: TWordField;
SourceCurrentTime: TWideStringField;
QueueBW: TSmallintField;
```

```
    EndTunnel: TADOTable;
    SimFlowControl3: TSimFlowControl;
    Congestion: TSimSink;
    QueueSeqNbs: TWideStringField;
    EndTunnelSequenceNb: TIntegerField;
    EndTunnelCurrentTime: TWideStringField;
    EndTunnelSimTime: TIntegerField;
    EndTunnelSize: TSmallintField;
    EndTunnelPacketsAggregated: TWideStringField;
    EndTunnelSeqNbs: TWideStringField;
    DestinationCurrentTime: TWideStringField;
    Label10: TLabel;
    Routers: TADOTable;
    RoutersSequenceNb: TAutoIncField;
    RoutersSimTime: TIntegerField;
    RoutersSize: TSmallintField;
    RoutersRouter: TSmallintField;
    RoutersPMTU: TSmallintField;
    RoutersBW: TIntegerField;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure EventControllerStop(Sender: TObject);
    procedure ZoominTBtnClick(Sender: TObject);
    procedure EventControllerSafeCall(Sender: TObject; SimTime: Int64);
    procedure EventControllerEventDone(Sender: TObject; SimTime: Int64);
    procedure EventControllerInit(Sender: TObject);
    procedure EventControllerReset(Sender: TObject);
    procedure EventControllerStart(Sender: TObject);
    procedure EventControllerStep(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    PROCEDURE KNAPAPPROX(N:INTEGER;VAR P,W,X:ARRN;VAR V,PROFIT:INTEGER;VAR
EPS:REAL);
    procedure SimSingleProcess1MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimStore1MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimGenerator1Generate_Int(Sender: TObject; SimTime: Int64);
    procedure SimSink1MUDestroy(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSource1MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimStore1MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    function SimFlowControl1FlowMethod(Sender: TObject): Integer;
    function whichRouterIsFree: Integer;
    function EmptyQueue(Queue: TsimStore): Boolean;
    procedure recycleMUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSink1MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure recycleMUDestroy(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure droppedMUDestroy(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure droppedMUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSink2MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSink3MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimMultiProcess1MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimMultiProcess2MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimMultiProcess3MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimStore1Blocked(Sender: TObject; SimTime: Int64);
    procedure SimSingleProcess1MUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimGenerator1Generate_Dur(Sender: TObject; SimTime: Int64);
    function SimFlowControl3FlowMethod(Sender: TObject): Integer;
    procedure CongestionMUEntrance(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSingleProcess3MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSingleProcess5MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSingleProcess4MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);
    procedure SimSingleProcess2MUExit(Sender: TObject; SimTime: Int64;
      MU: TMovableUnit);

  private
    { Private declarations }
    RunningECCount: integer;
    SynchEvent: TSynchronizeEvents;
  protected
    { protected declarations }
    procedure loaded; override;
  public
    { Public declarations }
    mudestroyed: integer;
    blockedmu: integer;
    countpackets: integer;
    constructor Create(AOwner: TComponent); override;
    procedure WriteStatus(Status: string);
  end;

var
  MainSimForm: TMainSimForm;
  N,        Nextint           :     integer;
       P,  W             :     ARRN;
     X                   :     ARRN;
     V,    PROFIT             :     integer;
       EPS               :     real;

implementation

{$R *.DFM}

{ Register all 'Child' Simulation Form Units Below!!!
  e.g. uses TestUnit; }
```

```
Function TMainSimForm.EmptyQueue(Queue: TsimStore): Boolean;
var
  i: integer;
  check: integer;
begin
        check := 0;
        for i := 0 to Queue.NumMu-1 do
          if Queue.Content(i) <> nil then
          else
                inc(check);
        if check = Queue.NumMU then
          Result := True
        else Result := False;

end;


Procedure TMainSimForm.KNAPAPPROX(N:integer;var P,W,X:ArrN;var
V,Profit:integer;var EPS:real);
var
  I,J,K,L,MaxP1,MaxP2,MaxP3,PP,Q,R,S,U,VV:integer;
  check: boolean;

