إقرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

Enhanced RED Protocol Using Multipath Routing

تصميم بروتوكول RED مطور باستخدام التوجيه متعدد المسارات

أقر بأن ما اشتملت عليه هذه الرسالة إنما هي نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل، أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أية مؤسسة تعليمية أو بحثية أخرى.

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification

Student's name:	اسم الطالب: مزيد عايد الدبرمه
Signature	التوقيع: [illegible]
Date:	التاريخ: 27/6/2015

Islamic University, Gaza, Palestine
Research and Postgraduate Affairs
Faculty of Engineering
Computer Engineering Department

Enhanced RED Protocol Using Multipath Routing

Farid K. O. AL Abraq

Supervisor
Prof. Mohammad Mikki

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

May 2015

بسم الله الرحمن الرحيم

الرقم: ج س غ/35/ Ref ....

التاريخ: 2015/06/09 Date ....

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **فريد كايد عودة الابرق** لنيل درجة الماجستير في **كلية *الهندسة*** قسم **هندسة الحاسوب** وموضوعها:

تصميم بروتوكول RED مطور باستخدام التوجيه متعدد المسارات

Enhanced RED Protocol Using Multipath Routing

وبعد المناقشة التي تمت اليوم الثلاثاء 22 شعبان 1436هـ، الموافق 2015/06/09م الساعة الحادية عشرة صباحاً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

أ.د. محمد أمين مكي	مشرفاً و رئيساً	[illegible]
د. أيمن أحمد أبو سمرة	مناقشاً داخلياً	[illegible]
أ.د. سامي ~~سليم~~ سليم أبو ناصر	مناقشاً خارجياً	[illegible]

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في **كلية *الهندسة*/ قسم** وموضوعها: **هندسة الحاسوب**.

واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.

والله ولي التوفيق ،،،

مساعد نائب الرئيس للبحث العلمي وللدراسات العليا

[illegible]

أ.د. فؤاد علي العاجز

شئون البحث العلمي والدراسات العليا – الجامعة الإسلامية غزة – Research & Graduate Affairs

DEDICATION

To My Father, Mother

To My brothers,

To my wife,

To my Sisters,

To my Child's Anas, Zinab, Yosef,

To My Friends,

ACKNOWLEDGEMENTS

Praise is to Allah, the Almighty for having led me at every point of my biography. And best pray and peace on Prophet Mohammed.

*I would also like to take this chance to thank my research supervisor, **Prof. Mohammad Mikki**, for guiding and helping me throughout this research and other courses.*

*As well I would like to thank the discussion committee, **Dr. Aiman Abu Samra** and **Prof. Samy Abu Naser**, for their guides and commentaries.*

ملخص الدراسة

البوابات المحسنة عشوائية الكشف المبكر المقترحة (ERED) تتجنب الازدحام في علبة تبديل الشبكات. تكتشف البوابة بداية الازدحام عن طريق حساب متوسط حجم الطابور بعد تحويله إلى قائمة الانتظار الموازي الثانوي. البوابة يمكن أن تخطر الاتصالات من الازدحام سواء من خلال إسقاط وصول الحزم الى البوابة أو انتقائية توجيه مسار بديل. عندما يتجاوز متوسط حجم الطابور الحد الفاصل المحدد مسبقا، تقوم البوابات بتعليم او حذف كل الحزم الواصلة باحتمالية معينة، وهذا في قائمة الانتظار الموازي الثانوي، حيث الاحتمالية مبنية حقيقة على متوسط حجم الطابور.

تقوم بوابات ال ERED بالحفاظ على انخفاض متوسط حجم قائمة الانتظار في الوقت الذي تسمح بالتقطعات العرضية بين الحين والآخر من الحزم في قائمة الانتظار. أثناء الازدحام، احتمالية أن البوابة تعلم بورود اتصال معين يحد من نافذتها يتناسب تقريبا مع حصة هذا الصدد من عرض النطاق الترددي من خلال البوابة.
تم تصميم بوابات ERED لملازمة طبقة نقل بروتوكول التحكم في الازدحام مثل TCP . بوابة ERED لا يوجد لديها تحيز ضد الحركة المتقطعة وتجنب التزامن الشامل من العديد من الاتصالات التي تنقص نوافذها في نفس الوقت. بالمقارنة مع بروتوكول RED نحقق بعض المزايا:

1. تحقيق إنتاجية عالية، وانخفاض متوسط التأخير.
2. تحقيق انخفاض حذف الحزم.
3. تحقيق خاصية تجنب الازدحام.

ونحن نعتقد أن ERED قوية بما فيه الكفاية للنشر في أجهزة التوجيه. استخدام جهاز محاكاة omnett++ لتوضيح أداء بوابات ERED .

كلمات مفتاحية :
تجنب الازدحام، المسارات المتعددة التوجيه ، ERED ، RED ، TCP .

.

ABSTRACT

Our proposed Enhanced Random Early Detection (ERED) gateways for congestion avoidance in packet switched networks. The gateway detects incipient congestion by computing the average queue size after converting to secondary parallel queue. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or selecting alternative path routing. When the average queue size exceeds a preset threshold, the gateway drops or marks each arriving packet with a certain probability, this in secondary parallel queue, where the exact probability is a function of the average queue size.

ERED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection’s share of the bandwidth through the gateway.

ERED gateways are designed to accompany a transport layer congestion control protocol such as TCP. The ERED gateway has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their window at the same time. By comparison with RED protocol we realize some advantages:

1) Achieve high throughput and low average delay.

2) Achieve less drop packets.

3) Achieve congestion avoidance property.

We believe that ERED is sufficiently robust for deployment in routers. Simulator omnett++ 4.2.2 are used to illustrate the performance of ERED gateways.

Keywords: Congestion Avoidance, Multipath routing, RED, ERED ,TCP.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AOMDV	Ad-Hoc On Demand Multipath Distance Vector
AQM	Active Queue Management
ALOHA	Advocates of Linux Open-source Hawaii Association
avg	Average Queue Size
BSD	Berkely Software Distribution
CSP	Congestion Status Packet
EDAPR	Early congestion Detection and Adaptive Routing in MANET
EDCSCAODV	Early Detection of Congestion and Self Cure routing protocol for wireless Ad hoc networks
ERED	Enhancement Random Early Detection
LSMP	Link State Multi-Path
MANET	Mobile Ad hoc Networks
max_p	Maximum value for probability
max_{th}	Maximum threshold for queue
min_{th}	Minimum threshold for queue
MRED	Modified Random Early Detection
MRED	Modified Random Early Detection
NHN	Non-congested Neighbors
NRT	Neighbor Routing Table
OMNET	Object-oriented Modular discrete event Network simulation framework
Pa	Current packet-marking probability
PD	Proportional Derivative
PRT	Primary Routing Table
q	Current queue size
q_time	Start of the queue idle time
RED	Random Early Detection
TCP	Transmission Control Protocol
TOS	Type Of Service
w_q	Queue Weight

LIST OF ALGORITHMS

LIST OF FIGURES

1 CHAPTER 1: INTRODUCTION

This thesis implements an algorithm called ERED. ERED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the gateway.

By now we have seen enough layers of the network protocol hierarchy to understand how data can be transferred among processes across heterogeneous networks. We now turn to a problem that spans the entire protocol stack how to effectively and fairly allocate resources among a collection of competing users. The resources being shared include the bandwidth of the links and the buffers on the routers or switches where packets are queued awaiting transmission. Packets contend at a router for the use of a link, with each contending packet placed in a queue waiting its turn to be transmitted over the link [1]. When too many packets are contending for the same link, the queue overflows and packets have to be dropped or take another route as in enhanced protocol. When such drops become common events, the network is said to be congested.

Most networks provide a congestion control mechanism to deal with just such a situation. A mechanism, called random early detection (RED), invented by Sally Floyd and Van Jacobson in the early 1990s. We note that several modifications have since been proposed both by the inventors and by other researchers. However, the key ideas are the same in basic, and we propose a mechanism to avoid dropping packet save it at least and we learn it to select another route that have less than congestion .

For many years ago a lot of algorithms are implemented for finding the shortest path from a source to a destination, these algorithms work under assumption that the network is like a graph consist of vertices and links, most of those protocols pay attention to the link status like its length, however according to [2], another research [3] note that the bottlenecks in networks appear in nodes rather than links.

In real networks we should note the following:

- The propagation delay that is directly proportional to link length is actually very small.
- Compared with the link propagation delay, delays experienced in nodes may be very large.
- The number of parameters associated with a link is small.

Many function blocks are added to nodes, rather than links. We can admit that some of the nodal parameters can be reassigned to links but only in case no shared resources are used.

Many routing algorithms schemes try to find the best optimal path for the routing according to some criteria's, the best means that the traffic is always routed over a single path, that leads to waste the network resources and makes the links congested with traffic.

Multipath Routing is an alternative approach that distributes the traffic among several "good" paths instead of routing all traffic along a single "best" path.

Multipath routing is the routing technique of using multiple alternative paths through a network, which can yield a variety of benefits such as fault tolerance, increased bandwidth, or improved security. The multiple paths computed might be overlapped, edge-disjointed or node-disjointed with each other. Extensive research has been done on multipath routing techniques, but multipath routing is not yet widely deployed in practice [2]. Unlike traditional routing schemes that route all traffic along a single path, multipath routing strategies split the traffic among several paths in order to ease congestion. It has been widely recognized that multipath routing can be fundamentally more efficient than the traditional approach of routing along single paths. Yet, in contrast to the single-path routing approach, most studies in the context of multipath routing focused on heuristic methods [3].

Congestion control and resource allocation involve both hosts and network elements such as routers. In network elements, various queuing disciplines can be used to control the order in which packets get transmitted and which packets get dropped. The queuing discipline can also segregate traffic; that is, to keep one user's packets from unduly affecting another user's packets. At the end hosts, the congestion-control mechanism paces how fast sources are allowed to send packets. This is done in an effort to keep congestion from occurring in the first place, and should it occur, to help eliminate the congestion.

1.1 Problem Statement

In network elements, various queuing disciplines can be used to control the order in which packets get transmitted and which packets get dropped. To avoid dropping packet we suppose ERED protocol using multipath routing and make less than congestion and to deal with dropping packets. The dropping packets are a big problem in networks. This is done in an effort to keep congestion from occurring in the first place, and should it occur, to help eliminate the congestion [1].

1.2 Objectives

The goal of this thesis is to implement ERED(enhancement RED protocol using multipath routing) to make congestion avoidance and to verify its compatibility on network compared to the existing system.

We aim to achieve two objectives:

- The ERED protocol makes less than dropping of packets when we comparing with the default basic RED (random early detection) protocol.
- Using multipath routing property solve congestion problem when the packet dropped or the source notify by gateway, we take another route to solve it.

1.3 Scope and Limitations

The RED algorithm is a congestion avoidance technique highly used in modern communication networks for avoiding network congestion. Compared to existing algorithms, the RED system monitors network traffic loads in an effort to anticipate and avoid congestion at common network bottlenecks i.e. the system triggers before any congestion actually occurs. Moreover, RED uses AQM techniques currently deployed in large IP networks. It takes advantage of TCP's congestion control mechanism to avoid congestion in network where packets are dropped probabilistically prior to periods of high congestion.

The thesis aimed towards the realization of an enhanced RED protocol algorithm for

- Improving by minimizing dropping packets.
- Improving by using multipath routing property.
- Improving by making low average queue.
- Improving by the rate of operation for heavily congested.
- Implements the existing drop tail congestion avoidance algorithm and shows the relationship between *avg* and *Pa*.

1.4 Research Methodology

In our research, we intend to achieve our specific research objectives using the following methodology shown in Figure 1.1.

- **Research and Survey**:

  This includes reviewing the recent literature related to RED protocol and multipath routing. Based on the survey, we formulate the ERED protocol with multipath routing property.

- **RED protocol and multipath routing collections:**

  We collect largest freely public related RED protocols and recent publication papers.

- **Studying OMNET++ simulator:**

  We use OMNeT++ simulator version omnetpp-4.2.2 and omnetpp is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains:

  - Modeling of wired and wireless communication networks.
  - Protocol modeling.
  - Modeling of queueing networks.
  - Modeling of multiprocessors and other distributed hardware systems.
  - Validating of hardware architectures.
  - Evaluating performance aspects of complex software systems.
  - In general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages.

