FPGA Implementation of Congestion Control Routers in High Speed Networks

Fariborz Fereydouni-Forouzandeh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the requirements
for the Degree of Master of Applied Science (Computer Engineering) at
Concordia University
Montreal, Quebec, Canada

February 2005

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Fariborz Fereydouni-Forouzandeh

Entitled: FPGA Implementation of Congestion Control Routers in High Speed Networks

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair

______________________________ Examiner

______________________________ Examiner

______________________________ Supervisor

Approved by ______________________________
Chair of Department or Graduate Program Director

____________ 20____ ______________________________
Dean of Faculty

Abstract

FPGA Implementation of Congestion Control Routers in High Speed Networks

Fariborz Fereydouni-Forouzandeh

Receiving large number of data packets at different baud rates and different sizes at gateways in high-speed network routers may lead to a congestion problem and force gate routers to drop some packets. Several algorithms have been developed to control this problem. Random Early Detection (RED) algorithm is commonly used to eliminate this problem. It has been recommend by IETF (Internet Engineering Task Force) for next generation Internet gateways. In this thesis we present an FPGA implementation of a modified version of the RED algorithm. Furthermore, we discuss three enhancements of the RED algorithm leading to a better performance suitable for FPGA implementation. We have conducted several simulations to show that our proposed algorithm improves the response time and reduces the risk of global synchronization in gateways. The implementation is fully FPGA compatible and is targeting Xilinx Virtex-II Pro family devices. Finally we present the implementation that can run as fast as 10 Gbps.

Acknowledgments

I would like to thank my dear supervisor, Dr. Otmane Ait Mohamed with utmost respect for his knowledgeable supports concerning my thesis and his unending patience, kindness and willingness at any time was concerning my weaknesses. This has been a wonderful learning experience that has helped me to continue my studies. I am thoroughly indebted to you. I am thankful to Dr. Sadegh Jahanpour as well as Dr. Yassine Mokhtari who have given me good feedbacks on my research. I thank Donglin, Asif and our other HVG members at Concordia University who have contributed to my knowledge and success.

To my dear wife Zohreh and my little daughter Parissa, thank you for your unconditional love, patience, support and belief in my ability to go the distance. To my brother Faramarz for all his kindly helpful supports, and to the rest of my extended family, thank you all for your love and all your individual supports.

I always knew that school is fun.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	Application Specific Integrated Circuit
ATDM	Asynchronous Time Division Multiplexing
ATM	Asynchronous Transfer Mode
CA	Cellular Automaton
CBR	Constant Bit Rate
DLL	Delayed Locked Loop
EPD	Early Packet Discard
FDE	Full Duplex Ethernet
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
Gbps	Giga bit per second
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP Core	Intellectual Property Core
IP/Packet	Internet Protocol packet
LFSR	Linear Feedback Shift Register
LPF	Low Pass Filter
LUT	Look Up Table
ODA	Over Drop Avoidance

PPD	Partial Packet Discard)
RED	Random Early Detection
RFC	Request For Comments
RTL	Register Transfer Level
SFD	Start Frame Delimiter
SODA	Stepped Over Drop Avoidance
SoC	System on Chip
TCP/IP	Transmission Control Protocol / Internet Protocol
TDM	Time Division Multiplexing
TTM	Time To Market
VBR	Variable Bit Rate
VC	Virtual Channel
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter – 1

Introduction

The impact of electronic communications, internet and networking applications is increasing globally and very broadly. It sounds to be out of control, if the appropriate enhancements do not cover its requirements. Especially in terms of speed getting higher and higher, and the number of users becoming larger and larger. It certainly creates huge data traffic at the gateways or routers, where a great number of sources send their packets to be routed to the appropriate destination. Hence, controlling this huge data traffic becomes heavily significant on traffic manager which are responsible in controlling the congestion at the gateways.

In this work we investigate various mechanisms used to control the congestion at the high speed gateways, and also we describe a hardware implementation of traffic management scheme at high speed and high performance networks. Our implementation targets a Field Programmable Gate Arrays (FPGAs).

While the technology of a multi-million gate in new devices is advancing very fast, both FPGAs and Application Specific Integrated Circuits (ASICs) are competitively demonstrating their capabilities in very large and high speed applications. Consequently, choosing the right technology to implement a given design is becoming the key question for several applications. In one hand, with ASICs one can implement multi-million gates in a small are of silicon using a library of reusable hardware and software blocks as Intellectual property (IP) cores [1]. In the other hand, FPGAs have also satisfied wonderfully the requirements for fulfillment of large complex designs in their today's multi-million gate ranges. Large variety of high performance IP cores (microprocessors, microcontrollers, intellectual functional logics and etc…) as well as high speed memories and much more are accessible in today's FPGAs [4]. These features facilitate the implementation of a large complex system designs in FPGAs. Since the FPGAs are reprogrammable, this indeed lowers the cost of any required changes or modifications in the design for future, and considering the importance of shorter Time To Market (TTM) in industry, it is of a great benefit.

1.1 – Motivations

The motivation of this work is to explore the issues involved in congestion control routers as existing traffic manager mechanisms in high speed networks, and also implementing a higher performance traffic manager using FPGAs. First we compare different congestion control mechanism in high speed gateways in terms of efficiency and throughput. After this analysis we have chosen the Random Early Detection (RED) mechanism which was first introduced by Floyd and Jacobson [6]. And secondly, we discuss the challenges in implementing such mechanisms using FPGAs. Since, the achieving high speed is the biggest challenges in FPGAs high speed implementations. Our objective is to achieve a speed of 10 Gbps. This is done by improving the RED algorithm significantly to be suitable for such speed in FPGA.

1.3 – Contributions

We can summarize our contributions in this thesis as:

1. The modification of the RED algorithm in order to achieve higher performances, and to facilitate its implementation. We prove that our new modified RED algorithm is more efficient with higher throughput and improves the response time when congestion appears at the gateway. Also, we will prove that our algorithm is more reliable in terms of packets drops than the original RED algorithm.
2. In order to verify and simulate the main features of our implementation we have designed and implemented an environment inside the same FPGA device. Generating artificial random patterns is always hard to achieve. We have produced a broad random vector set by mixing of two standard mechanisms to generate patterns of random numbers. Our 14-bit random pattern generator is required for calculations in RED algorithm.And our random pattern generator has been used as well in our environment to generate continuously pseudo random packets as emulating a real gateway to simulate the design.
3. Our high speed FPGA implementation deals with traffic up to 10 Gbps. Also, we introduced some heuristics in the hardware design to optimize some components especially in terms of speed of operations.

1.4 – Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 discusses the concept of the congestion and how it appears at the gateways. Chapter 3 describes the existing congestion avoidance mechanisms and compares their advantages and disadvantages. In chapter 4 we describe the chosen algorithm (RED algorithm). We analyze the algorithm and discuss certain ideas that lead to an efficient implementation. In chapter 5 we present our proposed algorithm and its implementation in FPGAs. Finally, Chapter 6 concludes the thesis.

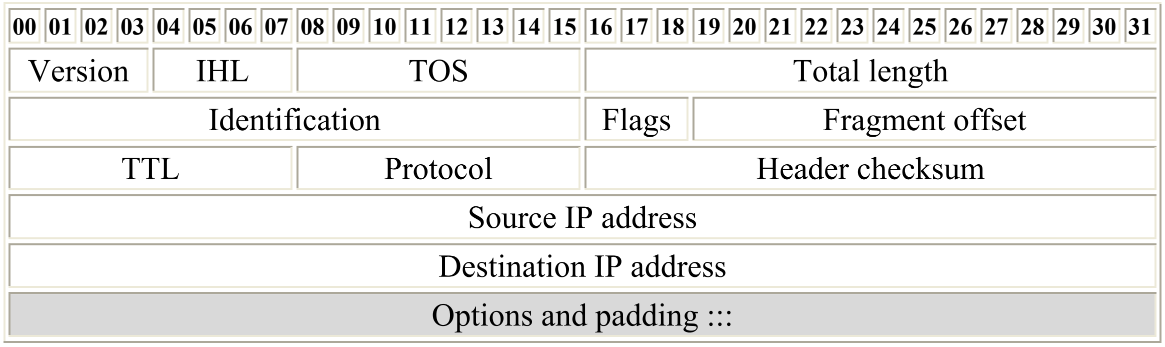
Chapter Two

Data Congestion in high speed Networks

2.1 – Effective factors causing Data Congestion in high speed Networks

High speed data transfer between several FTP (File Transfer Protocol) sources through a gateway with normal bandwidth implies unavoidable traffic congestions. Especially, increasing the number of high speed network stations in the world has brought on many problems in controlling the traffic between these stations. Data congestion always occurs in high speed network gateways with large bandwidth.

This thesis is discussing the IP Packets (Internet Protocol Packets) arriving at high speed gateways from several FTP sources which their protocol is TCP/IP (Transmission Control Protocol / Internet Protocol). The Internet protocol defines how information is transferred between systems and gateways across the Internet. The packets are decomposed to several parts that can be transferred using standard Internet Protocol (IP) over the internet. The data to be sent is partitioned into several IP packets and is reassembled on the receiving computer. Each packet starts with a header containing its specifications, e.g. address and system control information [11], [3]. Unlike uniform Asynchronous Transfer Mode (ATM) that breaks the packets into smaller standard sizes of 53-Byte cells [3], IP packets vary in length depending on the data that is being transmitted. IP header format of the IP Packet is shown in Table 2.1. The source and destination IP addresses are numbers that identify computers on the Internet.



<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="8">TOS</td><td colspan="16">Total length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment offset</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">Protocol</td><td colspan="16">Header checksum</td></tr>
<tr><td colspan="32">Source IP address</td></tr>
<tr><td colspan="32">Destination IP address</td></tr>
<tr><td colspan="32">Options and padding :::</td></tr>
</table>

Table 2.1 – IP Header format.

These numbers are usually shown in groups separated by periods, for instance: 123.123.23.2 consists of four separated values each between 0 and 255, i.e., each position in this address range could carry one Byte of address. All resources on the Internet must have an IP address or else they do not belong to the Internet.

FTP sources that send IP Packets to the gateway may vary in bit rates. If a great number of sources are active and sending IP Packets to the gateway with high bit rates, then there will be a bursty traffic of arriving packets and consequently a serious congestion will happen at the gateway. The congestion control is somehow based on the types of traffic sources at the Ingress port of the gateway. All IP Packets that have successfully passed through the gateway will be sent to the Egress port which represents the output port of the gateway. Considering the present internet system, there is no specific management between the gateways where traffic flow is controlled. Every IP Packet is processed as soon as a processor is available to send it between the gateways and a transmission port is free and ready to carry the packet. If the packets have to wait before processing, they are held in a queue, subject to the size limit and the restrictions on the way they are queued and enqueued in and from the buffer [17]. Once the size of the queue reaches a predefined limit according to the congestion control mechanism, then usually the next arriving packet will be discarded. This is discussed in details in chapter three where we review several congestion avoidance mechanisms.

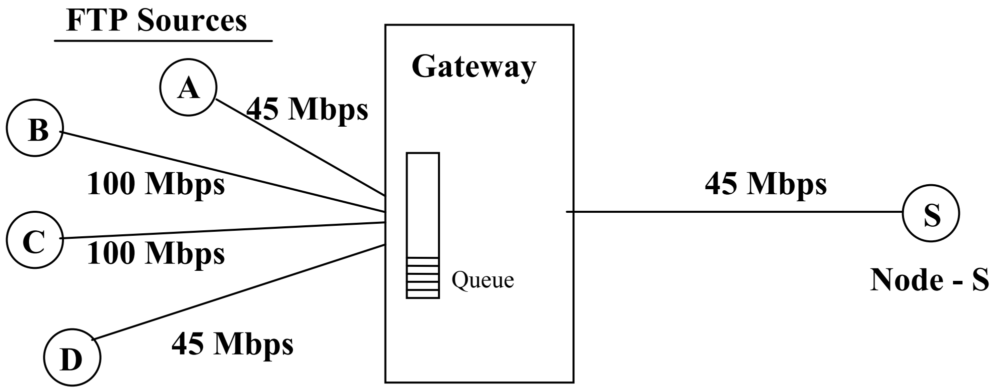


Figure 2.1 – Example of some FTP connections in a Network illustrating how congestion could appear at the gateway.

A simple example in figure 2.1 shows a gateway where several FTP connections are shown with different data transfer rates and burstiness. It is not always possible to transfer every arriving packet through such gateways, because the bandwidth of the node S is not capable to handle overall bandwidth of nodes A to D. In a glance, that could be easily a reason to encounter data congestion for such a gateway.

Gateway in Figure 2.1 looks like a firehouse connected to a straw through a small funnel as shown in figure 2.2 [2]. If the volume of the input flow is greater than the flow that can leave through the output of the straw, then funnel fills up and causes overflow.

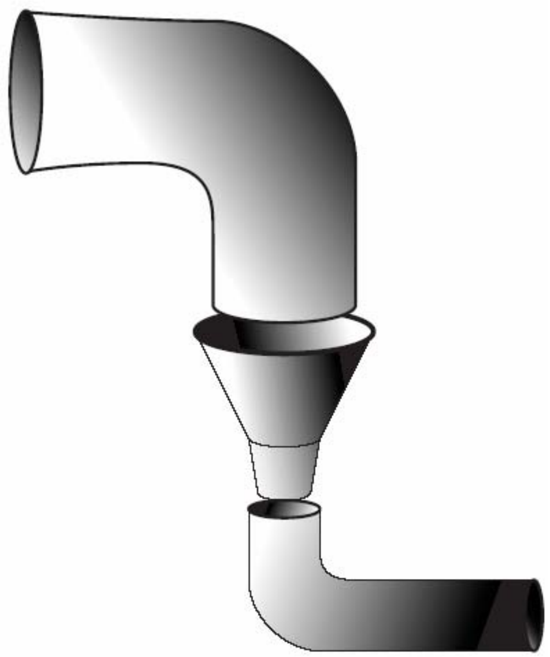


Figure 2.2 – Illustrating the Similarity of the congestion appearance in a gateway.

Consider data transfer in an internet protocol for example in Figure 2.1 that consists of four different sources A to D. The packets traverse between the nodes through the Gateway asynchronously, for instance using Time Division Multiplexing (TDM) algorithm. Asynchronous Time Division Multiplexing (ATDM) is better illustrated in figure 2.3. Packets with equal sizes may arrive from each node at any time sequentially. Arriving packets are enqueued in the buffer (queue) with respect to the size of the buffer. And they are dequeued and transmitted through the Gateway using First In First Out (FIFO) scheduler to distribute the unique delay among all packets especially during the period of burstiness. As shown in Figure 2.3 all arriving packets from nodes A, B, C, and D are supposed to be queued and sent in appropriate order like the one shown in line ATDM as asynchronous multiplexing in order (that's asynchronous because no packet is supposed to arrive at a certain expected time to be multiplexed from any node). Applications like ATDM could be observed in very messy transfer protocols to distribute arriving data from several high speed VCs (Virtual Channels) efficiently.

So, there is no fixed time relation between the multiplexed form of arrival cells or packets [3]. Considering the congestion appeared at the gateway as the result of arriving packets, it is not possible to accommodate all of them in the queue. That's why the data congestion forces the gateway to drop some packets. The way to make the decision to drop certain packets is based on the traffic manager algorithm and the congestion control mechanism implemented at the gateway.

Therefore the congestion is detected when the gateway starts to drop packets. The major factors which have main role in generating the congestion are increasing the number of high speed networks with large bandwidth and the

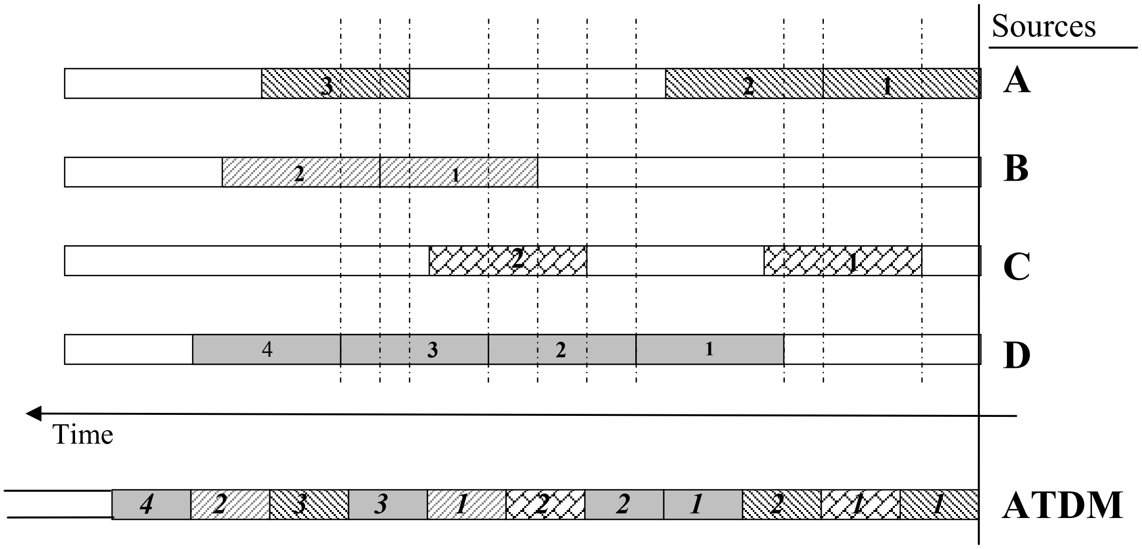


Figure 2.3 – Example of queuing the arrived packets from four different FTP in buffer as asynchronous time division multiplexing (ATDM).

bottleneck of the gateways. Such problems motivated the researchers to propose some mechanisms as the solution. Some of the more important mechanisms between them are described in next chapter.

Chapter Three

Congestion Avoidance Mechanisms

3.1 – Introduction to Congestion Avoidance

In order to avoid data congestion problem at high speed gateways, there are several mechanisms which have been proposed. Some of them are used extensively in traffic control in high speed gateways.

These mechanisms drop packets at the gateways when the packet queue reaches certain threshold. Size of the arrival packets in a fixed standard range [14] and also the baud rate, are discussed in next chapters. Since there is a queue as a buffer in every mechanism, so, while the arriving packets are being enqueued, meanwhile some packets could be dequeued through egress port. Therefore depending on the ratio of ingress and egress baud rate, size of the queued packets in the buffer may exceed some certain threshold value and consequently cause to drop some arriving packets. These threshold levels vary in different mechanisms, like queue overflow or a value less than overflow. We will see some examples in this chapter.

All of this congestion control process is to achieve a feasible higher throughput and lower average queue size. In the subsequent chapters regarding the implementation of the congestion control routers, we will discuss the impact of the queue size in regard with the congestion control at the gateways in high speed networks.

There are several standard queuing algorithms to control the congestion which determine the followings [10]:

- How packets are buffered

- Which packets get transmitted
- Which packets get marked or dropped
- Indirectly determine the delay at the router

3.2 –Congestion Avoidance Mechanisms

This section introduces and reviews briefly some existing congestion control mechanisms by describing their advantages if any. Finally, we present the algorithm that we choose to design and to implement. As we'll see all existing congestion control mechanisms drop some packets arriving at the gateway if the size of the stored packets in the buffer (queue) reaches a certain value. Some algorithms fill up the queue completely or make it overflow and then start dropping, and some verify on a threshold level. All of them choose an appropriate packet to drop but each of them do it in their own different way. Generally there are two major factors concerning these traffic managers;

- When decide to drop the packets in respect to the instantaneous size of the queue
- Which packets are chosen to drop in respect to the stored packets in the queue

Below is a list of very commonly used congestion avoidance mechanisms in high speed gateways

1 – Drop Tail
2 – Random Drop
3 – ERD (Early Random Drop)
4 – PPD (Partial Packet Discard)
5 – EPD (Early Packet Discard)
6 – IP Source Quench
7 – DEC bit
8 – RED (Random Early Detection)
… and more . . .

In the following section we briefly review and compare these mechanisms.

3.2.1 – Drop Tail

This is one of the simplest mechanisms with fewer throughputs. It is a FIFO queuing mechanism that starts to drop the packets from the tail of the queue once the queue is full [6]. It means, after recognizing the condition to drop a packet is satisfied, the last arrived packet at the gateway will be dropped.