   Procedure LB (var G, H, Q, U: integer);
   var
     K: integer;
   begin
      K := 0;
      repeat
        K := k + 1;
        if (K <> G) and (K <> H) and (W[K] <= U) then
        begin
          Q := Q + P[k];
          U := U - W[k];
        end;
      until k = N;
   end;

    procedure Max;
    begin
      if P[i] > MaxP1 then
      begin
        MaxP3 := Maxp2;
        Maxp2 := Maxp1;
        Maxp1 := P[i];
      end
      else
        if P[i] > Maxp2 then
        begin
          Maxp3 := Maxp2;
          Maxp2 := P[i]
        end
        else if P[i] > Maxp3 then
          Maxp3 := P[i]
   end;

begin
```

```
  i := 1;
  u := v;
  profit := 0;
  maxp1 := 0; maxp2 := 0; maxp3 := 0;
  check := false;
  while (w[i] <= u) do
  begin
    u := u - w[i];
    max;
    x[i] := 1;
    profit := profit + p[i];
    i := i + 1;
    check := true;
    if i = N then break;
  end;
{  if check = false then
  begin
    exit;
  end;
}
  i := i - 1;
  S := i;
  repeat
    i := i + 1;
    if w[i] <= u then
    begin
      u := u - w[i];
      x[i] := 1;
      profit := profit + p[i];
    end
    else
      x[i] := 0;
      max;
  until i = n;
  q := profit;
  k := 0;
  L := 0;
  for i := S to n do
    if x[i] <> 1 then
    begin
      vv := v - w[i];
      pp := p[i];
      LB(i, i, pp, vv);
      if pp > profit then
      begin
        profit := pp;
        k := i;
      end;
    end;
    r := s;
    for i:= 1 to n-1 do
    begin
      if i > s then
        r := i;
      for j := r+1 to n do
      begin
        vv := v-w[i] -w[j];
        if vv >= 0 then
        begin
          pp := p[i] + p[j];
          LB (i, j, pp, vv);
          if pp > profit then
          begin
            profit := pp;
            k := i;
            l := j;
          end
        end
      end
    end;
    if profit > q then
    begin
      if k > 0 then
      begin
        v := v -w[k];
        x[k] := 1;
      end;
      if l > 0 then
      begin
        v := v - w[l];
        x[l] := 1;
      end;
      for i := 1 to n do
        if (i <> k) and (i <> l) then
          if w[i] <= v then
          begin
            x[i] := 1;
            v := v - w[i];
          end
          else
            x[i] := 0;
    end;
    EPS := maxp3/profit;
    if EPS > 0.33333 then
        EPS := 0.33333
end;

procedure TMainSimForm.loaded;
begin
  inherited;
  WriteStatus('Idle');
end;

constructor TMainSimForm.Create(AOwner: TComponent);
begin

  { Register all 'Child' Simulation Forms Below!!!
    e.g. RegisterClass(TestUnit.TTestSimulationForm);
    Note: Place code BEFORE 'inherited' statement }

  inherited;
  Caption := '''.' + Name + ''' - Main Simulation Form';
end;
```

```
procedure TMainSimForm.WriteStatus(Status: string);
var
  Counter: integer;
  Component: TComponent;

  procedure RecurseWriteStatus(Component: TComponent);
  var
    Counter: integer;
    Control: TComponent;
  begin
    for Counter := 0 to Component.ComponentCount - 1 do
    begin
      Control := Component.Components[Counter];
      if (Control is TSimulationForm) then
      begin
        (Control as TSimulationForm).MainSBar.Panels[1].Text :=
MainSBar.Panels[1].Text;
        RecurseWriteStatus(Control);
      end;
    end;
  end;

begin
  MainSBar.Panels[1].Text := 'Simulation Status: ' + Status;
  if not (csDesigning in ComponentState) then
  begin
    for Counter := 0 to Application.ComponentCount - 1 do
    begin
      Component := Application.Components[Counter];
      if (Component is TSimulationForm) then RecurseWriteStatus(Component);
    end;
  end;
end;

procedure TMainSimForm.EventControllerSafeCall(Sender: TObject;
  SimTime: TSimulationTime);
begin
  case SynchEvent of
    seZoomIn: begin
                dec(RunningECCount);
                if RunningECCount <= 0 then ZoominTBtnClick(ZoominTBtn);
              end;
    seZoomOut: begin
                 dec(RunningECCount);
                 if RunningECCount <= 0 then ZoominTBtnClick(ZoomoutTBtn);
               end;
  end;
  SynchEvent := seNone;