- **Design ERED protocol:**

  After reading and collection information about RED protocol, we design an enhancement protocol called ERED with multipath routing property.

- **Development of our proposed algorithm:**

  For this stage, we implement our algorithm using omnetpp-4.2.2 simulator and we will talk in details in chapter4.

- **Testing and validation:**

  For this stage, after finishing encoding stage we will make test and validate the ERED protocol, and make the results and analysis.

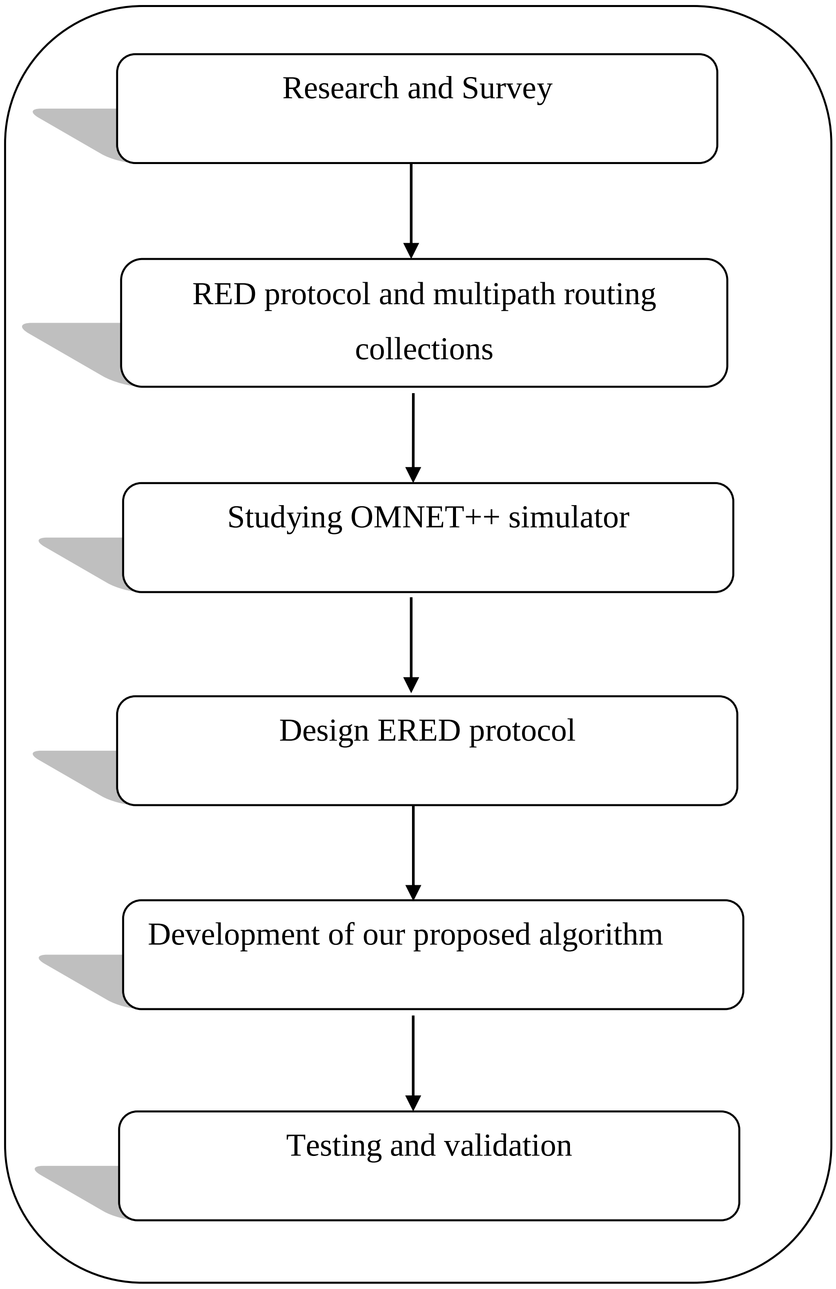
Research and Survey
RED protocol and multipath routing collections
Studying OMNET++ simulator
Design ERED protocol
Development of our proposed algorithm
Testing and validation

Figure 1.1 : Research Methodology

1.5 Research Format

The research report is organized as follows: Chapter 2 discusses the state of the art and literature survey. Chapter 3 includes the theoretical foundation of the research. Chapter 4 presents the proposed ERED protocol (enhancement on Red using multipath routing). Chapter 5 presents the experimental results and evaluation. Finally, Chapter 6 presents conclusions and future work.

2 CHAPTER 2: RELATED WORKS

In this chapter, related works that address the problem of dropping packets and avoid congestion by enhancement in RED protocol with multipath routing property are reviewed.

2.1 Random Early Detection Gateways for Congestion Avoidance

Several researchers have studied early random drop gateways as a method for providing congestion avoidance at the gateway

In [4], presents random early detection (RED) gateways for congestion avoidance in packet switched networks. The gateway detects incipient congestion by computing the average queue size. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or by setting a bit in packet headers. When the average queue size exceeds a preset threshold, the gateway drops or marks each arriving packet with a certain probability, where the exact probability is a function of the average queue size.

RED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the gateway. RED gateways are designed to accompany a transport-layer congestion control protocol such as TCP.

The RED gateway has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their window at the same time. Simulations of a TCP/IP network are used to illustrate the performance of RED gateways.

Hashem [5] discusses some of the shortcomings of random drop and drop tail gateways, and briefly investigates early random drop gateways. In the implementation of early random drop gateways in [5], if the queue length exceeds a certain drop level, then the gateway drops each packet arriving at the gateway with a fixed drop probability. This is discussed as a rough initial implementation. Hashem [5] stresses that in future implementations the drop level and the drop probability should be adjusted dynamically, depending on network traffic.

Hashem [5] points out that with drop tail gateways each congestion period introduces global synchronization in the network. When the queue overflows, packets are often dropped from several connections, and these connections decrease their windows at the same time. This results in a loss of throughput at the gateway. The paper shows that early random drop gateways have a broader view of traffic distribution than do drop tail or random drop gateways and reduce global synchronization.

It suggests that because of this broader view of traffic distribution, early random drop gateways have a better chance than drop tail gateways of targeting aggressive users. The conclusions in [5] are that early random drop gateways deserve further investigation. For the version of early random drop gateways used in the simulations in [6], if the queue is more than half full then the gateway drops each arriving packet with probability 0.02.

Zhang [6] shows that this version of early random drop gateways was not successful in controlling misbehaving users. In these simulations, with both random drop and early random drop gateways, the misbehaving users received roughly 75% higher throughput than the users implementing standard 4.3 BSD TCP.

The gateway congestion control survey [7] considers the versions of early random drop described above. The survey cites the results in which the early random drop gateway is unsuccessful in controlling misbehaving users [6]. As mentioned in [8], early random drop gateways are not expected to solve all of the problems of unequal throughput given connections with different roundtrip times and multiple congested gateways, when we take in our account the congestion in a network may occur at any interval[9].

In [7], the goals of early random drop gateways for congestion avoidance are described as “uniform, dynamic treatment of users (streams/flows), of low overhead, and of good scaling characteristics in large and loaded networks". It is left as an open question whether or not these goals can be achieved.

Among existing congestion control schemes, the benefits and the simplicity of the RED gateway algorithm make it an attractive solution for ATM networks [10]. RED is an active queue management scheme [11, 12], was proposed to solve the global synchronization problem in TCP connections sharing a congested router and to reduce the bias against bursty connections.

On other hand, mobile terminals start their transmissions at the beginning of each mini slot, in a way similar to a slotted ALOHA system, .with the difference that the channel can hold multiple simultaneous transmissions, leading to a slotted ALOHA spread spectrum system. In the related literature there exist several medium access control protocols that might be applied to this access interface - see for example [13], [14] and [15].

It was an improvement over the previous proposals such as random drop and early random drop [5–7]. It has been implemented in various routers and has been recommended for queue management by Internet Engineering Task Force (IETF), in order to reduce the increasing packet loss rates caused by an exponential increase in network traffic [12,16].

The behavior of RED is still not fully understood [17]; the relationship between its parameters and performance in terms of throughput is still being investigated. [11, 18, 20].

Most of the existing studies are obtained with simulations suggesting modifications to the RED algorithm.

For example, Feng et al. [21] proposed modified random early detection (MRED) gateways for congestion avoidance in TCP/IP networks, MRED aims at providing better control over the burstiness level while maintaining the advantages of RED. The numerical results demonstrate that MRED can achieve higher link utilization than RED without sacrificing its advantages in all simulation scenarios. This improvement becomes more significant under bursty traffic.

Sun et al. [22] proposed a new active queue management (AQM) scheme to improve the performance of the RED AQM. The new AQM is based on the proportional derivative (PD) control principle, and is called PD–RED. In PD–RED the authors introduced minimal changes to RED. They demonstrated the improvement in performance of PD–RED over the recently introduced Adaptive RED AQM by simulations.

Christiansen et al. [23] studied the effects of RED on the performance of Web browsing. They empirically evaluate RED across a range of parameter settings and offered loads. May et al. [24] modeled the throughput of the RED. They have shown that under heavy load, the throughput is inversely proportional to the load. Throughput can be increased by tracking the state of individual connections/links [18, 19, 25, 26] or by selecting appropriate parameters for RED [27–29].

Suter et al. [18] studied the throughput of RED under per-flow queue management for several cases. It is found that with a large number of TCP connections, the throughput of RED is generally low. Moreover, with a mixture of bursty and greedy sources, RED suffers from unfairness and low throughput. When TCP has to compete with more aggressive sources, or is used in asymmetric networks with a perpetually congested reverse path, RED'S throughput is very low.

To solve the problem of unfairness among links [10,30], Kim and Lee [25] studied the Fair Buffering RED (FB-RED) for running TCP over ATM connections, which takes advantage of ATM to improve the fairness characteristics of the RED gateway algorithm. The developed algorithm calculates the bandwidth-delay product of each connection and based on this value it ensures fair drop probabilities. Although FB-RED results in fairness among links, it needs to track the information for all the links. This results in scalability problem, which makes it impractical in a large network Lin and Morris [26] proposed Fair RED (FRED) to enhance the performance of RED gateways. The algorithm relies on usage of buffer spaces by the different traffic flows (per-active-flow accounting) to determine the drop rate of the flow. Although it achieves a fair drop rate for different flows, it needs to track the state of each flow which results in a scalability problem similar to those in [25].

To solve the scalability problem of FB-RED [25] and FRED [18], Ott et al. [19] proposed Stabilized Random Early Drop (SRED) which, like RED, preemptively discards packets with a load dependent drop probability when the buffer in a router gets congested. SRED can stabilize the buffer occupancy, over a wide range of load levels, independent of the number of active connections. SRED therefore overcomes the scalability issues of [25,26]. It, however, even with small number of traffic flows, suffers from low throughput.

Lakshman et al. [27] carried out a simulation of TCP/IP over ATM to study the throughput of RED. It was found that an exponential drop function is better than the single linear drop function of RED. However, an exponential drop function requires more computing power and is not easily implemented in hardware.

Feng et al. [28] showed that the effectiveness of RED depends, to a large extent, on the appropriate selection of the RED parameters. They showed that there is no single set of RED parameters that work well under different congestion scenarios and therefore they proposed an adaptive RED that self-parameterizes itself based on the traffic mix. However, adaptive determination of the RED parameters complicates buffer management of high speed router/gateways.

Parris et al. [29] pointed out that RED is not effective in the case of UDP sources. They proposed Classed Based Threshold (CBT), which sets the buffer thresholds for packet dropping according to their traffic types (TCP vs. UDP) and priority classes. Their scheme tags UDP traffic which has its own drop thresholds that are different from the thresholds used for TCP traffic. The performance of TCP traffic is thus protected in the presence of UDP traffic.

May et al. [17] studied the queuing delay and delay variance (jitter) of RED. It is found that RED has a large delay variance which is also very sensitive to the weight parameter w_Q of RED. The smaller the value of w_Q, the larger the delay variance. Further, for real-time application, RED'S delay is large. To study the stability of the RED algorithm, Firoiu and Borden [30] modeled RED as a feedback control system, and pointed out that RED will induce network instability and major traffic disruption if not properly configured.