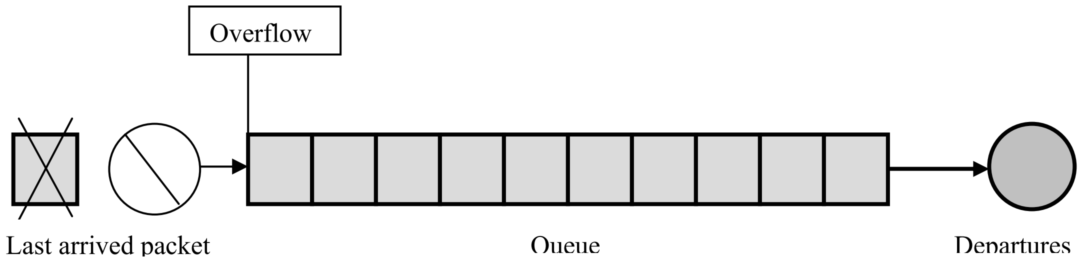


Figure 3.1 –Drop Tail overflowing and drop order.

A major problem of Drop Tail is global synchronization. This is because dropping packets from several VCs (Virtual Channels) forces these sources to resend them again later, and then consequently resending them which may cause congestion again at the gateway and that's why the global synchronization appears by such mechanisms. Figure 3.1 simply illustrates proceeding of the Tail drop congestion control gateways.

3.2.2 – Random Drop

Another congestion control called Random Drop, which gives feedback to the sources by dropping packets at the gateway, based on statistical situation of the gateway.

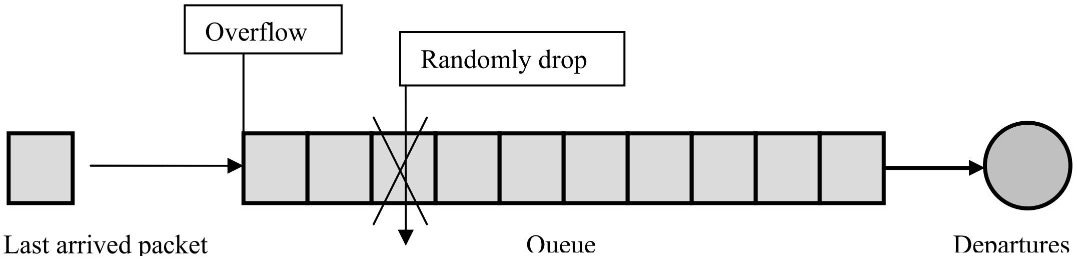


Figure 3.2 – Random Drop, overflowing and drop order.

Unlike the Drop Tail mechanism, instead of dropping the last arrived packet, in Random Drop a packet is selected randomly from all incoming sources arrived at the gateway to be dropped. Consequently such packets belong to those particular users with a probability proportional to the average rate of data transmission [18]. Therefore, dropping packets occurs on such users whose traffic generation is much more than those generating less traffic. In other word, the users who generate less amount of traffic, experience smaller amount of packet loss. As mentioned in the reference [17], Random Drop which was originally proposed by Van Jacobson, did not improve the congestion recovery behavior of the gateways. And the performance was surprisingly worse than the other corresponding mechanisms in a single gateway bottleneck. Figure 3.2 simply illustrates the behavior of the Random Drop gateway.

3.2.3 – ERD (Early Random Drop)

This mechanism of congestion avoidance has been first investigated briefly by Hashem in [5]. In this mechanism the packets are dropped at the gateway with a fixed drop probability once the size of the queue exceeds a certain threshold level. Many active researchers in this regard believe that both drop level and drop probability in ERD congestion avoidance should be adjusted dynamically according to the network traffic of the gateway [6].

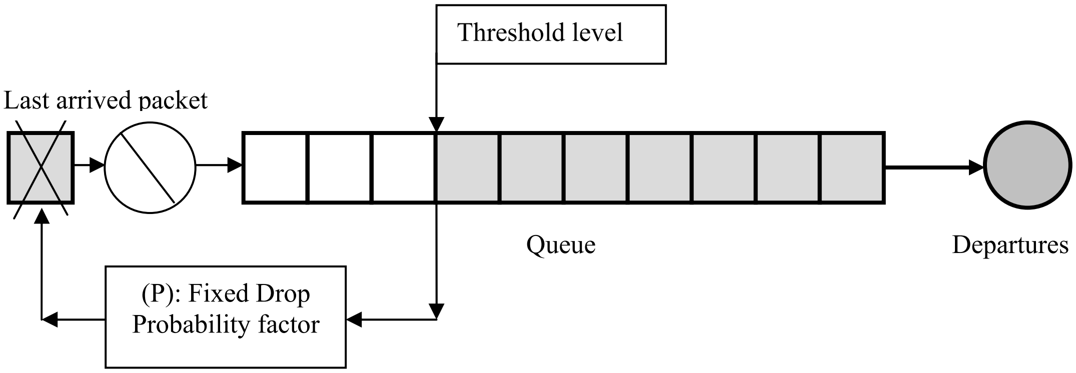


Figure 3.3 – Early Random Drop, Threshold level and fixed drop probability (P) and drop order.

Hashem [5] found out with Drop Tail gateways that since packets are dropped on queue overflow, it results decreasing the windows of these connections at the same time. And consequently results a loss of throughput at the gateway. Therefore, Early Random Drop gateways have better chance and are more recommended versus Drop Tail because of their broader view of traffic distribution. However it suffers from some disadvantages. For example, it has not been well successful in controlling the misbehaving of the users in the way they send data. Figure 3.3 simply illustrates the proceeding of Early Random Drop gateway.

3.2.4 – PPD (Partial Packet Discard) and EPD (Early Packet Discard)

These congestion avoidance mechanisms are related to Cell-discard in coexisting of ATM (Asynchronous Transfer Mode) and TCP/IP. ATM interoperates with TCP/IP because of its popularity in a great area of applications. Such legacy applications operating under ATM that must support the TCP/IP protocol, are based upon TCP/IP/ATM platform [7]. ATM drives the packets after they are decomposed into small fixed size segments named CELL. Normally the size of a Cell running over the ATM is 53 Bytes which is consisted of a 5-Byte header and a 48-Byte of data [3]. Then, if TCP/IP is run over ATM, all TCP/IP packets are segmented and decomposed to such fixed size Cells. And as stated in [7] if an ATM switch drops a cell from an arriving packet because of overflowing the buffer, the rest of the cells belonging to the same packet whose cell has been discarded will still transmit. Therefore after all these cells are arrived at the destination, the destination will fail while reassembling such packet that has lost a cell belonged to. According to the TCP mechanism if a packet is not received properly, it will be reported by a feedback and will be retransmitted. Thus if a packet is run over such TCP/IP/ATM whose one cell is lost, it will waste the whole remaining time of transmitting such useless packet resulting a loss of throughput. Sally Floyd and Romanow were those who observed this phenomenon in 1995 and then they proposed the two mechanisms named PPD (Partial Packet Discard) and EPD (Early Packet Discard) to enhance the efficiency of the TCP over ATM [9].

According to these congestion avoidance mechanisms, first in PPD (Partial Packet Discard), when a cell is dropped from a switch buffer, all cells except the last one in the arriving packet are discarded even if there is enough room to accommodate them in the buffer. The destination uses the last cell to get the information regarding the boundaries of the discarded packet. Therefore PPD could eliminate the time wasting in network concerned the damaged packet; however some parts of the damaged packet may have already been reached the destination before dropping the discarded cell.

And second about the EPD (Early Packet Discard), if the instantaneous size of the queue reaches a predefined threshold level, the next arriving cells will be discarded and the entire packet will be dropped. On the other hand, in case of overflowing the buffer while receiving the cells belonged to a packet, all subsequent cells of the same packet will be discarded as well as in the PPD described before.

A comparison between PPD and EPD has shown that when the packet length is short, the performance of the PPD is better than the EPD, and when the packet length is long, the performance of the EPD which uses a predefined and optimized threshold level is better than PPD [7].

3.2.5 – IP Source Quench

There are some different methods of congestion control mechanisms that are being used at the high speed gateways in the Internet that send some kind of a feedback to the senders reporting the congestion at the gateway. This is a congestion recovery policy. IP Source Quench is one of these methods that use such policy as described completely in [18] (RFC 1254). According to its definition, whenever a gateway responds to congestion by dropping an arrived packet, it sends a message to its source to notify the existing congestion at the gateway. This message in IP Source Quench is called ICMP (Internet Control Message Protocol). But basically the packets are not supposed to be dropped during the normal operation of the network gateway. Thereby it is very desirable to control the Sources before they overload the gateways.

A question is when to send an ICMP message. RFC 1254 says that according to the experiments based on a reasonable engineering decision, Source Quench should be applied when about half of the queue (buffer space) is filled up. However, it could be arguable to try to find another threshold, but they have not found it necessary yet.

By the way, there are some other gateway implementations generating the message not on the first packet discard, but after few packet discards. However it is not recommended by the engineers as they consider it undesirable [18].

Another question is what to do when an ICMP Source Quench is received. First, TCP or any other protocol will be informed of receiving such message. Then it demands the TCP implementations to reduce the amount of their data transmission rate toward the gateway.

3.2.6 – DECbit Gateway

DECbit is another method that uses a recovery policy by sending a feedback to the sender. But instead of the message in IP Source Quench, DECbit sends just a 1-bit feedback as a congestion indication bit which has already been allocated in the header of the packet used to inform the sender of existing congestion at the gateway. The congestion indication bit will be enabled whenever the average queue length reaches normally 1 or greater than 1 after every arriving packet, and average queue length is calculated during last “busy +idle” period plus the current “busy” period. (The busy period means the gateway is transmitting the packets and the idle means no transmission is in process) [6].

In order to control the congestion in DECbit, if the indication feedback bit is enabled in at least 50% of the packets, then it means notifying the congestion. And if it decreases the sending window by 87.5% otherwise it is increased linearly by one packet [6], [8].

As seen in this congestion control mechanism, the destination has to echo the congestion indication bit to the source, and it means that there must be the constraint of existing such special bit in the header of every arriving packet, and this could be a disadvantage versus those mechanisms not requiring such constraint.

3.2.7 – RED (Random Early Detection)

Random Early Detection (RED) is another congestion control mechanism proposed by Sally Floyd and Van Jacobson in early 1990s [6] that the major discussion in this work is focused on. Although it has been proposed many years ago, nevertheless because of its efficiency and considerable throughput in congestion avoidance at the gateways, it is still being used, however it has been modified several times till now.

The RED mechanism is introduced in this section briefly and we will compare it to other mechanisms mentioned earlier in this chapter, by showing its advantages over the other mechanisms. Then it will be explained in more details in next chapter, since it is the algorithm that we choose to implement.

The RED Gateways control the congestion by computing the average queue size in the networks based on packet switching. It computes the average queue size after every arriving packet at the gateway and detects the congestion and notifies the sources by dropping arrived packets. The congestion detection in RED mechanism is based on two threshold levels on average queue size. These thresholds in RED are named as “Minimum Threshold Level” and “Maximum

Threshold Level". After every arriving packet, the RED gateway computes the average queue size. Once the computed average queue size reaches the Minimum Threshold Level, then the arrived packet may be dropped based on a certain probability which depends on the average queue size [6]. If the average queue size is reached or is greater than the Maximum Threshold Level, the arrived packet will be dropped. Therefore there are three areas in computed average queue size separated by these two thresholds as shown in figure-3.4.

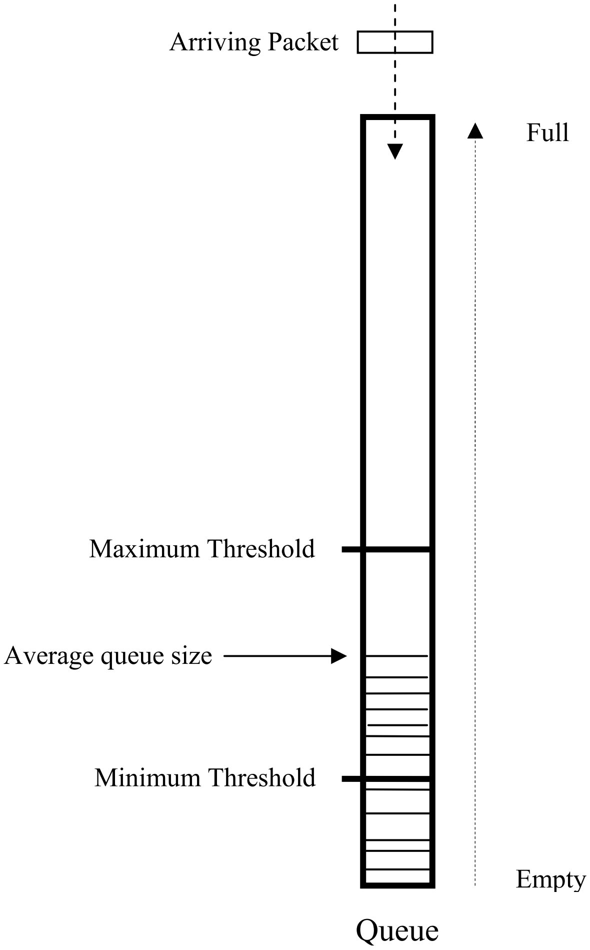


Figure 3.4 – Illustrating the RED buffering mechanism.

Figure 3.4 illustrates three different situations regarding packet drop decision. As long as the average queue size is below the minimum threshold level, no packet is dropped. If average queue size is above maximum threshold level, every arrived packet is dropped. And the major part of these areas pinpointed to, in the RED, is while the average queue size is between minimum threshold and maximum threshold levels. In this case average queue size is computed after every arriving packet and then a probability "Pb" is calculated based on average queue size and these two threshold levels, and finally it is compared to a generated random number to decide if the arrived packet should be dropped or not [6]. We will see that using the random number in drop decision in RED is an important factor in drop decision.

Next section describes how the RED mechanism is chosen as a better algorithm to implement in this work.

3.2.8 – Choosing the congestion avoidance mechanism to implement.

All congestion Avoidance mechanisms mentioned in this chapter try to prevent the congestion in high speed gateways in their own different ways. Most of them use large queue sizes to accommodate transient congestion while they can not keep it at a low level after any transient congestion.

In order to guaranty that all arriving packets could be accommodated while in transient congestion in high speed networks, it demands to provide large buffers for the queues at the gateways, but on the other hand, large queue sizes make large delay bandwidth problem which is undesirable for high speed networks. However, it is very important to have small instantaneous queue sizes to eliminate the large delay bandwidth problem at high speed gateways. But any way it is always required to guaranty the incipient or transient congestions. Thus, by increasing the high speed networks, it is strongly necessary to have such congestion mechanisms keeping high throughput with an average queue size as lower as possible.

To find a better mechanism between those mechanisms explained earlier, it is possible to compare their major advantages and/or disadvantages. For example, for the first one, as Hashem mentions in [5], Drop Tail gateways have fewer throughputs and are not successful at the gateways. And also as shown in [6], the Early Random Drop (ERD) gateways (described earlier in section 3.2.3), have a broader view of traffic distribution than the Drop Tail or Random Drop (described in section 3.2.2), because of reducing the global synchronization. Since packet dropping in Early Random Drop (ERD) happens by a fixed probability when the queue is exceeded a certain threshold, therefore ERD could have better chance to eliminate the global synchronization than the Random Drop. But even for the ERD gateways, Zhang has used this mechanism in simulations [6], [12]. In these simulations, when the queue exceeds the half of the queue size, then the gateway drops the arrived packets with the probability of 0.02. Then Zhang has shown that the ERD gateways were not successful in controlling the congestion at the gateways, because of unavoidable some misbehaving users.

Considering the PPD (Partial Packet Discard) and EPD (Early Packet Discard) mechanisms that have control on the cell portions of the arriving packets (described in section 3.2.4), each of them suffers from a lack of proper decision regarding the size of the arriving packets, because the PPD gateways suffer from lower performance on arriving larger packets and the EPD gateways from smaller packets.

And also about the IP Source Quench and DECbit mechanisms discussed in sections 3.2.5 and 3.2.6, both of them use a message to send the source as a feedback notifying the situation, regarding the existing of congestion at the gateway. Although there is a difference between them concerned to the fixed threshold level which in IP source Quench is compared to the instantaneous queue size and in DECbit, it is compare to average queue size.

IP Source quench sends the feedback to the source before the queue reaches a predefined certain level forcing the sender to decrease its window and before packets are dropped. First considerable point in this view is that both of these mechanisms are able to respond only to those sources whose arrived packets contain appropriate place or at least one bit as congestion indication bit to support the feedback message. On the other hand, the IP Source Quench does not respect to critical situations as well, such as incipient or transient congestion in bursty traffic at the gateway that always happens repeatedly in high speed networks, because it never let the queue to exceed much more than the fixed threshold level. However the DECbit computes the average queue size to compare with the threshold level and it is such an advantage over the other mechanisms that don’t do, to accommodate the transient congestion in the queue. But since it uses the last (busy + idle) period plus the current busy period to compute the average queue size, the queue size in this way could be some times averaged over a short period of time [6]. While in high speed networks with large buffers at the gateway, it is desirable to consider an appropriate portion of time constant for computing the average queue size which is done in RED gateways. However we will see later in our RED implementation, we present a new method to compute the average queue size which is faster and easier to implement as part of contribution in this thesis to improve the efficiency of the RED. And there is another difference between RED and DECbit regarding the way they use to send the feedback to the sources. In DECbit there is no relation between the way it recognizes the existing congestion at the gateway and the way it chooses the sources to send them a feed back message. Once a packet arrives at the gateway whose computed average queue size is too high, then its congestion indication bit in the header is enabled while may not be sent from one of those sources that brought the bursty traffic. This problem has been alleviated in RED by using the randomization method in drop decision. We will find enough explanation regarding the RED mechanism in next chapter. The randomized results in RED drops the packets randomly from different sources, consequently reduces the global synchronization which happens numerously in TCP/IP protocols.

Finally in this investigation on discussed congestion avoidance mechanisms, the RED mechanism is recognized as a more efficient method showing a better performance to implement. Because it avoids generating global synchronization better than the other mechanisms by accommodating the transient congestion as well and keeping control on the average queue size to decrease bandwidth delay.

Based on these results the RED gateway is the chosen mechanism to implement in this work. It is introduced more clearly in next chapter which gives us a better conceptual aspect of it.

Besides, we will see how some modifications have been proposed by other authors concerning its improvement and what kind of the challenges its implementation has.

Furthermore, the proposed contributions introduced and explained later in this thesis regarding how to improve its performance especially in terms of achieving more adequate response in high speed networks that has always been a major problem for RED high speed gateways to implement, are interpolated in the implementation as well.

Chapter Four

RED (Random Early Detection)

4.1 – Introduction

As seen in the previous chapters, one of the major problem for all the algorithms is the global synchronization at the gateways which is due to insufficient space in the buffer (queue) to accommodate the incipient congestion or transient congestion. Increasing the size of the queue will not solve the problem, because this will require a long delay for transmitting all packets in the queue.

The solution adopted by the RED algorithm consists in calculating the average queue size based on a low pass filter. The average queue size follows the instantaneous queue size very slowly using a coefficient constant value which is named as weight of the queue (w). Therefore the incipient or transient congestion could come over and easily pass. Such a situation carrying the temporary congestion at the gateway causes to increase significantly the instantaneous size of the queue, but after accommodating and traversing such burstiness of arriving packets at the gateway, it enters to a normal situation that controls and keeps the average queue size at a low level to eliminate the big delay bandwidth problem at the gateway.

On the other hand the RED mechanism drops the packets based on the average queue size and a random number while the average queue size reaches an area between two fixed thresholds level both chosen at very low levels of average queue size in order to control and keep it around this area. This is constitutes a property of a RED gateway.

The use of a Random Number by the RED algorithm distributes the drop decision between all the sources. Since the baud rate of the sources is different, the high bandwidth source will not be penalized too much.

In the following section we will present the RED algorithm in details, and we will discuss the challenges of its hardware implementation.

4.2 –RED algorithm

In the RED algorithm, the average queue size has the main role in the calculations of the drop decision. The average queue size is calculated of every arrived packet. Then the calculated average queue size is compared to two threshold levels (Minimum Threshold Level and Maximum Threshold Level), as illustrated in figure 4.1 (Min_th, Max_th).

