DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SIMULATION AND COMPARISON OF A NEW NETWROK PROTOCOL

by

Bayisa Kune MAMADE

September, 2015

İZMİR

SIMULATION AND COMPARISON OF A NEW NETWORK PROTOCOL

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University in
Partial Fulfillment of the Requirements for the Degree of Master of Science in
Computer Engineering Program

by
Bayisa Kune MAMADE

September, 2015
İZMİR

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"SIMULATION AND COMPARISON OF A NEW NETWORK PROTOCOL"** completed by **BAYISA KUNE MAMADE** under supervision of **DR. M. KEMAL ŞİŞ** and we certify that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

.....

Dr. M. Kemal ŞİŞ

Supervisor

.....

Prof. Dr. Haldun Karaca

(Jury member)

.....

Prof. Dr. Süleyman Sevinç

(Jury member)

Prof. Dr. Ayşe OKUR

Director

Graduate school of Natural and Applied Sciences

ACKNOWLEDGMENTS

I am grateful to Dr. M. Kemal ŞİŞ, my research supervisor, for his valuable and constructive suggestions during the planning and development of this research work.

I thank God for all the providences during my stay in Turkey.

Bayisa Kune MAMADE

SIMULATION AND COMPARISON OF A NEW NETWORK PROTOCOL

ABSTRACT

The active queue management (AQM) algorithms in use today are found to be vulnerable to a type of denial of service attack called low-rate denial of service (LDoS) attack which degrades TCP connection throughput. Robust Random Early Detection (RRED), an AQ M algorithm, is designed to filter out these attacking packets before they are fed to the RED. Performance of this algorithm is evaluated and compared with other active queue management algorithm during the presence of LDoS attack. However, the analysis only focused on average throughput of TCP during the attack period. In this work, we adopted the detection and filtering part from RRED and integrated it to our Orange AQM and expanded the analysis to instantaneous and average TCP throughput; used sensitivity, specificity and accuracy measures to determine the performance of the filter. In all the cases, the result of the analysis shows that Robust Orange algorithm performs better than RRED in terms of TCP throughput; Robust Orange is more sensitive in filtering out attack packets than RRED and Robust Orange is better in receiving normal user's packet than RRED. The accuracy measure of Robust Orange is better than that of RRED.

Keywords: Active queue management, sensitivity, specificity, accuracy, network attack

YENİ BİR AĞ PROTOKOLÜNÜN SİMULASYONU VE KARŞILAŞTIRMASI

ÖZ

Kullanım bugün aktif kuyruk yönetimi (AQM) algoritmaları TCP çıkışını alçaltır hizmet saldırı düşük oranlı reddi olarak adlandırılan hizmet reddi saldırısı bir tür karşı savunmasız olduğu bulunmuştur. Sağlam Rastgele Erken Teşhis (WRED), onlar RED beslemek önce bu saldıran paketleri filtrelemek için tasarlanmış bir AQM algoritması. Bu algoritma performansı değerlendirilerek yerel durum yoğunluklarındaki saldırı varlığı sırasında diğer etkin sıra yönetimi algoritması ile karşılaştırılır. Ancak, analiz sadece saldırı döneminde TCP ortalama verimi üzerinde duruldu. Bu çalışmada, RED'i gelen algılama ve filtreleme parçası benimsemiş ve bizim Portakal AQM entegre ve anlık ve ortalama TCP throughput analiz genişletilmiş; Kullanılan duyarlılık, özgüllük ve doğruluk filtresi derecesini ölçmek için. Her durumda, analizin sonucu Sağlam Portakal algoritması TCP throughput açısından RED daha yüksek performans gösterir; Robust Orange RED daha saldırı paketi filtreleyerek daha hassastır ve Sağlam Turuncu RRED daha normal bir kullanıcının paket alma iyidir. Sağlam Portakal filtreleme saldırı paket derecesinin ölçülmesi daha doğrudur.

Anahtar kelimeler: Aktif kuyruk yönetimi, duyarlılık, seçicilik, doğruluk, ağ saldırısı

CONTENTS

Pages

LIST OF FIGURES

Pages

LIST OF TABLES

Pages

CHAPTER ONE
INTRODUCTION

1.1 Evolution of Internet

As noted in (Leiner, 2009) shift from circuit switching to packet switching research, collaboration of government, research institution and industry in the evolution and deployment of Internet technology has brought exciting and drastic changes to the way day-to-day works are carried out. Bringing people together irrespective of distance, as a collaborative medium, dissemination of information and the largest and easy broadcasting capabilities are few of the advantages to mention. Prior to these capabilities, invention of core technology of communication among which telegraph, telephone, radio and computer has facilitated the discovery of unsought idea about integrating them which has unleashed the tremendous today's inter-connected computer and related devices world-wide.

After becoming the first head of computer research program at DARPA, in October 1962, Licklider had written memos on things that could be done through networking of computer. His insight was to interconnect computers globally and share programs and data in a quick and easy way. Publication of the first of its kind, a paper on packet switching theory in July 1961 and a book in 1964 Kleinrock persuaded theoretical feasibility of communications using packets rather than circuits; a stride toward computer networking. On the attempt to make computer to talk to each other, TX-2 and Q-32 computers were connected using low speed dial-up telephone line making it the first wide area network of computer (Marill, & Roberts, 1966). As a result, two very important and eye opening facts were discovered. The first being, the possibility of running programs, retrieving data as needed on remote computer from time-sharing computers and the second was the circuit switched telephone system was incapable; which gave proof for the new technology envisioned; packet switching (Marill, & Roberts, 1966). Robets, in late 1966 went to DARPA with a plan to develop computer network concept, called ARPANET (Roberts, 1967). At end of 1969, construction of Internet was straight up

on the ground on its own, with four host computers connected together to ARPANET.

1.2 Development of Protocol

Even though it was not a novel idea to work on packet switching network to share expensive and scarce network resources as indicated in (Cerf and Kahn, 1974), there were several works on designing and implementing protocols that solves problems of sharing resources within the same network (Despres, 1972; Kahn, Robert, & William, 1972; Carr, Crocker, & Cerf 1970). Here the researchers focus on sharing of resources which resides on among several distinct and different packet switching networks. Fundamental issues concerning interconnection of different packet switching networks have been discussed; and simple, however, very powerful and flexible protocol quenching network packet size, failures in transmission, sequencing related problems, flow control and lastly not the least creation and destruction of process to process were discussed. Specifically hosts of differing capacity can support the implementation of the proposed protocol.

As (McKenzie, 2012; Carr, Crocker, & Cerf, 1970), indicated Network Control Program (NCP), the code that had to be added to the operating system (OS) with each host to enable it to interact with its Interface Message Processor (IMP) and manage multiple connections, was developed during the years 1969, 1970, 1971 and 1972 and was officially declared to ARPANET protocol. A small general purpose computer called Interface Message Processors (IMPs) were used to route messages, check errors in the transmission medium, and provide asynchronous digital interface to the main computer as noted in (Heart, Kahn, Ornstein, Crowther, & Walden, 1970). The NCP did not have any error checking mechanism for control of errors. It was not enough to specify meaningful dialog between dissimilar hosts. The upgrade to facilitate such host to host communication was initiated and the responsibility to do this was given to Network Working Group (NWG). They developed a layered approach Table 1.1 to combat the problem. The primary function of NCP is to establish connections, break connections and control data flow over the connections, interrupt transmission and respond to test questions.

Table 1.1 Layered protocol approach

No.	Protocol	Usage
1	Host/IMP	Physical and logical message formatter specification
2	Host/Host	Establish, manage and interrupt communication path and used by all higher level processes
3	Initial Connection Protocol (ICP)	To gain simultaneous access to some hosts
4	Telecommunication network (TELENET)	Mapping arbitrary keyboard, printer to network virtual terminal. Uses ICP
5	Data Transfer (DTP)	Formatting data to be sent through network
6	File Transfer (FTP)	Specifies, reading, writing and update data stored remotely. Uses ICP
7	Graphics protocol (GP)	Specifying graphics exchange and display information
8	Remote Job Services (RJSP)	Submitting input, receiving output, and exercising control over Host

Network Working Group (NWG) built the first host to host protocol for the ARPANET in 1970, naming their product Network Control Protocol (NCP). Immediately application programs were developed by the network users. The first application being electronic mail, initiated by the ARPANET developers to facilitate coordination was introduced during 1972.

Referencing processes in another Host, each host with the ability to map internal process identifies into its portion of this name space. The name space is known as sockets. It forms one end of a connection and fully represented by a pair of sockets.

The idea of open architecture networking, pioneered first by Kahn in 1972 at DARPA, was originally part of packet radio network program. In open architecture networking, choosing any individual network technology was not determined by

particular network architecture rather freely determined by provider and make sure it interwork with other networks through a meta-data level. Due to some technical incapability of NCP with packet radio network, for instance, addressing machines further downstream than a destination IMP on the ARPANET, and the absence of end-to-end host error control which may completely halt the operation of ARPANET, Kahn came up with new version of the protocol assuring the demand of open-architecture network. Having listed points that are critical to his thinking of this new communication protocol, he delved into understanding more about different operating system so as to help him in implementing or incorporating the protocol into the systems software. Cerf's prior knowledge in the development and design of NCP and know-how on interfacing operating systems with Khan helped to distill the details of what would become TCP/IP.

It was also presented in (Cerf, Dalal, & Sunshine, 1974), Cerf and Kahn published a paper on a protocol know as TCP. It was supposed to provide all the transport services such as to reliable sequenced delivery to a datagram services and forwarding services in the Internet. The first implementation effort produced a version that could allow only virtual circuits-reliable sequence delivery, which work very good for file transfer and remote login applications. Other advanced application suffered from the packet loss particularly packet voice, implying reorganizing of the TCP to support these type of applications. But the need to transfer voice over the network required rethinking of the TCP design and Danny Cohen argued retransmitting voice packet end to end was not useful and the network may suffer the worst delay for the retransmission. In addition to the above mentioned problem, faults in the design of the first TCP by Bob Khan and Cerf V. were discovered by Ray Tomlison. The start and the end to the connections were random exchange of packets. He proposed the three-way handshake in order to identify the start of a new TCP connection. The original TCP was separated into two protocols. The Internet Protocol (IP), for the function of addressing and forwarding of individual packets through the network; and the Transmission Control Protocol (TCP), part to deal with services functionalities such as flow control and recovery from lost packets. An alternative protocol was also designed and developed for those applications that do not need the

services of TCP, called User Datagram Protocol (UDP). The transition from NCP to TCP/IP has helped ARPANET to be split into two networks called MILNET and ARPANET. By the year 1985, large communities of researchers, developers and other communities has begun to use Internet for day to day work. It served as a core communication protocol until 1983 and was superseded by TCP/IP.

Internally, the network was used to share software resources among which MATLAB, Natural Language Processor, LEAP, LC and ALGOL were some. The second was accessing the Network Information Center (NIC), which was an information repository on all systems connected to the network. It also provided file space and system accessing and updating the dynamics of the information. The last thing done on the network was to measure and experiment on the network itself. At the same time the external ARPA-funded research communities were given privilege to use the ARPANET for the sharing of data management system or retrieval system, giving birth to distributed database system. Table 1.2 displays the information in tabular form.

Table 1.2 ARPANET usages

No	Internal	External
1	Sharing of software resources such as MATLAB, LEAP, etc	Sharing of data management or retrieval system.(distributed database to be accessed by all the communities)
2	Access Network Information Center's (NIC) repository	
3	Provide file space and system accessing privilege	
4	Updating the dynamics of Information	
5	Measuring and evaluating the network itself	
6	email	

Externally, continental connections have been made between countries such as England, Norway, France, etc.

In addition to the layered approach developed by NWG, in the paper by (Crocker, Heafner, Metcalfe, & Postel, 1972), categories of layered protocols in the ARPANET were depicted as high and low level. This grouping were based on function and thus called function oriented protocol. The high level protocols are most closely matched to functions, related to substantive use and states about the meaning and timing; while the low level protocols deal with communication devices.

1.3 Flow Control in NCP

It is stated in (Kahn, & Crowther, 1972; McKenzie, 2012) establishment of connection facilitates sending and receiving of messages. Network Control Program (NCP), at the receiving end accepts message from IMP and queues to apply required function processes. To quench the rate of flow from the sender, host, which may be faster than the accepting rate of the receiving host, flow control mechanisms are required. At the same time the receiving host is required to specify buffer space for each connection and informs the sender the available room. Thus, the sender never transfers more data than the receiver can accept.

1.4 TCP and Congestion Control

The problems existed in the NCP: inability to address downstream after IMP, lack of end-to-end error checking (control), and incapacity to communicate meaningfully between dissimilar hosts' triggered research in more comprehensive, and robust layered protocol approach which were used in TCP/IP design and development.

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks (Postel, 1981). TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered

hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks. TCP is based on concepts first described in (Cerf, & Icahn, 2005). The TCP fits into layered protocol architecture just above a basic Internet Protocol as in (Postel, 1981) which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartimentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

The existence of network congestion was recognized as a problem of complex networks as stated in (Jacobson, 1995; and Stevens, Allman, & Paxson, 1999). Before the transfer from NCP to TCP, congestions at the routers, the most vulnerable devices on the net, were managed by IMP. IMP in the ARPANET had uniform bandwidth and same switching modes with substantial access capacity. As network expands, the benign behavior of IMP systems needed improved congestion control mechanism to handle overloaded traffic flows.

The first experiences of Internet meltdown or collapse were reported in the mid-1980s (Braden, Clark, Crowcroft, Davie, Deering, Estrin, et al., 1998). It initiated curiosity of researchers and they came up with the addition of congestion avoidance algorithms to the TCP, which informs TCP flows to reduce their sending rate in case congestions are detected through packets loss signal. The pioneer and prominent

researcher of the time (Jacobson, 1988), proposed four algorithms to solve the issue, which has become the base for TCP congestion control in the current implementation. These are slow start, congestion avoidance, fast retransmit and fast recovery. This newly added algorithm on top of TCP has been credited for the success of the Internet. End to end congestion control determines the stability of the Internet. However, many new applications do not use TCP as a transfer protocol or any TCP friendly algorithms. Three advantages are identified by using end to end congestion control. Prevention of Internet collapse, fair bandwidth sharing by flows for the links that are congested, and performance metrics optimization of a flow.

The collapse of Internet in the mid-1980s could be caused by two events: unnecessary retransmission of packets by TCP connections, which may have been in the transit or has been receipt but not acknowledged. This case is called classical congestion collapse. The fix was through the improvement in the TCP timing and congestion control mechanisms in the modern TCP implementation. The second cause is undelivered packets; wasting bandwidth to transfer packets to be dropped before arriving to the destination. This problem is still unresolved and may cause Internet collapse in future. The end nodes effective end to end implementation of congestion control, end to end guarantees of bandwidth and disfavoring aggressive algorithms by the application developers could solve the concern of today's Internet meltdown.

Indeed very important and powerful technique to avert the congestion collapse problem; but it was not enough and sufficient for all circumstances. Its limit was controlling network from the edge. It was hard and almost impossible to control congestion from far end which has happened on the intermediate devices such as routers. Router based congestion control mechanism ought to be implemented to get rid of such problems. Routers use two closely related but rendering different performance issues. These are queue management to manage packet queue length by dropping them when appropriate and scheduling algorithm to determine which packet to send next; with the main purpose to manage allocation of bandwidth among flows.

1.5 Active Queue Management

The old mechanism which has served on the Internet well for years has brought two very serious demerits known as full queue and lock-out. It was implemented by fixing the maximum queue length of packets on routers when configuration is done. When the limit is reached, the fate for the newly incoming packets was to be rejected inseparably until free spaces is available on the buffer. It is called DropTail.

In addition to the droptail, drop-front when full and random drop on full disciplines are used to solve limitations mentioned in droptail mechanism. Particularly drop front when full solves the lock-out caused by the drop tail; but queue fullness remained unsolved. An active queue management, proactive approach, implemented on routers drops packets in a queue before it becomes full. This informs TCP and TCP friendly flows to reduce transfer rate. Thus the full queue problem is solved through active queue management (AQM). Implementing AQM provides the following advantages: reducing number of packets dropped in routers, providing lower-delay interactive service, higher link utilization, correction of bias against burst flows and lock-out behavior.

With the purpose to improve TCP performance, numerous active queue management algorithms have been designed and implemented for congestion management (Floyd, & Jacobson, 1993; Mahajan, Floyd, & Wetherall, 2001; Feng, Kandlur, Saha, & Shin, 2001; Kunniyur, & Srikant, 2004). These algorithms' robustness in different network conditions were tested and proven to be worth implementing them in real network. However good they were, new attacks against network have degraded their services. Identifying network attacks such as DoS and DDoS are relatively easy because of the sledge hammer nature of their traffic (Chang, 2002). As noted in (Zhang, Yin, Cai, & Chen, 2010; Kuzmanovic, & Knightly, 2006 and Guirguis, Bestavros, & Matta, 2004), new kind of denial of service attack called low-rate denial of service (LDoS) attack has been distinguished to be potential attack weapon to the performance of TCP through the exploitation of inherent TCP's retransmission time out (RTO) mechanism. Forcing TCP flow to

enter RTO state using high-rate data, short duration and re-do it periodically. Thus the TCP throughput decreases at the receiver node while identifying the attackers are difficult because their average rate remain small. RED and RED-like algorithms have been found vulnerable to LDoS attacks.

An efficient algorithm has been developed to alleviate LDoS attack as indicated in (Zhang, Yin, Cai, & Chen, 2010). The purpose of our study is to apply LDoS detection and filtering mechanism mentioned in Zhang to Orange active queue management model which is developed by (Çatalkaya, & Şiş, 2010). Motive: TCP targeted DoS attacks are easy to launch remotely, stealthy, may be severely catastrophic, and can cause traffic oscillations.