2.2 Early congestion detection and adaptive routing in MANET

In [9], Ad hoc mobile networks are composed of mobile nodes communicating through wireless medium, without any fixed backbone infrastructure. In these networks, congestion occurs in any intermediate node when data packets travel from source to destination and they incur high packet loss and long delay, which cause the performance degradations of a network. This paper proposes an early congestion detection and adaptive routing in MANET called as EDAPR. Initially EDAPR constructs a NHN (non-congested neighbors) neighbors list and finds a route to a destination through an NHN node. All the primary path nodes periodically calculate its queue_status at node level. While using early congestion detection technique, node detects congestion that is likely to happen and sends warning message to NHN nodes. The ancestor NHN node is aware of this situation and finds an alternate path to a destination immediately by applying adaptive path mechanism. Thus, EDAPR improves performance in terms of reducing delay, routing overhead and increases packet delivery ratio without incurring any significant additional cost. The performance of EDAPR was compared with EDAODV and EDCSCAODV using the Ns-2 simulator. The result reveals significant improvement over EDAODV and EDCSCAODV routing schemes[9].

Early congestion detection and adaptive routing in MANET called EDAPR is a unicast routing protocol for MANET. It reduces network congestion, by minimizing the flooding of traffic, find a non-congested path between source and destination. We present this as complete design with an in-depth evaluation using EDAPR routing protocol. When the source host wants to transmit a data packet to a destination, EDAPR protocol first constructs a NHN set (non-congested neighbors) which connects both one hop and two hop neighbors and initiates the route discovery procedure by using NHN set to find a non-congested path to a destination [31,32]. After the route discovery, data packet is transmitted to the destination. Hence, the

EDAPR can reduce the overhead and automatically finds the non-congested path and decreases the flooding packets. EDAPR consists of the following components

1. NHN set construction.
2. Route discovery.
3. Adaptive routing.

- **Bidirectional path discovery**

Kumaran and Sankaranarayanan [9] view that : the source discovers the route to the destination; it broadcasts an RREQ packet toward the destination, the destination responds to the first arrived RREQ and sends back an RREP packet. The RREP will travel back in the path the RREQ previously travelled and adds a new entry in its routing table. This path becomes the primary route between the source and the destination. Each node has two routing tables, primary table (denoted as PRT) and alternate path routing table (denoted as ART). PRT is used to direct packets on the primary route, while ART directs packets on alternate path routes. If ART = 0 for a node that does not appear on an alternate path route of any connection .

A primary path of a node predicts its congestion status and periodically broadcasts a congestion status packet (CSP) to its neighbors with TTL =1.

The CSP packet contains the node's congestion status and a set of parameters (source S, destination D, previous ZoneI node P_ZoneI, Previous ZoneI hop count P_Zhop, Next ZoneI node N_ZoneI, Next ZoneI hop count N_Zhop), each for a destination appearing in the routing table.

- **Self cure routing**

Kumaran and Sankaranarayanan [9] view that : the source discovers the route to the destination; it broadcasts an RREQ packet toward the destination, the destination responds to the first arrived RREQ and sends back an RREP packet. The RREP will travel back in the path the RREQ previously travelled and adds a new entry in its routing table. This path becomes the primary route between the source and the destination. Each node has two routing tables, primary table (denoted as PRT) and neighbors table (denoted as NRT).

PRT is used to direct packets on the primary route. A primary path of a node predicts its congestion status and periodically broadcasts a congestion status packet (CSP) to is neighbors with TTL= 1. The CSP packet P contains the node's congestion status and a set of parameters (Source S, Destination D, Hop Count hop, Sequence Number Seq, Congestion Status Cong and Neighbors information N_list), each for a destination appearing in the routing table.

2.3 Congestion control and resource allocation

Early congestion detection technique Congestion in a network may occur at any interval, when the number of packets coming to a node exceeds its buffer capacity, the node becomes congested and starts losing packets. We can use a variety of metrics at a node to monitor congestion status. For instance, we can be based on the percentage of all packets discarded for lack of buffer space and the average queue length. We use an early congestion detection technique at a node to detect the congestion well in advance. An early congestion detection technique is a queue management algorithm with an optimization of random early detection (RED) model that makes use of direct measurement congestion status well in advance in a network [4].

In[1], say the Congestion control and resource allocation are two sides of the same coin. On the one hand, if the network takes an active role in allocating resources—for example, scheduling which virtual circuit gets to use a given physical link during a certain period of time—then congestion may be avoided, thereby making congestion control unnecessary. Allocating network resources with any precision is difficult, however, because the resources in question are distributed throughout the network; multiple links connecting a series of routers need to be scheduled. On the other hand, you can always let packet sources send as much data as they want, and then recover from congestion should it occur. This is the easier approach, but it can be disruptive because many packets may be discarded by the network before congestion can be controlled. Furthermore, it is precisely at those times when the network is congested—that is, resources have become scarce relative to demand—that the need for resource allocation among competing users is most keenly felt. There are also solutions in the middle, whereby inexact allocation decisions are made, but congestion can still occur and hence some mechanism is still needed to recover from it. Whether you call such a mixed solution congestion control or resource allocation does not really matter. In some sense, it is both.

Congestion control and resource allocation involve both hosts and network elements such as routers. In network elements, various queuing disciplines can be used to control the order in which packets get transmitted and which packets get dropped. The queuing discipline can also segregate traffic; that is, to keep one user's packets from unduly affecting another user's packets. At the end hosts, the congestion-control mechanism paces how fast sources are allowed to send packets. This is done in an effort to keep congestion from occurring in the first place, and should it occur, to help eliminate the congestion [1].

2.4 Multipath Routing

A novel view of routing protocols **[3]** concludes that nodes are more important than links when deciding routes. Taking those findings into consideration they have proposed a Link State Multi-Path (LSMP) routing protocol that shows many advantages over traditional link state routing protocols such as OSPF. LSMP has very simple algorithms. It easily supports local optimization (ILSMP) and can achieve much better traffic balancing in large and complex networks.

Recent studies show that the network would be more efficient and robust if routers could flexibly divide traffic over multiple paths. Unlike traditional routing schemes that route all traffic along a single path, multipath routing uses more than one path simultaneously to carry traffic between source and destination or ingress and egress pairs. Each path conveys a portion of data from the source, and the sink assembles the data fragments received from the various paths. If some paths fail to deliver data, then as long as the scale of failure is modest, the sink could still receive data using other routing paths. Therefore, the network would be more efficient and robust if routers could flexibly divide traffic over multiple paths **[33]**. Multipath routing could solve the algorithmic question of paths selection, avoid link congestion, and rebuild route quickly **[34]**.

Recently, traffic congestion avoidance has become a topic of recent investigation in multipath routing algorithms **[34]**.

Congestion control for Heterogeneous Network: [35]

A Heterogeneous network is a collection of wired, wireless and mobile networks. The convergence of this network is the success of the next generation networking. It has some challenge like:

(1) Maximize network resources utilization, and Minimize operational costs, delay and bandwidth on all the types of wired-wireless-mobile networks.
(2) Mix the QoS associated with Fixed, Mobile and Core networks.

Consider a heterogeneous network that is represented by a directed graph G = (V, E). Where V is the set of wired and wireless nodes and E is the set of links between different networks. Each link (i,j) € E is associated with a primary cost parameter c(i,j) j) and p is additive QoS parameters P = 1,2,…..p. The problem is to find a shortest path from source to destination.

1- A Heterogeneous network is a collection of wired, wireless and mobile networks. It is denoted as N = (N1, N2, N3 ……Nn).

N is the heterogeneous network. N1, N2 …. Nn may be wired, wireless and mobile node.

2- Multiple paths are denoted as P. P = P1, P2…..Pn n is a number of paths. Message passed over multiple paths. Load is shared between various links between source to destination

3- The set Ri=R1, R2, R3…..Rn represents the distribution of the load across the set of the resources. Total traffic sent from the source is R = ∑ Ri.

The research in **[35]** proposed a new improved QoS multipath routing algorithm for Ad-Hoc On Demand Multipath Distance Vector (AOMDV).

The first step of the algorithm is route disjoint nodes, so path disjointness will be discovery, Route Request packets are forwarded throughout the network. Hence multiple paths are established at destination node and at the intermediate nodes. The advertised hop count method at each node is helpful to get Multiple Loop free path. Route advertisements of the destination passed through hop count. An alternate path to the destination is selected by a node if the hop count is less than the advertised hop count for the destination.

In AOMDV algorithm is used to compute multiple paths during route discovery. The AOMDV protocol has two main components:

1- Establish and maintain multiple loop-free paths at each node of the network.
2- Compute link-disjoint paths

In **[34]**, a multipath routing is formulated as an optimization problem of minimizing network congestion. The authors have proposed a polynomial time algorithm that approximates the optimal solution for restricting a number of paths to be chosen and their quality which is based on the length of a path. Though it showed the efficiency of optimizing a congestion reduction, this algorithm employs centralized routing; hence, all routing decisions were determined at the source node. Although centralized routing might be simple and intuitive, it is not proper for mobile networks.

2.5 Summary

In this chapter, related works that address the problem of dropping packets and avoid congestion by enhancement in RED protocol with multipath routing property are presented.

In the next chapter, the fundamental concepts which represent the basis for understanding our research will be presented.

3 CHAPTER 3: THEORETICAL FOUNDATION

In this chapter, the fundamental concepts which represent the basis for understanding this research are presented. First, RED protocol is presented and then multipath routing is shown. Finally, an overview of used simulator performance and implementation is presented.

3.1 Random Early Detection (RED)

In [1], a mechanism called random early detection (RED), its scheme in each router is programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. RED, invented by Sally Floyd and Van Jacobson in the early 1990s, it have two properties. The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it implicitly notifies the source of congestion by dropping one of its packets. The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK. In case you haven't already guessed, RED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts (or some other means of detecting packet loss such as duplicate ACKs). As the "early" part of the RED acronym suggests, the gateway drops the packet earlier than it would have to, so as to notify the source that it should decrease its congestion window sooner than it would normally have. In other words, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on. Note that RED could easily be adapted to work with an explicit feedback scheme simply by marking a packet instead of dropping it, as discussed below on Explicit Congestion Notification.

Explicit Congestion Notification (ECN)

While current deployments of RED almost always signal congestion by dropping packets, there has recently been much attention given to whether or not explicit notification is a better strategy. This has led to an effort to standardize ECN for the Internet. The basic argument is that while dropping a packet certainly acts as a signal of congestion, and is probably the right thing to do for long-lived bulk transfers, doing so hurts applications that are sensitive to the delay or loss of one or more packets. Interactive traffic such as telnet and web browsing are prime examples. Learning of congestion through explicit notification is more appropriate for such applications.

Technically, ECN requires two bits; the proposed standard uses bits 6 and 7 in the IP type of service (TOS) field. One is set by the source to indicate that it is ECN capable, that is, able to react to a congestion notification.

The other is set by routers along the end-to-end path when congestion is encountered. The latter bit is also echoed back to the source by the destination host. TCP running on the source responds to the ECN bit set in exactly the same way it responds to a dropped packet. As with any good idea, this recent focus on ECN has caused people to stop and think about other ways in which networks can benefit from an ECN style exchange of information between hosts at the edge of the networks and routers in the middle of the network, piggybacked on data packets. The general strategy is sometimes called active queue management, and recent research seems to indicate that it is particularly valuable to TCP flows that have large delay-bandwidth products.

The second property is in the details of how RED decides when to drop a packet and what packet it decides to drop. To understand the basic idea, consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we could decide to drop each arriving packet with some drop probability whenever the queue length exceeds some drop level. This idea is called early random drop. The RED algorithm defines the details of how to monitor the queue length and when to drop a packet.

In the following paragraphs, we describe the RED algorithm as originally proposed by Floyd and Jacobson. We note that several modifications have since been proposed both by the inventors and by other researchers; some of these are discussed. However, the key ideas are the same as those presented below, and most current implementations are close to the algorithm that follows. First, RED computes an average queue length using a weighted running average similar to the one used in the original TCP timeout computation. That is, **AvgLen** is computed as

$$\mathbf{AvgLen} = (1 - \mathbf{Weight}) \times \mathbf{AvgLen} + \mathbf{Weight} \times \mathbf{SampleLen}$$

where $0 < \mathbf{Weight} < 1$ and **SampleLen** is the length of the queue when a sample measurement is made. In most software implementations, the queue length is measured every time a new packet arrives at the gateway. In hardware, it might be calculated at some fixed sampling interval.