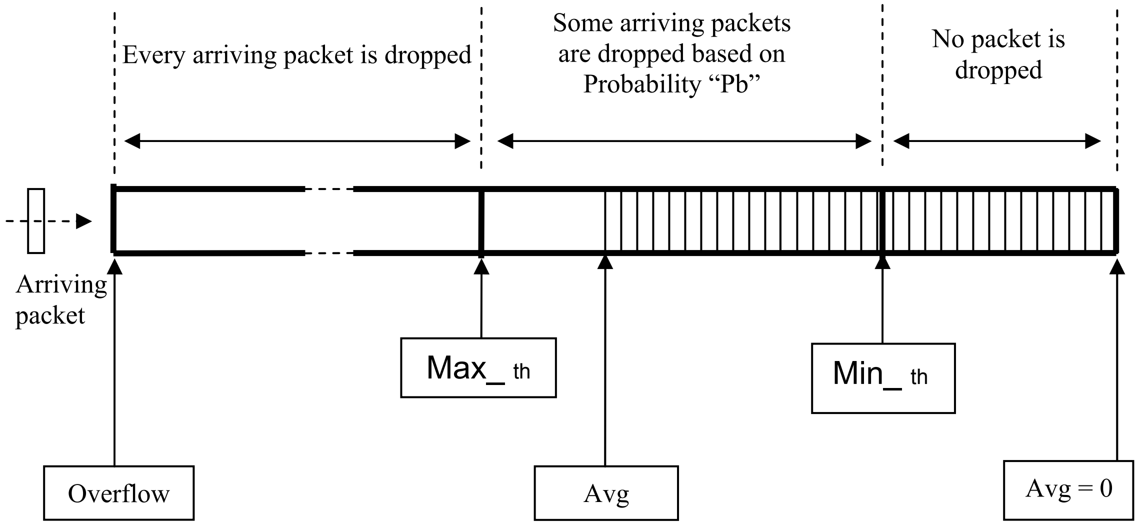

Every arriving packet is dropped
Some arriving packets are dropped based on Probability "Pb"
No packet is dropped
Arriving packet
Max_ th
Min_ th
Overflow
Avg
Avg = 0

Figure 4.1 – Illustrating detailed RED buffering mechanism.

If the average queue size (Avg) is below the minimum threshold level, no packet is dropped. If the average queue size is over the maximum threshold level, every arrived packet is dropped. And when the Avg is between the minimum threshold and maximum threshold levels which is the special case of the RED algorithm, the drop decision is based on the main calculations in the algorithm explained in section 4.3.

Let us note that goal of the RED algorithm is to keep the average queue size at a very low level in order to avoid global synchronization by dropping packets fairly to keep the average between minimum and maximum threshold levels. Now let's see how the average queue size is computed in RED.

4.2.1 – Average calculation in RED

After every arrived packet at the gateway, the new average queue size is computed through a low pass filter as shown the following formula [6].

$$\text{New\_Avg} \Leftarrow (1-w_q) \times \text{Old\_Avg} + w_q \times q \qquad (4.1)$$

It could be written and used in this form too;

$$\text{New\_Avg} \Leftarrow \text{Old\_Avg} + w_q \times (q - \text{Old\_Avg})$$

Where q is the instantaneous size of the stored packets in the buffer until the last arrived packet and the w_q is a fixed constant value which represents the weight of the queue which is the main parameter of the rate in low pass filter. According to this formula the average queue size follows the queue size very slowly if w_q is chosen a small constant value. Several different cases are discussed to find a suitable range for choosing the constant w_q. It has been measured in a range of a minimum of “0.001” and maximum “0.0042” in the case where all the packets are of equal sizes of one k-byte at the gateway [6], in a moderate baud rate, not at a high speed gateway. However it couldn’t be considered as a real situation in current high speed gateways. RED algorithm is very efficient mechanism, but it has not published complete solution in a real high speed gateway involving real arrival packets in different sizes and gap-times. In fact, it is very proportional to the size of the arrival packets and the baud rate in high speed gateways. Because both could rise up the queue size very quickly while it must be controlled before the queue overflows. Figure-4.2 shows a simulation [6] by considering the arriving packets in equal sizes of one K-Byte.

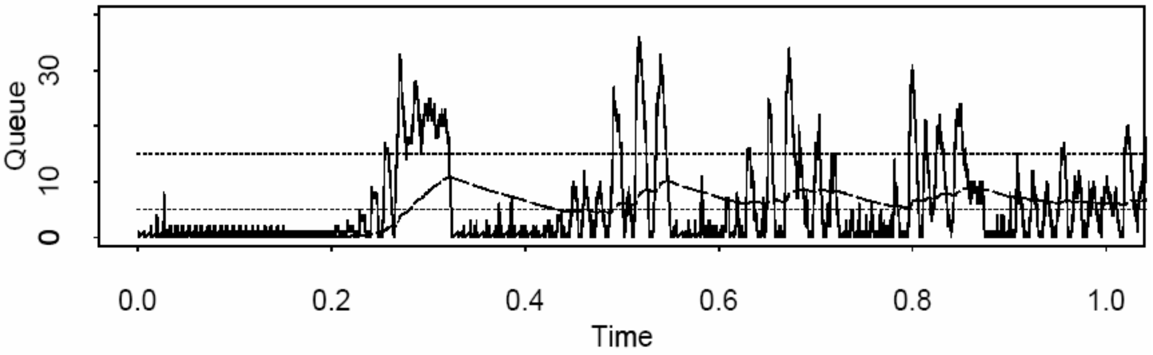


Figure 4.2 – A one second simulation of RED receiving equal packet sizes of 1 K-Byte each, the solid line is queue size and the dashed line is average queue size both in terms of number of packets.

As shown for “one second” period of time in this figure, the minimum threshold has been chosen as 5 packets and the maximum threshold as 15 packets. The authors have distinguished the empty queue to compute the new average queue size, from none empty queue. Because, when there is no packet arrival for a long time while the stored packets in the queue are being sent from egress port, this could empty the queue, but since there has been no packet arrival, therefore no new average queue size has been calculated since last packet arrival. Hence they have proposed a modification to calculate it in such cases. In case of a packet arrives at an empty queue (i.e. q=0), then the form of the equation (4.1) turns to the below formula:

$$\text{New\_Avg} \Leftarrow (1-w_q) \times \text{Old\_Avg} \qquad (4.2)$$

They modified it in [6] by calculating the new average queue size in this case through the equation (4.3).

$$\text{New\_Avg} \Leftarrow (1-w_q)^m \times \text{Old\_Avg} \qquad (4.3)$$

$$m = \text{Idle\_time} / S$$

Where “Idle_time” is the period of the time that the gateway has not been receiving any new packet and “S” is a typical required transmission time for a small packet. Obviously it is an approximation that calculates almost the number of the packets that could be sent from the gateway in this time period. Therefore their modified average queue size in equation (4.3) tries to compensate for the error in the calculation after an idle time period. Larger Idle_time gives larger m and larger m gives lower average queue size after Idle_time. In the next chapter we will present a new method to deal with this problem significantly in our implementation of the RED algorithm. This represents a contribution on the way the calculation is done since it’s more effective and more reliable.

4.2.2 – Drop decision

The average queue size is compared to the minimum and maximum threshold levels for the drop decision with respect to three regions described earlier as follows:

1- $(\text{Avg} \wedge \text{Green})$ → Keep arrived packet

2- $(\text{Avg} \wedge \text{Yellow})$ → Drop by Probability

3- $(\text{Avg} \wedge \text{Red})$ → Drop arrived packet

where:

Green = $\text{Avg} < \text{Min\_}_{\text{th}}$

Yellow = $\text{Min\_}_{\text{th}} \leq \text{Avg} < \text{Max\_}_{\text{th}}$

Red = $\text{Avg} \geq \text{Max\_}_{\text{th}}$

4.3 – Describing the drop decision in RED algorithm

RED algorithm tries to drop some arrived packets fairly based on the average queue size, number of previously accommodated packets in the queue and also randomization method to distribute the packet droppings between all sources sending packet to the gateway. And this is done by considering the following three parameters:

1 – A probability factor “P_b”, a function of the average queue size which is mathematically a distribution function of uniform distribution over (Min_ $_{th}$ and Max_ $_{th}$). Both of these levels are the boundaries for the average queue size to be controlled in.

In case of the average queue size between minimum threshold and maximum threshold levels, P_b is defined as below:

If:

$$Min\_{}_{th} \leq Avg < Max\_{}_{th}$$

Then if P_{th} is defined as:

$$P_{th} = (Avg - Min\_{}_{th}) / (Max\_{}_{th} - Min\_{}_{th})$$

Thereby:

$$P_{th} \in [0, 1]$$

Then Pb is defined as:

$$P_b = Max\_{}_{P} \times P_{th} \qquad (4.4)$$

Where:

P_{th} : Distribution function over [0, $(Max\_{}_{th} - Min\_{}_{th})$]

$Max\_{}_{p}$: Maximum value for Probability P_b

Then:

$$P_b = (Max\_{}_{p}) . (Avg - Min\_{}_{th}) / (Max\_{}_{th} - Min\_{}_{th}) \qquad (4.5)$$

2 – R which is a random number in [0, 1], is the second parameter used in RED calculation for drop decision.

$$R \in [0, 1]$$

3 – C which is third parameter represents a counter of the number of arrived and enqueued packets since last drop. It determines how many packets have been accommodated by that time. As we will see later in the algorithm, if the counter (C) is big, then it increases the probability of drop decision as expected to control the average queue size.

Now by considering the RED algorithm [6] which is given in figure-4.3 we will see how these parameters are used to determine if a packet should be dropped or not. The procedure is also shown simply through the flow chart in figure-4.4. All steps of the algorithm from line 4 to 11 in figure-4.3 are the same as explained before except for lines "2" and "3" which represent an initialization step. In line 12 the RED looks for the average queue size if it is in Yellow area ($Min\_{th} \leq Avg < Max\_{th}$). There is Counter "C" in the calculation which is initialized to "-1". C is incremented by one after every arrived packet in this area (line 13). And also C becomes zero once a packet is dropped in this area (lines 17 and 25 in figure-4.3). Thus, C represents the number of accommodated packets arrived at the gateway since last drop or first entrance to this area.

After the first packet is arrived in yellow area, "C" becomes zero. Line 15 calculates probability "Pb" for every arrived packet in this area. The condition "C > 0" in line 17 never lets to drop the first arrived packet in this area. Then, the value of C becomes greater than zero for the subsequent arrived packets. In this case, the upcoming packet will be dropped depending on the probability P_b and on the random number “R”.

```
Initialization:
    $Avg \Leftarrow 0$
    $C \Leftarrow -1$
for each packet arrival calculate the new average queue size “avg” :

if the queue is nonempty then
    $Avg \Leftarrow Avg + w .( q - Avg)$
else
    $m \Leftarrow Idle\_time / S$
    $Avg \Leftarrow Avg. (1 - w)^m$
end if
if $Min\_th \le Avg < Max\_th$ then
    increment $C$
    using new “Avg“ and “C “ calculate probability ”Pb”:
    $Pb \Leftarrow (Max\_p) . [(Avg - Min\_th) / (Max\_th - Min\_th)]$

    If $C > 0$ and $C \ge$ Approx[R/Pb] then
        Drop the arrived packet
        $C \Leftarrow 0$
    end if

    if $C=0$ then
        Random number [R] $\Leftarrow$ Random[0,1]
    end if

else if $Avg \ge Max\_th$ then
    Drop the arrived packet
    $C \Leftarrow 0$
else $C \Leftarrow -1$

end if

when queue becomes empty then
    start counting the Idle_time
end
```

Figure 4.3 – Effective RED algorithm to implement.

The second condition in line-17 is the main condition for the drop decision which is verified after every arriving packet. This is a comparison between C and the result of R divide by P_b. The random number “R” in this comparison in the RED algorithm randomizes the final results and is used to distribute the drop decisions fairly between all arrived packets from all sources. R gets a new random number, once a packet is dropped in this area (line 23). And the Pb which is calculated in equation (4.4) is directly proportional to the Avg in this area. So in the comparison in line 17 it implies that if Pb is large (Avg is large), then probability of drop high and could happen by less number of accommodated packets (stored in C). Otherwise it demands to accommodate larger number of packets (larger C).

The rest of the algorithm is when the average queue size is in Red area (Avg $\geq Max\_th$) which drops every arrived packet and resets the C to zero. And finally at the end of the algorithm, while in Green area (Avg $< Max\_th$), the RED does nothing except reinitializing the counter C to “-1”.

Not only all simulations presented in [6] by Floyd and Jacobson are based on the number of packets (instead of number of bytes), but also the arriving packets are in equal size of one K-Byte. But in the real high speed networks at the gateways it is different. A real gateway receives packets in different sizes. On the other hand, as notified in RED paper, it is optional to do all processes based on the number of bytes instead. As we will see in chapter five, the implementation of the RED algorithm in this thesis is designed and implemented to manage the real traffic by handling different packet sizes arriving at different times. This means that our implementation accommodates random time-gaps between packet arrivals.

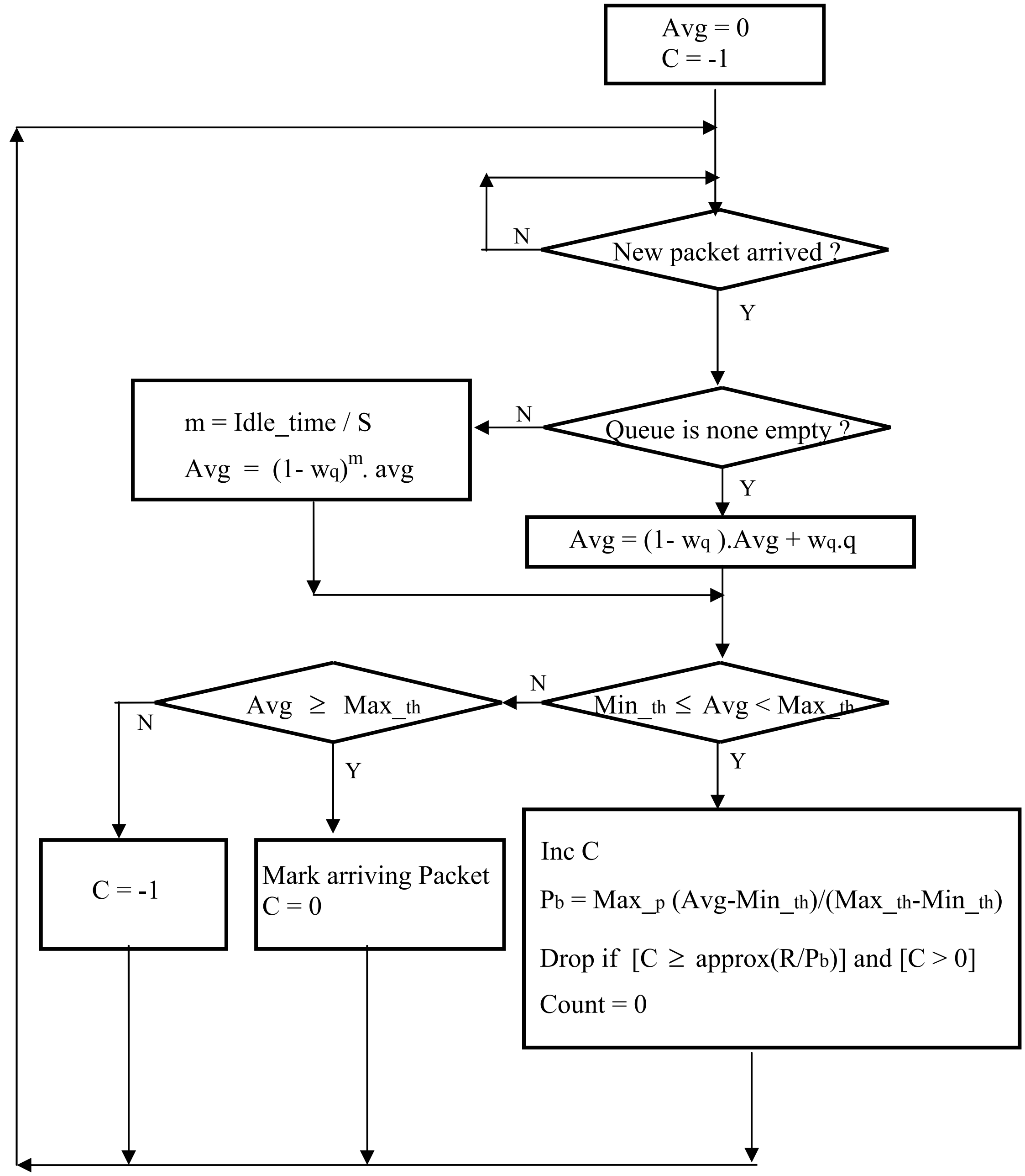


Figure 4.4 – Illustrating the flow chart of the RED mechanism.

The *Min_th and Max_th* must be chosen such that:

1. *Min_th*: must be large enough to accommodate bursty traffic
2. *Max_th*: must not lead to long average delay
3. rule of thumb: set *Max_th* at least twice *Min_th*

And for the constant "Max_{p}" in equation (4.4), it's set to "1/50". This value has been found by simulations [6]. However, it could be chosen depending on the traffic conditions and the gateway requirements. Depending on the queue size and the threshold levels and also the weight of the queue (w), an optimized value could be chosen for Max_{p} especially since the implementation of the RED algorithm is much approximated. Therefore it is required to choose it as a negative power of two [6]. Because if it is supposed to work in high speed gateways, it takes long time to calculate any complex arithmetic calculation, but if the calculation is based on powers of two, all division and multiplications could be done simply by using shift and add operations.

Finally here we can recall some benefits and principals of RED mechanism:

- Provides both congestion recovery and congestion avoidance
- Avoids global synchronization against bursty traffic
- Maintains an upper bound of average queue size

- Works with TCP and non-TCP transport-layer protocol
- Monitors the average queue size
- Uses randomization method to distribute the congestion notification
- Accommodates both transient and longer-lived congestion

In the next chapter we will present a complete description of the implement of the RED algorithm using FPGAs, and also introducing new modifications to improve its efficiency.

Chapter Five

New Algorithm
and
FPGA Implementation

5.1 – Introduction

In this chapter, we present our new modified RED Algorithm which has been implemented. These modifications are necessary in order to get an efficient hardware implementation. To ensure that our enhancement preserves the functionality of the original RED, we developed a behavioral model for both the original RED and the modified one. Then we compared those using VHDL simulations.

Our design targets the traffic speed of 10 Gbps. In other word, the goal of the algorithm is to compute the final drop decision for every arrived packet and issue the output result in the appropriate time. That means, before the actual packet is completely received, the drop decision must have been made and issued. This minimum time corresponds to the worst case of the reception of small size packet.

Thus the minimum available time for our design is the time that the smallest packet requires to be stored in the queue, starting from the edge of the packet arrival input signal. The minimum available time determines the timings of the design and is discussed in the next section. Figure 5.1 illustrates how incoming packets are processed through our traffic manager, dropped or stored in the FIFO queue and then sent to the egress port. Packets may arrive from several sources at different speeds, and then after serializing them in the interface they are forwarded to the FIFO queue and the traffic manager. The traffic manager calculates the average queue size on every change in the queue whether a new packet is arrived or a packet is sent. As we will see in our implementation, calculating the average queue size on every change in the queue in this design is an improvement upon the regular RED algorithm that calculates the average queue size only on every packet received.