The thesis is organized as follows: chapter 2 discuss thorough on related works, chapter 3 presents Robust Orange AQM, compares it with Orange, RRED and RED. Chapter 4 analyze TCP throughput with that of RRED. Finally chapter 5 concludes the thesis and proposes future work.

1.6 Random Early Detection (RED)

An active queue management called RED was first proposed by Floyd and Jacobson. It is developed to overcome congestion through continuous controlling of the average queue size. It is very sensitive to its parameters like min threshold and maximum threshold. All incoming packets are allowed to enter transmission if the average queue size is less than the min threshold. On the other hand, when the average queue size is greater than the maximum threshold, all the incoming packets are dropped. If the average queue size is between min and max threshold, packets are dropped with probability increasing as it approaches the max threshold. Global synchronization does not occur when using RED as active queue management (Viradiya, Vaghela, & Dhangar, 2014). The main disadvantage of RED is the number of parameters used which does not work uniformly for all the systems. In addition, the effectiveness of early detection depends heavily on the rate at which congestion notification is provided to the sources (Feng, Kandlur, Saha, & Shin,

1999). The probability of notifying by the routers in the situation of congestion to reduce its windows is roughly proportional to the connection's bandwidth share. Average queue size is kept low to allow occasional bursts of packets in the queue.

CHAPTER TWO
RELATED WORK

2.1 Introduction

Network attacks are set of malicious activities to disrupt, degrade, deny, or destroy information and service residing on networked computers. Executed through streams of data networks, its purpose is to compromise integrity, confidentiality or availability of network systems' resources. The attacks can be viruses as an attachment, reconnaissance, denial of services, Internet worms and unauthorized usage of the system resource according to (Ghorbani, Lu, & Tavallaee, 2010). Any intended attempt to prevent legitimate users of a network from particular network resource is termed as denial of service (DoS) attack as described in (Luo, & Chang, 2005). The first description of DoS attack was given in operating system environment in 1983. In the study by (Abliz, 2011), Denials of service (DoS) attacks are types of attack that have directed their goal at blocking availability of computer systems or services. One of the core concepts of network security is availability. Availability of network resources can be defined as the ability to use the desired information or resources in a reliable and timely manner. The attack can take one of the following forms: scares resource consumption, changing configuration information or destruction, and destruction or alteration of physical components of the network. The scares resources of a network are bandwidth, CPU, memory, I/O bandwidth, disk space or combination of the above.

According to (Ehlert, Geneiatakis, & Magedanz, 2010) study, the openness and scalability of IP and Internet design has contributed to the success of today's World Wide Web. However, its security mechanisms were not considered. Packets are sent end to end using the header information, IP, to the destination address by routers. Its authenticity is not checked and it created opportunity for the attackers to launch denial of service attack disguising their real identity.

Network attack in general and specifically DoS attack is poising serious problems to the Internet. Availability of Internet is under real threat from such categories of attacks. Discovery of a new type of DoS attack called low-level denial of service attack adds benzene on the fire already burning bushes. In the following sections we will discuss the LDoS attack mechanisms, detecting, and defense and assess the impact of such attack.

2.2 DoS Attack Launching Methods

Flooding attacks, sometimes called brute-force attacks, are type of attack that try to deny service to legitimate users of a service requested by sending to the victim traffic rate that exhausts one or more resources of the network. Such types of attacks are mostly done by recruiting compromised computers to mobilize traffic rate that can overwhelm the capacity of the victim. Even though it is not impossible to distinguish between DoS attack packets and the benign packets it is not easy to identify and prevent the attack. In (Maciá-Fernández, Díaz-Verdejo, & García-Teodoro, 2009), Denial of Service (DoS) vulnerability (semantic) attacks makes use of the flaws in a mechanism that enforces the policy to be implemented by the target systems and consume much of the resources of the target by sending carefully crafted request. DoS attack can make use of the combination of mechanisms mentioned to launch attack.

2.3 TCP Targeted Attack Category and Model

Reduction of Quality (RoQ) attack is a more general class of low-rate denial of service attack targeting TCP (Guirguis, Bestavros, & Matta, 2004). Arbitrary pulses of frequencies are sent by the attacker to exploit TCP's Additive Increase Multiplicative Decrease (AIMD) mechanism to damage TCP's goodput reduction. It can severely degrade service quality. The attack does not operate at a known frequency and this leads to the difficulty in detecting the attack. Such vulnerabilities are quantified through control-theoretic. From the results of the experiments it is

highly recommended that the needs for adaption mechanisms which are resilient to the attack are proposed.

The second one is LDoS attack, as noted in (Kuzmanovic, & Knightly, 2003); investigation into the low-rate denial of service attacks characteristics is conducted. Global network infrastructure is under threat from denial of service attacks ever before. The attacks emanate from the faulty assumptions of end system cooperation of TCP's congestion control algorithm leading to vulnerabilities. Because of the nature of the attacks, LDoS are difficult for both routers and counter-DoS system to detect them. Through the use of analytical modeling, simulations and experiments on the Internet it has been shown that TCP's retransmission time out mechanisms can dwindle flows of TCP when under LDoS attack to a small fraction of the ideal rate while disguising detection. It was found that low-rate DoS attacks are successful against both short and long lived TCP aggregates. Both network-router and end-point-based mechanisms can only mitigate, but cannot eliminate the effectiveness of the attack. They have demonstrated that TCP's retransmission timeout mechanism can be exploited by using maliciously chosen low-rate denial of service attack traffic to throttle it to small flows. The following are requirements for low-rate TCP targeted attack: Integer multiple of inter-burst period coinciding with minimum retransmission timeout value of TCP, attack peak traffic magnitude large enough to cause packet loss, and sufficiently long burst length to cause loss. Burst length needs to be longer than round-trip time (RTT) of TCP flows.

In LDoS attack attacker periodically sends short burst of packets to overflow a router's queue and cause packet loss of legitimate users. A well behaving TCP source will back off to recover from the congestion and retransmit only after one Retransmission Timeout (RTO). If the attacker congests the router again at the times of retransmission, little or no real user traffic can get through. Hence, by synchronizing the attack period to the RTO duration, the attacker can essentially shut off most legitimate TCP sources even though the long term rate of attack can be very low (Mathew, & Katkar, 2011). The objective of the low-rate attack is that for a short duration burst time (Tb), the attack packets will fill up the buffer of a victim

router with the high burst rate (Rb) so that packets of any TCP flows have to be discarded by the router. The packet loss will force most, if not all, TCP flows to enter the retransmission state. Also note that to be considered a low-rate TCP attack, the ratio of Rb/Ta has to be small (Sun, Lui, & Yau, 2004). Otherwise, system administrators can easily detect an attack by its high traffic volume. Figure 2.1 below shows the attack model. Attack category and vulnerability used by the attacker is shown in Table 2.1 below.

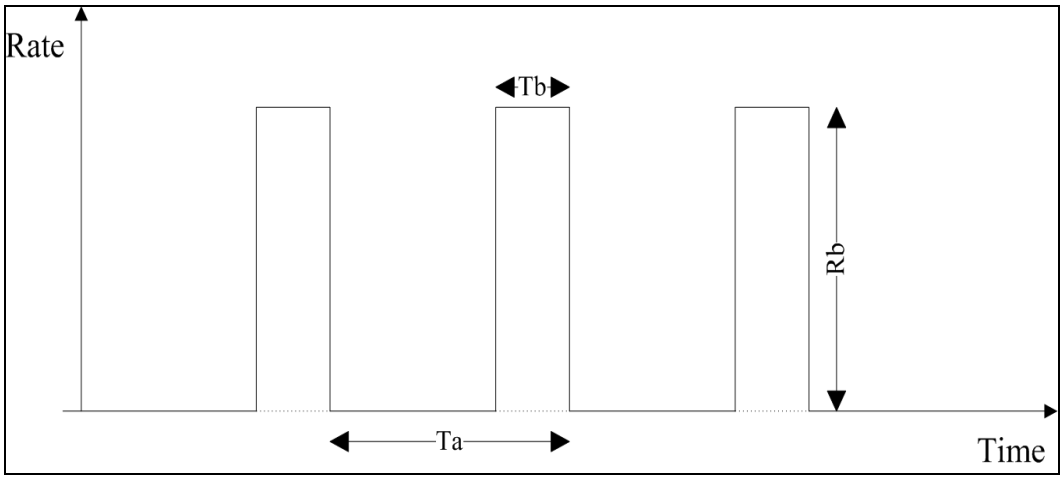

Figure 2.1 Low-rate denial of dervice attack model

Table 2.1 Attack category

Attack types	**Vulnerability used**
Reduction of Quality (RoQ)	Additive Increase Multiplicative Decrease (AIMD)
LDoS	TCP's retransmission timeout

2.4 AQM Integrated Detection and Filtering LDoS Attacks

Improving the throughput of TCP against low-rate denial of attack (LDoS) by using RED and RED-like active queue management algorithms are found to be exposed (Zhang, Yin, Cai, & Chen, 2010). The attackers exploit TCP retransmission time out (RTO) mechanism which forces the TCP to re-enter the RTO periodically. In effect it reduces TCP throughput without being detected. A novel approach to avert LDoS was to use the behavior of TCP after dropped packets signal which holds

back sending packets immediately and with the reduction in window size. Incoming packets within the short-time periods after the packet loss from same sender puts it under the suspicion of being attacker. Robust RED was proposed to improve TCP throughput through augmenting RED AQM algorithm with the detection and filtering algorithm before the incoming flows are supplied to the RED algorithm. If the incoming packets of any flow are suspected to be an attacking packet, it will be filtered or dropped before being fed to the RED. Robust RED is highly robust when routers are under LDoS attacks. It nearly fully preserved TCP throughput under LDoS attacks, outperforming existing RED-like active queue management algorithms with TCP traffic remaining at its ideal rate.

In (Shashidhara, & Balaji, 2014), a survey was conducted to find out existing active queue management algorithm that possess the ability to counter attack the threat poised to TCP throughput and web traffic by low-rate denial of service (LDoS) attacks. The seriousness of the problem is that such an attack known as LDoS are undetected by the already implemented AQM algorithms, disguising itself, degrades performances of web traffic and TCP Services in particular. Several AQM algorithms were examined and their performance against LDoS attack was evaluated. The research contributed that RED-PD, can only mitigate the effect of Denial of Service attack. However it fails to prevent LDoS attacks; while Robust RED (RRED) was not designed for distributed attack, it detects and prevents LDoS attacks. The use per-flow analysis used by RRED could increase the overhead for distributed attacks. It is exposed that various LDoS attacks detection and prevention algorithms are in use with differing degree of performance and various methods: for instances, partial and per-flow analysis method. Some are still in the laboratory. The limitation of the work is the discussion of the existing algorithms are very shallow, it fails to develop a new anti-LDoS attack algorithm as proposed.

2.5 Detection Mechanisms of LDoS Attack

The study in (Zhang, Mao & Wang, 2007) shows that most of the detection algorithms depend on signal analysis. Proposed algorithms are not sufficiently accurate and scalable for real networks deployment. Effective mitigation of such an attack is still unsolved problem. In the work by (Luo, & Chang, 2005), detection of timeout and AIMD attacks is based on an unusually high deviation in the incoming data traffic and radical decrease in the outgoing ACK traffic observed when attacks are executed. A wavelet transform is used to extract desired frequency components of the data and ACK traffic; then change points detection in the extracted components are carried out. The simulation and testbed experiment showed effectiveness of the mechanism and the low time computation complexity of both methods enables the scheme for online detection.

Even though shrew attacks are very low in volume, but periodical and stealth, they can cause damage to the victim servers without being detected for long. In (Luo, & Chang, 2005), they have proposed digital signal processing (DSP) approach to detect such an attack mixed among legitimate flows. Use of frequency-domain enables collaborative detection across multiple routers. It could be implemented on hardware. LocalCast – a network layer multicast protocol is developed, to support collaborative detection to ease the burdening on the end hosts. The application of autocorrelation-based statistical signal analysis helped in high accuracy of detecting shrew attacks. Shrew attacks are mostly distributed in the low frequency band in contrast to wideband distribution of legitimate traffic.

A novel approach to detect low-rate denial of services attack flows at edge routers was proposed. Traditional flood-based DoS attack solutions are not capable to solve this new type of attack called low rate TCP denial of service attack which exploit TCP retransmission time out (RTO) (Shevtekar, Anantharam, & Ansari, 2005). To curb this attack, two methods are designed. Flow with periodic pattern with burst length greater than or equal to RTTs of other connections with the same server is marked malicious. The second is flow with periodic pattern and time period equal to

fixed minimum RTO is also marked as malicious. Use of light weight data structure to store necessary flow history at edge routers rather than modifying TCP congestion control algorithms was contributed. Thus it was found that the proposed solution is able to detect patterned flows which satisfy the criteria with no changes to TCP.

In (Sun, Lui, & Yau, 2004), an approach of distributed detection mechanism which depend on dynamic time warping (DTW) method to robustly and accurately identify the existence of this sort of attack is considered. It is computationally efficient and robust techniques to compare similarity of template signature and input signal. It even does it effectively under conditions when input signals are subjected to changes in time scale and magnitude. As in the case of distributed DoS or intrusion detection, it cannot be performed at the victim site. This is due to the nature of the attack, which aims to intentionally throttle legitimate TCP flows destined for the destination server or hosts. Therefore, the attack aims to execute an attack on the upstream routers of the server to inflict damages to all the TCP flows passing through it. The detection mechanism employs matching current packet arrivals to the given attack pattern signatures and it is to be deployed on routers. The pattern matching detection consists of the following steps: statistical sampling of incoming traffic, noise filtering, feature extraction and signature matching at the last. If low-rate TCP attack is detected, the input port(s) should be determined by the routers. Detection will then be expanded to all the incoming ports of affected routers. Further detections to the upstream routers ports are also carried out to be closer to the source of the attack. If the attack is not found after push-back detection to the upstream, the attack is suspected to be distributed. The implementation of the mechanism requires routers in the network to collect periodic signature of attack flows. Table 2.2 below summarizes the attack and its detection mechanisms.

Table 2.2 Summary of attack type and detection mechanisms

No	Attack type	Detection mechanism
1	Timeout and AIMD	Wavelet transform for extracting desired frequency components
2	Shrew attack	Digital signal processing (DSP)
3	Low-rate DoS	Flow with periodic pattern burst length>=RTT of connection to the same server by the other flows. Flow with periodic pattern and time=RTO
		Uses dynamic time warping (DTW) for distributed detection, compares similarity of template signatures with input signal
		Use of per-flow arrival time of dropped and delivered packets

2.6 Impact of LDoS Attacks

The impact of low rate denial of service attack and the extent to which defense mechanisms are capable of mitigating them are studied as indicated in (Guirguis, Bestavros, & Matta, 2006). It was discovered that a new type of attacks having the capability to deny services to genuine users or impacting TCP flows significantly with very low attack traffic. To amend such hindrances, adopting of simple discrete-time model with a single TCP flow and non-oblivious adversary was proposed as a solution. The work has contributed discovery of new variants of low-rate denial of service attack that could potentially have high attack potency per attack burst. It was found that the upper bounds on the impact of low-rate denial of attacks are not real assessment under certain attack scenario.

Low-rate TCP targeted DoS attacks inflict severe impact on the Border Gateway Protocol (BGP) than on end hosts according to (Zhang, Mao, & Wang, 2007). The effect of the attack on BGP is devastating because they are global Internets' critical infrastructure for exchanging reachability information. It has been shown empirically that commercial routers are exposed to such an attack. It can remotely cause session resets and increased convergence delayed leading to global disturbance on routing

stability, and reachability problems of routing, and routing packet performance degradation. Lack of guaranteed bandwidth for routing traffic in today's routing protocol is the fundamental cause for the attack. Experiment on Internet and testbed are used to study the effect of such attack, feasibility of launching coordinated attack, and defense solutions by using existing router support. Protecting Internet infrastructure is very important; especially routing packets. The impact of this type of attack on inter-domain routing is severe. As indicated in (Kuzmanovic, & Knightly, 2006), even the router-supported mechanisms do not eliminate the attack impact without incurring excessively high false positives.

The effect of low-rate denial of service attack on TCP and all applications using TCP as a transport protocol will generally suffer much. During the attack period, losses to any TCP flow backs off the amount and time of data transfer leading to the TCP throughput almost nearing zero. As a result network resources are underutilized during the attack.

2.7 Defense against LDoS Attacks

In (Kuzmanovic, & Knightly, 2003), the only solution to the multiplicative increase of TCP's retransmission timeout behavior which LDoS attackers can easily tape in and wage war of sending high data rate in short time period and repeating it as to coincide with the RTO, is to randomize retransmission timeout. However, in the absence of an attack, randomization can degrade TCP connection performance as indicated in (Allman, & Paxson, 1999). In addition, the range of values for RTO randomization is limited to around 1 second; which shows that fully mitigating LDoS attack through randomizing alone is not enough. The defense method proposed by Kuzmanovic & Knightly was practically put to test by (Efstathopoulos, 2009). The evaluation result shows randomization hinders attacker from predicting the next RTO of TCP. However, in the study conducted by the same researchers (Kuzmanovic, & Knightly, 2006), it is stated that defending the system completely requires significant sacrifices to the systems performance in the absent of the attack. It is also noted that

inherent tradeoff between defense and attack time-scales are the vulnerability underlying; not retransmission timeout of TCP or poor design of DoS detection.