The reason for using an average queue length rather than an instantaneous one is that it more accurately captures the notion of congestion. Because of the bursty nature of Internet traffic, queues can become full very quickly and then become empty again. If a queue is spending most of its time empty, then it's probably not appropriate to conclude that the router is congested and to tell the hosts to slow down. Thus, the weighted running average calculation tries to detect long lived congestion, as indicated in the right hand portion of Figure 3.1, by filtering out short term changes in the queue length.

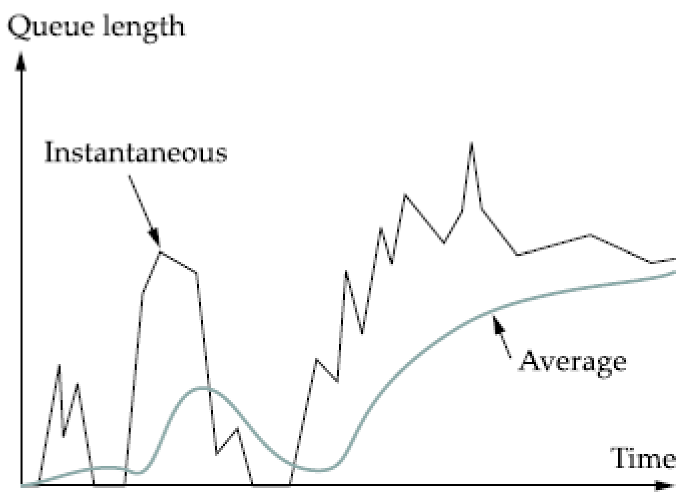

Figure 3.1: Weighted running average queue length[1] .

You can think of the running average as a low pass filter, where **Weight** determines the time constant of the filter. The question of how we pick this time constant is discussed below. Second, RED has two queue length thresholds that trigger certain activity: **MinThreshold** and **MaxThreshold**. When a packet arrives at the gateway, RED compares the current **AvgLen** with these two thresholds, according to the following rules:

```
if AvgLen ≤ MinThreshold
──→ queue the packet

if MinThreshold <  AvgLen <  MaxThreshold
──→ calculate probability P
──→ drop the arriving packet with probability P

if MaxThreshold ≤  AvgLen
──→  drop the arriving packet
```

That is, if the average queue length is smaller than the lower threshold, no action is taken, and if the average queue length is larger than the upper threshold, then the packet is always dropped. If the average queue length is between the two thresholds, then the newly arriving packet is dropped with some probability **P**. This situation is depicted in Figure 3.2. The approximate relationship between **P** and **AvgLen** is shown in Figure 3.3. Note that the probability of drop increases slowly when **AvgLen** is between the two thresholds, reaching **MaxP** at the upper threshold, at which point it jumps to unity. The rationale behind this is that if **AvgLen** reaches the upper threshold, then the gentle approach (dropping a few packets) is not working and drastic measures are called for, that is, dropping all arriving packets. Some research has suggested that a more smooth transition from random dropping to complete dropping, rather than the discontinuous approach shown here, may be appropriate.

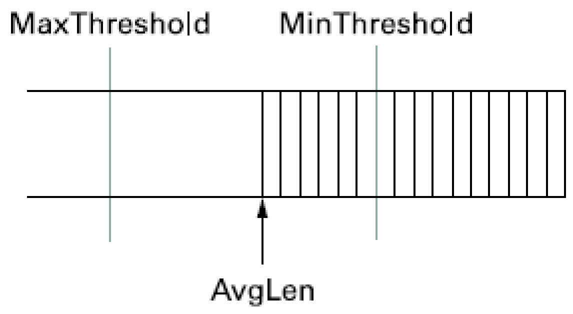

Figure 3.2: RED thresholds on a FIFO queue[1].

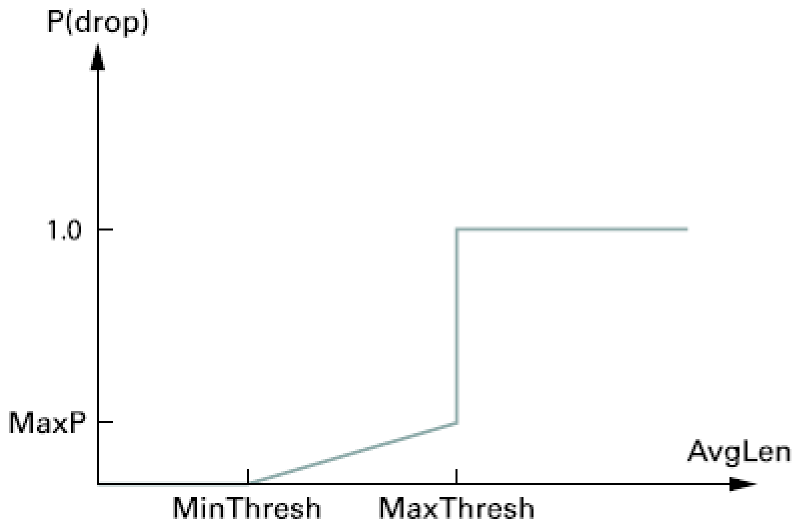

Figure 3.3: Drop probability function for RED[1].

Although Figure 3.3 shows the probability of drop as a function only of **AvgLen**, the situation is actually a little more complicated. In fact, **P** is a function of both **AvgLen** and how long it has been since the last packet was dropped. Specifically, it is computed as follows:

$$\mathbf{TempP = MaxP \times (AvgLen - MinThreshold) / (MaxThreshold - MinThreshold)}$$

$$\mathbf{P = TempP / (1 - count \times TempP)}$$

TempP is the variable that is plotted on the y-axis in Figure 3.3. count keeps track of how many newly arriving packets have been queued (not dropped) while **AvgLen**

has been between the two thresholds. **P** increases slowly as count increases, thereby making a drop increasingly likely as the time since the last drop increases.

This makes closely-spaced drops relatively less likely than widely-spaced drops. This extra step in calculating **P** was introduced by the inventors of RED when they observed that, without it, the packet drops were not well distributed in time, but instead tended to occur in clusters. Because packet arrivals from a certain connection are likely to arrive in bursts, this clustering of drops is likely to cause multiple drops in a single connection. This is not desirable, since only one drop per round-trip time is enough to cause a connection to reduce its window size, whereas multiple drops might send it back into slow start.

As an example, suppose that we set **MaxP** to 0.02 and count is initialized to zero. If the average queue length were halfway between the two thresholds, then **TempP**, and the initial value of **P**, would be half of **MaxP**, or 0.01. An arriving packet, of course, has a 99 in 100 chance of getting into the queue at this point. With each successive packet that is not dropped, **P** slowly increases, and by the time 50 packets have arrived without a drop, **P** would have doubled to 0.02. In the unlikely event that 99 packets arrived without loss, **P** reaches 1, guaranteeing that the next packet is dropped. The important thing about this part of the algorithm is that it ensures a roughly even distribution of drops over time.

Hopefully, if RED drops a small percentage of packets when **AvgLen** exceeds **MinThreshold**, the effect will be to cause a few TCP connections to reduce their window sizes, which in turn will reduce the rate at which packets arrive at the router. All going well, **AvgLen** will then decrease and congestion is avoided. The queue length can be kept short, while throughput remains high since few packets are dropped. Note that, because RED is operating on a queue length averaged over time, it is possible for the instantaneous queue length to be much longer than **AvgLen**. In this case, if a packet arrives and there is nowhere to put it, then it will have to be dropped. When this happens, RED is operating in tail drop mode. One of the goals of RED is to prevent tail drop behavior if possible.

The random nature of RED confers an interesting property on the algorithm. Because RED drops packets randomly, the probability that RED decides to drop a particular flow's packet(s) is roughly proportional to the share of the bandwidth that that flow is currently getting at that router. This is because a flow that is sending a relatively large number of packets is providing more candidates for random dropping. Thus, there is some sense of fair resource allocation built into RED, although it is by no means precise.

Note that a fair amount of analysis has gone into setting the various RED parameters—for example, **MaxThreshold**, **MinThreshold**, **MaxP**, and **Weight** all in the name of optimizing the power function (throughput-to-delay ratio). The performance of these parameters has also been confirmed through simulation, and the algorithm has been shown not to be overly sensitive to them. It is important to keep in mind, however, that all of this analysis and simulation hinges on a particular characterization of the network workload. The real contribution of RED is a mechanism by which the router can more accurately manage its queue length. Defining precisely what constitutes an optimal queue length depends on the traffic mix and is still a subject of research, with real information now being gathered from operational deployment of RED in the Internet. Consider the setting of the two thresholds, **MinThreshold** and **MaxThreshold**. If the traffic is fairly bursty, then **MinThreshold** should be sufficiently large to allow the link utilization to be maintained at an acceptably high level. Also, the difference between the two thresholds should be larger than the typical increase in the calculated average queue length in one RTT. Setting **MaxThreshold** to twice **MinThreshold** seems to be a reasonable rule of thumb given the traffic mix on today's Internet. In addition, since we expect the average queue length to hover between the two thresholds during periods of high load, there should be enough free buffer space above Max Threshold to absorb the natural bursts that occur in Internet traffic without forcing the router to enter tail drop mode. We noted above that Weight determines the time constant for the running average low-pass filter, and this gives us a clue as to how we might pick a suitable value for it. Recall that RED is trying to send signals to TCP flows by dropping packets during times of congestion.

Suppose that a router drops a packet from some TCP connection and then immediately forwards some more packets from the same connection. When those packets arrive at the receiver, it starts sending duplicate ACKs to the sender. When the sender sees enough duplicate ACKs, it will reduce its window size. So from the time the router drops a packet until the time when the same router starts to see some relief from the affected connection in terms of a reduced window size, at least one roundtrip time must elapse for that connection. There is probably not much point in having the router respond to congestion on timescales much less than the round-trip time of the connections passing through it. As noted previously, 100 ms is not a bad estimate of average round-trip times in the Internet. Thus Weight should be chosen such that changes in queue length over timescales much less than 100 ms are filtered out.

Since RED works by sending signals to TCP flows to tell them to slow down, you might wonder what would happen if those signals are ignored. This is often called the "unresponsive flow" problem, and it has been a matter of some concern for several years. Unresponsive flows use more than their "fair share" of network resources, and could cause congestive collapse if there were enough of them, just as in the days before TCP congestion control. There is a possibility that a variant of RED could drop more heavily from flows that are unresponsive to the initial hints that it sends; this continues to be an area of active research.

3.2 Multipath Routing

The other contribution is multipath routing. Calculating multiple paths between nodes to end host methods that utilize multiple paths to increase performance. Like dynamic metric single path routing, multipath routing offers potential performance increase over single path routing by better utilizing network resources. This section presents the multipath routing model and describes the various components needed for its implementation. Section 3.2.1 begins with multipath routing definitions to facilitate later discussions. Section 3.2.2 uses these definitions to discuss the advantages and disadvantages of multipath routing. Finally, Section 3.2.3 summarizes the necessary multipath routing components for increased network performance.

3.2.1 Multipath Routing Definitions

In [36], a glossary of terms to describe multipath routing models is presented in this section. The terms are structured to reflect the key parameters that define multipath networks and influence their performance. These parameters include

1. Basic definitions. The basic terms that describe a multipath routing model.

2. Path specification and calculation. Path specification describes the properties of the paths to be calculated between nodes. Example specifications are to calculate node disjoint paths, shortest K paths, and maximum flow paths between nodes. Path calculation is the actual algorithm that calculates the specified paths.

3. Multipath types. This parameter describes the paths a multipath routing algorithm provides between nodes. The two multipath types are multi-service paths where the routing algorithm provides different paths with different characteristics (example characteristics are high throughput and low delay) and multi-option paths where an algorithm provides multiple paths with the same characteristic. A routing algorithm that provides either or both multipath types is considered multipath routing algorithm.

4. Usage layer. The software layer responsible for using multiple paths to a given destination. This layer manages multiple paths by dictating which data packet should be sent on which path and when. Example layers in today’s Internet protocol stack are the network, transport, and application layers.