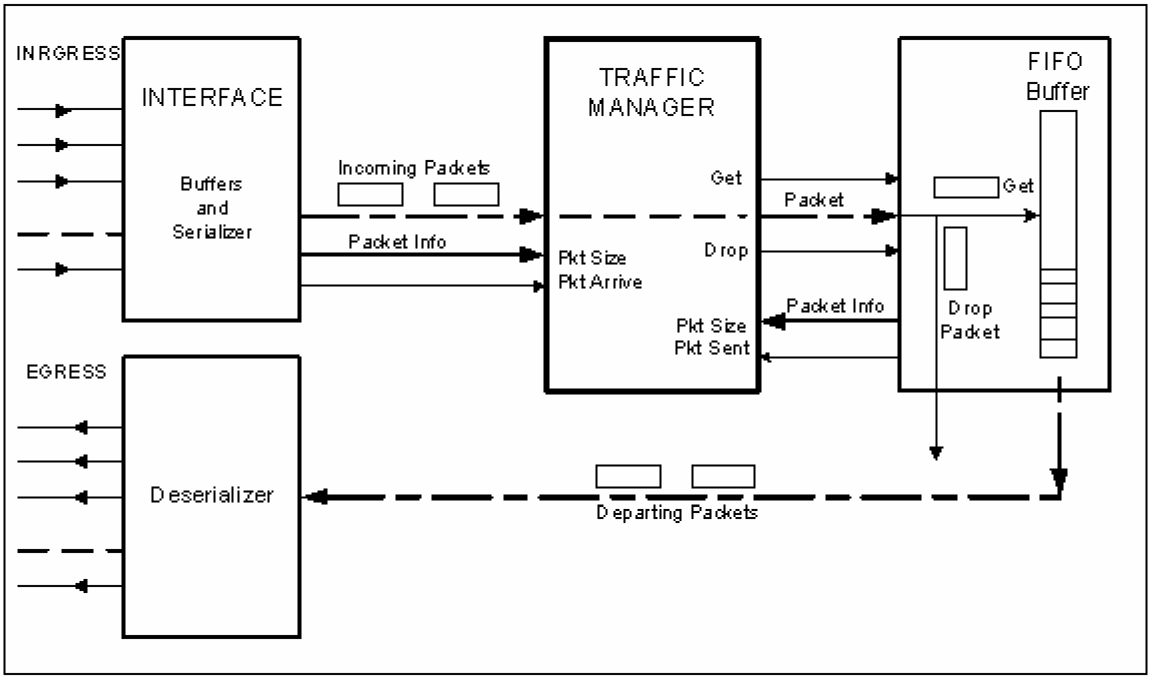


Figure 5.1 – Arrival packets are buffered in the Interface and then are serialized and sent to Traffic Manager in high speed rate, and then the traffic manager decides to drop or keep each packet in FIFO queue.

Then depending on the current situation, the traffic manager decides to drop or store the currently arrived packet in the queue. Current decision is determined by calculating the probability factor (Pb) which is a function of the average queue size (Avg), a random number (R), size of the arrived packet (Packet-Size) and some other parameters which will explain completely in the next sections. All these calculations are based on several arithmetic operations such as multiplication, division and powers. In order to afford these calculations in a high speed gateway it has to be done as fast as possible. That’s a major problem of the RED algorithm if used in high speed gateways, as pointed by Floyd in [6]. Therefore in order to decrease the calculation time, approximation methods are used in RED implementation as well as in our design. Our design is targeting a Virtex-II Pro device from Xilinx family. XC2VP30 device from this family has shown better performance and throughput especially in terms of routing delay as well as less logic delay in synthesis reports. However it could be downloaded in Virtex_4 family devices and consequently show much more throughput in terms of speed. The Virtex-4 is the new generation of Xilinx products released in 2004. Our design has been downloaded into a Xilinx Virtex-II Pro FPGA and its major features are tested. The timing reports are given later in this chapter, and demonstrate that our implementation is able to operate up to 10 Gbps properly.

5.2 – New features

In order to implement the RED algorithm as shown in Fig. 4.3 there are several operations that have to be executed for every packet arrival.

There are several other new features in our new modified algorithm which make significant differences with the original RED algorithm. In our case, we consider the number of byte rather than the number of packet to compute the drop decision. Since, the packets are of different sizes, this allow having more accurate calculations and hence more reliable drop decisions. Considering different packet sizes in the algorithm would affect directly the calculation of Pb in

(4.5). Because, in case of using the byte option and expecting the packets in different sizes, the algorithm would be modified to drop the packets with a probability proportional to the packet sizes [6]. That means if the arrived packet is too small, it shouldn't be dropped with the same probability as much as for the largest arrival packet:

$$\text{New\_Pb} \Leftarrow \text{Old\_Pb} \;.\; (\text{Pkt\_Ratio})$$

Where:

$$\text{Pkt\_Ratio} = \text{Packet\_Size} \,/\, \text{Maximum\_Packet\_Size}$$

Then:

$$P_b = (\text{Max\_}_{p}) \;.\; [(\text{Avg} - \text{Min\_}_{th}) / (\text{Max\_}_{th} - \text{Min\_}_{th})] \;.\; (\text{Pkt\_Ratio}) \qquad (5.1)$$

Since the design is targeting the FPGAs, implementing such high speed algorithm encounters with more complexity in terms of speed. However this implementation has afforded this challenge and has responded properly in 10 Gbps. In this regard special properties of the FPGAs are employed in the implementation, like internal high speed input/output buffers, and some special internal components. Digital Clock Managers (DCMs) as a property of the FPGAs and provide advanced clocking capabilities to FPGA applications. DCMs can multiply or divide the incoming clock frequency in order to synthesize a new clock frequency. DCMs also eliminate clock skew which could improve the system performance. Phase shifting is also another application of the DCMs, so it can delay the incoming clock by a fraction of the clock period [13]. Thereby a global low-skew clock is distributed in the FPGA. Delay Locked Loop (DLL) as a part of the DCM exploits a very low-skew clock from incoming clock and tries to shift the internal low-skew clock until reaches the corresponding edge of the incoming clock. Then it locks the low-skew clock in very few clock cycles. Consequently, DLLs (as parts of DCM) solve a variety of common clocking issues, especially in high performance and high speed implementations as well as in our design. Figure 5.2 summarizes clock synthesis options of a primitive DCM of Xilinx series. Using the CLK2X of the DCM in our implementation not only has improved the clock-skew, but also allowed us to provide a lower frequency for the external clock, and the duty cycle is also adjusted as well. Also we designed a verification environment to be downloaded in the same FPGA together with the design, in order to be able to test our design at high speed. Since the test equipments are very expensive for this kind of application. Our test environment, allow us to test the major features of our algorithm using the virtual packets which generates random packets with random arrival time.

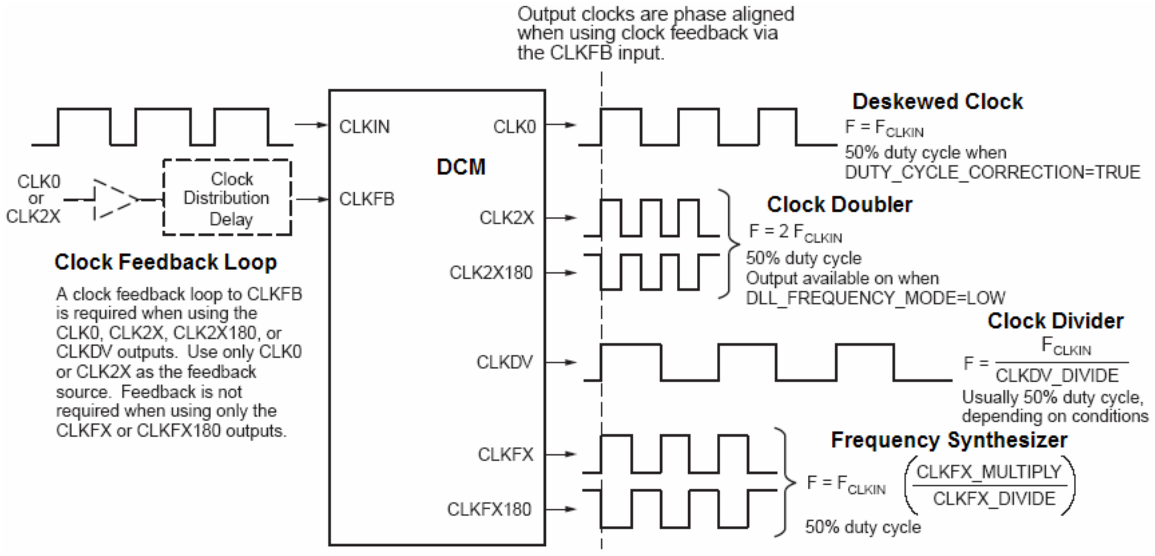

Output clocks are phase aligned when using clock feedback via the CLKFB input.
CLKIN
CLK0
DCM
CLKFB
CLK2X
CLK2X180
CLKDV
CLKFX
CLKFX180
CLK0 or CLK2X
Clock Distribution Delay
Clock Feedback Loop
A clock feedback loop to CLKFB is required when using the CLK0, CLK2X, CLK2X180, or CLKDV outputs. Use only CLK0 or CLK2X as the feedback source. Feedback is not required when using only the CLKFX or CLKFX180 outputs.
Deskewed Clock
F = F_CLKIN
50% duty cycle when DUTY_CYCLE_CORRECTION=TRUE
Clock Doubler
F = 2 F_CLKIN
50% duty cycle
Output available on when DLL_FREQUENCY_MODE=LOW
Clock Divider
F = F_CLKIN / CLKDV_DIVIDE
Usually 50% duty cycle, depending on conditions
Frequency Synthesizer
F = F_CLKIN (CLKFX_MULTIPLY / CLKFX_DIVIDE)
50% duty cycle

Figure 5.2 – Clock Synthesis Options in a DCM primitive.

It emulates randomly the sending of the packets, since our algorithm takes this information into account. The environment could not be configured by a ratio of “received packet / sent packet” in order to emulate real traffic in a given gateway.

5.3 – New modified algorithm and proposed contributions

In this section we discuss our major contributions for the RED algorithm.

5.3.1 – Impact of events for calculating the average queue size

The foremost open question on RED gateways involves determining the optimum average queue size for maximizing throughput and minimizing delay for various network configurations. The original RED algorithm showed that the new average queue size is calculated on every packet arrival at the gateway. This could cause some error in this calculation in case of no packet arrival for some periods of time, while stored packets in the queue are being sent through egress port. Because by sending the packets, the size of the queue is actually changed and consequently the corresponding average queue size should be changed too. Although by the next received packet at any later time the average queue size is calculated based on the new instantaneous queue size at that time, but the result is still different than the real calculation in our low pass filter algorithm. Our new algorithm calculates the average queue size on both packet arrival and packet sent, however it increases the complexity for the hardware implementation considering the speed constraints.

In case of both the packet arrival and the packet sent happen at the same time, the instantaneous queue size is affected because of both of them and stores the real current queue size in a register. In such condition the average queue size must not be calculated twice and once is enough. Our implementation by detecting such condition calculates the average queue size only once.

In the following, we present several simulation results in a graphical format as shown in figures 5.3 to 5.10. They compare the original RED algorithm (calculating the average queue size only on every packet arrival), with our algorithm (which calculates the new average queue size on either packet arrival or packet sent). All source codes in this thesis are written in VHDL and simulated under ALDEC HDL Model Simulator.

These simulations are executed in different conditions by changing the ratio of the packets sent to the incoming packets, as well as changing the weight of the queue (w). There is an Over Drop area percentage obtained from the simulation for each case specified in each graph. As every similar pair of the simulations is done under the same conditions, our new algorithm demonstrates less percentage of packet drops. Here are the constants used in these simulations;

Min_th = 8 Kbytes

Max_th = 24 Kbytes

Max_p = 1/32

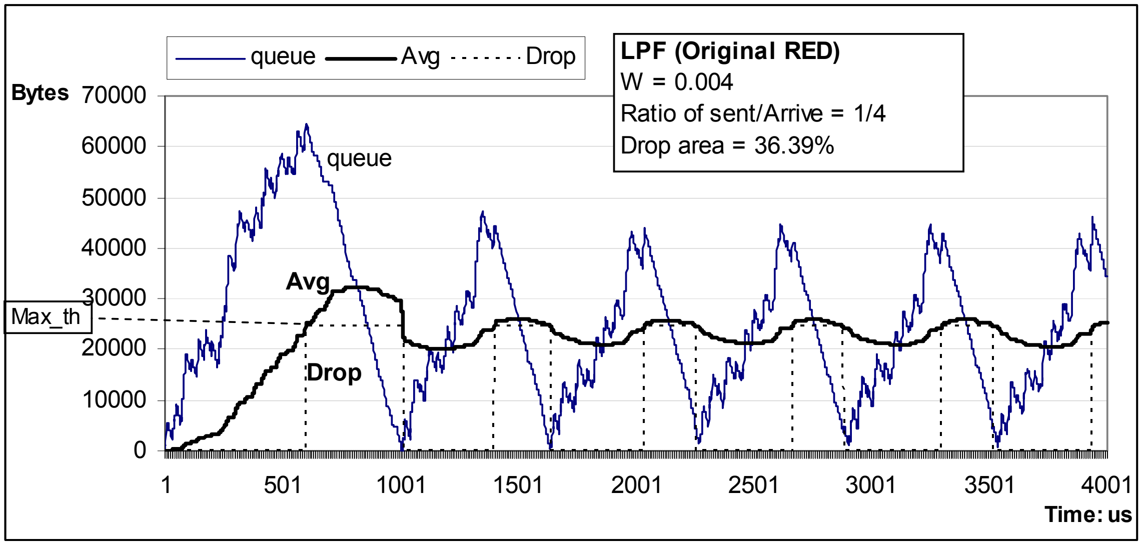


Figure 5.3 – Simulation result of Original RED algorithm with "w = 0.004" and ratio of "Packet sent / Packet arrive" = ¼ , (obtained Over-Drop rate = 36.39% in terms of percentage of the time).

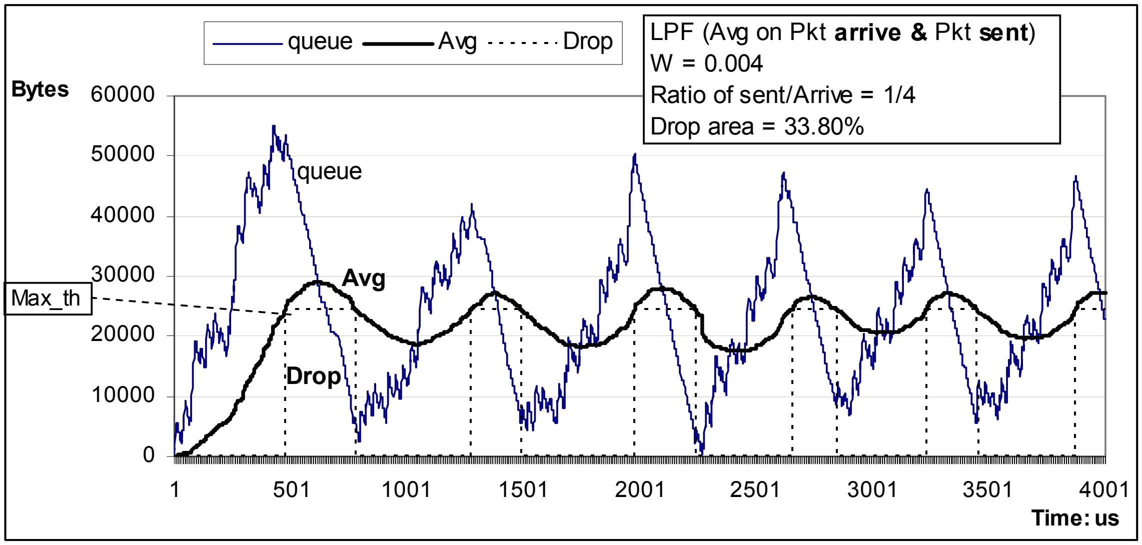


Figure 5.4 – Simulation result of modified RED algorithm calculating the average queue size on both packet arrival and packet sent, with "w = 0.004" and ratio of "Packet sent / Packet arrive" = ¼ , (obtained Over-Drop rate = 33.80% in terms of percentage of the time).

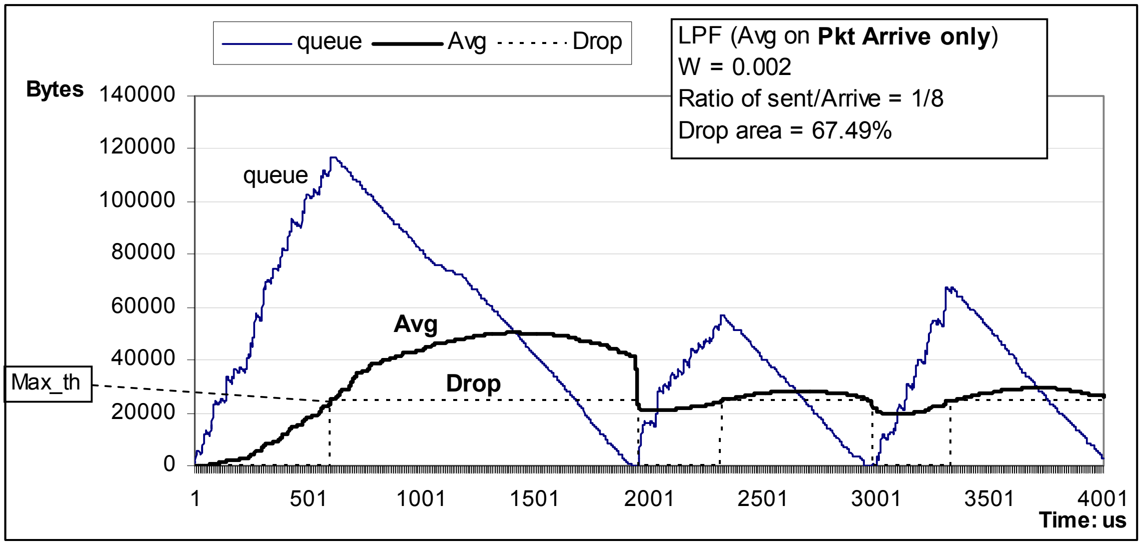


Figure 5.5 – Simulation result of Original RED algorithm with "w = 0.002" and ratio of "Packet sent / Packet arrive" = 1/8 , (obtained Over-Drop rate = 67.49% in terms of percentage of the time).

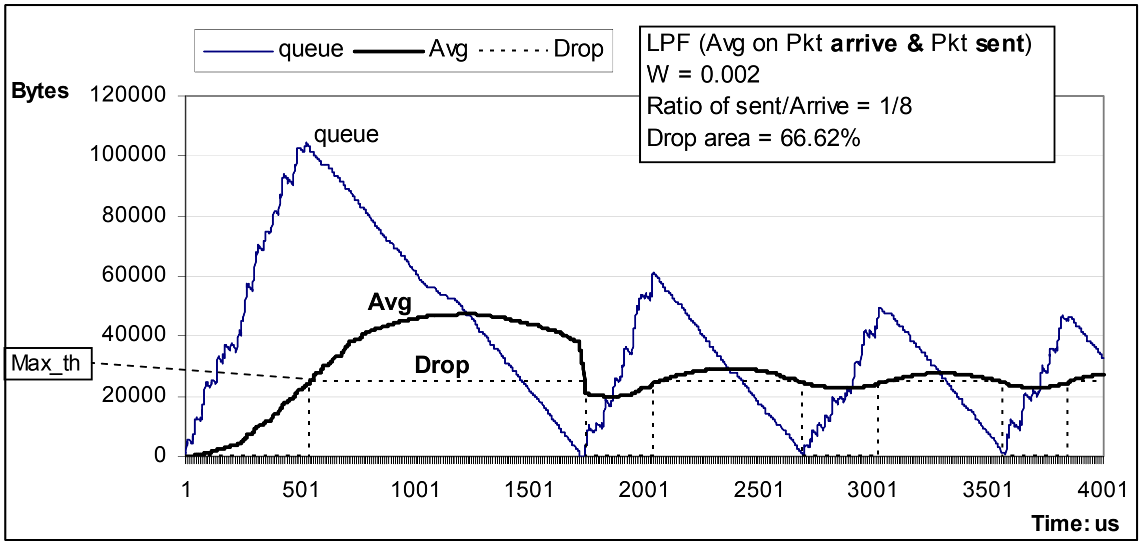


Figure 5.6 – Simulation result of modified RED algorithm calculating the average queue size on both packet arrival and packet sent, with "w = 0.002" and ratio of "Packet sent / Packet arrive" = 1/8 , (obtained Over-Drop rate = 66.62% in terms of percentage of the time).