It will reduce the effect of the attack on throughput. It is also indicated in (Yang, Gerla, & Sanadidi, 2004), because of commonly fixed value of RTO in most operating system is 1 second injecting bursty packets on the queue of the bottleneck push TCP connection timeout. This reduces its TCP's throughput. Randomizing incapacitates attackers not to predict the TCP's RTO.

Pattern matching approaches are used to prevent LDoS attack according to (Liu, & Guan, 2010). Patterns of input traffics are compared with attack traffic flows and if match is found, the flow is suspected to be an attacking flow. But the arithmetic operation used to detect attacking flow pattern is very expensive. The study by (Sarat, & Terzis, 2005) has shades light on the effect of buffer size on LDoS attack identification. They claim that increase in the buffer size of interconnecting edge routers can reduce the effect of the attack. The limitation however is that the mechanisms to detect LDoS attacks are absent and could not be could not be cost effective.

In the work by (Zhang, Yin, Cai, & Chen, 2010), they have proposed a novel approach to counter LDoS attack. Detection and filtering algorithm is added to the most widely used Random Early Detection (RED) active queue management algorithm. Packets pass through detection and filtering block before they are forwarded to RED block. Arrival of the last packet dropped by detection and filter as well as RED block is recorded. Packets are suspected to be an attacking if their arrival time falls in the short time interval computed from maximum of the drop time plus constant time T*. Keeping track of all the flows needs per-flow analysis and this may cause complexity in detection as well as increase execution delay. Another group of researchers (Mohan, John, & Bijesh, 2013), used preferential dropping RED and used partial analysis to detect LDoS flows.

In (Sun, Lui, & Yau, 2004), use the deficit round-robin (DRR) algorithm to protect the TCP flows and isolate them from the attack traffic is demonstrated through experiment. It provides bandwidth allocation and protection between flows. DRR isolate ill-behaved sources at a very low implementation cost with very high accuracy. Table 2.3 shows summaries defense mechanisms.

Table 2.3 Summary of defense mechanisms

No.	Mechanism	Effect (Result)
1	Randomization	Degradation of TCP connection, very small range of RTO
2	Pattern matching approaches	Very expensive operations are required
3	Increasing buffer size of edge routers	Not cost effective
4	Partial analysis of flows of packets (RED-PD)	Only mitigates the attack, cannot prevent
5	Per-flow analysis of arrival and drop time of packet (Robust RED)	Preserves TCP throughput even under LDoS attack (complexity of per-flow analysis)
6	Deficit round-robin (DRR) bandwidth allocation and protection between flows	Isolation of ill-behaving flows

2.9 Summary

Network attacks come in various form from simple disruption to destruction of computer network components. It causes compromises to the integrity, confidentiality or availability of resources on the net. Denial of service attack is one which hampers genuine rights of the legitimate users from accessing any or specific type of resources on the network. The two known TCP targeted attacks are reduction of quality (RoQ) exploiting AIMD and low-rate denial of service (LDoS) manipulating or misusing TCP's retransmission time out mechanism.

Integration of detection and filtering mechanisms to the already existing active queue management (AQM) was done on RED and RED-PD. The evaluation shows that the RED-PD mitigates the effect, but fails to prevent LDoS attacks. Robust

RED, even though per-flow method poise overhead to the distributed LDoS detection system, it effectively preserves TCP throughput under LDoS attack.

There are two board detection mechanisms to the low-rate denial of service attack. The first category consists of wavelet transform for extraction of desired frequency components, digital signal processing (DSP) – use of frequency domain statistical signal analysis, and dynamic time warping (DTW) – compares template signature with input signal for similarity. The second form of detection is when periodic burst length pattern is greater than or equal to round trip time (RTT), attack period overlaps with retransmission time out (RTO) and per-flow analysis of arrival time of both dropped and received packets.

The first defense against low-rate denial of service was randomization of retransmission time (RTO). It was practically implemented on UNIX system to see the performance of TCP throughput under LDoS attack and it achieved the desired result because in randomized RTO attackers cannot estimate or predict the next RTO to unleash the attack. It has demerits for both the range of the RTO is very small and under normal condition, TCP connection performance will suffer. Pattern matching approach, which tries to see matching between incoming signals with template signature, has also been proposed. Flows that match with the template signature will be suspected and removed. Increasing the buffer size at router was another method employed. Per-flow and partial analysis are the two flows related technique used to defend such an attack. In the per-flow analysis, arrival time of both dropped and received packets are recorded.

The impacts that can possibly be caused by LDoS attack may swing from under-utilization of network resources: nearly zero throughput of all applications using TCP as transport services to devastating effects to the boarder gateway protocol: global disturbance of routing stability.

CHAPTER THREE
ROBUST ORANGE ACTIVE QUEUE MANAGEMENT

3.1 Active Queue Management

An active queue management (AQM) has been introduced formally as complementary to IP networking in 1993(Adams, 2013). It was a complement to the end-system Transmission Control Protocol (TCP) in controlling congestion as network utilization increases; to limit packet loss and delay. Congestion, total demand for a resource greater than the capacity endowed; or faster arrival rate at the link interface greater than departure rate; causing latency in data transfer, waste of resources due to packet losses, even worst case congestion collapse. Tow mechanisms are required to solve the problem: congestion control to bring network out of already congested state and congestion avoidance to keep the network at optimal state or low delay and high utilization.

Initially, congestions were attempted to be managed at endpoint, termed as endpoint congestion control using TCP. However, there were limitations to this method: the first failure of such mechanism is that it is reactive- seeking to rescue after the occurrence of congestion. The second is time lag due to distances between the router and end-system. It takes significantly enough amount of time between the time packets are dropped at the routers and sources detecting the loss during which sources continue sending high transmission rate to the already congested network; exasperating the number of packet losses. Endpoint congestion control limitations are magnified by the increasing number of multimedia and peer-to-pear application. There are demands in quality of service (QoS) guarantees in terms of delay, delay variation or jitter, loss volume and bandwidth variability.

Incapacity of current endpoint congestion control to guarantee such demands led to the exploration of more efficient method to counter congestions. This led to development of active queue management (AQM) and packet or queue scheduling. AQM is a proactive congestion control mechanism. It sends information to the sources when congestion signals are detected. Random Early Detection (RED) was

the first formally and fully proposed AQM. Several schemes were proposed; some embraced to improve RED, others attempted to rigorously analyze RED and hinted its drawbacks; still a few devoted their time and energy to come up with completely new AQM schemes. Generally, there are three design approaches to AQM algorithms: heuristic, control-theoretic and deterministic optimization.

A new proposed, designed and developed active queue management algorithm called Orange will be augmented with detection and filtering mechanism to defend TCP flows from attacks targeting them. It is proactive algorithm. In the next section it will be discussed in detail first Orange and then Robust Orange.

3.2 Orange AQM

Orange is defined and analyzed in (Çatalkaya & Şiş, 2010) as queuing discipline designed for systems with two heterogeneous servers making use of threshold type policy because use of threshold helps a system to react to workload changes to improve the cost of performance ratio (Golubchik & Lui, 2002). Packets are preferably routed to the faster server in a threshold queuing discipline. Slower server remains idle and packets are allowed to queue up until queue size attains a certain threshold value. Then the packets are sent to the slower server for services. Thus, threshold value plays critical role in controlling systems overall performance.

Virtual drop server to drop incoming packets when the actual queue (on average) exceeds the threshold level is described fresh in (Çatalkaya & Şiş, 2010) work. It has service time which is the only configurable parameter based on changes in the network conditions. The use of IP level congestion control, higher throughput and lower queuing delays are the objectives planned or intended to be obtained.

A new packet dropping or marking approach is developed by the use of threshold and virtual server. Orange waits for a fixed amount of time after a dropping occurs before another one may be considered. Incoming packets are allowed to go to transmission if the queue size is below the threshold (Orange limit). Two types of

limits are used in the Orange AQM model. The first one is queue limit. If the computation of queue length after the arrival of a packet is equal to the queue limit, the packet will be dropped. If the queue size is between the second limit called Orange limit, and the maximum limit (queue limit), checking the status of the timer is required. If the status of the timer is found to be idle, the packet will be discarded and timer will be set on to count. Even if the newly arriving packet falls between the queue limit and Orange limit, packets are not discarded while timer is busy and therefore they are passed to the transmission server. If the queue length is less than the Orange limit, packets are automatically allowed for transmission. Figure 3.1 and Figure 3.2 below show the algorithm using pseudo code and block diagram respectively.

```
If(queue limit) then
        Drop(packet)
Else
        If(Orange limit) then          If(timer is idle) then
                                       Drop(packet)
                                       Begintimer()
                Else
                                       Enque(packet)
                End if
        Else
                Enque(packet)
        End if
End if
```

Figure 3.1 Pseudo code of Orange algorithm

Orange as an IP level AQM algorithm can be deployed on current Internet routers. It could be used in uncooperative transport control protocol to control average queue size. It is designed for a network where a single marked or dropped packet is sufficient to signal the presence of congestion. With cooperation with transport control protocol Orange is effective for congestion avoidance on routers. The rate at which the algorithm marks packet to be dropped depends on service time of the alternative server. This approach avoids synchronization that could have resulted from many connections. Virtual drop server is used only when queue size exceeds a

threshold (Queue limit). Orange and RED have similarity in their early drop action to inform TCP friendly flows about the congestion.

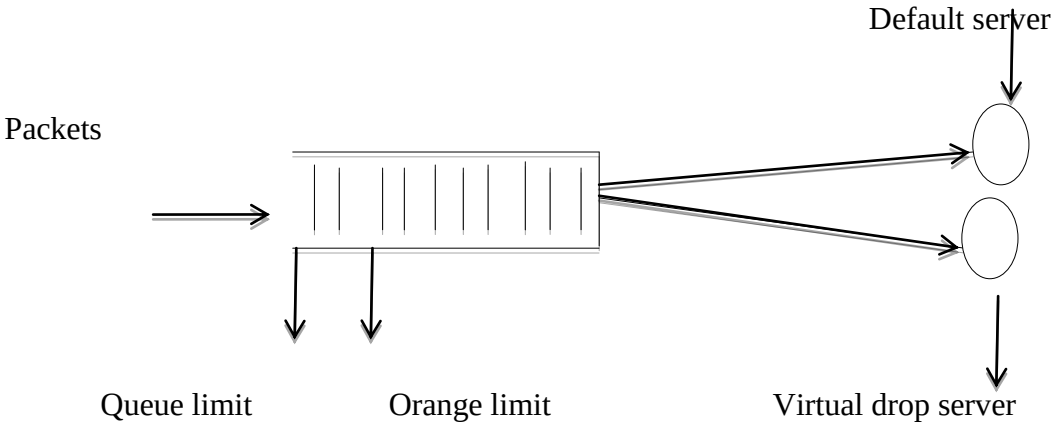

Figure 3.2 Simple model of Orange AQM

In (Zhang, Yin, Cai, & Chen, 2010), a variant of random early detection (RED) algorithm, which is designed to counter low-rate denial of service attack (LDoS) has been developed. The algorithm is called Robust RED (RRED). It preserves the ideal TCP throughput under LDoS attack. In this study, we adopted the same method suggested in Robust RED to counter LDoS attacks on Orange active queue management model. The next section will explain the integration of LDoS attack detection and filter mechanism into Orange in detail.

3.3 Robust Orange AQM

Robust Orange is an active queue management model which is the result of the combination of Orange and detection and filtering mechanism to protect TCP flows from low-rate denial of service attacks. Orange is a novel approach to design and develop a new active queue management model by (Çatalkaya & Şiş, 2010). Robust Orange AQM is an adoption from Robust RED. In Robust RED, detection and filtering block is added in front of traditional RED block. Thus, all incoming packets from all flows should go through the detection and filtering before fed into the RED algorithm which works by computing the average queue length. In this work, the

same principle is used to filter-out low-rate denial of service attack packets before they are fed into the Orange AQM.

The detection and filter algorithm added to the Orange works as follow: arrival time of the last packet from flow *f* dropped by the detection and filter block is registered for each flow. It is kept in a local variable called $f.T_1$. The arrival time of the last packet from any flow that is dropped by the Orange block is recorded in T_2. A packet from flow f is suspected to be an attacking packet if it arrives within a short-range after a packet from f that is dropped by the detection and filter block or after a packet from any flow that is dropped by the Orange block. Where the short-range is defined as $[T_{max}, T_{max} + T^*]$, in which T_{max}= MAX ($f.T_1$, T_2). If an incoming flow's arrival time falls into the above range, the packet is suspected to be an attacking packet. This is how suspected packets are distinguished from the normal TCP flows in the robust Orange model.

The assumption is that at the start of the flow, there is no drop. Thus, the values of f.T1 and T2 are assigned to zero. Whenever there is a drop, corresponding time of the flow will be updated. Then MAX is computed using the arrival time of the last packet dropped. All incoming flows will be checked against the Tmax. Computation of Tmax should depend on certain event. The event is the packet drop or loss. This needs the MAX computing algorithm in a function so as to call wherever required.

Once a packet from flow f is considered to be an attacking packet, the value of a variable which is used to indicate the status of the flow, *f.I,* is decreased by one. If the packet is assumed normal, the *f.I* is increased by one. Packets from a flow with negative *f.I* are filtered out and packets with positive *f.I* are further feed into the Orange. A very simple and clear way to observe what happens to flow indicator is displayed in Table 3.1 below.

Table 3.1 Summary of flow indicator and actions

Time	Flow indicator (f.I)	Action
Initial	Zero (0)	Feed to Orange
In short-range	Decreased by one (--1)	Filtered by detection & filter
Delayed	Increased by one (++1)	Feed to Orange

The need to analyze packets leads to recording all flows packet arrival time. To make use of space efficiently hashing the flow is required. Hash function is any function that can map arbitrary digital data size to fixed data size. It is a mapping function. Arbitrary digital data length will be manipulated by the function to produce a hash code, hash value or hash sum. Insignificant changes to the source data will produce very significant change in the output. Speed constraint applications which looks up for a value or range of value from the bulk of digital data benefits a lot from the utilization of hash function; because it creates hash indices-similar to database indices. Where the application only needs to search or look up throughout the file, in cryptography- to verify the validity of some input data against the stored hash value. Hash function accelerates look up by detecting duplicated records in the big files. In the Robust Orange, each flow is hashed to a hash table of size hundred. Packets with the same source and destination address will be hashed together because the function uses source and destination address to compute the hash index or value. The following Figure 3.3 shows hash algorithm.

Probabilistic space efficient to compactly store bits called bloom filter technique is used to handle numerous potential of incoming packets. It is array based hash data structure which. It computes k hash functions on each item.

```
1. iphdr<--access incoming packet
2. scrip<-- source address
3. dstip<-- destination address
4. sum<--(srcip + dstip)
5. module<-- (sum % Max)
6. return module
```

Figure 3.3 HashPkt pseudo code

As noticed in Figure 3.3 and Figure 3.4 of hashPkt, the procedure in hashing the flows based on the source and destination addresses of the incoming packets are simple and straight forward. Accessing incoming packet and ripping the IP header, from the IP header access both source and destination addresses which in NS2 case is represented as normal integer. Adding them together is the next step; then computing the module using Max (99) as hash table size. The pseudo code of Robust Orange is shown in Figure 3.5 as shown below.

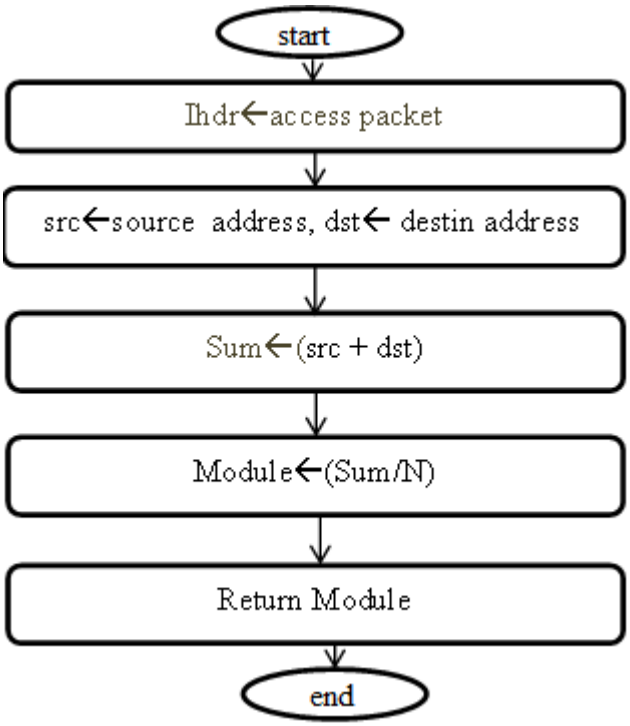

Figure 3.4 HashPkt flow chart

```
RobustOrange-ENQUE (pkt)
1.    index←RobustOrange-hashPkt(pkt)
2.    T_max ←MAX (flw[index].T_1 , T2orange)
3.    if pkt.arrivaltime is within [T_max + T*, T_max] then
4.                  decrease flw[index].findicator by1
5.    else
6.                  increase Flw[index].findicator by1
7.    end if
8.    if flw[index].indicator < 0 then
9.          flw[index].T_1 ← pkt.arrivaltime
10.         drop(pkt)
11.   else
12.         Orange-ENQUE (pkt)
13.         if Orange drops pkt then
14.               T2orange← pkt.arrivaltime
15.         end if
16.   end if
17.   return
```

Figure 3.5 Psuedo code of ROrange

Robust Orange and Robust RED algorithms are the same except Orange is used instead of RED in Robust Orange and RED in Robust RED. RED algorithm drops a packet in two conditions. First: if average queue length is greater than RED min threshold with probabilistic dropping rate increasing as average queue length increases until it reaches maxthreshold. Second case is when average queue length is greater than the max threshold. It always drops in the second case. Robust RED algorithm updates T2 in these two conditions. Similarly Orange algorithm drops a packet in two conditions. First: if queue length is greater than Orange (limit) threshold and lower than queue limit. In addition to the first condition, the status of the timer determines packets fate. The second case of drop is: if queue length is equal to or greater than queue limit. Robust Orange algorithm's flowchart is shown in Figure 3.6.