5. Multipath usage. The way an end host (or the usage layer of the end host) uses multiple paths to transmit data.

3.2.1.1 Basic definitions

The basic definitions of multipath routing are given here. First, a *multipath routing model* is defined as a routing model where the routing algorithms provide potentially multiple paths between node pairs and allows the end-hosts (or applications) to choose how to use these paths. We require that end-hosts have control over which path to use because this control offers the most flexibility in using multiple paths. This flexibility allows applications to use multiple paths in ways to best maximize their performance. Using this definition, dynamic metric, single path routing algorithms do not implement the multipath routing model; although these algorithms may route packets between a node pair on different paths, end-hosts do not control the path a particular packet will travel. For the same reason, networks with backup paths, such as telephone networks [37], do not implement the multipath routing model. A *multipath routing algorithm* refers to a routing algorithm that provides multiple paths between nodes so that data sent on a path travels that path through the network. A *path set* refers to the set of paths that a routing algorithm calculates for a particular network topology, and *multipath networks* are networks with routers that execute a multipath routing algorithm. That is, multipath networks are networks that offer multiple paths between node pairs.

3.2.1.2 Path Specification and Calculation

In order to calculate paths between nodes, one must first specify the characteristics of the paths to calculate. The different path characteristics depend on the intended use of the paths. For example, paths intended to maximize end-to-end throughput should be specified such that, for any node pair, the aggregate throughput obtain on multiple paths is maximized. In contrast, paths intended to minimize transmission delay should be specified such that, at any given time, there exist at least one low delay path between node pairs. Path specification specifies the characteristics a particular path set.

A path calculation algorithm is the algorithm that actually calculates the paths specified by path specification. A practical path calculation algorithm takes into account the operating resources and environmental constraints such as the distributed nature of a network. Examples of path calculation algorithms are Dijkstra's shortest path algorithm [38], Topkis's initial link disjoint paths algorithm [39], and the Bellman-Ford distributed shortest path algorithm [40, 41].

3.2.1.3 Multipath Types

Path type specifies the relationship among the paths a routing algorithm provides between node pairs. There are two path types: multi-service and multi-option. The first path type, multi-service paths, denotes paths between nodes that have different characteristics (i.e. different path specifications). Example services that a network could provide are low delay and high bandwidth path services. Since applications may have different demands on the network, providing paths with different characteristics allows applications to choose paths that best fit their communication demands. The second path type, multi-option paths, denotes the scenario where a routing algorithm provides multiple paths with the same path service. For example, an algorithm might provide four multi-option paths for the high bandwidth path service. That is, each end-host has, to each destination, four paths that can provide high bandwidth to a destination. Networks that support multi-service and/or multi-option paths are called multipath networks.

For example, a multipath network can be one that provides multiple service paths with only one path in each service (multi-service, single option), or one that provides only one path service with many paths within that service (single service, multi-option).

3.2.1.4 Usage Layer

Usage layer refers to the highest protocol layer responsible for managing multiple paths. The levels applicable in today's Internet protocol stack are the network, transport, and user/application layers. If the usage layer is the network layer, then it is the responsibility of this layer to decide which path a packet should travel and to do this transparently to the protocol layers above. Similarly, if the multipath usage layer is the transport layer, this means the transport layer has the freedom to send data on multiple paths.

The protocol layer that manages multiple paths needs to effectively use these paths to increase performance. The choice of usage layer depends on the tradeoffs between flexibility, performance, and the software engineering issues of implementing multipath management at a particular protocol layer.

3.2.1.5 Multipath Usage

Usage mode characterizes how multiple paths are used. There are two prototypical multipath usage modes: using paths concurrently or one at a time. The choice of which usage mode is application specific. For example, for applications interested in maximizing throughput, such as FTP, the right usage mode is to use all paths concurrently to obtain the aggregate bandwidth of all available paths. On the other hand, the appropriate usage mode for delay-sensitive applications, such as Telnet, is to use one path at a time, preferably the path with the lowest delay. Another application is one that needs to send urgent messages; here, the appropriate mode may be to send urgent messages on multiple paths concurrently, minimizing the message delivery time to the minimum time of all the paths used. In general, usage mode varies with application needs. In the foreseeable future, applications might require paths from different path services at the same time, thereby requiring different usage modes. For example, a Web session may have concurrent large file transfers and time critical user interactions. In this scenario, the appropriate usage mode may be to use paths from different services and multiplex data among these paths according to the type of the data.

One expects that as network applications become more sophisticated, their usage modes will increase in complexity as well. The multipath routing model is able to accommodate these complex applications because it does not place restrictions on usage modes. Thus, nodes are allowed to use the offered paths in ways that best fit their communication needs.

3.2.2 Multipath Routing Overview

This section presents a conceptual overview of the multipath routing model in order to describe the potential benefits and costs of providing multiple paths and to highlight the components needed to make multipath routing viable. This section is organized as follows. The first two subsections present the advantages and disadvantages of the multipath routing model and argue that the flexibility of the model offers significant network performance gains that outweigh the potential disadvantages. The latter two subsections, 3.2.2.3 and 3.2.2.4, present the key multipath implementation issues.

3.2.2.1 Multipath Advantages

The multipath routing model offers applications the ability to increase their network performance. Because of its multi-service paths, multi-option paths, and end-hosts freedom to use these paths, the model provides a flexible interface to network resources that enables applications with varying network demands to increase their performance.

In general, multipath performance improvements are obtained in two ways. First, multiservice paths allow an application to use paths within a service that best suit the application’s communication needs. Second, multi-option paths provide more network resources per path service, allowing applications to aggregate these path resources. These two general approaches are discussed below.

Providing the Right Paths

A multipath network with multi-service paths improves network performance because it allows applications to choose the paths that best suit their communication style. For example, an application such as FTP can improve its performance, measured in throughput, if it uses high-bandwidth service paths. Similarly, an interactive application, such as Telnet, can increase its performance, measured in response time, if it uses low-delay service paths. Because network performance depends on application demands, networks that provide paths with characteristics that fit these demands will be able to increase application network performance.

Since network demands vary with applications, the generality of a multi-service paths allows a multipath network to satisfy the needs of different applications. Providing the appropriate paths to increase performance will become more significant as the diversity of network applications increases. For example, in the foreseeable future, network applications such as IP telephony, real-time medical imaging, and video conferencing will become more prevalent. These applications need paths with very different characteristics from those of traditional applications. Specifically, many of these new applications need paths with QoS and real-time guarantees.

In this environment, a multi-service network might provide paths with different delivery guarantees, allowing applications to select the paths that best suit their needs. Notice that in a single path routing model, it is in general not possible to customize the one path between a node pair with characteristics suitable for all applications. In practice, single path routing algorithms calculate paths that compromise between throughput and delay [42].

Although the paths generated by this single path compromise are sufficient for today’s applications, it seems unlikely that these paths can effectively support future applications that need paths with radically different characteristics.

Aggregating Multiple Paths

Multi-option paths increase application performance by giving applications the freedom to use multiple paths within the same path service. Performance improvement is obtained in two prototypical ways: 1) aggregating resources of multiple paths and 2) selectively using the best available path. The descriptions of these two methods are given below.

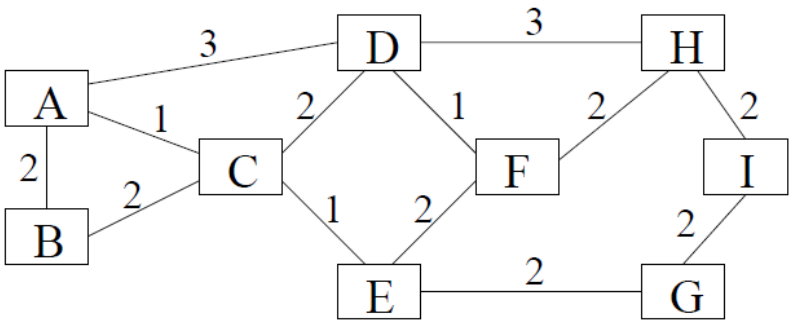

Figure 3.4: An example network topology. Boxes represent network nodes and edges represent links. The number above each link shows the link's capacity, and all links have uniform delay.

Using multiple paths, the simplest method to increase performance is to aggregate path resources. For example, consider node B maximizing its throughput to node D in Figure 3.4. In this figure, boxes denote nodes and the number above each link denotes the link's capacity. In a single, shortest path network, the maximum bandwidth between B and D is 2 units: either the path (B,A,D) or (B,C, D), but not both. However, a multipath network can provide both paths to B , allowing B to send data to D at 4 units of bandwidth. Similarly, an QoS application can increase the probability that the network satisfy its QoS request by combining the QoS reservations it makes on multiple paths. In general, this style of resource aggregation can benefit any application that can use multiple paths in parallel.

End-hosts can also use multi-option paths to increase performance by selectively using the available paths. For example, consider an application interested in low delay. In a single path model, the application has no choice but to incur the delay of the one path provided by the network to its destination. On the other hand, an application in a multipath network can attain lower communication delays by probing the delays among the available paths to the destination and then choosing the minimum delay path. Moreover, if low latency is one of the multi-service paths, a node can choose among the paths in that service category.

In summary, end-to-end performance is measured with respect to application demands; thus, different applications maximize their performance differently. These differences are accommodated by multipath routing model's multi-service and multi-option paths. This model offers a flexible interface to network resources that allows different applications to increase end-to-end performance compared to single path routing models.

3.2.2.2 Multipath Disadvantages

Routing algorithms play a major role in end-host resource usage because they allocate network resources (in terms of paths) between nodes. By construction, multipath routing algorithms offer more network resources to end-hosts than do single path routing algorithms, both to specific destinations and to sets of destinations. The previous section showed that the advantage of multipath routing is that these additional resources can be used to increase end-host performance; however, a potential disadvantage is that these same resources can also be used by a greedy or malicious user to deny other users their fair share of network resources. This section argues that although multipath routing may exacerbate resource denial, the problem is actually orthogonal to single or multipath routing.

For example, consider a FIFO datagram network that does not place any restrictions on how much data end-hosts can send. In this network, excessive resource consumption and denial of services cannot be prevented [43]. In today's Internet (a single path, FIFO datagram network), nodes can blast packets to random network destinations, consuming a significant amount of resources and drastically degrading overall network performance. Although malicious users in a FIFO multipath datagram network could deny more resources than in a FIFO single path data network, the fact remains that FIFO datagram networks do not have mechanisms to prevent resource abuses, whether single or multiple paths are provided.

In general, two approaches can prevent or reduce excessive resource denial: cooperative network communities and enforced network policies. In the cooperative approach, network users/applications agree not to consume excessive resources. The Internet uses this approach via congestion sensitive transport protocols (e.g. TCP [44]). These protocols attempt to share resources fairly by regulating their sending rates in response to the available bandwidth of the path they are using. Internet resource abuses are low because these protocols are used by most users [45]. The advantage of the cooperative community approach is that it does not require any network support. Internet routers do not need any additional mechanisms to prevent resource abuse because end-hosts voluntarily do not abuse resources. However, the disadvantage of this approach is that users (e.g. malicious ones) may not abide by the convention and thus can consume excessive resources. In order to control these users, a network needs mechanisms for admission and traffic control (described below). The important point is that the success or failure of the cooperative approach is independent of whether single or multiple paths are provided between nodes: if all hosts cooperate, excessive resource consumption will not occur in either single or multiple path networks. Similarly, if hosts do not cooperate, resource abuse can occur in both types of networks.

The second method to prevent/reduce resource abuse is for the network itself to enforce admission and traffic control policies [46, 47]. These networks enforce end-host resource usage by regulating the number of senders and/or the amount of data each sender sends. Regulating network traffic requires specific mechanisms in the network (e.g. in routers) to monitor and enforce end-host sending policies. For example, a pricing network implicitly discourages resource abuse by charging users for packets they transmit [48]. Users in these networks are unlikely to maliciously consume excessive resources because they have to pay for the resources they use.

Again, notice that the enforceability of traffic control in pricing networks is independent of whether single or multiple paths are provided between nodes. Users in a pricing network pay for packet transmission, regardless of whether they send their packets on different paths or on the same path. Other networks with mechanisms and policies to discourage/prevent excessive resource consumption can be found in [48,47,49,50,51]. Although the actual implementation of the admission and traffic control mechanisms may differ, the enforceability and effectiveness of admission and traffic control policies are orthogonal to the routing model.