  //Disconnect Event
  if (Sender is TEventController) then
    (Sender as TEventController).OnSafeCall := nil;
end;

procedure TMainSimForm.EventControllerEventDone(Sender: TObject;
  SimTime: TSimulationTime);
begin
```

```
      label10.caption := inttostr(simtime);
  WriteStatus('Finished');
end;

procedure TMainSimForm.EventControllerInit(Sender: TObject);
begin
  WriteStatus('Initialising');
  RunningECCount := 0;
  SynchEvent := seNone;
end;

procedure TMainSimForm.EventControllerReset(Sender: TObject);
begin
  WriteStatus('Resetting');
end;

procedure TMainSimForm.EventControllerStart(Sender: TObject);
begin
  mudestroyed := 0;
  blockedmu := 0;
  countpackets := 0;
  WriteStatus('Running');
end;

procedure TMainSimForm.EventControllerStep(Sender: TObject);
begin
  WriteStatus('Stepping');
end;

procedure TMainSimForm.EventControllerStop(Sender: TObject);
begin
  case SynchEvent of
    seClose: begin
               dec(RunningECCount);
               if RunningECCount <= 0 then
                 Windows.PostMessage(Handle, WM_CLOSE, 0, 0);
             end;
  end;
  SynchEvent := seNone;
  WriteStatus('Stopped');
end;

procedure TMainSimForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  Counter: integer;
begin
  with MainSimPnl do
    for Counter := 0 to ControlCount - 1 do
      if (Controls[Counter] is TEventController) then
      begin
        if (Controls[Counter] as TEventController).ThreadStopped then
          (Controls[Counter] as TEventController).ShutDown
          else
          begin
            inc(RunningECCount);
            (Controls[Counter] as TEventController).Stop;
          end;
      end;

  //Check if they have all stopped
  if RunningECCount = 0 then
    Action := caFree
    else
      begin
        Action := caNone;
        SynchEvent := seClose;
      end;
end;

procedure TMainSimForm.FormCreate(Sender: TObject);
begin
  inherited;
  Application.HintHidePause := 5000;
  RunningECCount := 0;
  SynchEvent := seNone;
end;

procedure TMainSimForm.ZoominTBtnClick(Sender: TObject);
var
  Counter: integer;
begin
  with MainSimPnl do
    for Counter := 0 to ControlCount - 1 do
      if (Controls[Counter] is TEventController) and
        not assigned((Controls[Counter] as TEventController).OnSafeCall) and
          not (Controls[Counter] as TEventController).ThreadStopped then
      begin
        (Controls[Counter] as TEventController).OnSafeCall :=
EventControllerSafeCall;
        inc(RunningECCount);
      end;

  //Check if they have all stopped
  if RunningECCount = 0 then
    inherited
    else
      begin
        if (Sender = ZoominTBtn) then
          SynchEvent := seZoomIn
          else if (Sender = ZoomoutTBtn) then
            SynchEvent := seZoomOut;
      end;
end;

procedure TMainSimForm.SimSingleProcess1MUExit(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
  contentind: integer;
  stockobj: TSimStore;
  size: integer;
  SeqNb: Integer;
  Agg: byte;
begin
```

```
  inherited;
        stockobj := simstore1;
        if stockobj.nummu >= stockobj.capacity then
         begin
           mu.SimProperties.Items[0].GetValue(size);
           mu.SimProperties.Items[3].GetValue(Agg);
           mu.SimProperties.Items[2].GetValue(SeqNb);
           CongestionDropped.Append;
           CongestionDroppedSimTime.Value := SimTime;
           CongestionDroppedSize.Value := Size;
           CongestionDroppedAgg.Value := Agg;
           CongestionDroppedSequenceNb.Value := SeqNb;
           CongestionDropped.Post;
        end;
end;

procedure TMainSimForm.SimStore1MUExit(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
var
 PAcketSize, TTLValue: Integer;
 InnerPackets: string;
begin
  inherited;
 label3.caption := IntTostr(simstore1.nummu);

end;