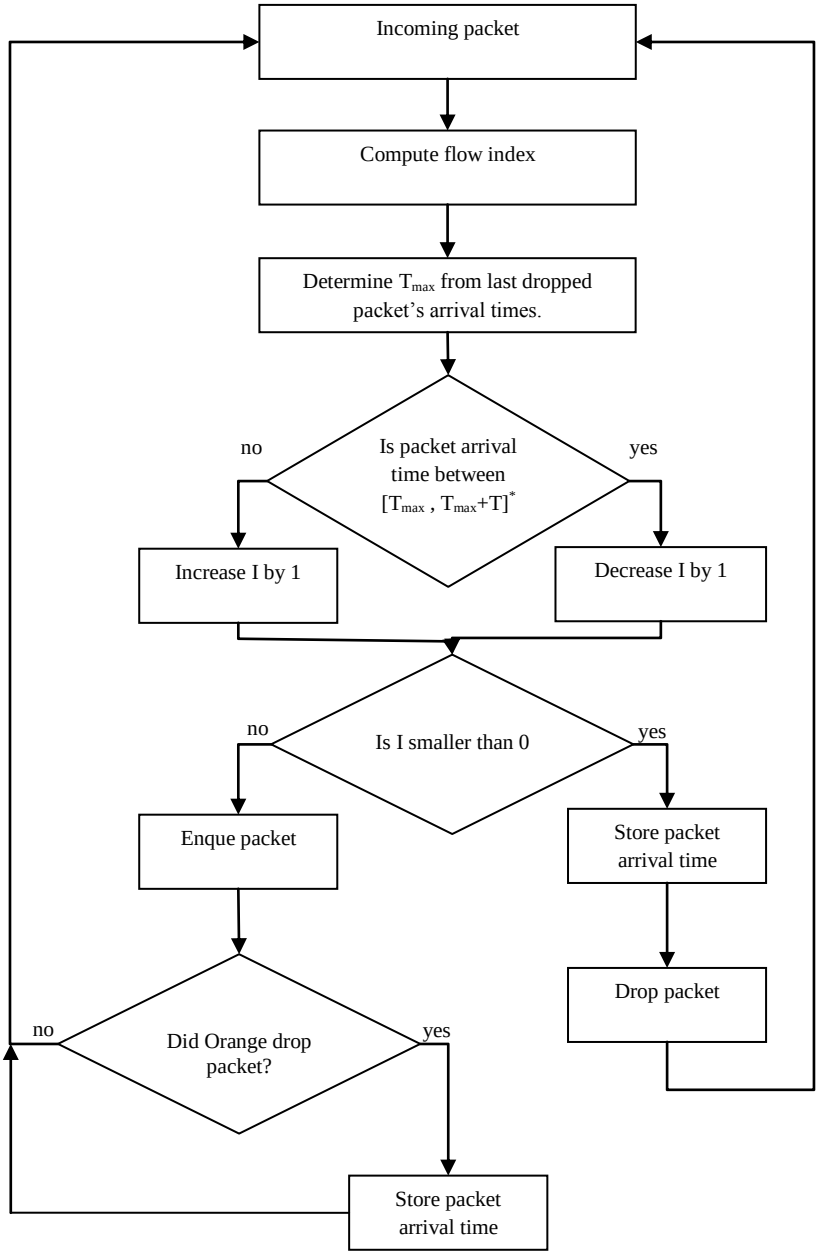

Figure 3.6 Flow chart of ROrange

3.4 Summary

Came into being as a complementary to TCP, which is implemented on the end-systems and reactive, AQM is proactive to congestion and provide quality of services. Orange is proactive AQM which uses threshold virtual server to mark or drop packets. Deployable on Internet routers to control uncooperative transport protocols; good for congestion avoidance; a single marked packet signals the existence of congestion.

CHAPTER FOUR
METHOD AND PERFORMANCE EVALUATION

4.1 Introduction to Simulator

Simulation recreates real-world scenarios using computer programs. It is used in various applications ranging from operations research, business analysis, manufacturing planning, and biological experimentation, just to name a few. Compared to analytical modeling, simulation usually requires fewer simplification assumptions, since almost every possible detail of system specifications can be incorporated in a simulation model. When the system is rather large and complex, a straightforward mathematical formulation may not be feasible. In this case, the simulation approach is usually preferred to the analytical approach. The essence of simulation is to perform extensive experiment and make convincing argument for generalization. Due to the generalization, simulation results are usually considered not as strong as the analytical results.

A simulation is, more or less, a combination of art and science. That is, while the expertise in computer programming and the applied mathematics accounts for the science part, the very skills in analysis and conceptual model formulation usually represent the art portion.

4.2 Network Simulator 2

Among the available network simulator, network simulator 2 (ns2) is an event-driven simulator. Its usefulness in studying the dynamic-nature of communication network has been proven. It has functionalities for simulating wired and wireless networks with regard to protocols such as routing algorithms, TCP, UDP and AQM to mention few. When using NS2, users may add their own protocols, simulation scenarios and algorithms since ns2 provides flexible environment to the users by its developable structure.

Since 1989, its popularity in networking research community has remained constant for its flexibility and modularity. Substantial contributions from the players such as university of California, Cornell University, development support from Advanced Research Project Agency (DARPA) and National Science Foundation (NCF) have marked the growing maturity to the tool. Group of researchers and developers in the community are another part bringing NS2 to be strong and versatile.

NS2 uses two languages: Otcl and C++. Otcl is used to define comprehensive network topologies, and configuration for simulating scenarios. On the other hand, detailed definitions of communication protocols are written in C++. C++ processes a huge volume of data in a short time. The Otcl and C++ languages are linked together using TclCL (Fall & Varadhan, 2011). Otcl and C++ languages have their advantages and NS2 benefits from them (Altman & Jimenez, 2003). C++ provides simulation efficiency and reduces event processing time. However, C++ is a bit difficult to learn and slow when user has to adjust some parameters and re-run the simulation. On the other hand, OTcl is slower while running the codes but it is very convenient to change parameters and re-run (Altman, & Jimenez, 2012; Chung & Claypool, n.d).

NS2 interprets user TCL scripts and produces the output in the form of a formatted text file called as trace file. Trace file length depends on created simulation topology and data traffics. With the increase of number of nodes and traffics, trace file length is increased. A trace file may have tens of thousands of lines. We need a text processing tool to interpret this file. We prefer to use awk, which is the high capable programming language used for NS trace analysis. To analyze trace files generated by the NS2, we develop awk script to calculate average throughput. Graphing tool used is gnu plot.

4.3 Integrating Robust Orange into NS2

We choose NS2 as Simulator and install version 2.35 of all-in-one to Ubuntu 12.0.4. In order to integrate Robust Orange to NS2, firstly we integrated Orange to NS2. Orange integration into NS2 is indicated in (Gökhan & Çatalkaya, 2012). We have followed these instructions. After the development of the required files such as C++, header and OTCL, visit the ns home directory. Mostly ns home directory found at nsallinone-2.35/ns-2.35. Among several files under the ns home directory, go to the Makefile file and open with any text editor. There are three sections in this file. The first part is called OB_CC, which represents C++/ C objects files. To simplify it, go to any line after the OBJ_CC = \ and add the name of the C++/C source code adding .o extension name. The header file is not included in the Makefile. The next file to be added to the Makefile is the OTCL file ending with .tcl extension. Section starting with NS_TCL_LIB = \ is where this file is going to be added. The third and final thing to add is your created directory where the files reside if any. The include directory is added following the INCLUDES = \ and the format is –I./<direcoryName>\. It includes the directory to the compiling path so that there is no need to type full path every time you compile or run the program.

The next procedure is copy both the header and C++/C source codes to the directory called queue under the home directory of ns2. In our case there are couples of default values to be set in the nsallinone-2.35/ns-2.35/tcl/lib/. Under this directory open the file with the name ns-default.tcl. Modification to the already existing values can be changed; as well as new default values which has been defined in the header file, bind to its otcl object under the constructor method are required to be set.

NS2 needs to be recompiled in two cases. The first is when there is modification to the existing algorithm especially to the header and C++ files. The second is when new module, protocol, added. In both cases, make clean, make depend and make commands must be executed respectively and if there is any error message need to be debugged very cleanly to execute the new or modified protocol or module added to the ns2. If execution is successful, object file is created in the queue directory.

4.4 Simulation Topology

In order to test Robust Orange model and compare it with other models we adapted a network topology used in (Zhang, et al., 2010). The topology is shown in Figure 4.1. As indicated in the Figure 4.1, the network topology has fifty (50) nodes representing the personal computers and two (2) more nodes representing routers. Thirty (30) of the computers are connected to the NewReno TCP agent with ftp packet generating application. A packet size of 1000 bytes is generated from each user. They are considered normal computer without any malicious traffic. The remaining twenty (20) are the LDoS attack computers. They are all connected to the UDP agent with CBR packet generator. Each attacker sends a packet size of 50 bytes. The size of the bottleneck queue size is 50 packets and the new active queue management algorithm we developed called Robust Orange will be used on this queue. Other queues use DropTail. The RED AQM used in our case is packet byte rather than packet count mode. This due to the fact that Orange AQM algorithm is based on byte, we ought to use the packet byte, which makes comparison possible.

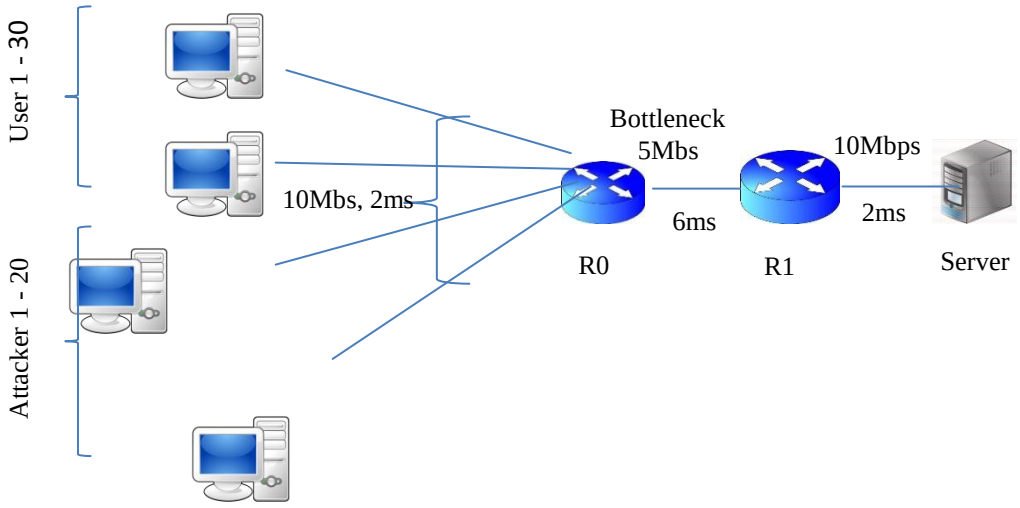

Figure 4.1 Experimental topology

4.5 Performance Evaluation

NS2 simulator is used to measure the performance of the proposed Robust Orange AQM algorithm to counter low-rate denial of service attack. Simulation topology is depicted in the Figure 4.1 above with all the parameters. But for the three parameters of the LDoS attack, we choose T_a= 1s, T_b = 200ms and R_b = 0.25Mbps so that the aggregate R_b of 20 attackers is equal to the bottleneck bandwidth of the network (5Mbps) as in (Zhang, et. al., 2010). Varying the values of these three parameters, three scenarios are conducted and compare the performance of Robust Orange and other AQM algorithms such as Robust RED. For each scenario, we fixed two values and vary the other value. LDoS parameters are shown in Table 4.1. We implemented two types Robust Orange with variations in the data structure used. In the first one array of struct that holds flow indicator, flow arrival time, and flow index for each flow are used. All the three parameters are initialized to zero (0) at the start. The second implementation is direct use of the Robust RED detection and filtering algorithm. In this implementation bloom filter technique is used for its space efficient on real system. Simulation results of both implementations are explained in the following graphs basing on the same LDoS parameters. Performance of the first implementation is greater than that of the second.

Table 4.1 LDoS parameters

Experiments	T_a (**s**)	T_b (**ms**)	R_b (**Mbps**)
Scenario one	[0.2 - 2]	200	0.25
Scenario two	1	[0 - 600]	0.25
Scenario three	1	200	[0.1 - 0.5]

Scenario one shows that as the attack period (Ta) increases, the throughput of Robust RED maintains its ideal link bandwidth throughout during the given attack time period as shown on Figure 4.2 below. It is resistant to LDoS attack. It performs better than RED. RED active queue management algorithm's performance sharply

increases approximately for the 0.5 seconds and continues gradual increase in performance as the attack period increases.

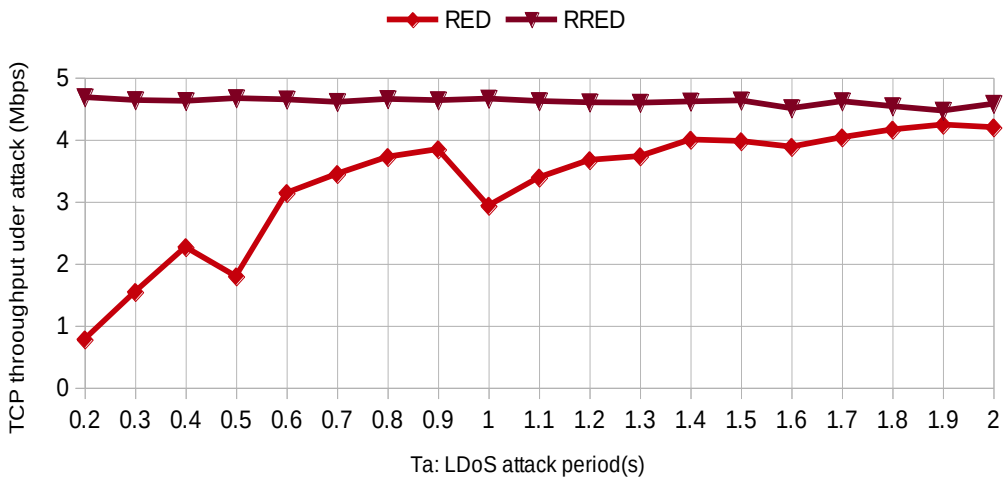

Figure 4.2 Simulation results of scenario one for RED and RRED

Scenario one of Robust Orange, detection and filtering added, and Orange AQM algorithm is shown in Figure 4.3 below. As the attack period (Ta) increases, TCP throughput of Robust Orange maintains its initial performance throughout the attack period as that of Robust RED depicted on Figure 4.2 above. Orange active queue management, as the attack period (Ta) increases, their TCP throughput performance increases sharply for the first 0.5seconds and keeps gradual increase up to 4Mbps at 0.9 second. It falls to 3Mbps and start climbing though out but finishes with 4Mbps throughput.

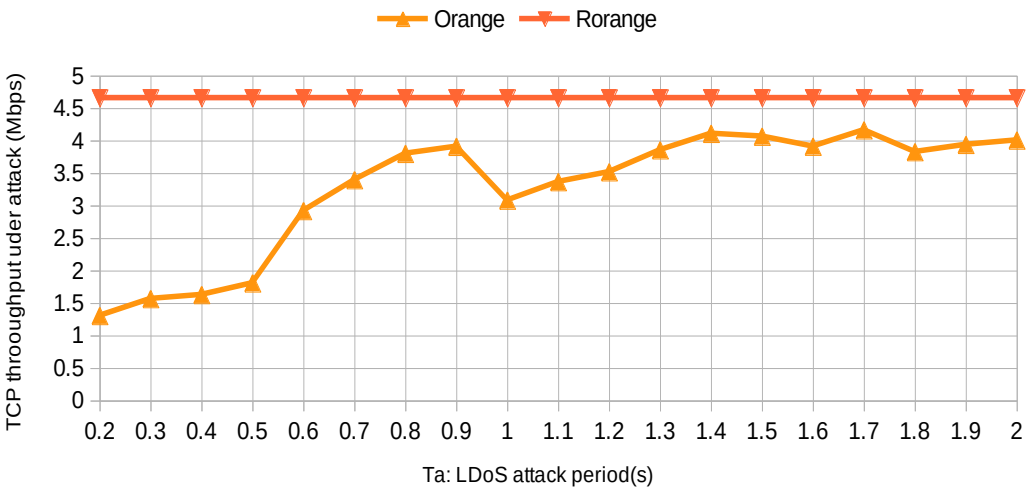

Figure 4.3 Simulation results of scenario one Orange and ROrange

As depicted in Figure 4.4 below, the increase in attack period (Ta), the throughput of Robust RED and Robust Orange maintains its ideal link bandwidth throughout the given attack time period However, Robust RED's TCP average throughput fluctuate showing a slight lower performance than Robust Orange starting from 1.5 seconds up to 2 seconds. Both resilient to LDoS attack which targets TCP's throughput. The performance of both RED and Orange active queue management algorithm sharply increases approximately for the 0.5 seconds and continues gradual increase until the end. Except for the first 0.5seconds the performances of Both RED and Orange overlaps for the larger remaining time.