The advantage of the network enforcement approach is that it does not rely on the cooperation of end-hosts; thus, it is much more robust and can prevent malicious users from abusing the network. The primary disadvantage is that it requires network mechanisms to regulate traffic and prevent abuse. These monitoring and enforcement mechanisms increase network cost and may decrease performance because of additional packet processing.

In summary, although multipath routing can exacerbate resource abuse in certain types of networks, the core issues of resource abuse are orthogonal to both single and multipath routing. That is, the fact that a network is prone to resource abuse is independent of whether it provides single or multiple paths between nodes. Because of this independence, we assume end users greedily maximize their resource usage but attempt to avoid network congestion. This is the same end-user assumption used in today's Internet.

3.2.2.3 Multipath Implementation Cost

The advantages of multipath routing come at a cost. Recall that routing is a two-step process: 1) calculating paths and 2) forwarding data on those paths. Implementing these two routing tasks incurs the following three cost categories:

1. The cost of computing multiple paths
2. Per packet path specification overhead in bytes
3. Router overhead of processing and forwarding data packets.

The first category corresponds to the cost of path computation, and the latter two to the cost of forwarding data on the computed paths. The three costs are described below.

Computing Multiple Paths

The first cost category, computing multiple paths, is measured in terms of routing messages (in bytes) needed to propagate routing information and the router CPU time needed to compute multiple paths. The number of messages and the amount of CPU usage depends on the path calculation algorithm and the base routing algorithm.

For example, the number of routing messages needed to compute multiple paths heavily depends on whether the routing algorithm is based on LS or DV. In an LS environment, routing message overhead is generally low because path computations are done with the knowledge of the network topology. Thus, the number of messages needed to disseminate topology information is the same independent of whether multiple or single paths are calculated. In contrast, DV based algorithms use routing messages (in the form of Distance Vector packets) as the mechanism to propagate paths; therefore, computing multiple paths generally requires more DV messages than computing single paths.

The amount of router CPU time to compute multiple paths has similar dependencies. In LS, because path computations are centralized, standard complexity analysis of centralized algorithms suffice to measure LS router CPU usage. As examples, calculating multiple K initial link disjoint paths takes $O(K*E*lg(E))$ and calculating the shortest K loop free paths takes $O(n*E*lg(E))$ [39,52, 53], where n is the number of nodes (or routers), and E is the number of network edges. In DV, the analysis is not so straightforward because path computations are distributed. In the worse case, the message and CPU complexities are exponential [54]. However, it has been shown that the average message and CPU complexities is $\Theta(nM^3 ln(M)^2)$ [55, 56], where M is the average number of neighboring routers.

Specifying Multiple Paths

Because there are multiple paths between nodes, every packet needs to specify not only its destination, but also a particular path to that destination. This is in contrast to single path networks where a destination address uniquely specifies a packet's path.

The second multipath cost category refers to this additional per packet cost of path specification. The specification cost is measured in the number of bytes needed in order to ensure that a packet travels its specified path. It is critical to minimize this cost because it is incurred on every data packet.

Forwarding Multiple Paths

Finally, the per packet path specification implies that more router processing is needed to forward each packet. The additional processing is needed for a router to decide, given the packet's destination address and path specifier, which outgoing link the router should forward the packet to. This additional processing may slow router forwarding speed and decrease network performance; thus it is critical to minimize this processing time. The cost of this additional processing is called the router forwarding overhead. Not surprisingly, this overhead is closely tied to how paths are specified in data packets.

3.2.2.4 Multipath Benefits

The previous subsections list the potential advantages, disadvantages, and costs of a multipath network; this subsection concludes the multipath discussion by addressing how the benefits of a multipath network can be obtained. To obtain multipath benefits, 1) multipath networks need to calculate appropriate paths, and 2) end-hosts need to effectively use these paths. These two properties are described below.

Path Calculation

The extent of performance improvement users can obtain from a multipath network depends on the quality of the calculated paths. For example, an application can obtain higher throughput only if the multiple paths calculated actually provide greater combined bandwidth than the one provided by a single path routing algorithm. Similarly, an application can increase its probability of establishing a QoS connection only if the calculated paths have, either individually or combined, a higher probability of satisfying QoS requests compared to the probability of a single path.

For example, consider the three paths (A_D_H_ I), (A_ C_ D_ H_ I), and (A_ C_ D_ F_ H_ I) from node A to node I in Figure 3.4. These three paths are not well chosen if A wishes to increase its network throughput to node I because all three paths traverse the same bottleneck link (H_ I) with capacity 2 . On the other hand, if a multipath network provides paths (A_D_H_ I), (A_B_C_E_ G_ I), and (A_D_F_E_ G_ I), then node A has 4 capacity units to node I.

As the example shows, a multipath network must provide the “right” paths in order for nodes to obtain higher performance gains, where the right paths depend on the applications that use the paths. In general, providing quality paths is a two-step process. First, determine the type of path services to calculate (path specification), then develop efficient algorithms to calculate them (path calculation algorithm). The complexities of both algorithms are linear in the number of paths calculated compared to computing single shortest paths.

Path Usage

The second component necessary for end-hosts to obtain increased performance in multipath networks is effective end-host usage of multiple paths. The fact that a network provides quality paths between nodes does not necessarily imply that nodes are able to effectively use these paths to maximize performance. This subsection shows the importance of effective multipath usage and its impact on network performance.

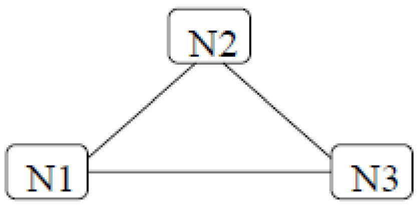

Figure 3.5: A simple three node network with full-duplex links. All links have equal bandwidth and latency

Consider the simple network in Figure 3.5 where all links have equal capacity and delay. Here, the multipath routing algorithm calculates two link disjoint paths (the one link and two link paths) between every node pair. Notice that the calculated paths provide the optimal paths for maximizing throughput. In this setting, the two paths calculated provide twice the capacity between nodes compared to single path routing; however, the effective throughput between nodes depends on how each node uses its paths.

To show the potential harm of naively using multiple paths, this three node network was simulated using TCP and a non-congestion aware, multipath striping protocol.

The multipath striping protocol clocks the sending of its data at full link capacity and distributes the data by striping them along the two available paths. That is, given N packets destined for destination D, the protocol sends packet $2i$ on the one-hop path and packet $2i+1$ on the two-hop path to D, $0 \leq i < N/2$. The single path TCP protocol sends all packets along the shortest path. In this experiment, nodes randomly select a neighbor and then send a burst of packets to that neighbor. The times between each burst are exponentially distributed.

Figure 3.6 shows TCP throughput versus the striping protocol’s throughput when all three nodes are sending data.

In Figure 3.6, the y-axis represents the average end-to-end throughput of all nodes as a percentage of link capacity, and the x-axis represents the aggregate sending rate normalized by the total network capacity.

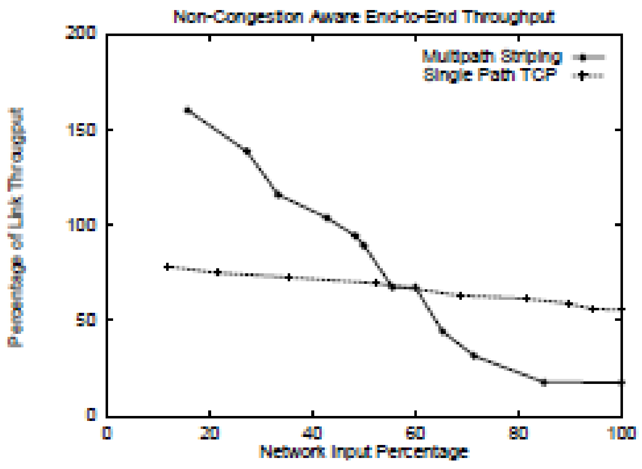

Figure 3.6: Simulation result using the network in Figure 3.5. The graph shows the performance of single path TCP versus a multipath striping protocol without congestion control.

First, the graph shows that TCP performance is very stable despite the increase in traffic. TCP does not achieve the maximum link bandwidth because its congestion and flow control mechanisms cautiously probes the network for available bandwidth and reduces its sending rate upon detection of congestion. On the other hand, the non-congestion aware multipath striping protocol achieves very high throughput at low network utilization levels.

The reason is that at low network loads, the amount of contention from other connections is low; therefore, multiplexing data between the one link and two link paths allows effective aggregation of path resources, resulting in higher throughput than the single path strategy.

However, the relative performance of these two strategies changes at higher network loads ($\geq$ 60%): here, the TCP's transmission strategy proves superior because at these utilization levels, packets in the multipath striping protocol experience enough contention from other connections to cause significant performance degradation (due to packet queuing in router buffers). This contention is due to packets traveling on the two link paths competing with packets from other connections. Furthermore, the contention increases as more packets are injected into the network, resulting in degradation of both aggregate and individual throughput. To address this problem, we develop a congestion-aware multipath transport protocol, MPTCP. The effectiveness of MPTCP is again compared against a single path TCP protocol on the same three node network. The results are shown in Figure 3.7. The x-axis and y-axis have the same representation as the previous figure. As this figure shows, MPTCP outperforms the single path protocol at all levels of network utilization.

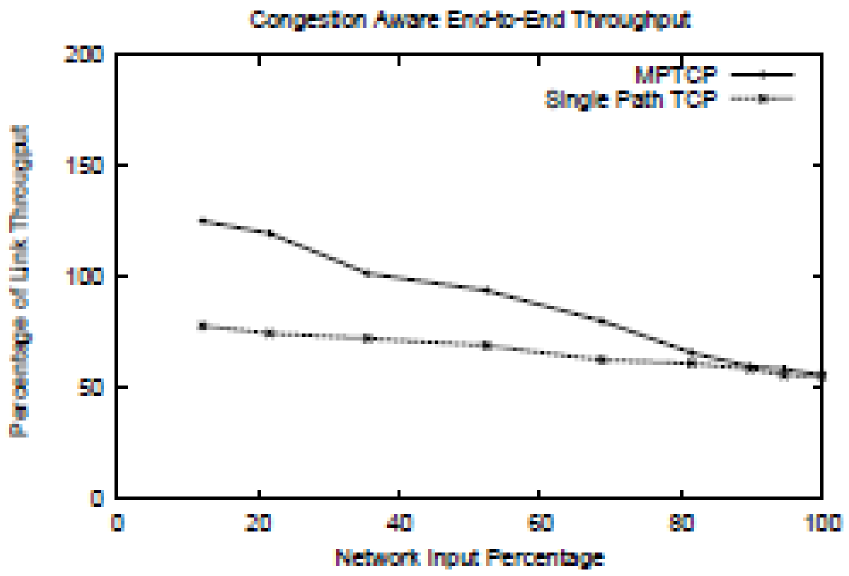

Figure 3.7: Simulation result using the network in Figure 3.5. The graph shows the performance of single path TCP versus MPTCP.

This demonstrates MPTCP’s ability to adapt to network conditions in order to increase throughput.

Compared to the multipath striping protocol, MPTCP does not achieve the same level of end-to-end throughput at low network utilization. This is because MPTCP is congestion aware and incrementally tests the network for available bandwidth. This cautious approach results in lower performance when the network is underutilized because it takes time for MPTCP to fully exploit available path bandwidth. However, the same cautious approach allows MPTCP to significantly outperform the striping protocol at high utilization levels. The conclusion of this section is that in order for end users to benefit from a multipath network, the network needs to provide not only the right paths, but the end-hosts also need to be able to take advantage of the additional paths. As shown by a naive multipath striping approach (Figure 3.6), wrongly using multiple paths can degrade not only end-to-end performance, but also the performance of other connections. This section also shows that correctly using multiple paths can increase network performance and avoid performance degradations.