procedure TMainSimForm.SimGenerator1Generate_Int(Sender: TObject;
  SimTime: Int64);
var
  contentind: integer;
  packet:  integer;
  stockobj: TSimStore;
  obj: TMovableUnit;
  counter: integer;
  whichpath: integer;
  router : TMAterialFlow;
  Aggregated: Integer;
  irand: integer;
  i,j: integer;
  PacketsAggregated, PAcketsFragmented: String;
  packetsize,size,test, index: integer;
  a:array[1..5] of integer;
  PMTU, BW, TTL: integer;
  PathSize: Integer;
  Seq: LongInt;
  SeqNbs : String;
begin

  stockobj := SimStore1;
  obj := nil;
  Listbox5.Clear;
  listbox3.clear;

  with stockobj do
  begin
        for contentind := 0 to NumMU - 1 do
        begin
             obj := content(contentind);
             if obj = nil then
               listbox5.items.append('0')
             else
             begin

Content(contentind).SimProperties.Items[0].GetValue(packetsize);
                  listbox5.items.append(inttostr(packetsize));
             end;
        end;

        if NumMu > 1 then
        begin

            EPS := CONSTANT;
            N := Stockobj.NumMu;
            whichpath := whichrouterisfree;
            if whichpath = 4 then
            begin
                // showmessage('congestion');
                 exit;
            end;

            router := simflowcontrol1.Successors.Items[WhichPath].Successor;
            router.SimProperties.items[0].getvalue(v);
            router.SimProperties.items[1].getvalue(BW);
           PathSize := v;

            for counter := 1 to N do
            begin
                P[counter] := StrToInt(listbox5.Items.Strings[counter-1]);
                W[counter] := StrToInt(listbox5.Items.Strings[counter-1]);
            end;

            if (EmptyQueue(simstore1)) and (countpackets
=simsource1.NumToGen)then
            begin
                simgenerator1.stop := 1;
                exit;
            end;

            KNAPAPPROX(N,P,W,X,V,PROFIT,EPS);

            for index := 0 to N do
              listbox3.Items.Strings[index] := inttostr(X[index+1]);

// fragmentation part
            index := 0;
            if (profit = 0) or (v < 0) then
            begin
                for index := 0 to listbox5.items.count-1 do
                begin
                  whichpath := whichrouterisfree;
                  if whichpath = 4 then
                  begin
                       // showmessage('congestion');
```

```
                    exit;
                end;
                router :=
simflowcontrol1.Successors.Items[WhichPath].Successor;
                router.SimProperties.items[0].getvalue(PMTU);
                Packet := strtoint(listbox5.Items.strings[index]);
                if packet > PMTU then
                begin
                  for i := 1 to 5 do
                      a[i] := 0;

                  j := Packet div PMTU;
                  for i := 1 to j do
                      a[i] := PMTU;
                  if Packet mod PMTU <> 0 then
                      a[i] := (Packet - j*PMTU);

                  i := 1;
                  PacketsFragmented := '';
                  SeqNbs := '';

                  stockobj.Content(i-1).Move(recycle);
                  listbox5.Items.Strings[i-1] := '0';

                  while a[i] <> 0 do
                  begin
                      fragment.SimProperties.Items[0].SetValue(a[i]);
                      fragment.CreateUnit.Move(router);
                      listbox7.Items.add(inttostr(a[i]));
                         inc(i);

                    Queue.Append;
                    QueueSimTime.Value := SimTime;
                    QueueAggSize.Value := a[i-1];
                    QueuePMTU.Value := PathSize;
                    QueueBW.Value := BW;
                    QueueAgg.Value := 3;
                    QueueTTL.Value := 5;
                    PacketsFragmented:= PacketsFragmented + ' ' +
Inttostr(a[i]);
                    QueuePacketsAggregated.Value := PacketsFragmented;
                    Queue.Post;
                  end;
          end;
      end;
end
//end of fragmentation
  else begin
          aggregated := 0;
          for counter := 1 to N do
          begin
           listbox3.Items.Strings[counter-1] := inttostr(X[Counter]);
           if X[counter] <> 0 then
              Aggregated := Aggregated + P[counter];
          end;
          router.SimProperties.items[1].getvalue(BW);
          aggregate.SimProperties.Items[0].SetValue(Aggregated);
```