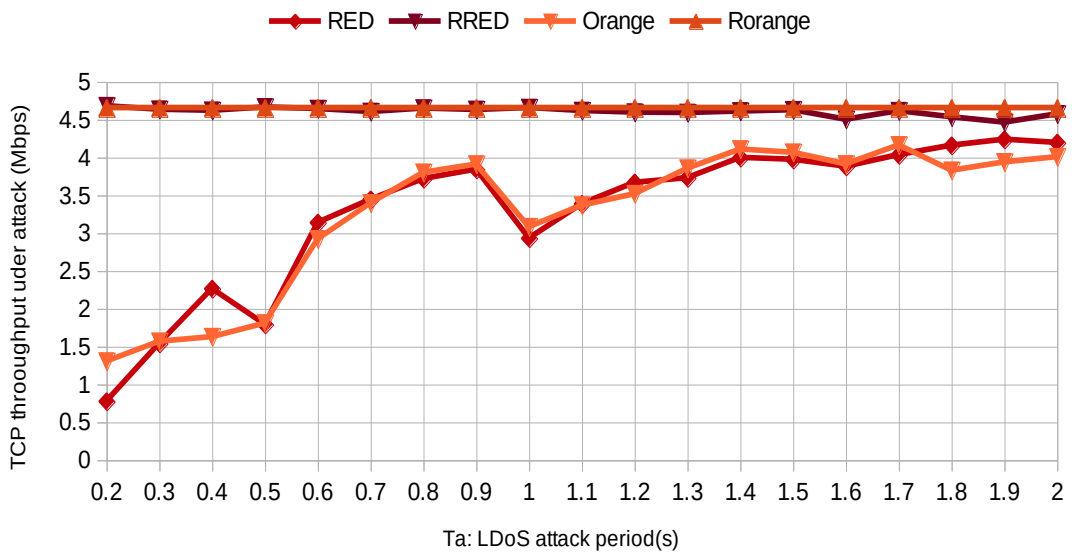

Figure 4.4 Simulation results of scenario one RED, RRED, Orange and ROrange

Scenario two shows that as the attack burst width (Tb) increases, the throughput of Robust RED maintains its ideal link bandwidth throughout the given time period as shown in Figure 4.5 below. It is resistant to LDoS attack. It performs better than RED. RED active queue management algorithm's performance declines as the attack period increases.

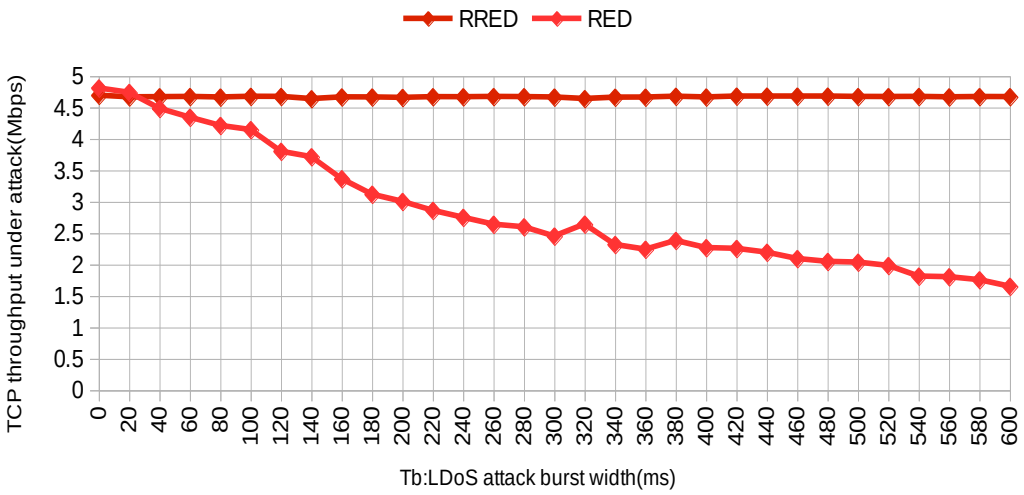

Figure 4.5 Simulation results of scenario two RED and RRED

Scenario two of Robust Orange's detection and filtering algorithm with Orange active queue management is shown in the Figure 4.6 below. As the attack burst width (Tb) increases, TCP average throughput of Robust Orange remains the same. Robust Orange has constant TCP throughput performance pattern of resisting the LDoS attack. Orange cannot resist the attack and as the attack burst width increases its TCP throughput decreases sharply from around 4.75Mbps to 1.5Mbps at the end of the attack duration.

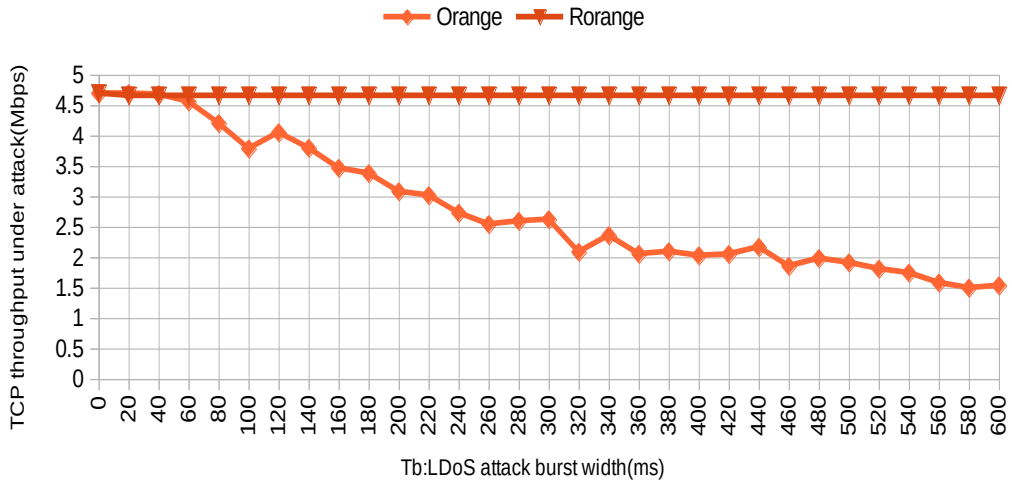

Figure 4.6 Simulation results of scenario two Orange and ROrange

Increase in attack burst width (Tb) does not affect the performance of Robust RED and Robust Orange throughout the attack. Their performance overlaps with almost no variations in the patterns of the line. On the other side, the performance of RED and Orange decreases as the attack burst width increases. The sharp decrease is observed after the first 40ms and up until the end of the attack burst width. Having an average TCP throughput of 4.75Mbps initially, but both finish the race against LDoS attack by 1.5Mbps.

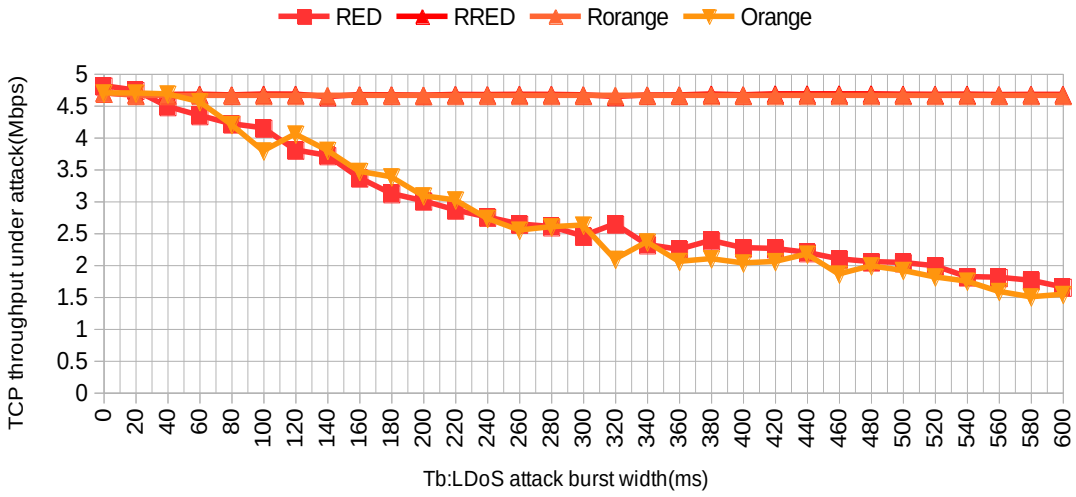

Figure 4.7 Simulation results of scenario two Orange and ROrange

Scenario three shows that as the attack burst rate (Rb) increases, the throughput of Robust RED maintains its ideal link bandwidth throughout the given time period as shown in the Figure 4.8 below. It is resistant to LDoS attack. It performs better than RED. Gradual decline is observed in RED active queue management algorithm's performance as the attack burst rate increases.

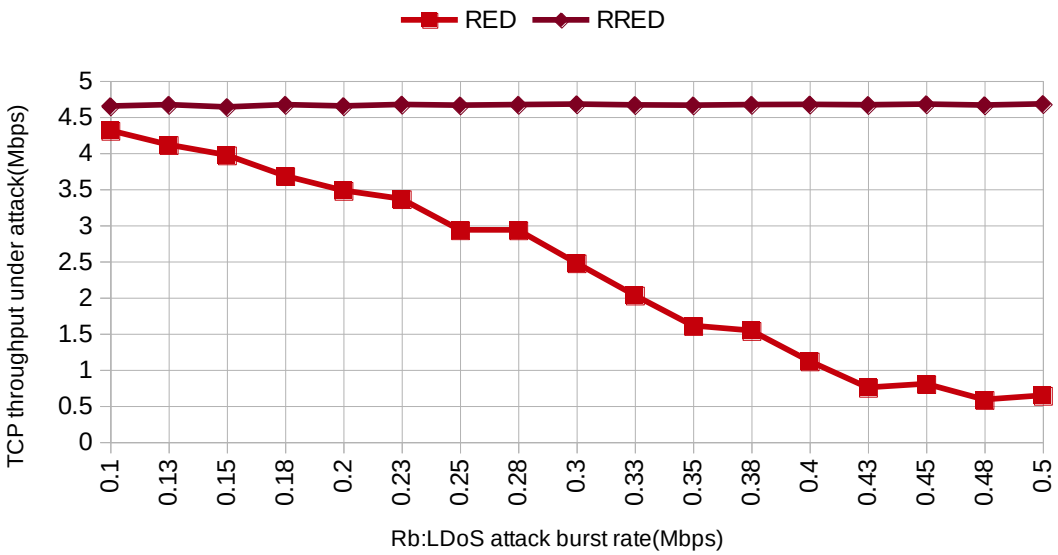

Figure 4.8 Simulation results of scenario three RED and RRED

Senario three of Robust Orange deteciton and filtering algorithm and Orange AQM is shown as in Figure 4.9. As shown in the Figure 4.9, as the attack burst rate (Rb) increases, Robust Orange maintains its throughput. This indicates that the detection and filtering block is functioning. Performance of Orange is highly fluctuating as the attack burst rate increases. It is vulnerabel to the attack. The average TCP throughput of Orange is around 3.25Mbps throughout the period implying that the attack highly influances the performance of AQM without deteciton and filtering algorithm.

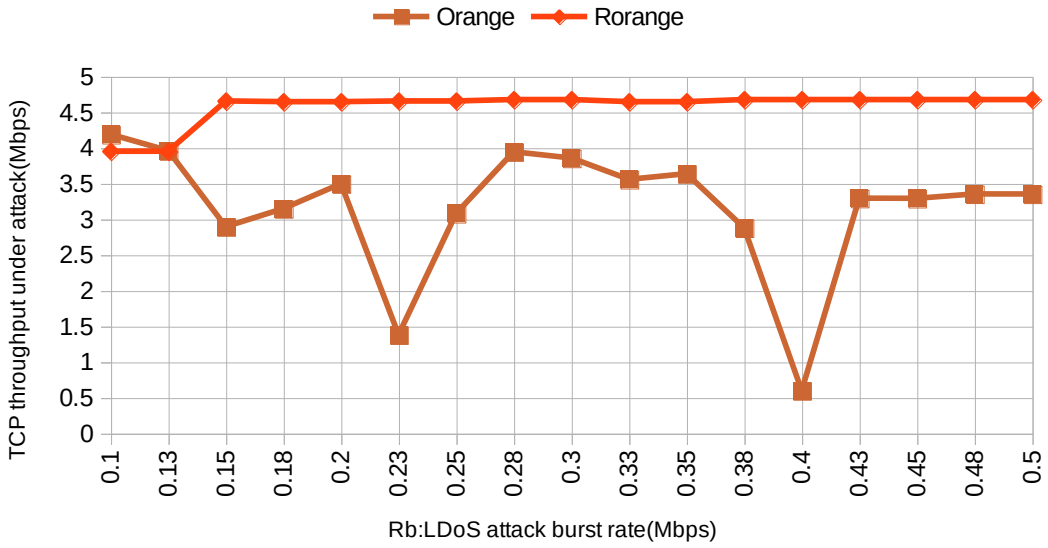

Figure 4.9 Simulation results of scenario three Orange and ROrange

Even if Robust RED performs better than Robust Orange for the first 0.15 seconds, they have the same performance pattern of resilience to the attack burst rate increase-continually filtering out the attack packets and letting as much of normal users' packets boosting TCP's average throughput performance. When it comes to the performances of both RED and Orange, Orange is fluctuating throughout the period, with an average throughput of 3.25Mbps and RED has much lower average throughput of 2.5Mbps throughout the attack period. Both RED and Orange are vulnerable to the attack which targets TCP and exploiting the retransmission time out.

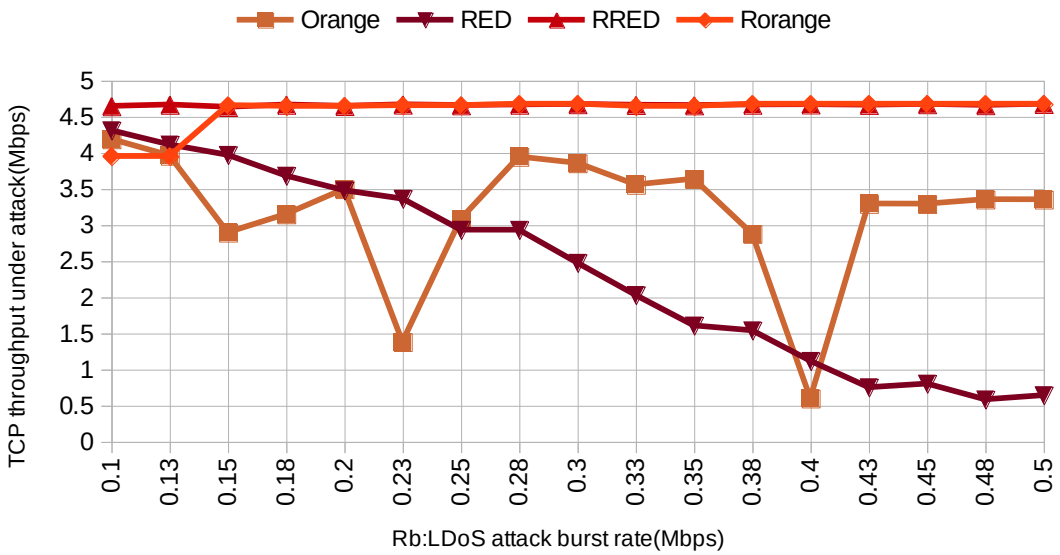

Figure 4.10 Simulation results of scenario three Orange, ROrange, RED and RRED

4.6 Instantaneous Throughput

The following figures show instantaneous TCP connection throughput of RED, Robust RED, Orange and Robust Orange. It also shows TCP drop for the above mentioned active queue management.

Figure 4.11 below convies instantaneous throughput of Robust RED and Robust Orange active queue management algorithms' instantaneous TCP throughput and TCP packet drops for the duration of the simulation periods. As the figure 4.11 shows clearly, both ROrange and RRED demonistrated almost identical performance before the LDoS attack streams are commenced. Immediately after the start of the attacks, both ROrange and RRED showed reduction in their instantaneous TCP throughput approximately for 25 seconds; however the stress caused by the attack is visibiley high in the RRED where the throughput fall bellow a thoudand kilo bits persecnd. For the next 75 seconds, theier perfomance almost overapped while for the rest of the time, ROrange showed slightly better performance. The instantaneous TCP connction throughput drop for both the AQM is also shown on the Figure 4.11. Less TCP connections' packets were dropped for RRED than ROrange before the

start of the attack. After the attack is initiated, slightly less TCP connection packets were dropped from the ROrange than RRED.