3.2.2.5 Static and Dynamic Metric Multipath Routing

Like single path routing, multipath routing algorithms can use either dynamic or static metrics. In dynamic metric multipath routing, routing algorithms monitor link costs and recompute paths upon detection of link cost changes. The dynamic metric multipath routing model cannot specifically address, it can be implemented by traditional dynamic metric triggering mechanisms; thus the methods presented for static metric multipath routing are also applicable to dynamic metric multipath routing. The fundamental difference between multipath routing and single path dynamic metric routing is that end-hosts in multipath networks control the use of its paths on a much finer time and path granularity. Thus, given appropriate paths and end host protocols, multipath end-hosts can dynamically detect poor path performance, and then switch and use other paths that provide better performance. For example, consider an application that wishes to minimize its communication delay to its destination, and the multipath routing algorithm calculates link disjoint paths. In single path dynamic metric routing, if the path to the destination is congested, then the application will incur the delay caused by the congestion, unless the routing algorithm recompute a better path. In multipath routing, however, the application can dynamically switch and use other paths to avoid the congestion without any router intervention.

Thus, on small time scales, the ability to control path usage allows end-hosts in multipath networks to dynamically adjust to transient traffic patterns. Because end-hosts can adapt to small time-scale traffic patterns, a dynamic metric multipath routing algorithm does not need to recompute paths in fine time granularities. Consequently, these routing algorithms should recompute paths that consider large time-scale traffic patterns. For example, a dynamic metric multipath routing algorithm might monitor traffic patterns for weeks and then recompute paths based on the gathered statistics.

3.2.3 Multipath Routing Summary

While it is clear that the static metric multipath routing model offers many advantages over single path routing models, it is unclear whether enough benefits could be extracted to offset the cost of their implementation. In short, in order to make multipath routing viable, the following questions need to be resolved:

1. What paths should be calculated between nodes and how?

2. How should routers efficiently provide multiple paths in a distributed routing environment?

3. How should end-hosts use multiple paths to gain higher performance?

The first question deals with the potential gains of a multipath network. As illustrated in Section 3.2.2.4, one of the necessary criteria for multipath networks to increase end to-end performance is to calculate the right paths.

The second question deals with the cost of providing multiple paths between nodes. The main cost of implementing multipath routing is solving the packet forwarding problem: how to efficiently forward packets to the same destination but on different paths? The novel solution developed uses routing overhead linear in the number of paths between nodes and has constant per packet path specification overhead. This low overhead is achieved by requiring that paths calculated by a multipath routing algorithm satisfy the *suffix matched* property.

The last question is how end-hosts should best use a multipath network in order to increase their performance. As demonstrated in the previous section, the way in which multiple paths are used has a dramatic impact on individual and aggregate performance. A congestion aware multipath transport protocol, MPTCP, is developed that effectively uses multiple paths to increase throughput.

3.3 What Is OMNeT++?

OMNeT++ is an object-oriented modular discrete event network simulation framework[57]. It has a generic architecture, so it can be (and has been) used in various problem domains:

- modeling of wired and wireless communication networks
- protocol modeling
- modeling of queueing networks
- modeling of multiprocessors and other distributed hardware systems
- validating of hardware architectures
- evaluating performance aspects of complex software systems
- in general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages.

OMNeT++ itself is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is a component architecture for simulation models. Models are assembled from reusable components termed modules. Well written modules are truly reusable, and can be combined in various ways like LEGO blocks.

Modules can be connected with each other via gates (other systems would call them ports), and combined to form compound modules. The depth of module nesting is not limited. Modules communicate through message passing, where messages may carry arbitrary data structures. Modules can pass messages along predefined paths via gates and connections, or directly to their destination; the latter is useful for wireless simulations, for example. Modules may have parameters that can be used to customize module behavior and/or to parameterize the model's topology. Modules at the lowest level of the module hierarchy are called simple modules, and they encapsulate model behavior. Simple modules are programmed in C++, and make use of the simulation library.

OMNeT++ simulations can be run under various user interfaces. Graphical, animating user interfaces are highly useful for demonstration and debugging purposes, and command-line user interfaces are best for batch execution.

The simulator as well as user interfaces and tools are highly portable. They are tested on the most common operating systems (Linux, Mac OS/X, Windows), and they can be compiled out of the box or after trivial modifications on most Unix-like operating systems.

OMNeT++ also supports parallel distributed simulation. OMNeT++ can use several mechanisms for communication between partitions of a parallel distributed simulation, for example MPI or named pipes. The parallel simulation algorithm can easily be extended, or new ones can be plugged in. Models do not need any special instrumentation to be run in parallel – it is just a matter of configuration. OMNeT++ can even be used for classroom presentation of parallel simulation algorithms, because simulations can be run in parallel even under the GUI that provides detailed feedback on what is going on.

OMNEST is the commercially supported version of OMNeT++. OMNeT++ is free only for academic and non-profit use; for commercial purposes, one needs to obtain OMNEST licenses from Simulcraft Inc.

3.4 Summary

In this chapter, the fundamental concepts which represent the basis for understanding this research are presented. Firstly, the fundamental concepts of RED are described and then multipath routing is shown. Finally, an overview of used simulator performance and implementation is presented.

In the next chapter, the proposed ERED approach will be presented. Also, all steps of the proposed ERED approach using algorithms and diagrams will be described.

4 CHAPTER 4: THE PROPOSED ERED APPROACH

In this chapter, the proposed ERED approach is presented and all steps of the proposed ERED approach using algorithms and diagrams are described. Firstly, an overview of the basic RED protocol is presented. Secondly, diagram and detailed algorithm is shown. Then proposed ERED is presented, the diagram and algorithm of basic queue and secondary parallel queue are presented. Finally, Dijkstra algorthim is shown, it used to select shortest path.

4.1 Overview of Random Early Detection (RED)

A protocol, called random early detection (RED), its scheme in each router is programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. RED as shown in Figure 4.1, it have two property. The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it implicitly notifies the source of congestion by dropping one of its packets. The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK. In case you haven't already guessed, RED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts (or some other means of detecting packet loss such as duplicate ACKs). As the "early" part of the RED acronym suggests, the gateway drops the packet earlier than it would have to, so as to notify the source that it should decrease its congestion window sooner than it would normally have[1]. In other words, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on.

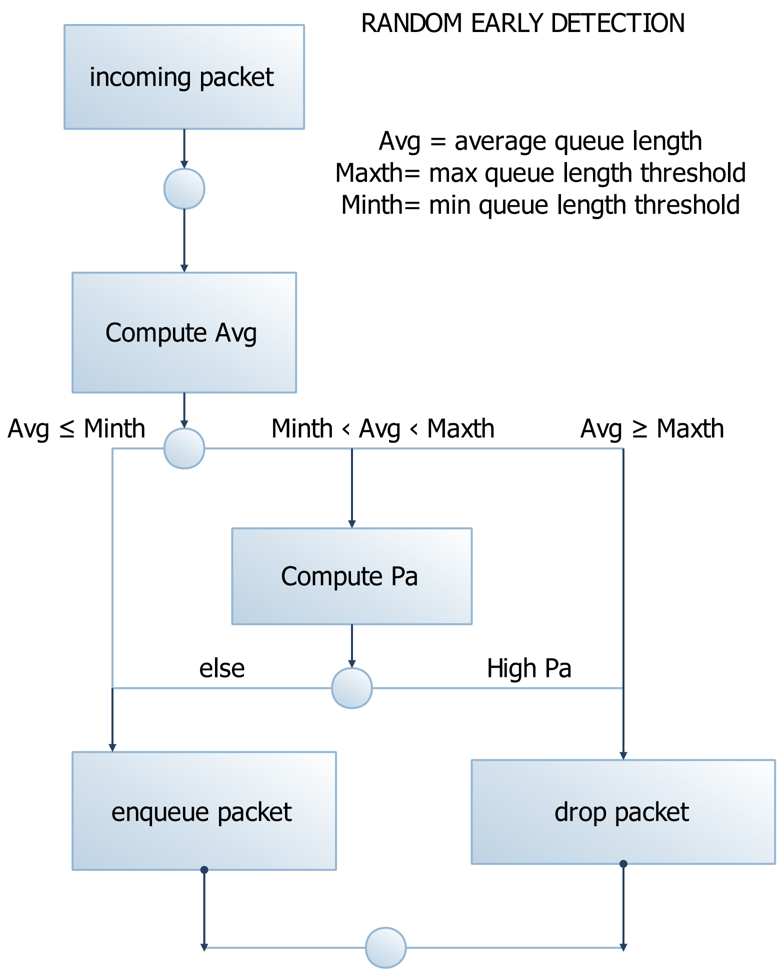

Figure 4.1 : Random Early Detection (RED)

Figure:4.1 Show the basic random early detection , when the packet incoming, the first step, calculate average queue size using a low-pass filter. Thus, the short-term increases in the queue size that result from bursty traffic or from transient congestion do not result in a significant increase in the average queue size. The low-pass filter is an exponential weighted moving average (EWMA):

$$avg \leftarrow (1-wq)avg+wq*q$$

The weight *wq* determines the time constant of the low-pass filter.

The second step, compare average queue size (avg) with *minth and maxth thresholds by* setting *minth* and *maxth values previously*. The optimal values for *minth* and *maxth* depend on the desired average queue size. If the typical traffic is fairly bursty, then *minth* must be correspondingly large to allow the link utilization to be maintained at an acceptably high level. For the typical traffic in our simulations, for connections with reasonably large delay bandwidth products, a minimum threshold of one packet would result in unacceptably low link utilization. The optimal value for *maxth* depends in part on the maximum average delay that can be allowed by the gateway. The RED gateway functions most effectively when *maxth-minth* is larger than the typical increase in the calculated average queue size in one roundtrip time. A useful rule of thumb is to set *maxth* to at least twice *minth*.

The finally step " decision ", the initial packet-dropping probability *pb* is calculated as a linear function of the average queue size as follows:

$$Pb \leftarrow maxp(avg-minth)/(maxth - minth)$$

The parameter *maxp* gives the maximum value for the packet-marking probability *pb*, achieved when the average queue size reaches the maximum threshold.

The pseudecode for detailed RED algorithm is described in Algorithm 4.1 below.

```
Initialization:
          $avg \leftarrow 0$
          $count \leftarrow -1$
for each packet arrival
calculate the new average queue size $avg$:
          if the queue is nonempty
               $avg \leftarrow (1-wq)avg+wq * q$
          else
               $m \leftarrow f(time - q\_time)$
               $avg \leftarrow avg *(1 - wq) \wedge m$
if $minth < avg < maxth$
          increment $count$
          calculate probability $Pa$:
          $Pb \leftarrow maxp(avg-minth)/(maxth - minth)$
          $Pa \leftarrow Pb/(1-count * Pb)$
          with probability $Pa$:
          mark the arriving packet
          $count \leftarrow 0$
else if $maxth \leq avg$
```

```
            mark the arriving packet
                  $count \leftarrow 0$
            else $count \leftarrow -1$
when queue becomes empty
            $q\_time \leftarrow time$

Saved Variables:
$avg$   : average queue size
$q\_time$: start of the queue idle time
$count$ : packets since last marked packet

Fixed parameters:
$wq$ : queue weight
$minth$: minimum threshold for queue
$maxth$: maximum threshold for queue
$maxp$: maximum value for $Pb$
Other:
$Pa$  : current packet-marking probability
$q$   : current queue size
$time$: current time
$f(t)$: a linear function of the time $t$
```

Algorithm 4.1: Pseudecode For Detailed RED algorithm

4.2 THE PROPOSED ERED APPROACH

In this section we will describe our enhancement protocol for RED protocol, we will show the basic queue of ERED and secondary parallel queue we illustrate it by using diagrams and algorithms.

4.2.1 Basic Queue of ERED

Our enhancement protocol for RED protocol, we called it ERED, its scheme contain two queue, the basic queue and secondary queue. We will describe the work of basic queue as shown in Figure 4.2 .The work of basic queue is similar to basic queue RED protocol but when queue decide to drop packet, the packet dropped will go to secondary parallel queue, and the secondary queue makes its calculation , we will describe diagram and pseudocode later.

In each router, the scheme of ERED must programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. The first is that rather than explicitly sending a congestion notification message to the source, ERED is implemented such that it implicitly convert to other parallel queue called secondary parallel queue, it implemented as the basic queue but it difference mechanism when reach to drop –as in basic RED- or (when reach high Pa or $maxth \leq avg$ called it drop sate) as in ERED. When the secondary parallel queue reach to drop sate the source implicitly change the path routing. The source after notified by the subsequent timeout or duplicate ACK it will change the path routing, if it found else the packet will be dropped. In case you haven't already guessed, ERED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts (or some other means of detecting packet loss such as duplicate ACKs) as in RED protocol.