```
            TTL := 5;
            aggregate.SimProperties.Items[1].SetValue(TTL);
            aggregate.CreateUnit.Move(router);
            PacketsAggregated := '';
            for counter := 1 to N do
              if (X[counter] <> 0) and (stockobj.content(counter-1)<>nil) then
              begin
               stockobj.content(counter-
1).SimProperties.Items[0].GetValue(packetsize);
               stockobj.content(counter-1).SimProperties.Items[2].GetValue(Seq);
               stockobj.Content(counter-1).Move(recycle);
               PacketsAggregated := PacketsAggregated + ' ' +
Inttostr(PacketSize);
               SeqNbs := SeqNbs + ' ' + Inttostr(Seq);
               stockobj.content(counter-
1).SimProperties.Items[4].SetValue(PacketsAggregated);
               listbox5.Items.Strings[counter-1] := '0';
              end;
              aggregate.SimProperties.Items[4].SetValue(PacketsAggregated);
              aggregate.SimProperties.Items[5].SetValue(SeqNbs);
               Queue.Append;
               QueueSimTime.Value := SimTime;
               QueueAggSize.Value := Aggregated;
               QueuePMTU.Value := PathSize;
               QueueAgg.Value := 2;
               QueueBW.Value := BW;
               QueueTTL.Value := 5;
               QueuePacketsAggregated.Value := PacketsAggregated;
               QueueSeqNbs.value := SeqNbs;
               Queue.Post;
        end;
  end;
 end;
        if mudestroyed = simsource1.NumToGen-1 then
          simgenerator1.stop := 1;
end;


procedure TMainSimForm.SimSink1MUDestroy(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
begin
  inherited;
//        inc(mudestroyed);
//        label4.caption := inttostr(mudestroyed);
end;

procedure TMainSimForm.SimSource1MUExit(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
    PacketSize: Integer;
    currenttime: ttimestamp;
    Seq: LongInt;

begin
  inherited;
        inc(countpackets);
        randomize;
```

```
//         PacketSize := random(1600);
//         while packetsize < 500 do
//             PacketSize := random(1600);
        packetSize := 576;
        Packet.SimProperties.Items[0].SetValue(packetsize);
         Listbox4.Items.Append(inttostr(packetsize));
         Source.Append;
         SourceSimTime.Value := SimTime;
         SourceSize.Value := PAcketSize;
         SourceTTL.Value := 10;
         SourceAgg.Value := 1;
         currenttime := DateTimeToTimeStamp(time);
         SourceCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
         Source.Post;
         Seq := SourceSequenceNB.Value;
         Packet.SimProperties.Items[2].SetValue(Seq);
end;

procedure TMainSimForm.SimStore1MUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
        packetsize: integer;
        i: integer;
        counter: Integer;
begin
  inherited;
  if simstore1.nummu = simstore1.capacity-1 then
  begin
      listbox5.Clear;
    for i := 0 to simstore1.nummu do
        begin
          simstore1.Content(i).SimProperties.Items[0].GetValue(packetsize);
          Listbox5.Items.Append(inttostr(packetsize));
        end;

    end;
  end;

function TMainSimForm.SimFlowControl1FlowMethod(Sender: TObject): Integer;
var
PathMtu, PacketSize : Integer;
begin

  inherited;

    simflowcontrol1.predecessor.MU.SimProperties.Items[0].GetValue(PacketSize);
     if packetSize < 1600  then
        result :=  1
     else if PacketSize <2000 then
        result :=  0
     else if packetsize < 2500  then
       result := 3
     else if packetsize < 3500 then
        result := 2;
end;

function TMainSimForm.WhichRouterisFree: integer;
```

```
var
  randi: integer;
begin
        randomize;
        randi := random(4);
        if (randi = 0) and (simsingleprocess3.NumMu = 0) then
            result := 0
        else if (randi = 1) and (simsingleprocess2.NumMU = 0) then
           result := 1
        else if (randi = 2) and (simsingleprocess4.NumMU = 0) then
           result := 2
        else if (randi = 3) and (simsingleprocess5.NumMU = 0) then
           result := 3
        else result := 4;

end;

procedure TMainSimForm.recycleMUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
        size: integer;
begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox1.Items.Append(inttostr(size));
end;

procedure TMainSimForm.SimSink1MUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
 var
        size: integer;
begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox2.Items.Append(inttostr(size));
end;