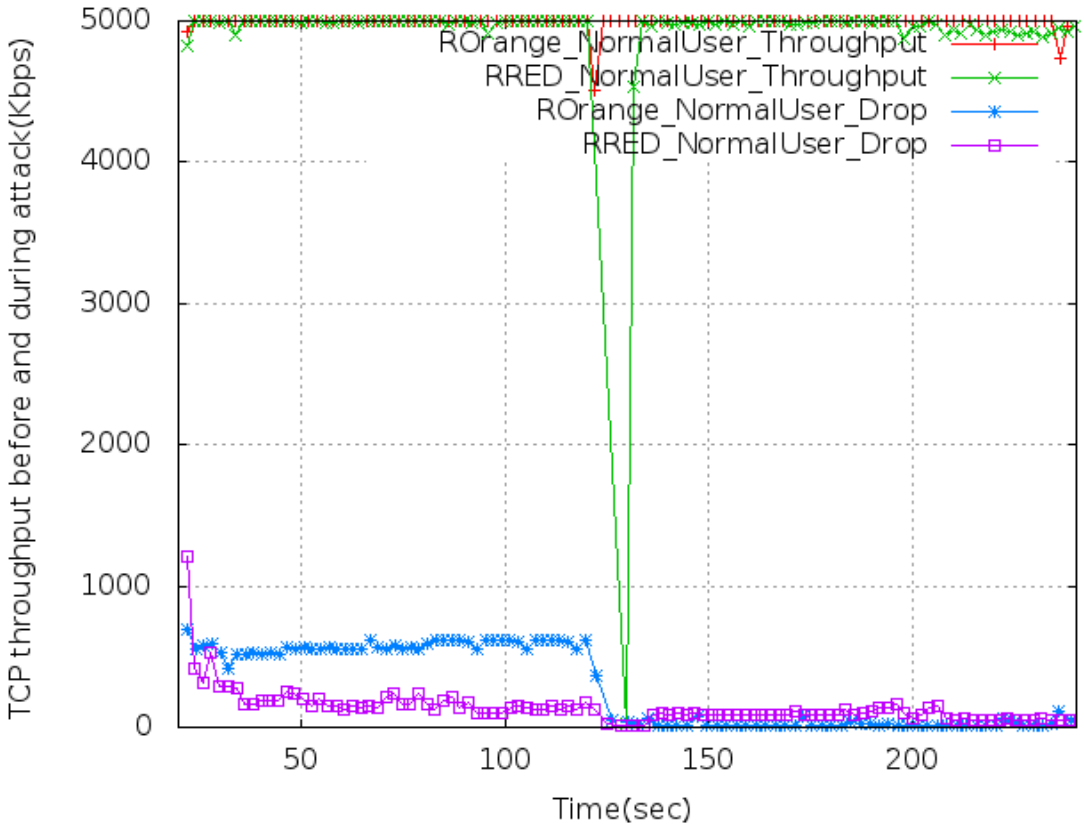

Figure 4.11 Instantaneous throughputs of RRED and ROrange

Figure 4.12 shows the instantaneous TCP connections throughput and the connections packets drop before and after the attack. The performance of both Orange and RED were identical before the start of the attack. However, immediately after the start of the attack, instantaneous TCP connection throughput of both AQM algorithms declined down below 4500Kbps. The difference between the TCP's instantaneous throughput before and after is approximately about 500Kbps. TCP connections instantaneous packet drops for both the algorithms showed a slight rise; however Orange TCP connections packet dropped much less than that of RED throughout the simulation periods.

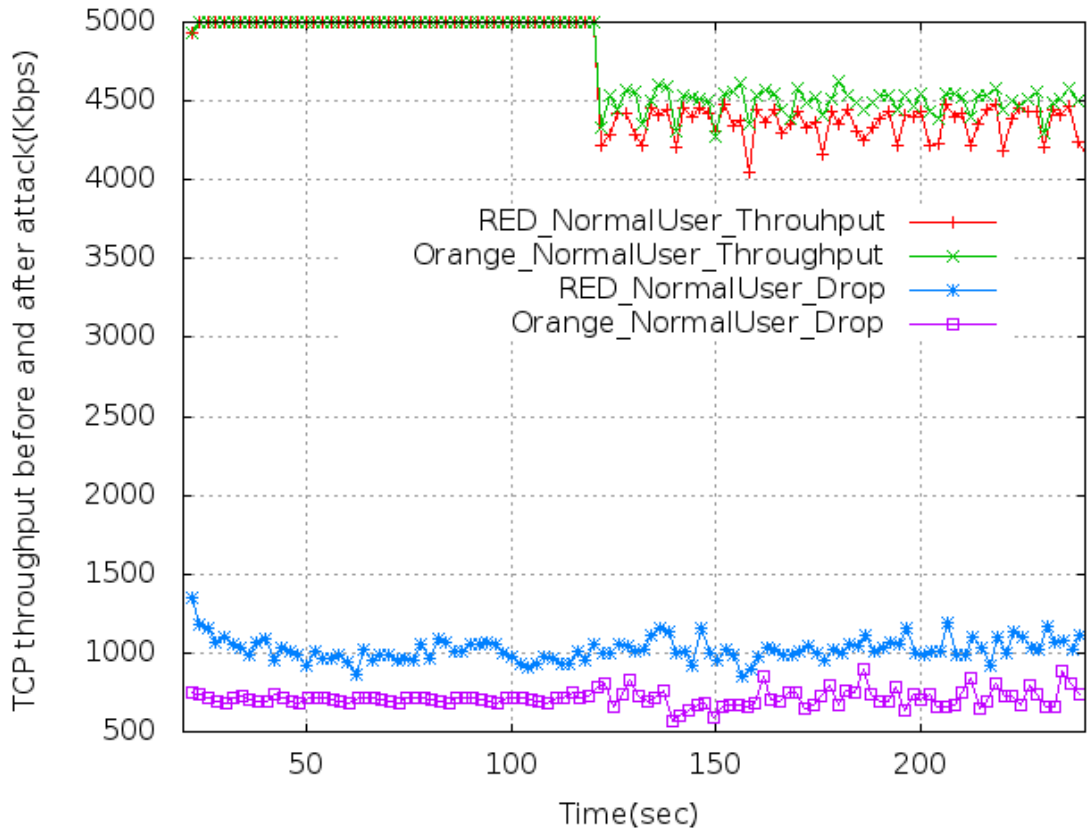

Figure 4.12 Instantaneous throughputs of RED and Orange

Figure 4.13 summarizes TCP connections instantaneous throughput performance and packets drop for both AQM with and without the attack filter. It is clear that for both the instantaneous TCP connections throughput and the drops, they maintain their initial performance up until encountered with the LDoS attack. It is obvious that ROrange and Orange slightly perform better in both instantaneous TCP throughput and the drop than the RRED and RED. Less TCP connections packets are dropped in both ROrange and Orange than RRED and RED.

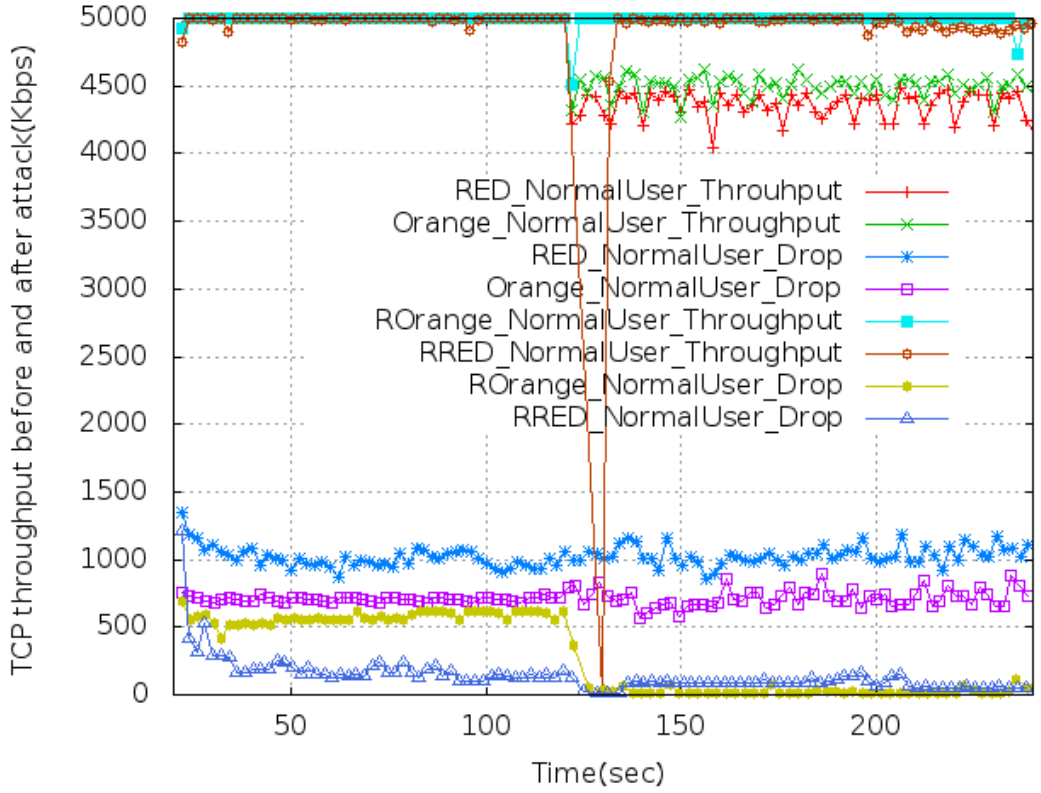

Figure 4.13 Instantaneous rhroughputs of RRED, ROrange, RED and Orange

4.7 Sensitivity, Specificity and Accuracy Mesures

Sensitivity is a true positive rate that measures the proportion of positives which are correctly identified. It is the probability of dropped packets given that the source is from the attacker. Specificity is the true negative rate that measures the proportion of negatives which are correctly identified. It is the probability of received packets given that the packet is from normal user.

The following section is dedicated to show the graphs of the sensitivity, specificity and accuracy measures of the filter for AQM with and without. The Figures show performance of RRED, ROrange, RED and Orange during the attack period (Ta) in second, attack burst width (ms) and attack burst rate (Rb) in Mbps.

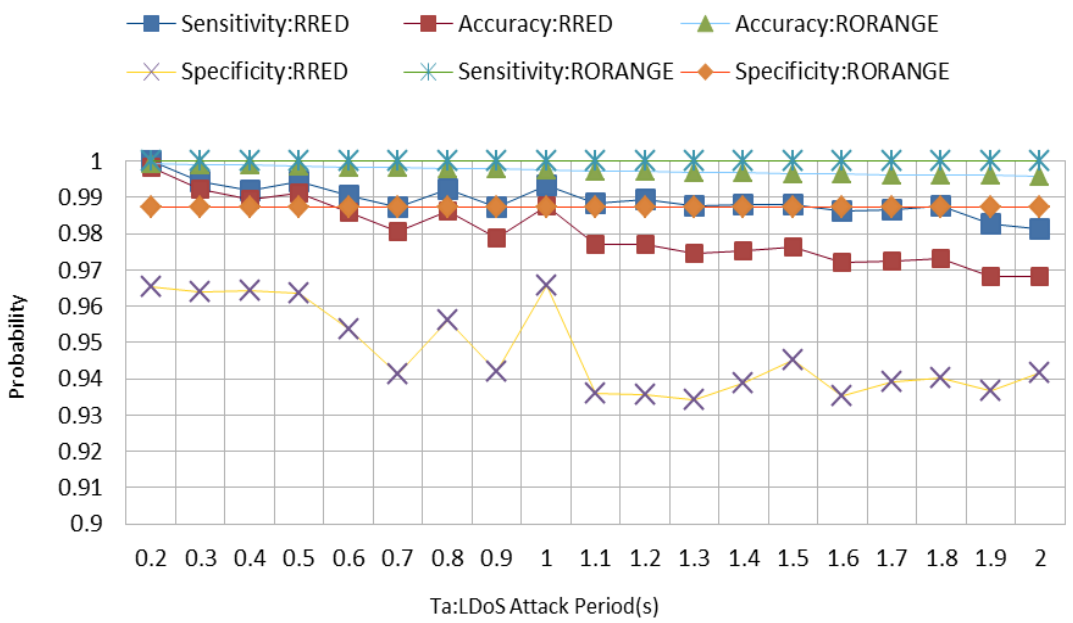

Figure 4.14 Sensitivity, specificity and accuracy of ROrange and RRED

Figure 4.14 above shows probability of sensitivity, specificity and accuracy measure of RRED and ROrange AQM algorithm during the attack period (Ta). As displayed on the figure above, ROrange is 99.99% sensitive to the attackers packets.Though sensitivity of RRED is the same with that of ROrange at the start, it slowly decreases as the attack period increases and falls below approximately 0.98. The specificity measure of RRED is much below that of ROrange, having an average 0.95 specificity while ROrange maintained almost 0.987 specificity during the attack period. Better measure of specificity shows the ability of the filter to allow normal users packet. The accuracy measure of both decreases; however, ROrange filters more accurately than RRED with almost 0.99 accuracy.

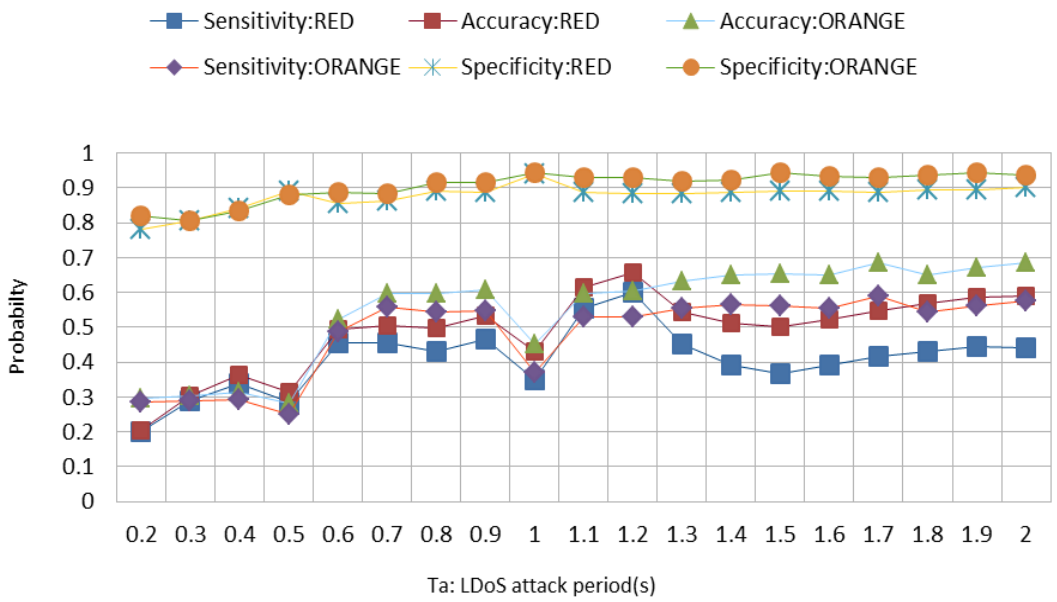

Figure 4.15 Sensitivity, specificity and accuracy of RED and Orange

Figure 4.15 is the probability measure of sensitivity, specificity and accuracy for Orange and RED without the filter block during the attack period (Ta). Sensitivity measure of both RED and Orange are both low. In both cases it slowly increases reaching on average 0.4 approximately from 0.25. Specificity measure of both the algorithms overlaps; starting with 0.8 and gradually increasing up to 0.9. Accuracy of both algorithm compete each other and their performance is similar to that of the sensitivity.

Except for the first forty (40) seconds, Figure 4.16 shows that sensitivity, specificity and accuracy measure of both RRED and ROrange algorithm have almost identical performance during the attack burst width (Tb) period. For the three measurements, the performance is approximately around 0.988. As the burst width of the attack increases, the sensitivity, specificity and the accuracy measure of the algorithm sharpens to perform best of their ability.

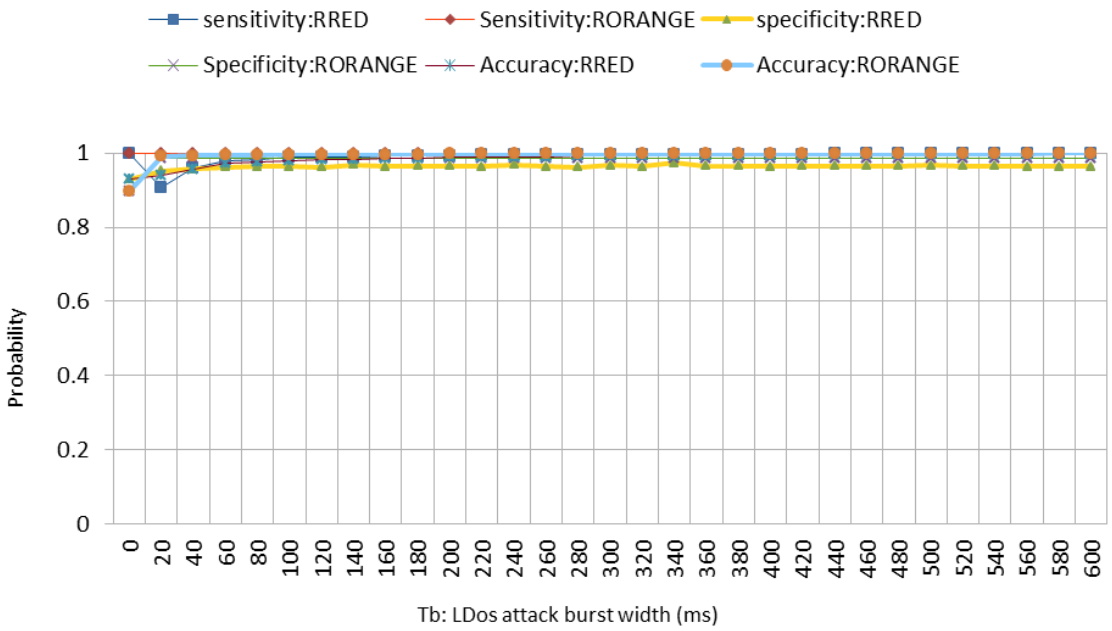

Figure 4.16 Sensitivity, specificity and accuracy of RED and ROrange

Figure 4.17 shows the probability measure of sensitivity, specificity and accuracy of RED and Orange AQM during the attack burst width (Tb). While specificity measure for both RED and Orange maintains 90% on average, both sensitivity and accuracy measure of RED and Orange decreases sharply during the first 320 milliseconds and slowly decrease up to the end of the attack burst width (Tb). For both RED and Orange they have around 99% values at the start but at the end of the burst width period it only performed 20%.