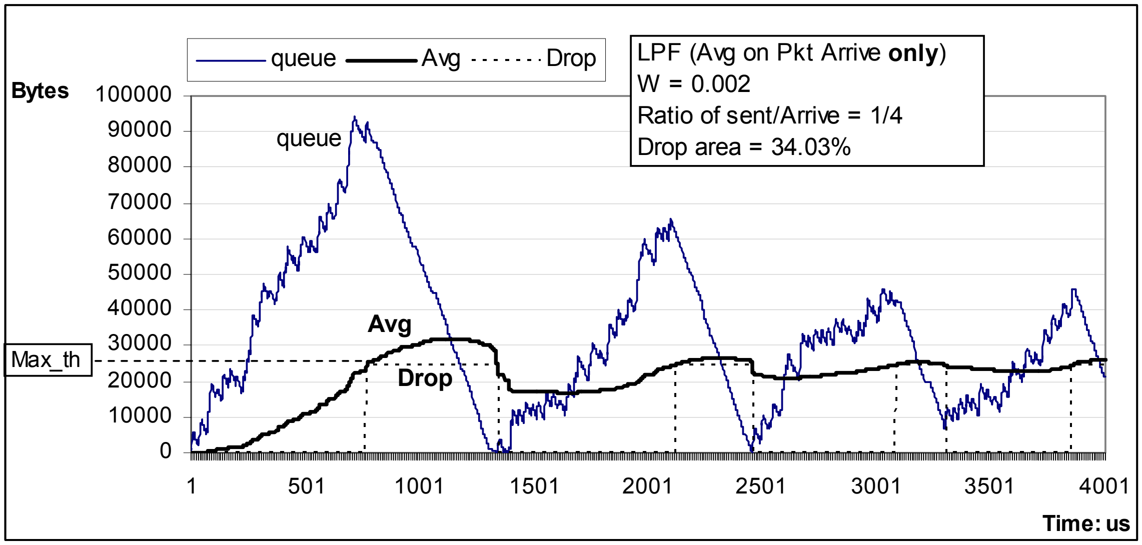


Figure 5.7 – Simulation result of Original RED algorithm with "w = 0.002" and ratio of "Packet sent / Packet arrive" = ¼ , (obtained Over-Drop rate = 34.03% in terms of percentage of the time).

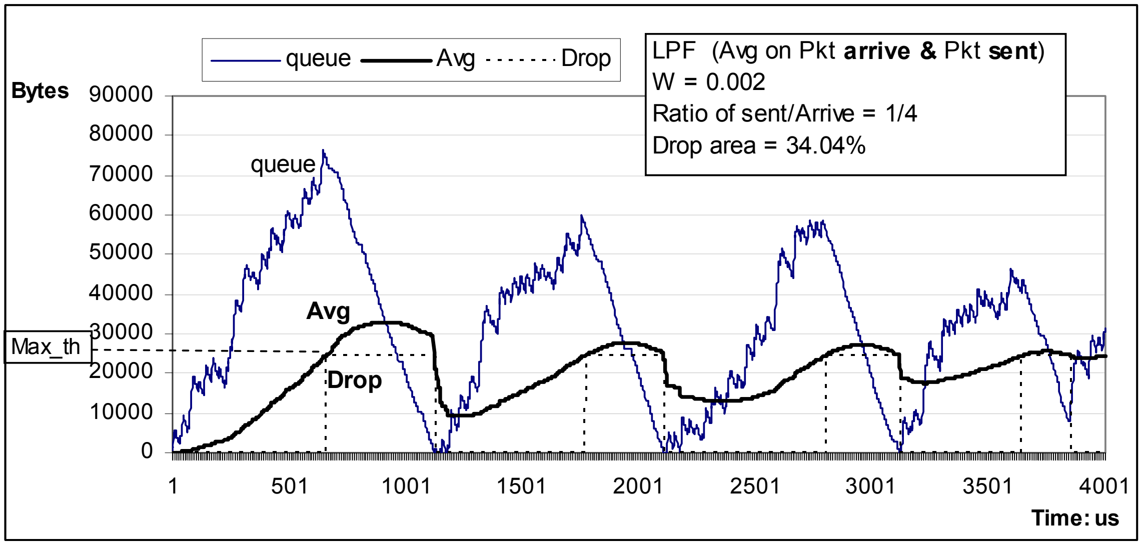


Figure 5.8 – Simulation result of modified RED algorithm calculating the average queue size on both packet arrival and packet sent, with "w = 0.002" and ratio of "Packet sent / Packet arrive" = ¼ , (obtained Over-Drop rate = 34.04% in terms of percentage of the time).

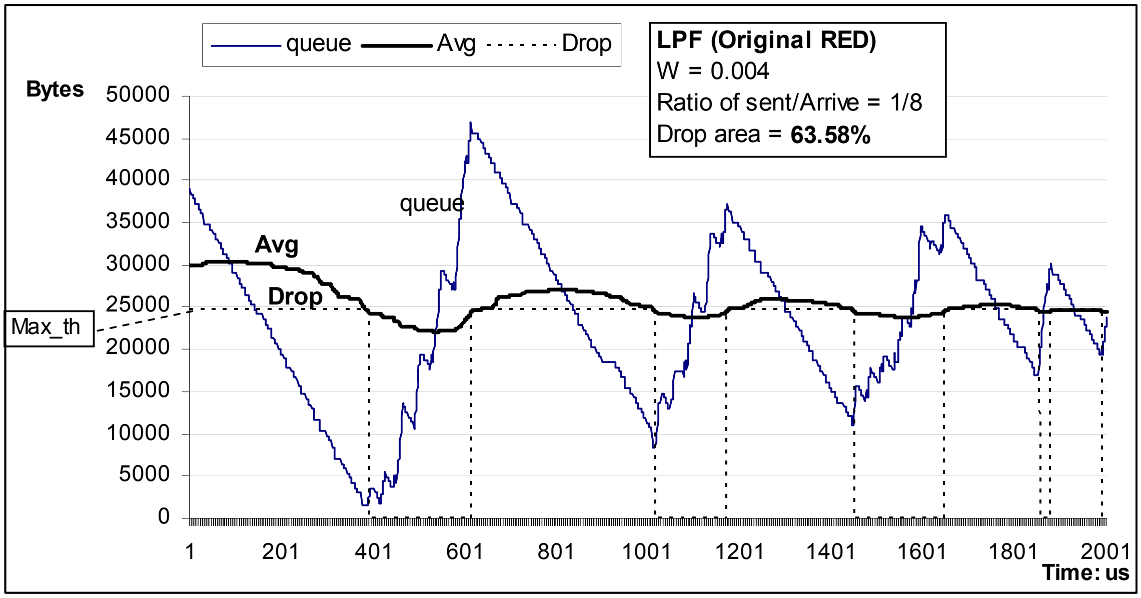


Figure 5.9 – Simulation result of Original RED algorithm with "w = 0.004" and ratio of "Packet sent / Packet arrive" = 1/8 , (obtained Over-Drop rate = 63.58% in terms of percentage of the time).

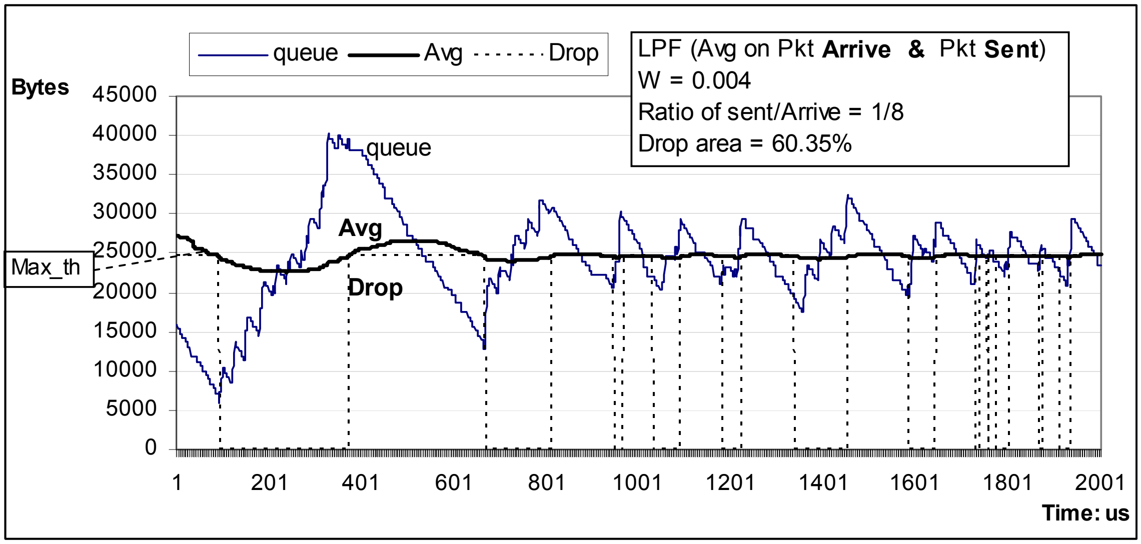


Figure 5.10 – Simulation result of modified RED algorithm calculating the average queue size on both packet arrival and packet sent, with "w = 0.004" and ratio of "Packet sent / Packet arrive" = 1/8 , (obtained Over-Drop rate = 60.35% in terms of percentage of the time).

A summary of these results are presented in Table-5.1.

Table 5.1 – This table gives the abstract results of the simulations on both original RED algorithm and our modified algorithm which calculates the average queue size on every arrival packet as well as on every sent packet, showing the improvement on Drop area.

Conditions	Drop % Original RED Avg affected on: Only Arrival	Drop % Modified RED Avg affected on: Arrival & Sent	Improvement of Drop %
Sent/Arrive = 1/8 W=0.004	63.58	60.35	+ 3.23
Sent/Arrive = 1/8 W=0.002	67.49	66.62	+ 0.87
Sent/Arrive = 1/4 W=0.002	34.03	34.04	- 0.01
Sent/Arrive = 1/4 W=0.004	36.39	33.80	+ 2.59

5.3.2 –SODA method to resolve the empty queue problem in RED

We are focusing on cases when there is congestion and it happens whenever the flow of incoming packets is more than outgoing packets. And we are interested in dropping as few packets as possible in congestion periods. Especially, it is much more important when the congestion is very big. So, our investigations are based on where the ratio of the packets sent to the received packets is small (less than “1”).

We proposed the Stepped Over Drop Avoidance (SODA) method in this thesis which is a modified version of Low Pass Filter/ Over Drop Avoidance

(LPF/ODA) mechanism given in [15]. The LPF/ODA discusses the compensation of the excessive packets drop after a long term congestion at the RED gateways. In the RED algorithm, the average queue size does not follow the instantaneous queue size as long as no packet arrives at the gateway. In other words, in such cases after long term congestion if no packet arrives at the gateway for a while, some queued packets in the buffer could have been sent meanwhile. And this causes to free up lots of spaces in the queue and make it ready to accommodate some new packets. On the other hand, since the RED algorithm calculates the average queue size only each time a packet arrives at the gateway, thus, the queue would have contained few packets or even have become almost empty. This is while the average queue size register, keeps a big wrong value since the last arrived packet after long term congestion, because the average queue size does not follow the instantaneous queue size on every sent packet. And consequently it does not let the buffer to accommodate new packets after a congestion term until the average queue size is decreased to below the maximum threshold level. In other words, the average queue size has been unnecessarily a large value for a while after long term congestion, causing to drop some packets excessively.

RED gateways in case of no packet arrival do not resolve this problem unless the queue becomes empty (zero). Floyd and Jacobson’s method compensates or modifies some what the obtained error in the average queue size as the result of the empty queue problem in RED algorithm [6]. It is done by calculating the new average queue size through (4.3) described in section 4.2.1

when a packet arrives at the gateway with empty queue. In order to avoid excessive packets dropping in this regard, a proposed algorithm named “LPF/ODA” algorithm [15], has stated that how well the original RED algorithm has satisfied the quick response to the end of the long term congestion. It shows that the original LPF of the RED algorithm satisfies accommodating the short term congestion as well. But for quickly response to the end of the long term congestion in order to avoid dropping of excessive packets, it does not satisfy this requirement very well. Because;

1- The Original RED attempts to modify the average queue size only after queue becomes zero. While there are many opportunities to compensate this error before the queue gets empty.
2- The proposed formula to compensate the error is too big and complex to quickly calculate and respond especially for high speed networks.

Bing and Mohammed in LPF/ODA [15] have proposed an algorithm to reduce the excessive packets dropping. They do that by halving the average queue size on every packet arrival after a long-term congestion, if the average queue size is above minimum threshold level (Min_th). They consider the end of the Long-term congestion if the instantaneous queue length being at a “low level” for a considerable period of time that average queue length is above minimum threshold level (Min_th). During this long-term congestion the average queue length is calculated by LPF of RED using (4.1), and after long-term congestion is gone and the Avg is greater than Min_th, it will be halved. Figure 5.11 shows the pseudo code for LPF/ODA algorithm [15].

```
for  each packet arrival:
     if   long term congestion      then
          $Avg \Leftarrow (1-w) . Avg + w.q$
     else
          if   $Avg \geq Min\_th$   then
               $Avg \Leftarrow 0.5 Avg$
          else
          $Avg \Leftarrow (1-w) . Avg + w.q$
          end if
     end if
end
```

Figure 5.11 – Pseudo code for LPF/ODA algorithm.

As stated in LPF/ODA, it waits for a considerable time period after long term congestion. It means that any way, while detecting the end of congestion, it does let the gateway to continue excessive packets dropping, and then it halves the Avg. This clearly implies that eliminating such wasting of this considerable time period, could achieve higher throughput by less excessive packet dropping. We have modified this algorithm by breaking down this period of time into several smaller steps instead. We called this method as Stepped Over Drop Avoidance (SODA). However, both LPF/ODA algorithm and our SODA algorithm, have less complexity in terms of calculating the average queue size after long term congestion, versus the complex formula in original RED algorithm. This algorithm tries to follow the average queue length so that if the instantaneous queue length is less than a predefined fraction of the average queue size, and if the Avg is above Min_th, then it decreases the Avg as this amount of fraction. Since the implementation is using approximation, the fraction of Avg could be based on a negative power of “2” which simplifies the divide calculation by using a shift instruction instead. LPF/ODA halves the Avg on such events, thus, the fraction in LPF/ODA algorithm is (2^{-1}), or one half (½) the Avg. But our SODA algorithm is flexible in choosing the number of the steps to break this long step down to smaller steps. Regarding the step resolution, number of steps in this algorithm could be “4”, “8” or “16” and etc., since these are negative powers of two. But we should note that, as the number of the steps gets larger and larger (i.e. makes very smaller steps), it will not necessarily affects much more on excessive packets dropping. Most of our SODA simulations are executed upon splitting this area to 8 steps, or using the fraction of ($1/8 = 2^{-3}$) and it has responded appropriately well. Although, our new algorithm calculates the new average queue length on every packet, whether it is incoming or outgoing. Our proposed modified algorithm for avoiding excessive packets drop is shown in Figure – 5.12, where “Steps” is the number of steps chosen used for step resolution which is a power of two. In our simulations the step resolution is ($1/8 = 2^{-3}$) or the number of states is “8”.

```
for each packet arrival or packet sent do:
    if $(Avg > Min\_th)$ and $(q < (Avg – (Avg/Steps)))$ then
        $Avg \Leftarrow Avg – (Avg / Steps)$
    else
        $Avg \Leftarrow (1-w) . Avg + w.q$
    end if
end do
```

Figure 5.12 – Pseudo code for SODA algorithm.

The simulations have shown considerable improvement on avoiding excessive packets drop as shown in Figures 5.13, 5.14 and 5.15. These simulations, shown in these three figures show a comparison between the three mentioned algorithms, original LPF of the RED, LPF/ODA, and SODA algorithms. All of them are simulated using the same conditions. They show the improvement getting better and better in terms of less excessive packets drop by measuring the percentage of the drop area in each case excluding the incipient congestion. For all cases, the conditions and values for constant parameters are as following;

W = 0.002	"mostly recommended by Floyd in [6]"
Congestion Factor = ~¼	"ratio of departing packets to arrivals"
Max_p = 1/32 = 2^{-5}	
Max_th = 24 KB	
Min_th = 8 KB	

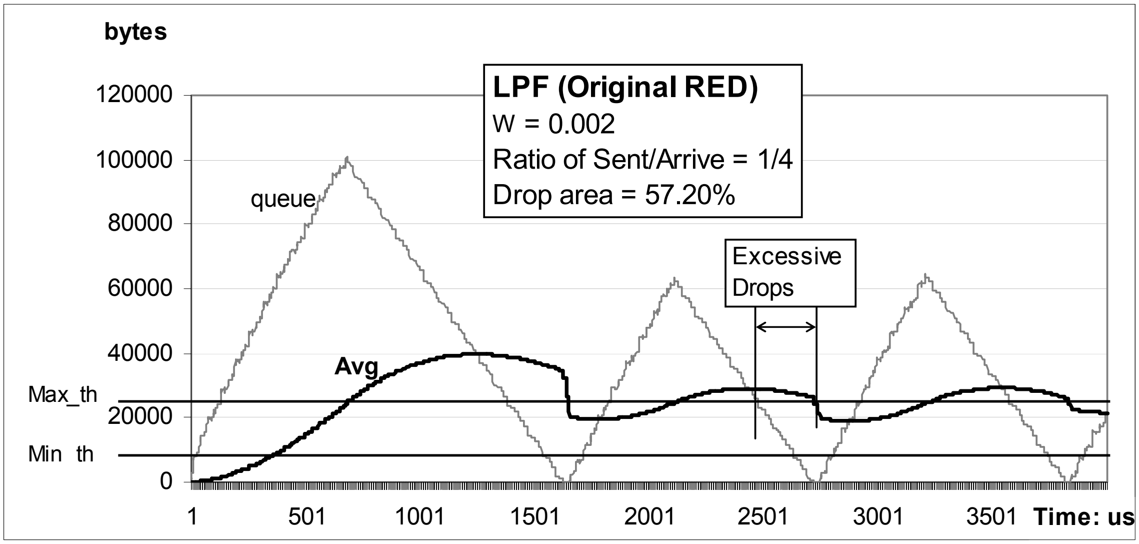


Figure 5.13 – Simulation result of Original RED algorithm, with "w = 0.002" and ratio of "Packet sent/Packet arrive" = ¼, showing more excessive packets drop with higher percentage of drop area versus LPF/ODA and SODA algorithms.

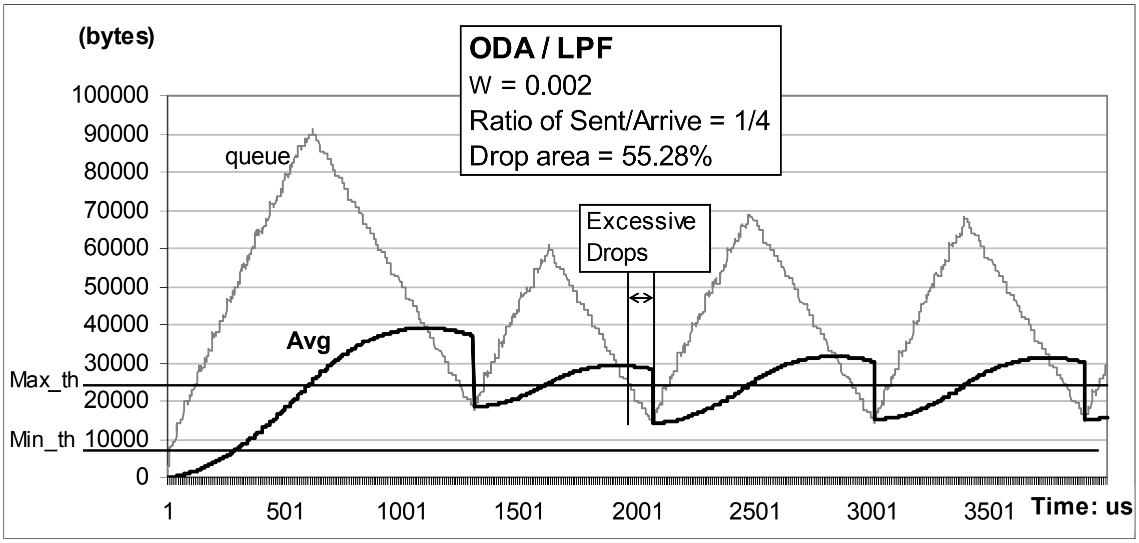


Figure 5.14 – Simulation result of LPF/ODA RED algorithm, with "w = 0.002" and ratio of "Packet sent/Packet arrive" = ¼, showing a sample time of excessive packets drop with normal percentage of drop area versus Original RED and SODA algorithms.