procedure TMainSimForm.recycleMUDestroy(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
begin
  inherited;
        inc(mudestroyed);
        label4.caption := inttostr(mudestroyed);

end;

procedure TMainSimForm.droppedMUDestroy(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
begin
  inherited;
//        inc(mudestroyed);
//        label4.caption := inttostr(mudestroyed);

end;

procedure TMainSimForm.droppedMUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
   size: integer;
begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox6.Items.Append(inttostr(size));
end;

procedure TMainSimForm.SimSink2MUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
        size: integer;
begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox2.Items.Append(inttostr(size));

end;

procedure TMainSimForm.SimSink3MUEntrance(Sender: TObject; SimTime: Int64;
  MU: TMovableUnit);
  var
  size: integer;
begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox2.Items.Append(inttostr(size));

end;

procedure TMainSimForm.SimMultiProcess1MUEntrance(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
  a:array[1..5] of integer;
  i,j: integer;
  Packet, PMTU: integer;
  TTLValue: byte;
  InnerPackets,Seqs: string;
  currenttime: ttimestamp;
begin
  inherited;
  MU.SimProperties.Items[0].GetValue(Packet);
  SimMultiProcess1.SimProperties.items[0].GetValue(PMTU);

//  MU.SimProperties.Items[4].GetValue(InnerPackets);
//  MU.SimProperties.Items[5].GetValue(Seqs);
    endtunnel.Append;
    endtunnelsize.Value := Packet;
    endtunnelSimTime.Value := SimTime;
//    endtunnelSeqNbs.Value := Seqs;
//    endtunnelPacketsAggregated.value := InnerPackets;
    currenttime := DateTimeToTimeStamp(time);
    EndTunnelCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
    endtunnel.post;
  if packet > PMTU then
  begin
  for i := 1 to 5 do
    a[i] := 0;

  j := Packet div PMTU;
  for i := 1 to j do
    a[i] := PMTU;
  if Packet mod PMTU <> 0 then
    a[i] := (Packet - j*PMTU);

  i := 1;
  while a[i] <> 0 do
  begin
      fragment.SimProperties.Items[0].SetValue(a[i]);
      fragment.CreateUnit.Move(simsink1);
      listbox7.Items.add(inttostr(a[i]));
      inc(i);
      MU.SimProperties.Items[1].GetValue(TTLValue);
      dec(TTLValue);
      Destination.Append;
      DestinationSimTime.Value := SimTime;
      DestinationSize.Value := a[i];
      DestinationAgg.Value := 3;
      DestinationTTL.Value := TTLValue;
      currenttime := DateTimeToTimeStamp(time);
      DestinationCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
      Destination.Post;
   end;
end
else begin
               MU.SimProperties.Items[1].GetValue(TTLValue);
               dec(TTLValue);
               Destination.Append;
               DestinationSimTime.Value := SimTime;
               DestinationSize.Value := Packet;
               DestinationAgg.Value := 2;
               DestinationTTL.Value := TTLValue;
               currenttime := DateTimeToTimeStamp(time);
               DestinationCurrentTime.Value :=
floattostr(timestamptomsecs(currenttime));
               Destination.Post;

end;

end;

procedure TMainSimForm.SimMultiProcess2MUEntrance(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
  a:array[1..5] of integer;
  i,j: integer;
  Packet, PMTU: integer;
    TTLValue: byte;
    InnerPackets, Seqs: string;
      currenttime: ttimestamp;
begin
```

```
  inherited;
  MU.SimProperties.Items[0].GetValue(Packet);
   SimMultiProcess2.SimProperties.items[0].GetValue(PMTU);

//    MU.SimProperties.Items[4].GetValue(InnerPackets);
//  MU.SimProperties.Items[5].GetValue(Seqs);
    endtunnel.Append;
    endtunnelsize.Value := Packet;
//    endtunnelPacketsAggregated.value := InnerPackets;
//        endtunnelSeqNbs.value := Seqs;
            endtunnelSimTime.Value := SimTime;
    currenttime := DateTimeToTimeStamp(time);
    EndTunnelCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
    endtunnel.post;

  if packet > PMTU then
  begin
  for i := 1 to 5 do
    a[i] := 0;

  j := Packet div PMTU;
  for i := 1 to j do
    a[i] := PMTU;
  if Packet mod PMTU <> 0 then
    a[i] := (Packet - j*PMTU);