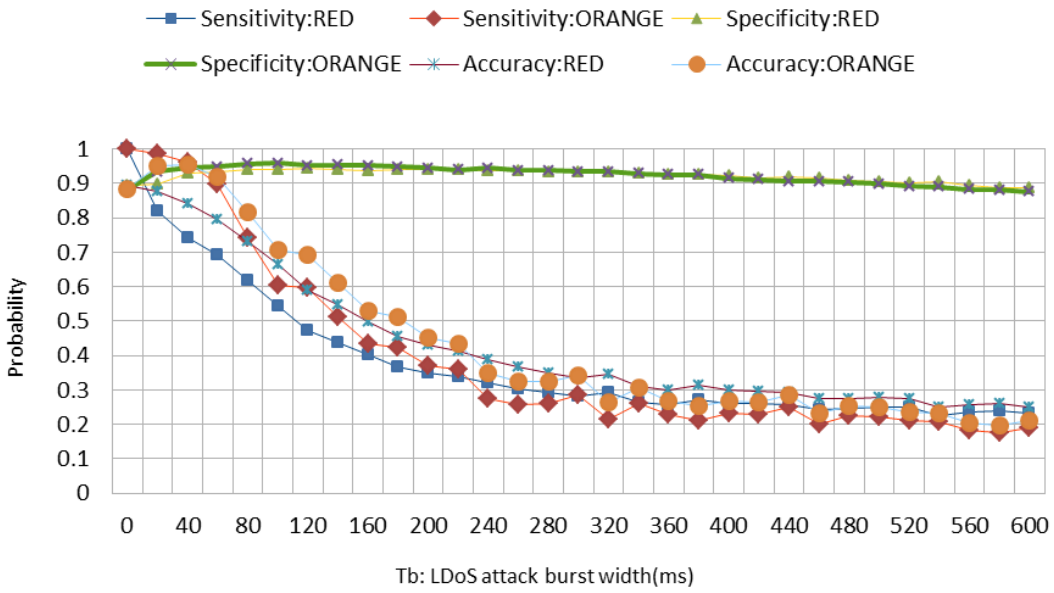

Figure 4.17 Sensitivity, specificity and accuracy of RED and Orange

Figure 4.18 shows the probability measure of sensitivity, specificity and accuracy of RRED and ROrange AQM algorithm for the duration of attack burst rate (Rb). While sensitivity of ROrange is measured to be 100%, RRED's sensitivity start at about 96.8% and sharply increases reaching much above 99% at the end. Performance of ROrange on specificity is much better than that of RRED which is down at 96%, while ROrange is around 98.5% approximately. The accuracy measure of ROrange (99.5%) is much higher than that of RRED which approximately follows the trend of its sensitivity.

Figure 4.19 depicts probability measure of sensitivity, specificity and accuracy of RED and Orange during the attack burst rate (Rb) period. Orange's sensitivity measure is 50% on average and RED's is 40%. The specificity of both RED and Orange which is 95% on average is much better when compared with both sensitivity and accuracy measure of RED and Orange. The accuracy for both the algorithm is approximately about 55% and it follows the trend of sensitivity.

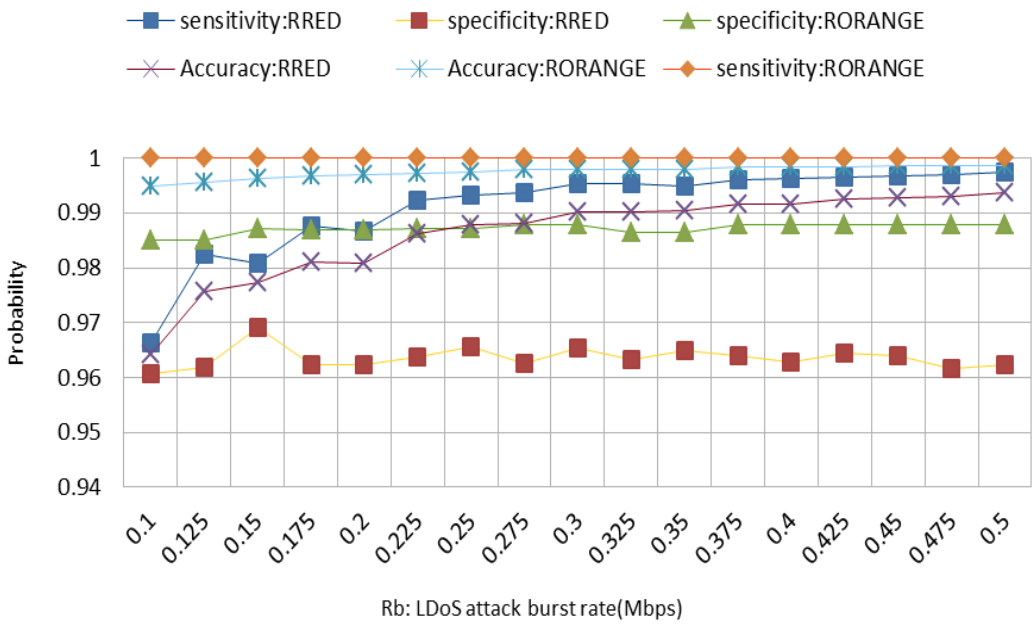

Figure 4.18 Sensitivity, specificity and accuracy of RRED and ROrange

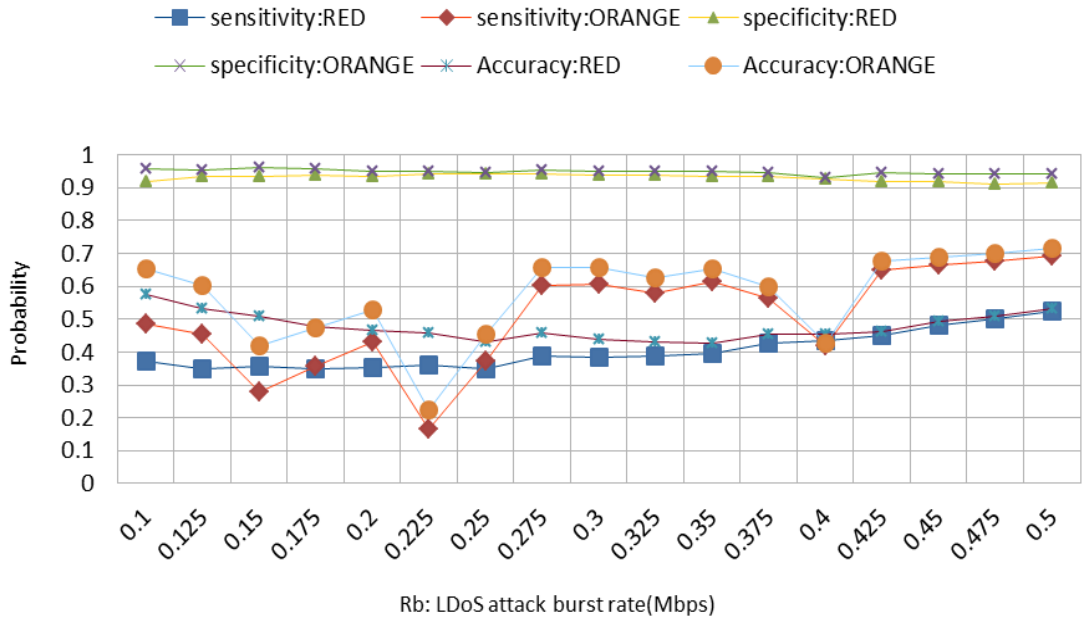

Figure 4.19 Sensitivity, specificity and accuracy of RRED and ROrange

CHAPTER FIVE
CONCLUSION AND FUTURE WORK

5.1 Conclusion

Varieties of AQM algorithm developed and in use today are with the objectives of increasing TCP throughput through managing congestions at bottleneck on the network. Among several network attacks available to disturb the smooth functioning of network traffic and infrastructure attackers has come up with tools that can systematically exploit designs of protocol and other software which fails to incorporate security issues. Low rate denial of service attack termed shortly, LDoS, is one that exploits TCP connections slow time scale dynamics of retransmission time out (RTO). This degrades performance by sending high-rate but short-duration bursts repeatedly to cause TCP connections enter RTO.

We adopted filtering technique employed in the Robust RED algorithm to counter this LDoS attack flow's packets into Orange and expanded the performance measurement to instantaneous TCP connection throughput because bursty traffics are best measured through such techniques. In addition, we sensitivity, specificity and accuracy measures are used to determine the performance of the filter itself.

Even though average TCP connections throughput are almost overlapping for all the three scenario of attack time, these do not lend to instantaneous TCP connections throughput where better performances are recorded. The simulation in the case of instantaneous throughput is done for fixed Ta, Tb and Rb.

Sensitivity, specificity and accuracy measure of ROrange during the attack period (Ta) found to be outperforming that of RRED. During attack burst width, the three measures are approximately overlapping except for the first 40 seconds. The performances are getting better as the attack burst width increases. As attack burst rate increases, sensitivity of RRED and specificity of ROrange increases gradually. Over performance of ROrange can be observed from close examination of the figure in all the three measures. From all the above supporting ideas, ROrange has shown

more robustness in filtering out the attack packets destined to degrade TCP connections throughput by exploiting RTO.

5.2 Future Work

The filtering algorithm in use within this thesis is based on the arrival time of packets. Packet sources such as ftp where the agent is the TCP respond to the existence of congestion within the network by delaying sending new packets and reducing the amount to send. On the other hand UDP agents with CBR sources of packets are non-responsive. They send packets irrespective of the existence of congestion in the network and the probability of packet drop from such source is high. In the future work, such unfair dropping ought to be examined closely.

REFERENCES

Abliz, M. (2011). Internet denial of service attacks and defense mechanisms. *University of Pittsburgh Technical Report*, 1-50.

Adams, R. (2013). Active queue management: A survey. *Communications Surveys & Tutorials*, *15*(3), 1425-1476.

Allman, M., & Paxson, V. (1999). On estimating end-to-end network path properties. In *ACM SIGCOMM Computer Communication Review*, 29, 4, 263-274.

Altman, E., & Jimenez, T. (2012). NS Simulator for beginners. *Synthesis Lectures on Communication Networks*, *5*(1), 1-184.

Baran, P. (1964). On distributed communications. *Volumes I-XI, RAND Corporation Research Documents*, 637-648.

Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., & Zhang, L. (1998). *Recommendations on queue management and congestion avoidance in the Internet* (No. RFC 2309). Retrieved May, 2, 2015 from http://www.rfc-editor.org/rfc/rfc2309.txt.

Çatalkaya, G. & Şiş, M. K. (2010*).* Analysis of Orange Active Queue Management Algorithm to Find Its Optimum Operating Parameters. *Istanbul University - Journal of Electrical & Electronics Engineering (IU-JEEE)*, 1243-1255.

Çatalkaya, G. (2010). *A new approach to IP level congestion control.* P.hD. Thesis, Dokuz Eylül University, İzmir.

Carr, C. S., Crocker, S. D., & Cerf, V. G. (1970). HOST-HOST communication protocol in the ARPA network. In *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, (589-597).

Cerf, V. G., & Icahn, R. E. (2005). A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, *35*(2), 71-82.

Cerf, V. (1993). *How the Internet came to be*. *The On-line User's Encyclopedia: Bulletin Boards and Beyond. Reading,* Massachusetts: Addison-Wesley.

Cerf, V., Dalal, Y., & Sunshine, C. (1974). *Specification of internet transmission control program* (72). INWG General Note.

Chang, R. K. (2002). Defending against flooding-based distributed denial-of-service attacks: a tutorial. *Communications Magazine*, *40*(10), 42-51.

Crocker, S. D., Heafner, J. F., Metcalfe, R. M., & Postel, J. B. (1972). Function-oriented protocols for the ARPA computer network. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference,* (271-279).

Despres, R. (1972). A packet switching network with graceful saturated operation. *Computer Communications: Impacts and Implications*, 345-351.

Efstathopoulos, J. T. P., (2009). Low-rate TCP-targeted Denial of Service Attack Defense. *Internet Technology and Secured Transactions. ICITST 2009. International Conference.*

Ehlert, S., Geneiatakis, D., & Magedanz, T. (2010). Survey of network security systems to counter SIP-based denial-of-service attacks. *Computers & Security*, *29*(2), 225-243.

Feng, W. C., Kandlur, D. D., Saha, D., & Shin, K. G. (1999). A self-configuring RED gateway. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.* 3, 1320-1328.

Feng, W. C., Kandlur, D. D., Saha, D., & Shin, K. G. (2001). Stochastic fair blue: A queue management algorithm for enforcing fairness. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies.* 3, 1520-1529.

Floyd, S., & Fall, K. (1997). *Router mechanisms to support end-to-end congestion control* technical report. Retrieved April 25, 2015 from http://wwwnrg. ee. lbl. gov/floyd/end2end-paper. html.

Floyd, S., & Fall, K. (1999). Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking (TON), 7*(4), 458-472.

Floyd, S., & Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions, 1*(4), 397-413.

Ghorbani, A. A., Lu, W., & Tavallaee, M. (2010). *Network attacks* (1-25). US: Springer.

Chung, J & Claypool, M. (n.d). *NS by example.* Retrieved May 1, 2015, from http://nile.wpi.edu/NS/.

Guirguis, M., Bestavros, A., & Matta, I. (2006). On the impact of low-rate attacks. In *Communications, 2006. ICC'06. IEEE International Conference,* 5, 2316-2321.

Guirguis, M., Bestavros, A., & Matta, I. (2004). Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference,* (184-195).

Heart, F. E., Kahn, R. E., Ornstein, S. M., Crowther, W. R., & Walden, D. C. (1970). The interface message processor for the ARPA computer network. In *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference,* (551-567).

Jacobson, V. (1995). Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review, 25*(1), 157-187.

Jacobson, V. (1988). Congestion avoidance and control. In *ACM SIGCOMM Computer Communication Review*, 18, 4, 314-329.

Kahn, R. E., & Crowther, W. (1972). Flow control in a resource-sharing computer network. *Communications, IEEE Transactions*, *20*(3), 539-546.

Khosroshahy, M. (2009). *Congestion avoidance and control technical report*. May, 2015 from http://www.masoodkh.com/_files/projects/networks/Congestion.pdf.

Kunniyur, S. S., & Srikant, R. (2004). An adaptive virtual queue (AVQ) algorithm for active queue management. *Networking, IEEE/ACM Transactions*, *12*(2), 286-299.

Kuzmanovic, A., & Knightly, E. W. (2003). Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 75-86).

Kuzmanovic, A., & Knightly, E. W. (2006). Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Transactions on Networking (TON), 14*(4), 683-696.

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., et al., (2009). A brief history of the Internet. *ACM SIGCOMM Computer Communication Review*, *39*(5), 22-31.

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., et al., (1997). The past and future history of the Internet. *Communications of the ACM*, *40*(2), 102-108.

Liu, Z., & Guan, L. (2010). Attack simulation and signature extraction of low-rate DoS. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium*, (544-548).

Luo, X., & Chang, R. K. (2005). *On a new class of pulsing denial-of-service attacks and the defense.* Retrieved on September 15, 2015 from http://www.isoc.org/ndss05/proceedings/papers/new_pulsing_DOS.pdf.

Luo, X., Chang, R. K., & Chan, E. W. (2005). Performance analysis of TCP/AQM under denial-of-service attacks. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium*, (97-104).

Maciá-Fernández, G., Díaz-Verdejo, J. E., & García-Teodoro, P. (2009). Mathematical model for low-rate DoS attacks against application servers. *Information Forensics and Security, IEEE Transactions*, *4*(3), 519-529.

Mahajan, R., Floyd, S., & Wetherall, D. (2001). Controlling high-bandwidth flows at the congested router. In *Network Protocols, 2001. Ninth International Conference*, (192-201).

Marill, T., & Roberts, L. G. (1966). Toward a cooperative network of time-shared computers. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference* (425-431).

Mathew, R., & Katkar, V. (2011). Survey of low rate dos attack detection mechanisms. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology* (955-958).

McKenzie, A. (2012). *HOST/HOST protocol for the ARPA network*. Retrieved April 30, 2015 from https://tools.ietf.org/html/rfc6529.

Mohan, L., John, J. K., & Bijesh, M. G. (2013). Shrew Attack Prevention in RED Queue with Partial Flow Analysis. *International Journal of Computer Applications*, *67*(8), 9-15.

Nagle, J. (1984). Congestion control in IP/TCP internetworks. *ACM SIGCOMM Computer Communication Review*, 14(4), 11-17.

Postel, J. (1981). *Internet protocol*; RFC791. *ARPANET Working Group Requests for Comments*, 791. Retrıeved August 2, 2015 from http://tools.ietf.org/html/rfc791?AFRICACIEL=djc4hn3j5o9hjth7cdv1q8v936.

Postel, J. (1981). RFC793: Transmission Control Protocol. USC. *Information Sciences Institute*, *27*, 123-150.

Roberts, L. G. (1967). Multiple computer networks and intercomputer communication. In *Proceedings of the first ACM symposium on Operating System Principles* (3-1).

Roberts, L. G., & Wessler, B. D. (1970). Computer network development to achieve resource sharing. In *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, 543-549.

Sarat, S., & Terzis, A. (2005). On the effect of router buffer sizes on low-rate denial of service attacks. In *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference*, 281-286.

Shashidhara, H. V., & Balaji, D. S. (2014). Low Rate Denial of Service (LDoS) attack–A Survey. *International Journal of Engineering Technology and Advanced Engineering*, *4*.

Shevtekar, A., Anantharam, K., & Ansari, N. (2005). Low rate TCP denial-of-service attack detection at edge routers. *Communications Letters, IEEE*, *9*(4), 363-365.

Stevens, W. R., Allman, M., & Paxson, V. (1999). *TCP congestion control*. *Consultant*. Retrieved August 3, 2015 from https://tools.ietf.org/html/rfc2581.