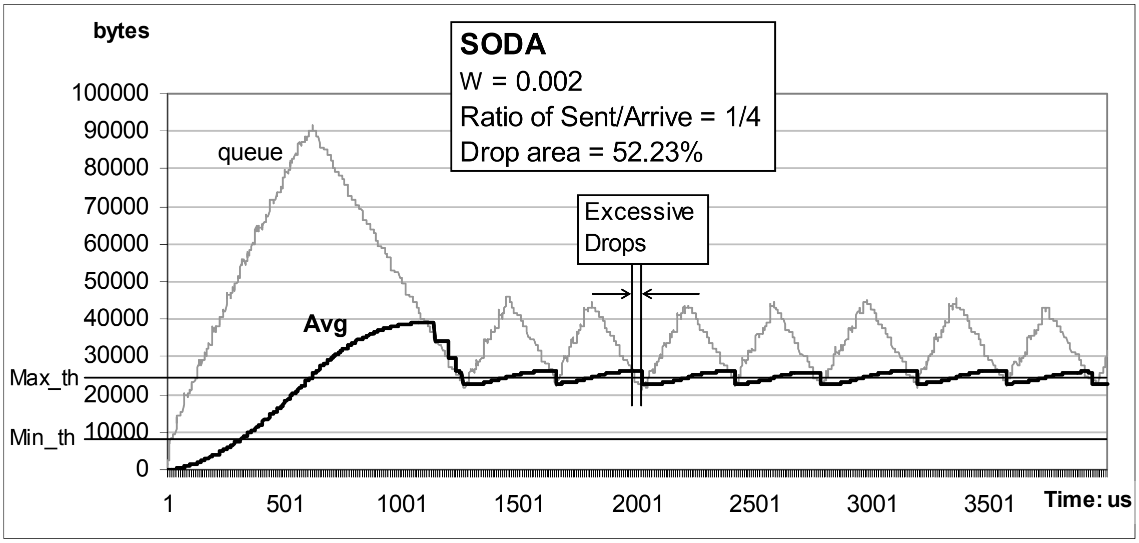


Figure 5.15 – Simulation result of SODA algorithm, with "w = 0.002" and ratio of "Packet sent/Packet arrive" = ¼, showing less excessive packets drop with less percentage of drop area versus Original RED and LPF/ODA algorithms.

A survey on the above figures indicates that after incipient congestion SODA has shown much better performance. It is improved in terms of percentage of drop area, maximum pick of instantaneous queue length which is directly concerned with the delay for packets transferring through the gateway. Also, if we look carefully to these three graphs, we will find that the LPF and LPF/ODA are experiencing two to three long term congestions after incipient congestion in the same similar time period for all cases. But the SODA algorithm shows its activity by larger number of shorter term congestions. And this implies better distribution of the congestion between all sources sending packets to the gateway, and consequently resulting less global synchronization. Table 5.2 gives briefly a better comparison between these results.

Table 5.2 – Comparison between the simulation results of the Original RED algorithm, LPF/ODA, and our final modified SODA algorithm which is using our Stepped Over Drop Avoidance and calculating the average queue size on both arrival and sent packets in three cases shown in figures 5.13, 5.14 and 5.15. The simulations are executed under the same conditions in 4 ms time period and with the weight of the queue "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor is "1/4".

Results: **Algorithms:**	**Drop %**	**Max q_Size**(KB) **(Delay concern)**	**Congestion terms** **(More Drop distribution, Less global Synch.)**
Original RED	57.20	60	2
RED LPF/ODA	55.28	65	3
RED (SODA)	**52.23**	**45**	**7**

The next three simulations shown in Figures 5.16, 5.17 and 5.18 represent the behavior of the algorithm after a long time after incipient congestion to exclude the slow start period.

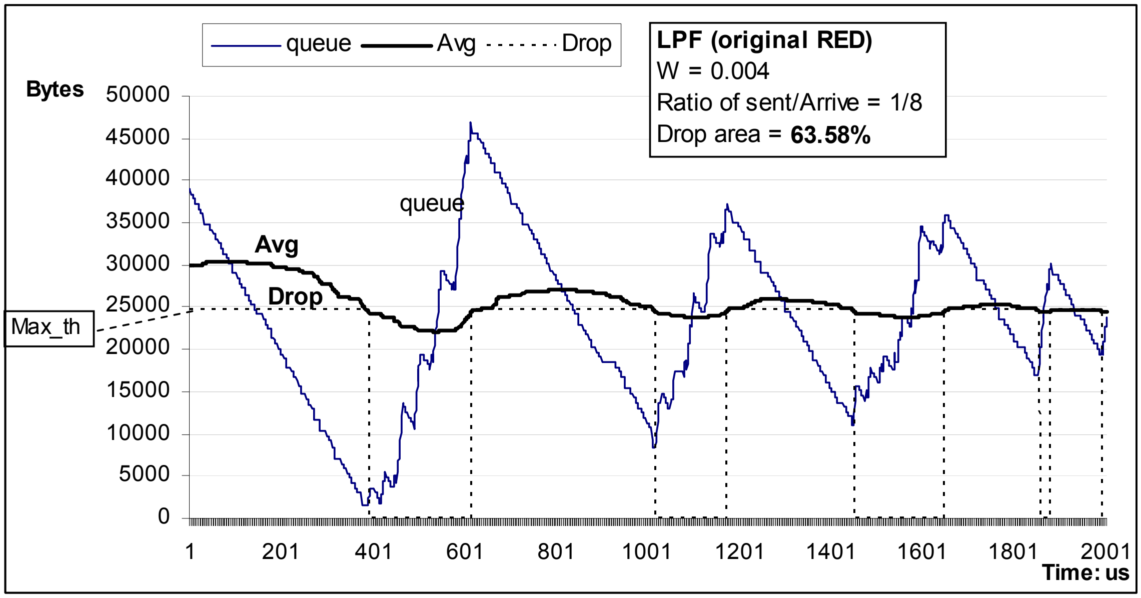


Figure 5.16 – Simulation result of original RED algorithm, with "w = 0.004" and ratio of Packets to the arriving Packets as the congestion factor which is "1/8".

These again show a better performance for our SODA algorithm versus other two algorithms. However the LPF/ODA algorithm sometimes reacts even worse than original RED (LPF), like here. Table 5.3 compares them briefly.

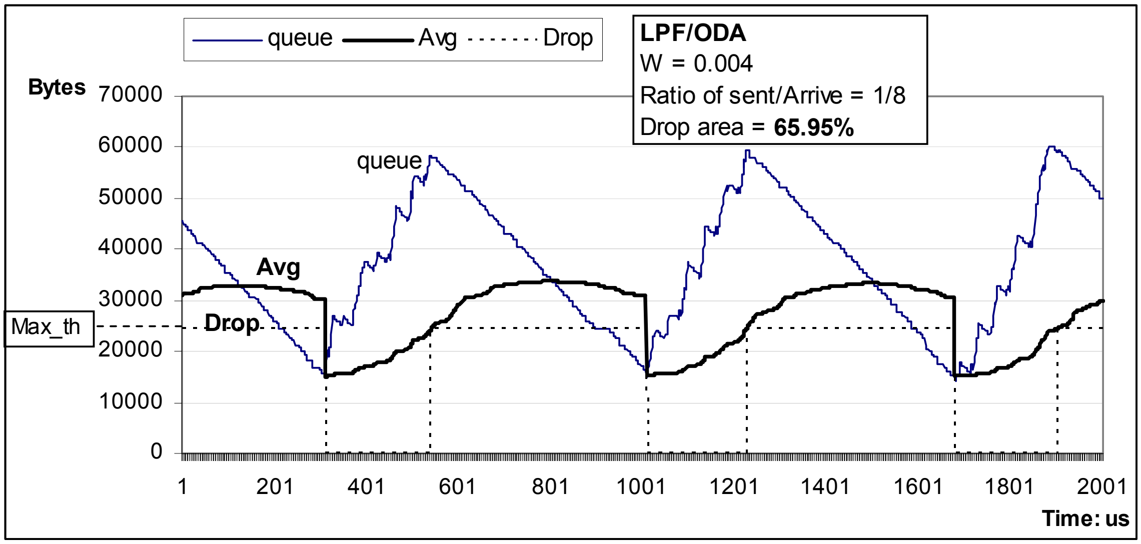


Figure 5.17 – Simulation result of LPF/ODA algorithm, with "w = 0.004" and ratio of Packets to the arriving Packets as the congestion factor which is "1/8".

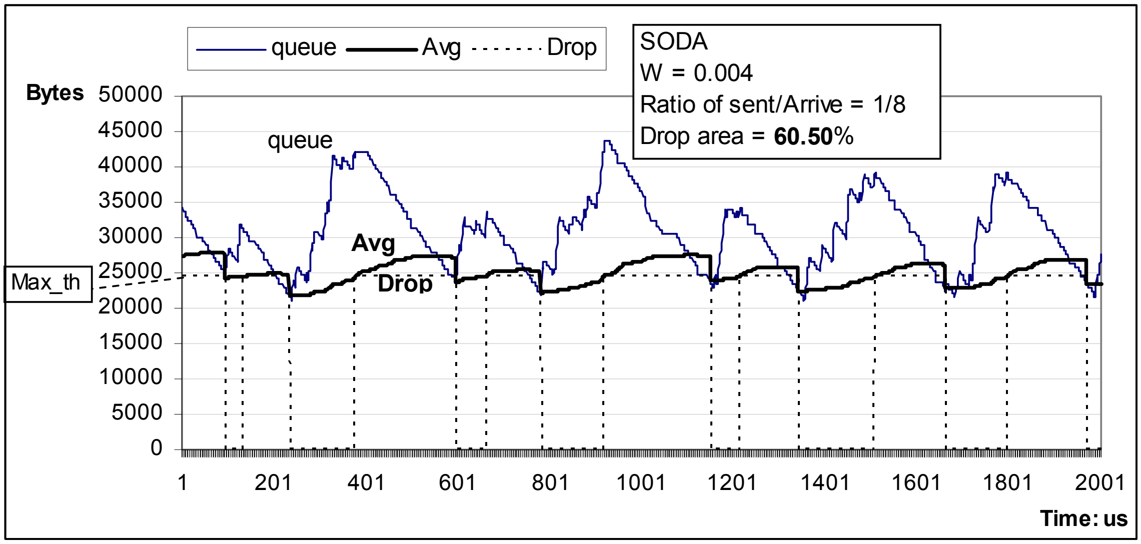


Figure 5.18 – Simulation result of SODA algorithm, with "w = 0.004" and ratio of Packets to the arriving Packets as the congestion factor which is "1/8".

Table 5.3 – Comparison between the simulation results of the Original RED algorithm, LPF/ODA, and our final modified SODA algorithm which is using our Stepped Over Drop Avoidance and calculating the average queue size on both arrival and sent packets in three cases shown in figures 5.16, 5.17 and 5.18. The simulations are executed under the same conditions in "2000 us" starting a long time after slow start or incipient congestion, with the weight of the queue "w = 0.004" and the ratio of sending Packets to the arriving Packets as the congestion factor which is "1/8".

Results: **Algorithms:**	**Drop %**	**Max q_Size**(KB) **(Delay concern)**	**Congestion terms** **(More Drop distribution, Less global Synch.)**
Original RED	63.58	~40	4
RED LPF/ODA	65.95	~60	3
RED (SODA)	**60.50**	**~35**	**7**

Finally we compare our complete SODA algorithm which is also affected by both arriving and sending packets for calculating the Avg, with original RED algorithm and other cases separately. Each of the other cases is using only one property of our contributions. Then there are four simulation results shown in Figures 5.19 through 5.22. First one is the LPF of the original RED algorithm, second is the LPF of the RED algorithm but calculating the average queue length on arriving and sending packets, third one is using only our stepped over drop avoidance (SODA) method, and the last one is our final modified RED algorithm using both methods; SODA, and calculating the average queue length on both arriving and sending packets. All are executed under the same conditions as the previous simulations with the same constants and initialization.

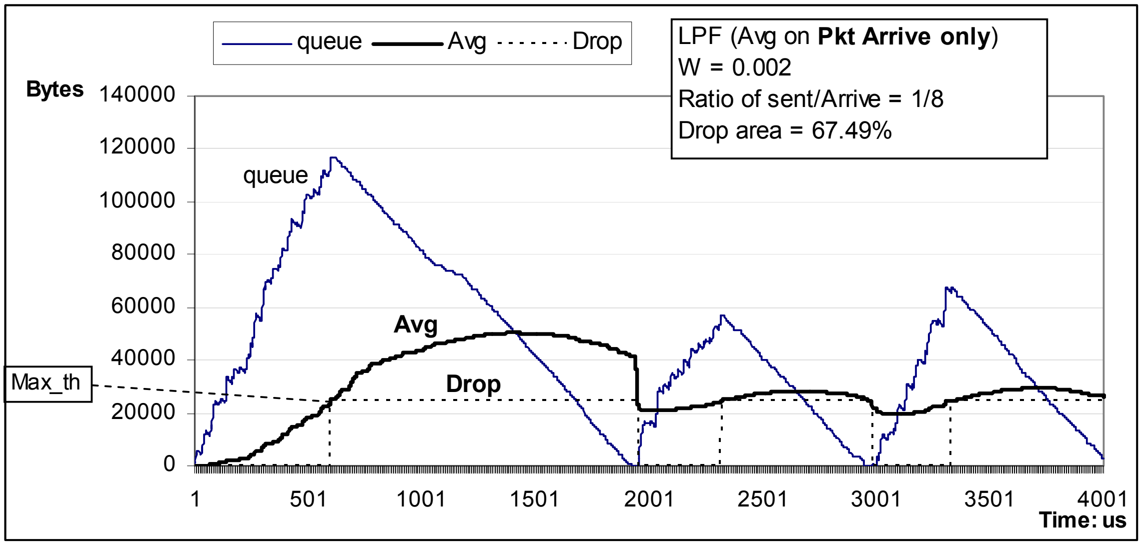


Figure 5.19 – Simulation result of LPF algorithm of original RED, with "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor which is "1/8".

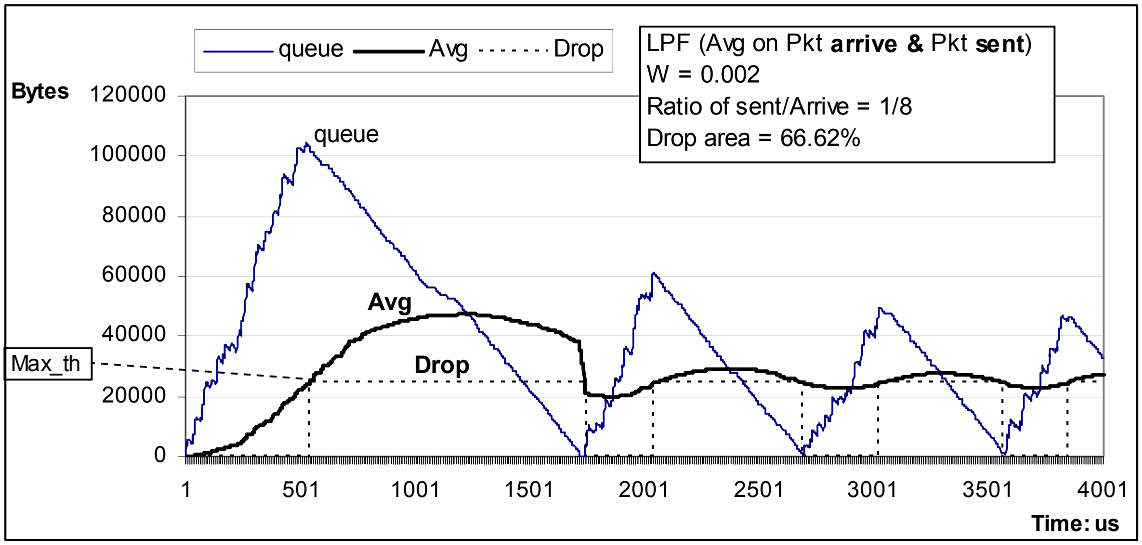


Figure 5.20 – Simulation result of the LPF algorithm of RED with calculating the average queue length on arriving and sending packets. Weight of queue is as "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor which is "1/8".

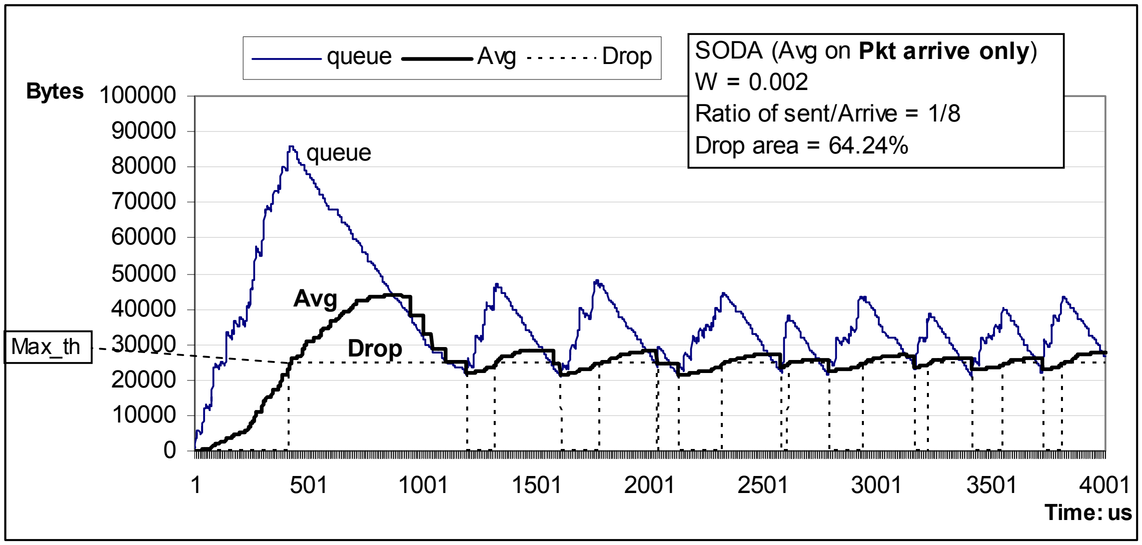


Figure 5.21 – Simulation result of the LPF algorithm of RED using only SODA method, with "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor which is "1/8".

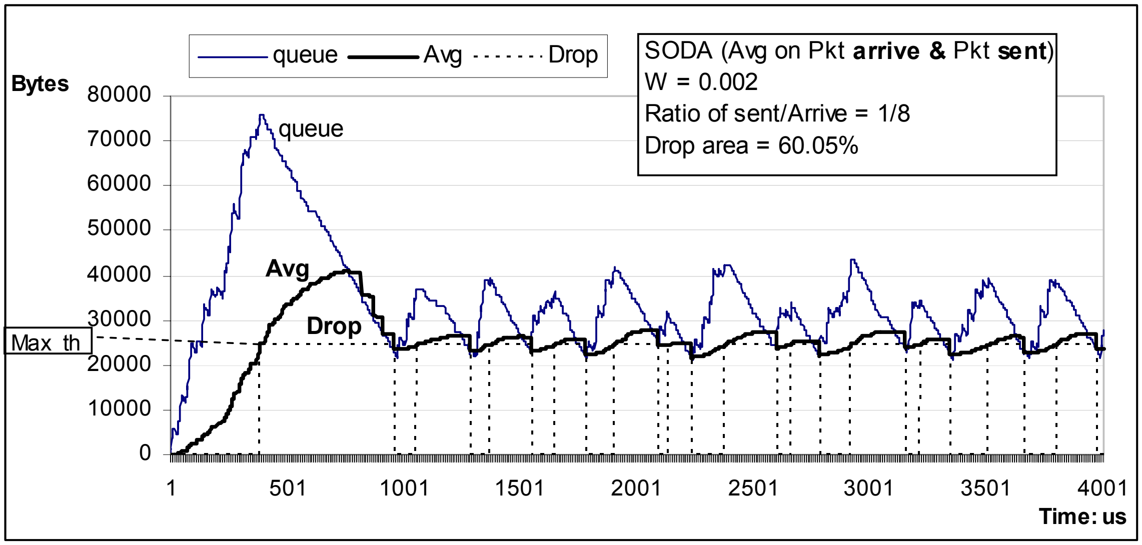


Figure 5.22 – Simulation result of final SODA-RED algorithm which calculates the average queue length on arriving and sending packets. It is using SODA method. Weight of queue is as "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor which is "1/8".

Table 5.4 shows briefly the complete comparison of these cases. As it is seen, the performance is improved on every step versus its previous step. The results are showing considerable improvements on all features in terms of less excessive packets dropping area, less delay in scheduling in the buffer, and less global synchronization because of splitting the long term congestions to several short term congestions, and consequently distributing shorter term congestions between the sources.

Table 5.4 – Comparison between the simulation results of the Original RED algorithm and our modified algorithm in three cases shown in figures 5.19 to 5.22; RED (PA, PS) which calculates the average queue size on both arriving and sending packets; RED (SODA) which is using our Stepped Over Drop Avoidance method, and our final modified algorithm (SODA, PA, PS) which is using our Stepped Over Drop Avoidance method and also calculates the average queue size on arriving and sending packets. The simulations are executed under the same conditions within 4000 us time period with weight of the queue "w = 0.002" and the ratio of sending Packets to the arriving Packets as the congestion factor being "1/8".