  i := 1;
  while a[i] <> 0 do
  begin
      fragment.SimProperties.Items[0].SetValue(a[i]);
      fragment.CreateUnit.Move(simsink2);
      listbox7.Items.add(inttostr(a[i]));
      inc(i);
      MU.SimProperties.Items[1].GetValue(TTLValue);
      dec(TTLValue);
      Destination.Append;
      DestinationSimTime.Value := SimTime;
      DestinationSize.Value := a[i];
      DestinationAgg.Value := 3;
      DestinationTTL.Value := TTLValue;
      currenttime := DateTimeToTimeStamp(time);
      DestinationCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
      Destination.Post;

   end;
end
else begin
      MU.SimProperties.Items[1].GetValue(TTLValue);
      dec(TTLValue);
      Destination.Append;
      DestinationSimTime.Value := SimTime;
      DestinationSize.Value := Packet;
      DestinationAgg.Value := 3;
      DestinationTTL.Value := TTLValue;
      currenttime := DateTimeToTimeStamp(time);
      DestinationCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
      Destination.Post;
```

```
end;
end;

procedure TMainSimForm.SimMultiProcess3MUEntrance(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
  a:array[1..5] of integer;
  i,j: integer;
  Packet, PMTU: integer;
  TTLValue: byte;
  InnerPackets, Seqs: String;
    currenttime: ttimestamp;
begin
  inherited;
  MU.SimProperties.Items[0].GetValue(Packet);
  SimMultiProcess3.SimProperties.items[0].GetValue(PMTU);

//     MU.SimProperties.Items[4].GetValue(InnerPackets);
//   MU.SimProperties.Items[5].GetValue(Seqs);
    endtunnel.Append;
    endtunnelsize.Value := Packet;
        endtunnelSimTime.Value := SimTime;
//     endtunnelPacketsAggregated.value := InnerPackets;
//         endtunnelSeqNbs.value := Seqs;
    currenttime := DateTimeToTimeStamp(time);
    EndTunnelCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
    endtunnel.post;


  if packet > PMTU then
  begin
  for i := 1 to 5 do
    a[i] := 0;

  j := Packet div PMTU;
  for i := 1 to j do
    a[i] := PMTU;
  if Packet mod PMTU <> 0 then
    a[i] := (Packet - j*PMTU);

  i := 1;
  while a[i] <> 0 do
  begin
      fragment.SimProperties.Items[0].SetValue(a[i]);
      fragment.CreateUnit.Move(simsink3);
      listbox7.Items.add(inttostr(a[i]));
      inc(i);
      MU.SimProperties.Items[1].GetValue(TTLValue);
      dec(TTLValue);
      Destination.Append;
      DestinationSimTime.Value := SimTime;
      DestinationSize.Value := a[i];
      DestinationAgg.Value := 3;
      DestinationTTL.Value := TTLValue;
      currenttime := DateTimeToTimeStamp(time);
      DestinationCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
      Destination.Post;
```

```
   end;
end
else begin
      MU.SimProperties.Items[1].GetValue(TTLValue);
      dec(TTLValue);
      Destination.Append;
      DestinationSimTime.Value := SimTime;
      DestinationSize.Value := Packet;
      DestinationAgg.Value := 3;
      DestinationTTL.Value := TTLValue;
      currenttime := DateTimeToTimeStamp(time);
      DestinationCurrentTime.Value := floattostr(timestamptomsecs(currenttime));
      Destination.Post;

end;
end;

procedure TMainSimForm.SimStore1Blocked(Sender: TObject; SimTime: Int64);
begin
  inherited;
   inc(blockedmu);
   label7.caption := inttostr(blockedmu);
end;


procedure TMainSimForm.SimSingleProcess1MUEntrance(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
  var
        sizevalue,TTLValue: integer;
begin
  inherited;
        mu.SimProperties.items[1].GetValue(TTLValue);
        dec(TTLValue);
        mu.SimProperties.items[1].SetValue(TTLValue);
        if TTLValue <= 0 then
        begin
          MU.Move(recycle);
          mu.SimProperties.items[0].GetValue(sizeValue);
         listbox8.Items.Append(inttostr(sizevalue));
        end;

end;

procedure TMainSimForm.SimGenerator1Generate_Dur(Sender: TObject;
  SimTime: Int64);
var
  stockobj: TSimStore;
  obj: TMovableUnit;
  counter,contentind: integer;
  TTLValue, sizeValue: integer;
  DropAgg:byte;
begin
  inherited;
    stockobj := SimStore1;
   if (stockobj <> nil) then
```