Subramani, B., & Chandra, D. E. (2010). A survey on congestion control. *Global Journal of Computer Science and Technology*, 9(5).

Sun, H., Lui, J. C., & Yau, D. K. (2004). Defending against low-rate TCP attacks: dynamic detection and protection. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference*, (196-205).

Viradiya, P., Vaghela, D., & Dhangar, D. (2014). Study of Low Rate Denial of Service (LDoS) attacks on Random Early Detection (RED). *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 3(6).

Yang, G., Gerla, M., & Sanadidi, M. Y. (2004). Defense against low-rate TCP-targeted denial-of-service attacks. In *Computers and Communications. Proceedings. ISCC 2004. Ninth International Symposium*, 1, 345-350.

Zhang, C., Yin, J., Cai, Z., & Chen, W. (2010). RRED: robust RED algorithm to counter low- rate denial-of-service attacks. *Communications Letters*, 14(5), 489-491.

Zhang, Y., Mao, Z. M., & Wang, J. (2007). *Low-rate tcp-targeted DoS attack disrupts internet routing*. Retrieved on September 15, 2015 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.5004&rep=rep1&type=pdf.

APPENDICES

Here are the attachments of the codes that are integrated into NS2 to simulate the configuration topology. There is also awk script to analyze the trace file.

Appendix A

RobustOrange C++ Header file code

```
#ifndef ns_orange_robust_h
#define ns_orange_robust_h
#include <string.h>
#include "queue.h"
#include "config.h"
#include <timer-handler.h>
#include <time.h>
#include <sys/time.h>
#define Max 99
#include "orange.h"
#include "red.h"
#define MAX(a,b)            ((a) > (b) ? (a) : (b))
struct Rflows { int findex; int findicator; double T1; };
class RobustOrange;
class robustOrangeTimer : public TimerHandler{
public:
        robustOrangeTimer (RobustOrange* que) : TimerHandler() { que_ = que; }
        virtual void expire(Event *evnt);
protected:
                RobustOrange* que_;
};

class RobustOrange :  public Orange { /* Inherited from Orange not from Queue*/
```

```
        friend class robustOrangeTimer;
public:
        RobustOrange();
        int t_status();
   ~RobustOrange() {
         delete q_;
}
protected:
void enque(Packet* pkt);
void initialize();
int hashPkt(Packet* pkt);
int detectionFilter(Packet* pkt);
int pktArrivalTime(Packet *pkt);
double updateDropTime(Packet* pkt);
void reportDrop(Packet* pkt);
robustOrangeTimer rorangeTimer_;
robustOrangeTimer rorangeStatus_;
void startTimer();//to begin timer after drop for orange_timer;
void reset();
int command(int argc, const char*const* argv);
Packet* deque();
PacketQueue *q_; /*underlying FIFO queue*/
int drop_front_;/*drop-from-front(rather than from tail)*/
int summarystats;
void print_summarystats();
int qib_ ;/*bool queue measured in bytes?*/
int mean_pktsize_;//configured mean packet size in bytes
int queue_limit;
double lastDrop;
double Tastrix_;
double drop_time_last_;
int byROrange;
```

```
int byDroptail;
double rorange_timer;
int rorange_limit;
struct Rflows flw[Max];
};
#endif
```

C++ code: source code

```
#ifndef  lint
static const char rcsid[] = "@(#) $header: /nsf/jade/vint/CVSROOT/ns-
2/queue/orange-robust.cc,v 1.15 2003/01/16 19:02:54 sfloyd Exp $ (LBL)" ;
# endif
#include "orange-robust.h"
#include <math.h>
#include <sys/types.h>
#include "config.h"
#include "template.h"
#include "random.h"
#include "flags.h"
#include "basetrace.h"
#include "hdr_qs.h"
#include <time.h>
#include <stdlib.h>
#include <sys/time.h>
double T2orange=0.0; /*Initializing T2,global; updated as T2 changes.*/
timeval tim;
static class RobustOrangeClass : public TclClass  {
public :
        RobustOrangeClass() :  TclClass("Queue/RobustOrange") {}
        TclObject* create(int,const char*const*) {
                return(new RobustOrange);
                }
} class_robustorange;
```

```
void robustOrangeTimer::expire(Event* evnt){que_->t_status();}
void RobustOrange::startTimer(){/*if(orangeStatus_.status()==TIMER_IDLE){
        }*/rorangeTimer_.resched(rorange_timer/1000);
}
int RobustOrange::t_status(){rorangeTimer_.status();}

RobustOrange::RobustOrange():rorangeTimer_(this), rorangeStatus_(this){
        q_ = new PacketQueue;
        pq_ = q_;
        bind_time("Tastrix_", &Tastrix_); // T*
        bind_time("drop_time_last_", &drop_time_last_);
        bind_bool("drop_front_", &drop_front_);
        bind_bool("summarystats_", &summarystats);
   bind_bool("queue_in_bytes_",&qib_);//boolean : q in bytes?
        bind("mean_pktsize_", &mean_pktsize_);
   bind("queue_limit_", &queue_limit);
        bind("rorange_limit_", &rorange_limit);
        bind("rorange_timer_", &rorange_timer);
}
void RobustOrange::reset()
{
    Queue::reset();
}
int RobustOrange::command(int argc, const char*const* argv) {
        if(argc==2) {
                if(strcmp(argv[1],"printstats")==0) {
                        print_summarystats();
                        return (TCL_OK);
                }
                }
        if(argc==3) {
                if(!strcmp(argv[1],"packetqueue-attach")) {
```

```
                delete q_;
                if(!(q_ =(PacketQueue*)TclObject::lookup(argv[2])))
  return (TCL_ERROR);
else {
            pq_ = q_;
            return (TCL_OK);
                }
                }
            }
            return Queue::command(argc,argv);
    }
void RobustOrange::initialize(){
        int i=0;
        for(i=0;i<Max; i++){
        f[i].findex=0;
        f[i].findicator=0;
        f[i].T1=0.0;
        }
}
void RobustOrange::enque(Packet* pkt) {
        if(pktArrivalTime(pkt)){
                updateDropTime(pkt);
                drop(pkt);
        }
        else {
                detectionFilter(pkt);
        }
return;
}
int RobustOrange::hashPkt(Packet* pkt){
        int hashIndex=0;  int ilevel=2;
        hdr_ip* iph=hdr_ip::access(pkt);
```

```
        unsigned int param1=(int)(iph->saddr());
        unsigned int param2=(int)(iph->daddr());
        hashIndex=((param1<<ilevel)+param2)%Max;
return hashIndex;
}
void RobustOrange::reportDrop(Packet* pkt) {
        double drop_time=updateDropTime(pkt);
        drop_time_last_=drop_time;
        return;
}
double RobustOrange::updateDropTime(Packet* pkt) {
        double t1= Scheduler::instance().clock();
        int dropIndex=0;  dropIndex=hashPkt(pkt);
        f[dropIndex].T1=(t1);
return t1;
}
int RobustOrange::pktArrivalTime(Packet* pkt){int i=0,index=0; int rtn=1;
        double Tnow= Scheduler::instance().clock();
        index=hashPkt(pkt);
        if(((Tnow - MAX(T2orange, f[index].T1) )<Tastrix_)){
        f[index].findicator-=1;
        } else{ f[index].findicator+=1; }
        if(f[index].findicator>=1){rtn=0;}
return  rtn;
}
int RobustOrange::detectionFilter(Packet* pkt ) {
   int curIn=0; int indx=0; int dropIndex=0;
        dropIndex=hashPkt(pkt); (tim.tv_sec) * 1000 + (tim.tv_usec) / 1000;

                if(summarystats)
                {
                        Queue::updateStats(qib_?q_->byteLength():q_->length());
```

```
                }
      int qlimBytes=queue_limit*mean_pktsize_; //queue limit in bytes
                int qlimBytes2=rorange_limit*mean_pktsize_; //((orange limit in
bytes
                if((!qib_&&(q_->length()+1)>=queue_limit)||
                (qib_ && (q_->byteLength() + hdr_cmn::access(pkt)->size())
>=qlimBytes)){
                //if the queue would overflow if we added this packet....
                if(drop_front_) {//remove from head of queue
                        q_->enque(pkt);
                        Packet* pp = q_->deque();
                        drop(pp);
                        gettimeofday(&tim, NULL); double t5=
Scheduler::instance().clock();
                        T2orange=abs(t5);
                        byDroptail++;
                } else {
                        drop(pkt);
                        double t4= Scheduler::instance().clock();
                        T2orange=abs(t4);
                        byDroptail++;
                }
                }  else {
    if((!qib_&&(q_->length()+1)>=rorange_limit) ||(qib_ && (q_->byteLength() +
hdr_cmn::access(pkt)->size())>=qlimBytes2)) {
                if(rorangeStatus_.status()==TIMER_IDLE){
              if(drop_front_){//remove from head of queue
                q_->enque(pkt);
                  Packet* pp = q_->deque();
                drop(pp);
                double t2= Scheduler::instance().clock();
                T2orange=abs(t2);
```

```
                byROrange++; startTimer(); //beginTimer() function.
        } else {
                                drop(pkt);
                                double t3= Scheduler::instance().clock();
                                T2orange=abs(t3);
                                byROrange++; startTimer();//beginTimer() function.
                        }
        }else {
             q_->enque(pkt);
       }else {
            q_->enque(pkt);
       }
    }
  }
return f[dropIndex].findicator;
}
Packet* RobustOrange::deque() {
        if(summarystats && &Scheduler::instance() !=NULL) {
        Queue::updateStats(qib_?q_->byteLength() : q_->length());
        }
        return q_->deque();
}
void RobustOrange::print_summarystats() {
        //printf("\nNumberof dropped packets by %d threshod: %d", orange_limit,
by/ROrange);
        //printf("\nNumber of dropped packets by droptail: %d", byDroptail);
                if(qib_) printf("(in bytes)");
        //printf("\ntime:%5.df\n", total_time_);
}
```

Appendix B

Awk trace files processing script. The calculation is from the beginning to the current average throughput, rather than the current instantaneous throughput for the last for loop part. The first for loop is for calculating the TCP throughput and the second for calculating the TCP/CBR drop. The fist and the second for loop does not compute the average TCP or CBR throughput or drop at the same time; needs to comment out one of them. They both produce a single number for average TCP throughput or drop. But for the case of the last for loop, it generates average throughput from the beginning to the current.

```
BEGIN {
  init=0;  i=0;
receivedPacketSum=0;startTime=0;endtime=0;dropPacketSum=0;
}
{
action = $1; time = $2; from = $3;
to = $4; type = $5; pktsize = $6;
flow_id = $8; src = $9; dst = $10;
seq_no = $11; packet_id = $12; #from==51 &&, flow_id==1
if(action=="r" && from==50 && to==51 && type=="tcp" ) {
        receivedPacketSum=receivedPacketSum + $6;
        pkt_byte_sum[i+1]=pkt_byte_sum[i]+ pktsize; # pktsize in bytes

        if(init==0) {
        startTime= time;
        start_time = time;
        init = 1;
        }       endTime= time;
        end_time[i] = time;
        i = i+1;
#time_value[i]=time; # && type=="cbr"
```

```
}
if(action=="d" && from==50 && to==51 && type=="tcp"){
dropPacketSum = dropPacketSum + $6;
        drop_pkt_byte_sum[i+1]=drop_pkt_byte_sum[i]+ pktsize;
        if(init==0) {
        startTime= time;
        start_time = time;
        init = 1;
        }
        endTime= time;
        end_time[i] = time;
        i = i+1;
        }
}
END {
print(receivedPacketSum, endTime);
receivedThroughput = (0.000008*receivedPacketSum / (endTime -1)); # received
packet is multiplied by 8 to change it to bits and
droppedThroughput = (0.000008*dropPacketSum / (endTime -1)); # divided by
1000000 to change it to mega unit
printf("receivedPacketSum=%10g\tdropPacketSum=%10g\tendTime=%10g\n",
receivedPacketSum,dropPacketSum,endTime);
printf("droppedThroughput=%.2f\n", droppedThroughput);
print("=======================================\n");
printf("receivedThroughput=%10g\tdroppedThroughput=%10g\n",
receivedThroughput, droppedThroughput);
sTime=start_time;
printf("receivedThroughput=%f\n",receivedThroughput);

{
printf(" \tendtime \t drop \tdrp_pkt_byte_sum[j] \ttTime\n");
printf("%.5f\t%.2f\n", end_time[0], 0);
```

```
        for(j=1 ; j<i ; j++){
        drop=(drop_pkt_byte_sum[j]/((end_time[j] - start_time)+0.01))*8/1000;#+
0.0001
        th = (pkt_byte_sum[j] / (end_time[j] - start_time))*8/1000; # divided by 1000
to change the unit to kilo bits per seconds.
        printf("%.5f\t%.2f\n", end_time[j], drop );
        }
printf("%.5f\t%.5f\n", end_time[i-1], 0);
}
}
```

Tcl Script

Tcl is a tool command language for writing the configuration of the topology. It is also used to provide parameters that are used while running the simulation.

```
#Create a simulator object
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
set nf [open out.nam w]
$ns namtrace-all $nf
set var [ open traceFileRobustREDPktMode.tr w ];
$ns trace-all $var
#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the NAM trace file
    close $nf
```

```
                # exec awk -f packetLoss1.awk traceFileRobustOrange.tr &
        exit 0
}
#Create 52 nodes
for {set i 0} {$i < 53} {incr i} {
        set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i < 50} {incr i} {
$ns duplex-link $n($i) $n(50) 10Mb 2ms DropTail
}
$ns duplex-link $n(50) $n(51) 5Mb 6ms RED/Robust;# Orange/Robust

$ns duplex-link $n(51) $n(52) 10Mb 2ms DropTail  ;# RED/Robust

Agent/TCP set packetSize_ 1000; #in bytes

for {set i 0} {$i < 30} {incr i} {
set tcp($i) [new Agent/TCP/Newreno]
$tcp($i) set class_ 2
$ns attach-agent $n($i) $tcp($i)
set sink($i) [new Agent/TCPSink]
$ns attach-agent $n(52) $sink($i)
$ns connect $tcp($i) $sink($i)
$tcp($i) set fid_ 1
set ftp($i) [new Application/FTP]
$ftp($i) attach-agent $tcp($i)
$ftp($i) set packetSize_ 1000; #in bytes
$ftp($i) set type_ FTP
}
for {set i 0} {$i < 20} {incr i} {
set a [expr $i+30]
```

```
set udp($i) [new Agent/UDP]
$ns attach-agent $n($a) $udp($i)

set null($a) [new Agent/Null]
$ns attach-agent $n(52) $null($a)
$ns connect $udp($i) $null($a)
$udp($i) set fid_ 2
set cbr($i) [new Application/Traffic/CBR]
$cbr($i) attach-agent $udp($i)
$cbr($i) set type_ CBR
$cbr($i) set packet_size_ 50; #in bytes
$cbr($i) set rate_ 0.25Mbps; # Rb PARAMETRESI 0.25Mbps
$cbr($i) set random_ true
}
set simulationfinishtime 60
for {set i 0} {$i < 30} {incr i} {
$ns at 1.0  "$ftp($i) start"
$ns at $simulationfinishtime "$ftp($i) stop"
}
set Ta 1.0; # Ta in sec
set Tb 0.20; # Tb in s but it is given as 200ms
#set count 0;
for {set i 0} {$i < 20} {incr i} {
#set count [expr {$count + 1}]
                # set j [expr {$j+$Ta}]
        for {set j 1} {$j < $simulationfinishtime} { set j [expr {$j+$Ta}]} {
                set artis [expr {$j+$Tb}]
                $ns at $j     "$cbr($i) start"
                #puts $j
                $ns at $artis "$cbr($i) stop"
                #puts $artis
        }
```

```
#puts $count
}
set bitis [expr {$simulationfinishtime+1}]
$ns at $bitis "finish"
$ns run
```

Instant Throughput Calculating Awk

Use to compute instantaneous throughput of TCP from the trace file generated by the simulator.

```
BEGIN {
        recv = 0
        currTime = prevTime = 0
        printf("# %10s %10s %5s %5s %15s %18s\n\n", \
             "flow","flowType","from","to","time","throughput")
}
{
        # Trace line format: normal
        if ($2 != "-t") {
                action = $1; time = $2; from = $3;
        to = $4; type = $5; pktsize = $6;
        flow_id = $8; src = $9; dst = $10;
        seq_no = $11; packet_id = $12;
        }
        # Trace line format: new
        # Init prevTime to the first packet recv time
        if(prevTime == 0)
                prevTime = time
        # Calculate total received packets' size
         if(action=="r" && from==50 && to==51 && type=="tcp") {
                # Rip off the header
```

```
                #hdr_size = pkt_size % pkt
                #pkt_size -= hdr_size
                # Store received packet's size
                recv += pktsize
                # This 'if' is introduce to obtain clearer
                # plots from the output of this script
                if((time - prevTime) >= tic*10) { # tic*10
                        printf("  %10g %10s %5d %5d %15g %18g\n", \
                                flow,type,from,to,(prevTime+1.0),0)
                        printf("  %10g %10s %5d %5d %15g %18g\n", \
                                flow,type,from,to,(time-1.0),0)
                print"(time - prevTime)="(time - prevTime);
                }
                currTime += (time - prevTime)
                #print "time=" time;
                #print "prevTime="prevTime;
                #print "currTime="currTime;
                if (currTime >= tic) {
                        print "recv="recv; print "currTime="currTime;
                        printf("  %10g %10s %5d %5d %15g %18g\n", \
                             flow,type,from,to,time,(recv/currTime)*(8/1000))
                        recv = 0
                        currTime = 0
                }
                prevTime = time
        }
}
END {
        printf("\n\n") }
```