Results: **Algorithms:**	**Drop %**	**Max q_Size**(KB) **(Delay concern)**	**Congestion terms** **(More Drop distribution,** **Less global Synch.)**
Original RED	67.49	60	2
RED (PA,PS)	66.62	45	3
RED (SODA)	64.24	44	8
RED (SODA, PA, PS)	**60.05**	**35**	**11**

The proposed final modified RED algorithm is using stepped over drop avoidance, and also considers both packet sent and packet arrivals to calculate the average queue size. We call it SODA_RED algorithm. Hence, as seen in all simulated processes in several different conditions, the SODA_RED algorithm has always shown better performance versus other discussed algorithms. Our modified RED algorithm has shown sufficient improvements in the simulations, especially in terms of less packet drops, less delay in packet transfer, and generating less global synchronization problem at the gateways.

5.3.3 – New modified algorithm

Here we propose the final modified SODA_RED algorithm to implement as shown in Figure 5.23. Later in this chapter we will see how well this algorithm performs especially in terms of responding in high-speed traffic.

SODA method not only has shown better performance than LPF/ODA or original LPF of RED, its algorithm is also easier to implement compared to extensive formula modification of empty queue problem proposed by Floyd in RED [6].

```
Initialization:
    $Avg \Leftarrow 0$
    $C \Leftarrow -1$
for each packet arrival and each packet sent calculate
the average queue size "avg" :
if $(Avg > Min\_th)$ and $(q < (Avg - (Avg / Steps)))$ then
    $Avg \Leftarrow Avg - (Avg / Steps)$
else $Avg \Leftarrow Avg + w .( q - Avg)$
end if

if $Min\_th \le Avg < Max\_th$ then
    increment $C$
using new “Avg“ and “C “ calculate probability ”Pb”:
    $Pb \Leftarrow (Max\_p) . [(Avg - Min\_th) / (Max\_th - Min\_th)] . (Pkt\_Ratio)$

    If $C > 0$ and $C \ge Approx[R/Pb]$ then
        Drop the arrived packet
        $C \Leftarrow 0$
    end if

    if $C=0$ then
        Random number [R] $\Leftarrow$ Random[0,1]
    end if

else if $Avg \ge Max\_th$ then
    Drop the arrived packet
    $C \Leftarrow 0$
else $C \Leftarrow -1$

end if
```

Figure 5.23 -- **Final modified SODA_RED algorithm for implement.**

5.4 – FPGA Implementation

This SODA_RED implementation is written in VHDL source code and has been synthesized and implemented through ISE of Xilinx application series. It is then downloaded into a Xilinx FPGA and some major features of the design have been tested within the FPGA device. Hence, a required environment has been designed in VHDL, implemented and downloaded inside the same FPGA, to perform the testing. It is expected to work in 10 Gbps gateways, and is targeting Virtex_II Pro family devices from Xilinx FPGAs. XC2VP30, a high performance device from this group has responded as well to our high speed implementation.

5.4.1 – Principal specifications

To be able to respond in 10 Gbps, this implementation has to execute a complete process of the SODA_RED algorithm in relatively small time. In other words, the drop decision on an arrived packet must be done in minimum available time before the next packet arrives at the gateway (worst case). This minimum available time is determined by minimum size of the arriving packets at the gateway. However, the required time for the external interface to scan the result must be taken into account. Packets may arrive at the gateway in different sizes with different gap-times between them. The worst case is when the gateway is receiving packets with smallest size and zero ns gap-time between them. According to the FDE (Full Duplex Ethernet) in [14] the minimum and maximum standard packet sizes are 72-Bytes and 1526-Bytes. However, since the 8-Bytes for the introduction (7-byte), and SFD (Start Frame Delimiter) field (1-byte) of the header for each packet are automatically generated by the recipient [14], then we consider the worse case as being the arrival packets with even less size. Then for our implementation, the expected range of packet sizes is;

$$\text{Minimum packet size} = 72 - 8 = 64 \text{ Bytes} \qquad (5.1)$$

$$\text{Maximum packet size} = 1526 - 8 = 1518 \text{ Bytes} \qquad (5.2)$$

If we call the minimum available time as “T_Cycle” for a complete cycle, then our 10 Gbps traffic manager is calculated as followed. Keep in mind that each complete cycle in this design (T_Cycle) is divided into eight stages (S_0 ~ S_7), thus the given clock period (T) determines the required input clock:

T_Cycle = 64 Bytes / (10 G. bits per second)

T_Cycle = 512 bits / (10^{10} bits / 10^9 ns)

T_Cycle = 512 bits / 10 bit/ns = 51.2 ns (5.3)

T = T_Cycle / 8 = 6.4 ns (5.4)

$F_{(Clock)}$ = 1/6.4 ns = 156.25 MHz (5.5)

In section 5.2 we discussed how we employed the DLL (Delayed Locked Loop) ,an internal property of advanced FPGAs, in our implementation. This improved clock skew, adjusted Duty-Cycle and CLK2X properties are utilized in our design. Therefore, required external clock to FPGA is half the frequency we calculated in (5.5), and thus, a 78.125 MHz external clock would be sufficient.

The complete cycle in this implementation could not be pipelined, due to different packet sizes and gap-times. Unfortunately, we cannot expect every arriving packet start at specified times, and as such; this makes our high-speed implementation more challenging. But at least by generating the stages (S_0 ~ S_7) in the controller unit, we can register the results of each stage to be used for any other stage if applicable. Figure-5.24 shows input signals such as; clock specifications, generated stage-pulses (S_0 ~ S_7) by controller, Packet-Arrive signal, Reset and the outputs Drop and PA-acknowledgment.

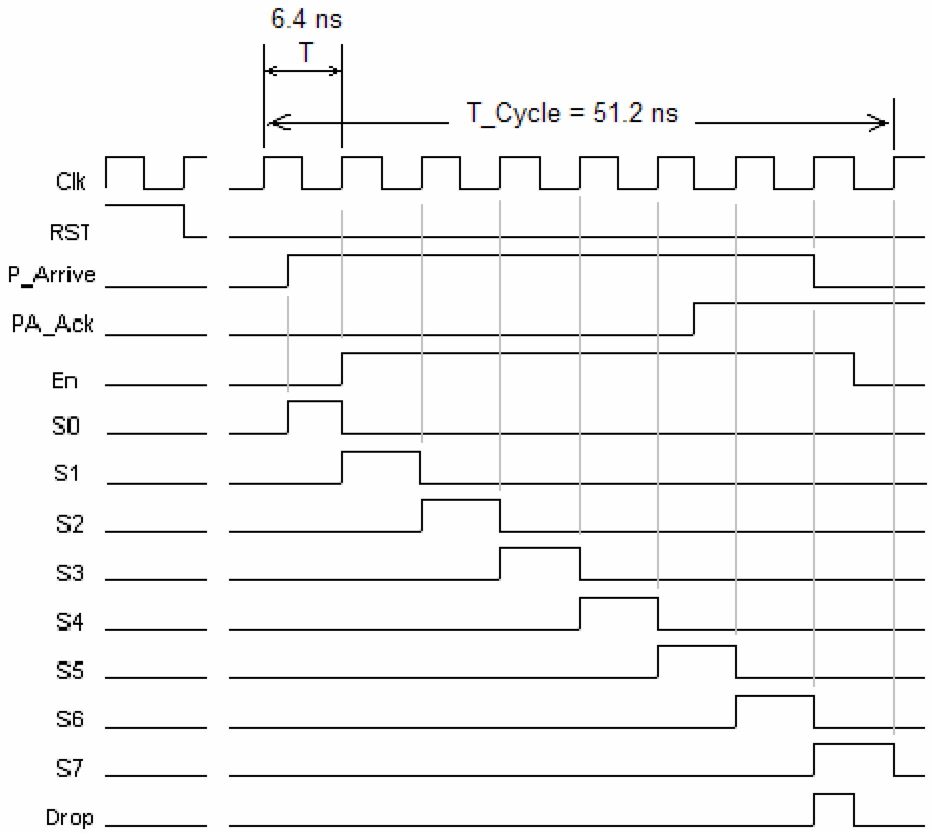


Figure 5.24 -- **Clock, stage-pulses and ~~some~~ timing controls for SODA_RED.**

Inputs to the system are two 16-bit vectors as PA_Size (Packet Arrive Size) and PS_Size (Packet Sent Size) which carry the size of the packets arrived or sent at the gateway, as well as PA and PS signals to inform the traffic manager about any incoming or outgoing packet, the Clock, the Reset, and a 2-bit width "W" which referring to four levels for weight of queue.

The system has been designed so that the weight of queue could be initialized flexibly through four fixed and predefined values. These fixed values for W are "2^{-9} =0.002, 2^{-8} =0.004, 2^{-7} =0.008 and 2^{-6} =0.016". The inputs PA_Size and PS_Size are 16-bit width, and since the maximum expected size of arrival packets is 1518 bytes, it covers the packet size width.

Outputs from the system are PA_Ack (Packet Arrive Acknowledge), PS_Ack (Packet sent Acknowledge), Drop (the main drop decision output signal to be used by the gateway) and the Valid_Drop signal to validate the final Drop decision for external devices waiting for decision.

General block diagram of our final SODA_RED algorithm to implement is designed as shown in figure-5.25.

The constant values used in the implementation are defined in following:

$$Min\_th = 8\ KB \qquad (8192\ bytes) \tag{5.6}$$

$$Max\_th = 24\ KB \qquad (24576\ bytes) \tag{5.7}$$

$$Max\_p = 1/32 \qquad (2^{-5}) \tag{5.8}$$

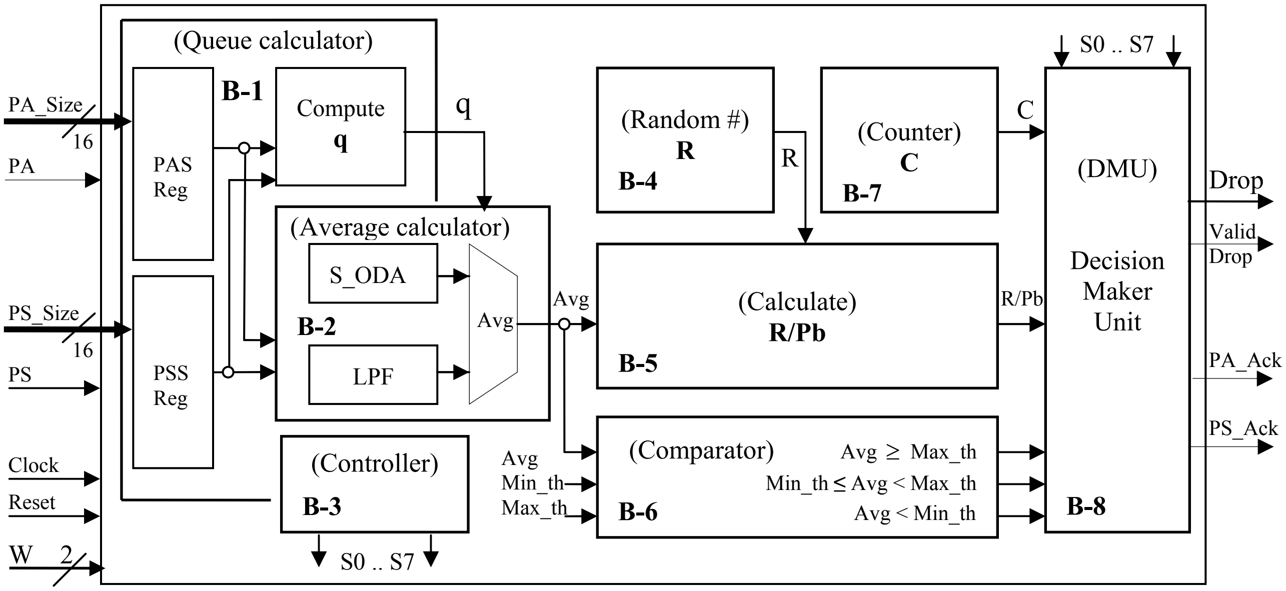


Figure 5.25 - General block diagram of the new modified SODA_RED

5.4.2 – Random pattern generator

There are several different methods to generate required random numbers for our algorithm. For example, Floyd in [6] has suggested a LUT (Look Up Table). For this design we designed a random number generator that generates a series of wide area random numbers. We employed two different standard mechanisms to generate the random pattern called CA (Cellular Automaton) and LFSR (Linear Feedback Shift Register) [16]. Each is consisted of several Flip-Flops which could be connected all together in series through any arbitrary number of feedbacks between them. Number of embedded Flip-Flops in them determines the bit-width of the generated random pattern. The constraint in each method is the flexibility of each number and the kind of arbitrary feedbacks between them determines its efficiency. The efficiency here for the generated random numbers depends on maximum coverage of all existing numbers belonging to the bit-width of the pattern generator. Although it is hard to obtain very large throughput, we mixed both of these methods heuristically together and found an appropriate topology to make a well efficient form of feedbacks between the Flip-Flops. The width of our designed pattern generator is 14-bit and its coverage of generated pattern determined in simulation is 15,890 (out of 2^{14} = 16384). This is 97% coverage of the total area of our 14-bit pattern generator. And that is indeed large enough to utilize for our calculations in the algorithm. The same pattern generator has been used to generate pseudo packet sizes and gap-times for verification in simulations through our designed environment.

5.4.3 – Design and Implementation of basic elements

Implementing the probability P_b Calculation in (5.1) and after that implementing the result of R/P_b division is the most important part of the design. These calculations are too large for high speed RED gateway and comprise of the difficult critical paths in hardware implementation. Therefore, eliminating such long path in the design is very effective to obtain the speed requirements for the implementation. However the original RED [6] has not considered the variety of arrival packets in terms of size, and the suggested implementation does not operate on byte option. Suggested approach is a method to calculate the P_b using two constants "C1" and "C2" and reformulating equation (4.5) as below:

$$C1 = Max\__{P} / (Max\__{th} - Min\__{th})$$

$$C2 = C1 \times Min\__{th}$$

$$P_b = C1 \times Avg - C2$$

By interpolating the effect of Pkt_Ratio of (5.1) it becomes;

$$P_b = (Pkt\_Ratio) \times (C1 \times Avg - C2)$$

Calculating the above equation encounters with very small floating point numbers. Because considering the expected range of constants $Max\__{P}$, $Max\__{th}$, and $Min\__{th}$ makes value for C1 as little as 2^{-20}. Although this is appropriate method suggested by original RED, considering 20 bits just for floating point and reserving some extra places for total calculation, it demands large hardware. It is considerably more difficult when the effect of variety in packet size is interpolated and it can add about 6-bits to floating point width.

We have proposed a more effective method to address this problem. As mentioned in [6] for RED implementation, all results of multiplications, divisions and powers are approximated to their closest power of “2”. Then, all operations are replaced by shift instructions. Instead, we compromise with a maximum error of 25% in results of calculations. Reason being, in worst-case scenario in related range, the calculated result is something between the maximum and half the maximum values. Therefore, the concept of binary based approximation is based on “catch on the closest power of 2” which is given in original RED implementation [6]. This idea could be illustrated in Figure-5.26. This method is used several times in our design implementation and is used as reference to all of our approximations in the design. The illustration shows how the distributed areas like A, B, C, etc. are assigned to their nearest power of two. The power of “n” could be either positive or negative.

In order to produce the result of R/P_b, consider the uniform average distribution part of equation (5.1) which is over [Min_{th} , Max_{th}]. we break down the equation to three components. We call the Avg portion as “Avg_Ratio”.

$$P_b = (Max\_{p}) . [Avg\_Ratio] . (Pkt\_Ratio) \qquad (5.9)$$

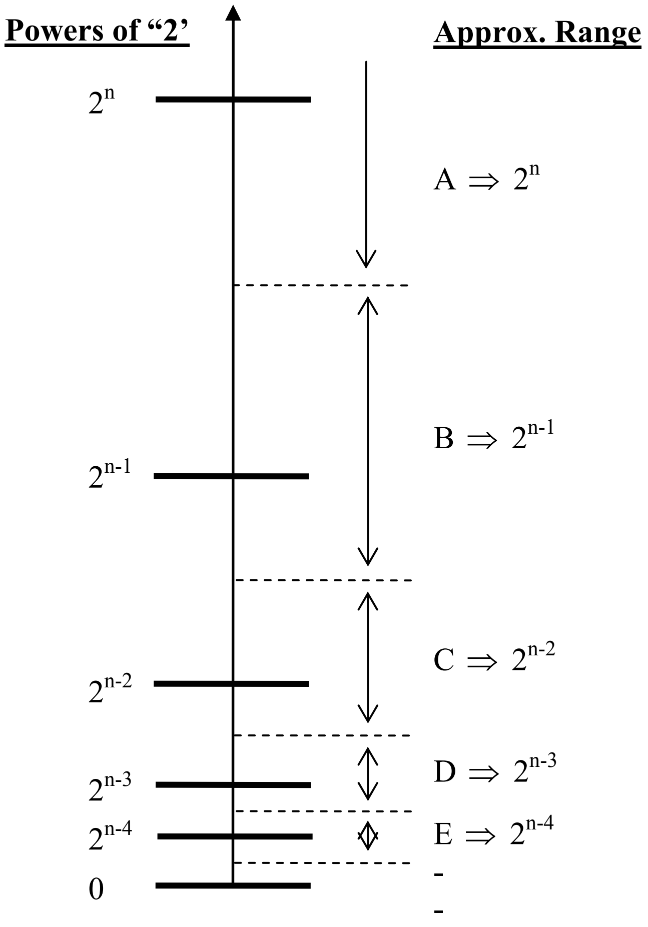


Figure 5.26 – **Binary based approximation determining assignment of range to areas of A, B, C, etc.**

To calculate P_b, we need to operate the binary approximation method on all three portions. Max_p is given in (5.8). Minimum value for Packet_Ratio according to (5.1) and (5.2) would be "64/1518 = 0.04216" (By the binary-based approximation, it is assigned to "2^{-4}"). And for the Avg Ratio, if we break the area between Min_th and Max_th as six times divide by two, then by binary-based approximation we have waived only less than 1% error just for approximating the smallest value in lowest area. And it's minimal versus 25% tolerance for higher area (like A or B in Figure-5.26). Then using the binary-based approximation (Figure-5.26), the range for each portion of (5.9) is determined as:

$$\text{Max\_p} = 1/32 = 2^{-5} \qquad (5.10)$$

$$\text{Avg\_Ratio} \in [2^{0}, 2^{-5}] \qquad (5.11)$$

$$\text{Pkt\_Ratio} \in [2^{0}, 2^{-4}] \qquad (5.12)$$

Then P_b in (5.9) would become

$$P_b \in [2^{-5}] . [2^{0}, 2^{-5}] . [2^{0}, 2^{-4}]$$

Or:

$$P_b \in [2^{-5}, 2^{-14}]$$

Finally we need the random number "R" previously produced by our random pattern generator to calculate the final result for R/p_b

$$R \in [0, 1]$$

$$R/p_b = R \;/\; [2^{-5}, 2^{-14}]$$

$$R/p_b = R \times [2^{+5}, 2^{+14}] \qquad (5.13)$$

Sum of the powers of “2” for equations (5.10), (5.11) and (5.12) which is considered probability p_b, could be easily produced using LUT based on binary-based approximation. The produced sum of powers of “2” which is an integer value between 5 and 14 will be used in (5.13). The key is, since “R/p_b“ is the result of the (R) multiplied by (14-bit p_b), then R/p_b could be easily obtained from the most 5 to 14 significant bits of a 14-bit random number “R” depending on integer value of p_b. Therefore this treatment has significantly simplified the implementation of these calculations, which have important role in dealing with the speed constraints. This is another facet of these thesis findings in terms of heuristic hardware minimization.

Calculating the average queue size (Avg) is done through following formula:

$$Avg \Leftarrow (1\text{-}w) \,.\, Avg + w.q$$

It could be arranged as below with respect to the old and new values:

$$Avg \Leftarrow (1\text{-}w) \,.\, Avg + w.q = Avg – w \,.\, Avg + w \,.\, q$$

$$Avg_{(new)} = Avg_{(old)} + w \,.\, (q_{(new)} - Avg_{(old)}) \qquad (5.14)$$

Required time to produce the $Avg_{(new)}$ in (5.14) based on stage-time (T) is first dependent on producing the $q_{(new)}$ which takes one stage-time “T” to compute.