```
    begin
         obj := nil;
          with stockobj do
            for contentind := 0 to NumMU do
            begin
              obj := content(contentind);
              if obj <> nil then
                break;
            end;
          if obj <> nil then
          begin
            obj.SimProperties.items[1].GetValue(TTLValue);
            dec(TTLValue);
            obj.SimProperties.items[1].SetValue(TTLValue);
            if TTLValue <= 0 then
            begin
              obj.Move(dropped);
              obj.SimProperties.items[0].GetValue(SizeValue);
              listbox8.Items.Append(inttostr(SizeValue));
              TTLDRop.Append;
              TTLDropSimTime.Value := SimTime;
              TTLDropSize.Value := SizeValue;
              obj.SimProperties.items[3].GetValue(DropAgg);
              TTLDropAgg.Value := DropAgg;
              TTLDrop.Post;
            end;
          end;
    end;

end;

function TMainSimForm.SimFlowControl3FlowMethod(Sender: TObject): Integer;
var
  stockobj: TSimStore;
begin
  inherited;
        stockobj := simstore1;
        if stockobj.nummu >= stockobj.capacity then
          Result := 1
        else Result := 0;
end;

procedure TMainSimForm.CongestionMUEntrance(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
size: integer;
  begin
  inherited;
        mu.SimProperties.Items[0].GetValue(size);
        listbox6.Items.Append(inttostr(size));
end;

procedure TMainSimForm.SimSingleProcess3MUExit(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
        Size, PMTU: integer;
        BW: longint;
```

```
begin
  inherited;
        MU.SimProperties.Items[0].GetValue(Size);
        SimSingleProcess3.SimProperties.Items[0].GetValue(PMTU);
        SimSingleProcess3.SimProperties.Items[1].GetValue(BW);
        Routers.Append;
        RoutersSimTime.Value := SimTime;
        RoutersSize.value := Size;
        RoutersPMTU.value := PMTU;
        RoutersBW.value := BW;
        RoutersRouter.value := 1;
        Routers.Post;

end;

procedure TMainSimForm.SimSingleProcess5MUExit(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
        Size, PMTU: integer;
        BW: longint;
begin
  inherited;
        MU.SimProperties.Items[0].GetValue(Size);
        SimSingleProcess5.SimProperties.Items[0].GetValue(PMTU);
        SimSingleProcess5.SimProperties.Items[1].GetValue(BW);
        Routers.Append;
        RoutersSimTime.Value := SimTime;
        RoutersSize.value := Size;
        RoutersPMTU.value := PMTU;
        RoutersBW.value := BW;
        RoutersRouter.value := 2;
        Routers.Post;
end;

procedure TMainSimForm.SimSingleProcess4MUExit(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
        Size, PMTU: integer;
        BW: longint;
begin
  inherited;
        MU.SimProperties.Items[0].GetValue(Size);
        SimSingleProcess4.SimProperties.Items[0].GetValue(PMTU);
        SimSingleProcess4.SimProperties.Items[1].GetValue(BW);
        Routers.Append;
        RoutersSimTime.Value := SimTime;
        RoutersSize.value := Size;
        RoutersPMTU.value := PMTU;
        RoutersBW.value := BW;
        RoutersRouter.value := 3;
        Routers.Post;

end;

procedure TMainSimForm.SimSingleProcess2MUExit(Sender: TObject;
  SimTime: Int64; MU: TMovableUnit);
var
```

```
        Size, PMTU: integer;
        BW: longint;
begin
  inherited;
        MU.SimProperties.Items[0].GetValue(Size);
        SimSingleProcess2.SimProperties.Items[0].GetValue(PMTU);
        SimSingleProcess2.SimProperties.Items[1].GetValue(BW);
        Routers.Append;
        RoutersSimTime.Value := SimTime;
        RoutersSize.value := Size;
        RoutersPMTU.value := PMTU;
        RoutersBW.value := BW;
        RoutersRouter.value := 4;
        Routers.Post;

end;

end.
```