Second, it depends on subtraction and shifting as it takes another stage-time "T" and finally in third "T" it will be added to the $Avg_{(old)}$.

$$T\ [Avg_{(new)}] = T\ [q_{(new)}] + T\ [\text{Sub \& Shift}] + T\ [\text{Add}] = 3\ T_{stage} \qquad (5.15)$$

Then, Avg takes 3 T_{stage} to produce. However, if we modify equation (5.14), we could substitute "q-old + new-Packet-size" with "q-new":

$$Avg_{(new)} = Avg_{(old)} + w\ .\ [\ (\text{new-packet-size} + q_{(old)}\ ) - Avg_{(old)}\ ]$$

$$Avg_{(new)} = Avg_{(old)} + w\ .\ [\ \text{new-packet-size} + (\ q_{(old)} - Avg_{(old)})\ ] \qquad (5.16)$$

$$T\ [Avg_{(new)}] = T\ [((q_{(old)} - Avg_{(old)}) + Pkt_{-Size)}\ \&\ \text{shift}] + T\ [\text{Add}] = 2\ T_{stage} \quad (5.17)$$

In (5.16), we do not have to wait for T_{stage} to calculate q_{new}. Instead we can produce the result from "($q_{(old)}$ - $Avg_{(old)}$)", since both parameters are already available. The new-packet-size is also available regardless of its size being of an arrived packet or a sent packet. That is why we have computed this two times in the implementation separately in order to handle arrival and/or sent packet sizes. Therefore, in first T_{stage} we could compute just the result of (Add & shift) and in the second T_{stage} we get the final result for Avg. Thus, comparing the (5.17) by 3-T_{stage} with (5.15) by 2-T_{stage} indicates ~~on~~ elimination of Avg calculation time from "3" T_{stage} to "2" T_{stage}.

5.4.4 – Implementation Block Diagrams

The block diagram of Figure-5.25 showed a general form of total design comprised of many smaller components. Each component or Small block generates calculation result or required information for other stage(s). Eight components are designed and concatenated together to execute our SODA_RED algorithm. However number of stages in one complete cycle in our design also consists of eight elements. However, there is no relation between number of these components and number of stages in one complete cycle because some components wait for more than one T_{stage} to complete the process. For example, Block (B-3) is the controller unit in design, and block (B-7) is counter controller which depends on past situation after last dropped packet and crucial in next drop decision. Figure-5.27 illustrates block (B-2) which contains block (B-1), because these two blocks are related in more common connections as shown. Block (B-1) contains two separated registers for PAS (Packet Arrival Size) and PSS (Packet sent Size). It could calculate the instantaneous queue size in one stage (S_1) by any arrived packet size whether it is informing of arrival or sending. It is designed to handle arriving and sending packets in the event they both happen at the same time. In stage (S_0) the system is waiting for arrival of the packet information vectors and signals. Block (B-2) uses the new calculated queue to produce the new average queue size (Avg) after every packet is arrived or sent. It executes the processes of equation (5.16) to calculate the Avg of LPF (RED) before the end of second stage (S_2). At the same time, the SODA unit has produced the Avg(SODA) ~~too~~ which depends on the condition where appropriate Avg($_{new}$) will transfer to the Avg-Register at the end of (S_2).

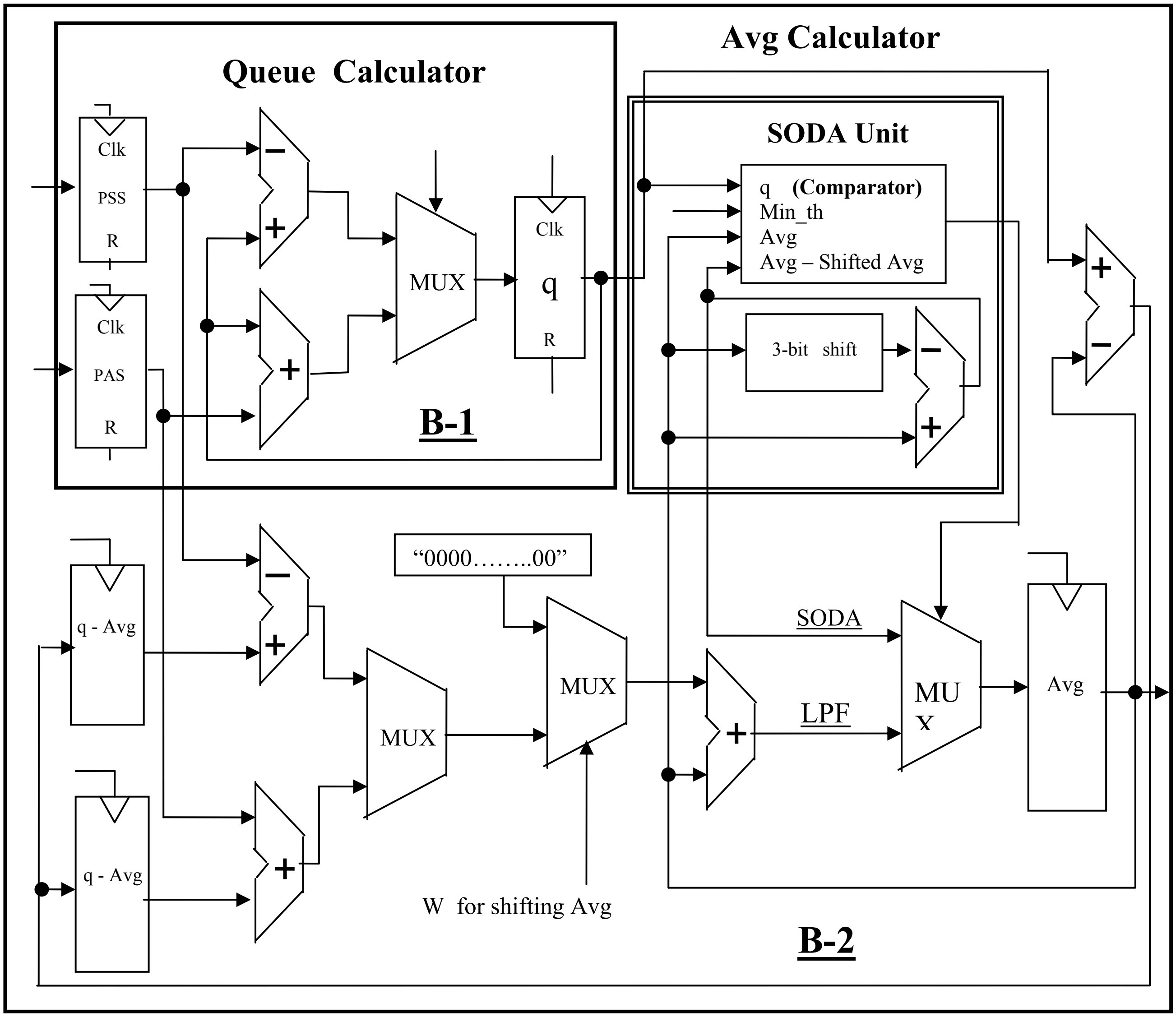


Figure 5.27 – Block diagrams for queue calculator (B-1) and Average calculator (B-2) for SODA_RED algorithm to implement.

Blocks (B-4) and (B-5) are shown in Figure-5.28. (B-4) is a mix of LFSR and Cellular Automaton random pattern generators as described in 5.4.2. And block (B-5) illustrates important part of the design which is described in detail in section 5.4.3. As described, it demonstrates how effortlessly and quick one complex part of the design is implemented to produce the calculation for R/Pb.

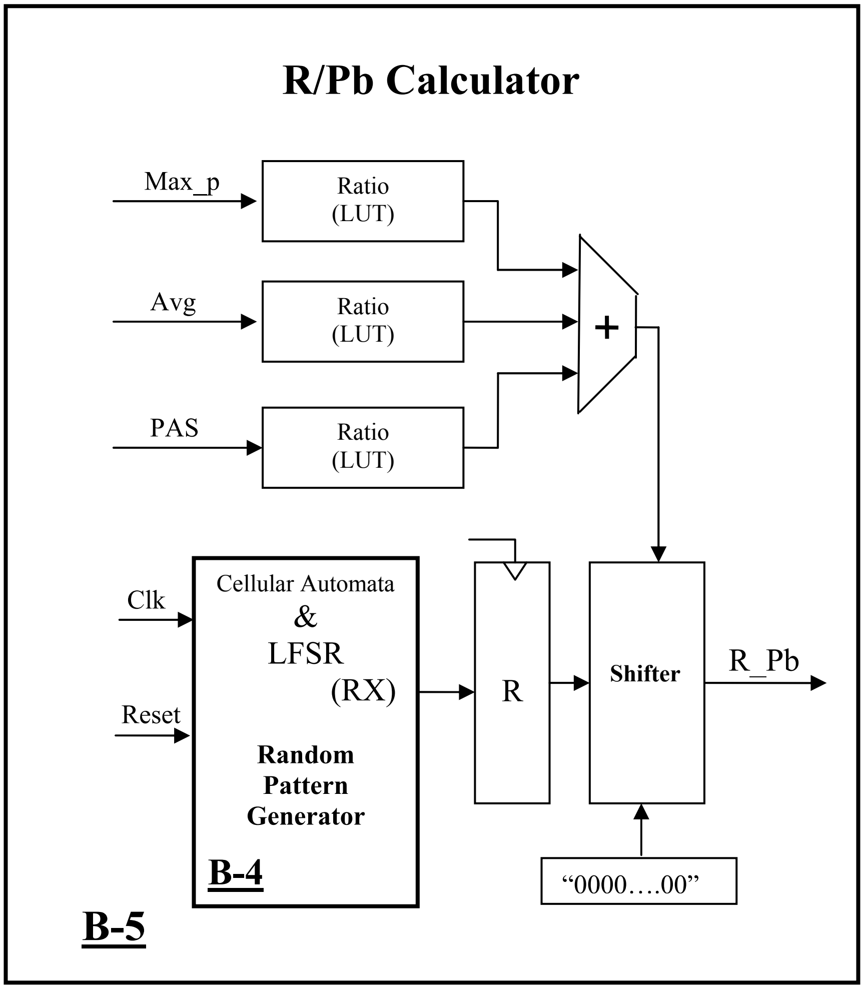


Figure 5.28 -- **Block diagrams for R_Pb calculator (B-5) and Random number generator (B-4) for SODA_RED algorithm to implement.**

Finally, Figure 5.29 shows the blocks (B-6) and (B-8) which gathers all produced results through the corresponded comparators to make the final decision for arrived packet. All these processes are done in only 6 stages starting from the beginning of (S_1) to end of (S_6), and it gives the chance as well to upgrade the speed through appropriate modifications. Final Drop decision accompanying with its Valid-Drop signal is issued with start of the last stage (S_7). Therefore, we have allocated large enough window for interfacing the external corresponding devices to safely hand shake with arrival information or to access issued results. This is possible since entire time for (S_0) is free to arrival of input signals, and the entire stage (S_7) is standby for sampling the result by the external device.

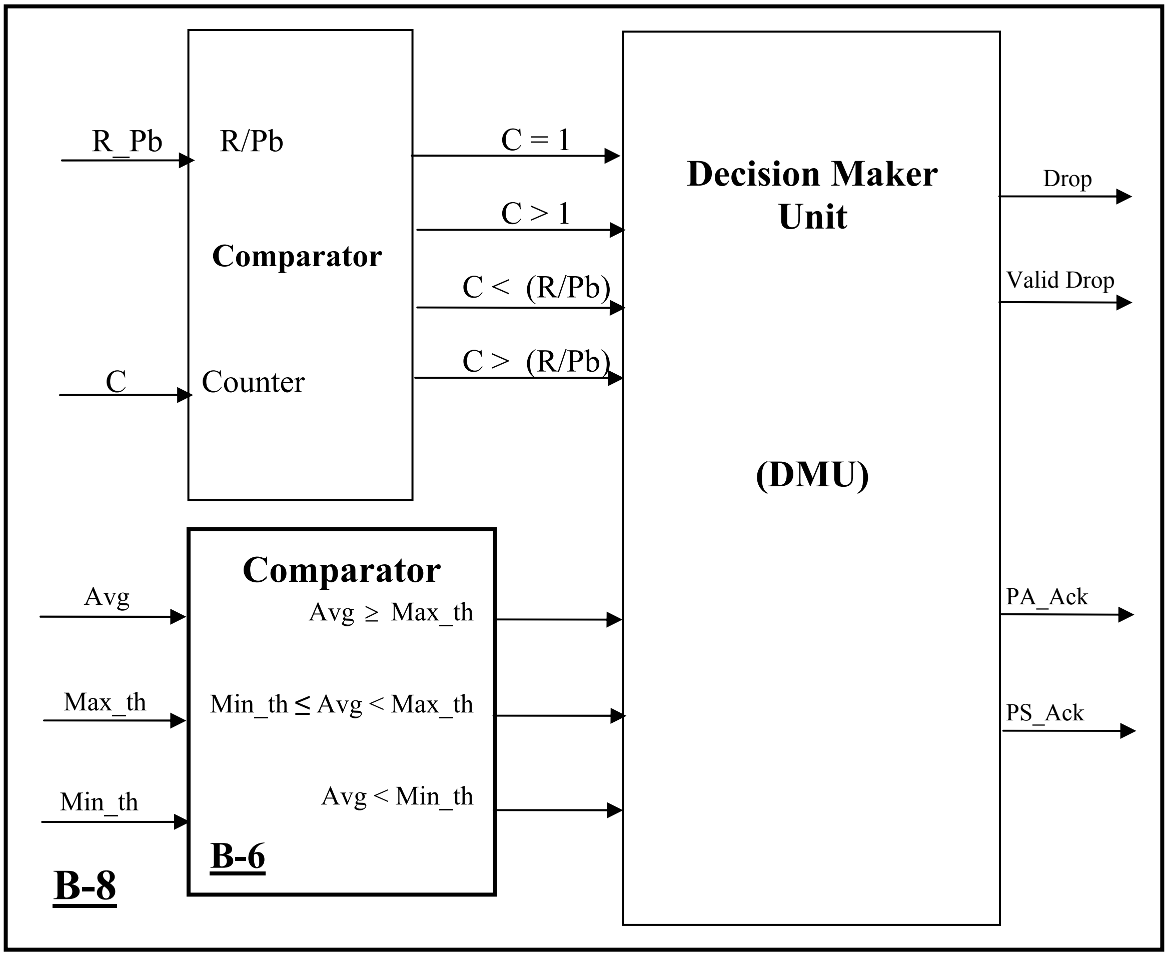


Figure 5.29 -- **Block diagrams for Comparator (B-6) and Decision Maker Unit (B-8) for SODA_RED algorithm to implement.**

5.4.5 – Synthesis reports

As mentioned earlier, FPGA implementation has been aimed at Xilinx device. VIRTEX-II PRO family devices are well known high performance FPGA devices especially in terms of less routing delay. The XC2VP30 is chosen to download the final bit-stream file has well satisfied the speed requirements. Table 5.5 contains final timing summary of synthesis report obtained by XILINX-ISE Navigator.

Table 5.5 – Timing Summary of synthesis report synthesized by Xilinx-ISE Project Navigator.

Xilinx Virtex II-PRO XC2VP30	Synthesis Report	Expected
Min. Clock Period (T)	6.216 ns	6.4 ns (Satisfied)
Min. Cycle Period (T_{cycle})	49.728 ns	51.2 ns (Satisfied)
Max. Clock Frequency	160.875 MHz	156.25 MHz (Satisfied)

Information about the area usage of the FPGA by this implementation is given in Table-5.6. Although the gate-count usage for our implementation is very trivial versus very large available area in XC2VP30 device, its advanced performance is the major reason for embedding our high speed design.

Table 5.6 – HDL Synthesis report synthesized by Xilinx-ISE Project Navigator, final summary report of area usage.

Xilinx Virtex II-PRO XC2VP30	**area of used Units**	**Usage**
Slices	256 out of (13696)	13%
Slice FFs	188 out of (27392)	< 1%
4-input LUTs	456 out of (27392)	< 2%
IOBs	78 out of (416)	18%
GClks	4 out of (16)	25%
CLKDLL	1 out of (16)	6%
IOs	78 out of (676)	11%
Gate count	~ 25000 out of (3000K)	1%

Chapter Six

Conclusion

Conclusion

This thesis has addressed congestion control router concerns by applying two approaches. First is investigation of congestion control and obtaining results among several commonly used standard congestion avoidance mechanisms. In this approach, the (RED) Random Early Detection congestion control method was found to be the algorithm of choice for this research. Second, in the research, in depth analysis of RED algorithm has yielded opportunities for improvements. Several significant enhancements were presented on RED algorithm, and improved algorithm with detailed calculations and higher throughput was proposed more which targeted higher speed networks with a line of 10 Gbps. Finally, presented algorithm in this thesis is implemented and validated through Xilinx FPGAs. This research demonstrates advantages of proposed enhanced RED algorithm. This is shown through improved high speed gateway efficiency response, reduced Drops, distribution of long term congestions to several shorter term congestions among all sources, less risk of global synchronization, and reduced packet queue time.

References

[1] Virtual Socket Interface Alliance- VSIA. Web Page: Fact Sheet.

[2] Van Jacobson, “Notes on using RED for Queue Management and Congestion Avoidance”, Network Research Group, Berkeley National Laboratory, Berkeley, CA 94720, NANOG 13, Dearborn, MI, van@ee.lbl.gov, June 8, 1998

[3] Khalid Ahmed, “Source book of ATM and IP internetworking”, IEEE press, ISBN: 0-471-20815-9.

[4] Crain Matsumoto, and Rick Merritt. “Analysis: FPGAs muscle in on ASICs”, embedded turf. *EE Times*, July 2000.

[5] Hashem, E., “Analysis of random drop for gateway congestion control”, *Report LCS TR-465*, Laboratory for Computer Science, MIT, Cambridge, MA, 1989, p.103.

[6] Sally Floyd and Van Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, Lawrence Berkeley Laboratory, University of California, floyd@ee.lbl.gov, van@ee.lbl.gov, To appear in the August 1993 IEEE/ACM Transactions on Networking

[7] Hairong Sun_ Xinyu Zang and Kishor S_ Trivedi, fhairong_ xzang_ kst_ee_duke_edug, “A Performance Model of Partial Packet Discard and Early Packet Discard Schemes in ATM Switches”, Center for Advanced Computing and Communications, Department of Electrical and Computer Engineering, Duke University, Durham_ NC 27708

[8] Justin Kutticherry U69-69-8804, Palak Patel U48-47-3644, “Random Early Detection”, SC 546 Fall 2001 Project

[9] A. Romanow, S. Floyd, “Dynamics of TCP Traffic over ATM Networks,“IEEE J –SAC. pp. 633-641, May 1995.

[10] Peterson, L. and Davie, B., Morgan Kaufmann “Congestion Control and Resource Allocation Lecture material” taken from “Computer Networks *A Systems Approach*”, Presented by Bob Kinicki, Third Ed. , 2003.

[11] RFC: 791, “INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION”, prepared for Defense Advanced Research Projects Agency, Information Processing Techniques Office,

1400 Wilson Boulevard, Arlington, Virginia 22209, By Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, September 1981

[12] Zhang, L., "A new architecture for packet switching network protocols", MIT/LCS /TR-455, laboratory for computer science, Massachusetts institute of technology, august 1989.

[13] Virtex-II PRO Platform FPGA Handbook, October 2002, Xilinx.

[14] Scott Karlin and Larry Peterson, “Maximum Packet Rates for Full-Duplex Ethernet”, Technical Report TR_645-02, Department of Computer Science, Princenton University, February 14,2002,

[15] Bing Zheng and Mohammed Atiquzzaman, “Low Pass Filter/Over Drop Avoidance (LPF/ODA): Int. J. Commun. Syst. 2002; 15:899-906 (DOI: 10.1002 / dac.571), School of Computer Science, Univ. of Oklahoma, Norman, OK 73019, USA. 18, Oct, 2002.

[16] Katarzina Radeka, and Zeljko Zilic, “Verification by Error Modeling Using Testing Techniques in Hardware Verification”, Kluwer Academic Publishers, Boston, Concordia and Mc Gill Universities, Canada, 2003.

[17] Allison Mankin, “Random Drop Congestion Control”, The MITRE Corporation, 7525 Colshhe Drive, McLean, VA 22102, mankin@gateway.mitre.org

[18] A. Mankin, MITRE, K. Ramakrishnan, “RFC (Request for Comments): 1254”, Networking Group, Digital Equipment Corporation Editors, August 1991