The Steps in Figure 4.2 are described as follows:

- The packet comes to basic queue.
- The average queue length of basic queue is calculated.
- After average queue length calculation, the comparison will be achieved.
- If avg≤ minth or Pa accepted it will enter basic queue else the packet will go to secondary parallel queue, Algorithm 4.2 presents pseudecode for basic queue of ERED as shown below.

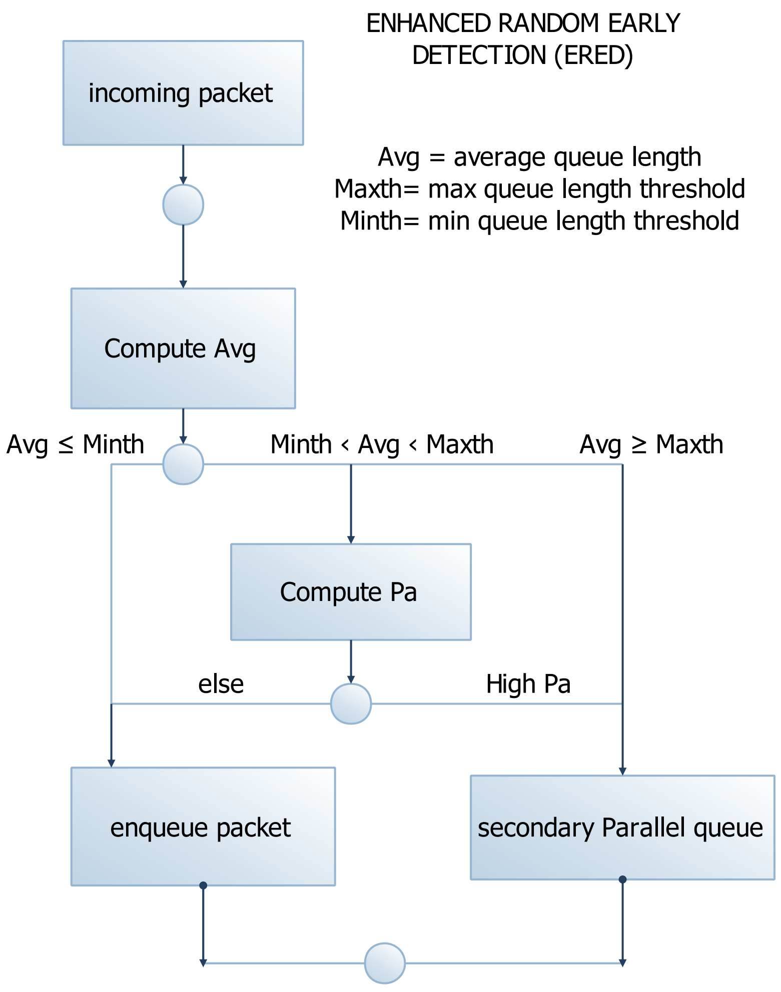
ENHANCED RANDOM EARLY DETECTION (ERED)
incoming packet
Avg = average queue length
Maxth= max queue length threshold
Minth= min queue length threshold
Compute Avg
Avg ≤ Minth
Minth ‹ Avg ‹ Maxth
Avg ≥ Maxth
Compute Pa
else
High Pa
enqueue packet
secondary Parallel queue

Figure 4.2: Basic Queue of ERED

```
Initialization:
          $avg \leftarrow 0$
          $count \leftarrow -1$
for each packet arrival
calculate the new average queue size $avg$:
          if the queue is nonempty
               $avg \leftarrow (1-wq)avg+wq*q$
          else
               $m \leftarrow f(time - q\_time)$
               $avg \leftarrow avg*(1 - wq) \wedge m$
if $minth < avg < maxth$
          increment $count$
          calculate probability $Pa$:
          $Pb \leftarrow maxp(avg-minth)/(maxth - minth)$
          $Pa \leftarrow Pb/(1-count * Pb)$
          with probability $Pa$:
          go to secondary parallel queue
          $count \leftarrow 0$
else if $maxth \leq avg$
          go to secondary parallel queue
               $count \leftarrow 0$
          else $count \leftarrow -1$
when queue becomes empty
          $q\_time \leftarrow time$
```

- **Note:**

The declared variables are described as in Algorithm (4.1)

Algorithm 4.2: Pseudecode For Basic Queue of ERED

4.2.2 Secondary parallel Queue of ERED

ERED scheme contains two queues, the basic queue and secondary queue. We will describe the work of secondary queue as shown in Figure 4.3 .The work of secondary queue is similar to basic queue but when queue reach to Pa high it decide to drop packet or change the path routing after checking another shortest path routing called alternative path routing.

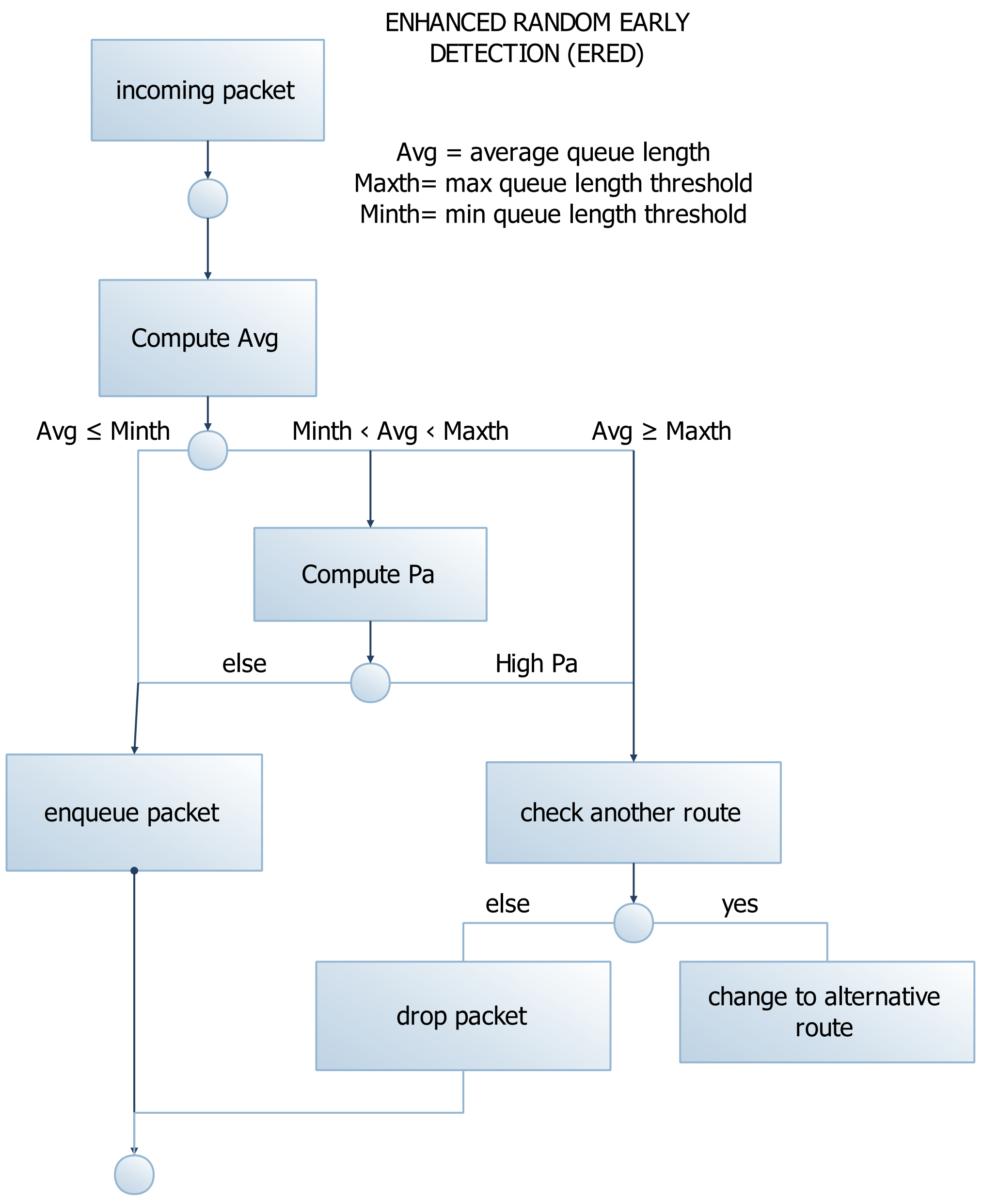
ENHANCED RANDOM EARLY DETECTION (ERED)
incoming packet
Avg = average queue length
Maxth= max queue length threshold
Minth= min queue length threshold
Compute Avg
Avg ≤ Minth
Minth ‹ Avg ‹ Maxth
Avg ≥ Maxth
Compute Pa
else
High Pa
enqueue packet
check another route
else
yes
drop packet
change to alternative route

Figure 4.3: Secondary parallel Queue of ERED

In each router, the scheme of ERED must programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. The first is that rather than explicitly sending a congestion notification message to the source, ERED is implemented such that it implicitly convert to other alternative path using Dijkstra algorithm.

When the probability reach high Pa or *maxth* ≤ *avg* called it drop sate as before, the source implicitly change the path routing. The source after notified by the subsequent timeout or duplicate ACK it will change the path routing, if it found else the packet will be dropped.

The Steps in Figure 4.3 are described as follows:

- The packet income to secondary queue.
- The average queue length of secondary queue is calculated.
- After average queue length calculation, the comparison will be achieved.
- If avg≤ minth or Pa accepted it will enter secondary queue else the packet will go to alternative path or packet will be dropped, Algorithm 4.3 presents Pseudecode For Secondary parallel Queue of ERED as shown below.

```
Initialization:
          $avg \leftarrow 0$
          $count \leftarrow -1$
for each packet arrival
calculate the new average queue size $avg$:
          if the queue is nonempty
               $avg \leftarrow (1-wq)avg+wq*q$
          else
               $m \leftarrow f(time - q\_time)$
               $avg \leftarrow avg*(1 - wq) \wedge m$
if $minth < avg < maxth$
          increment $count$
          calculate probability $Pa$:
          $Pb \leftarrow maxp(avg-minth)/(maxth - minth)$
          $Pa \leftarrow Pb/(1-count * Pb)$
          with probability $Pa$:
                  check if found another route
                     change routing to alternative route
                  else
                  drop packet
          $count \leftarrow 0$
```

```
else if $maxth \leq avg$
                    check if found another route
                        change routing to alternative route
                    else
                        drop packet

               $count \leftarrow 0$
          else $count \leftarrow -1$
when queue becomes empty
          $q\_time \leftarrow time$
```

- **Note:**

  The declared variables are described as in Algorithm (4.1)

Algorithm 4.3: Pseudecode For Secondary parallel Queue of ERED

4.2.3 Dijkstra Algorithm

Dijkstra algorithm is used to determine the shortest distance (or the least effort / lowest cost) between a start node and any other node in a graph. The idea of the algorithm is to continuously calculate the shortest distance beginning from a starting point, and to exclude longer distances when making an update [58]. It consists of the following steps:

1. Initialization of all nodes with distance "infinite"; initialization of the starting node with 0
2. Marking of the distance of the starting node as permanent, all other distances as temporarily.
3. Setting of starting node as active.
4. Calculation of the temporary distances of all neighbor nodes of the active node by summing up its distance with the weights of the edges.

5. If such a calculated distance of a node is smaller as the current one, update the distance and set the current node as antecessor. This step is also called update and is Dijkstra's central idea.

6. Setting of the node with the minimal temporary distance as active. Mark its distance as permanent.

7. Repeating of steps 4 to 7 until there aren't any nodes left with a permanent distance, which neighbors still have temporary distances.

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                    // Initialization
        dist[v] := infinity                        // initial distance from source to vertex
                                                   v is set to infinite
        previous[v] := undefined                   // Previous node in optimal path from
                                                   source
    dist[source] := 0                              // Distance from source to source
    Q := the set of all nodes in Graph             // all nodes in the graph are
                                                   unoptimized - thus are in Q
    while Q is not empty:                          // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:                  // where v has not yet been removed
                                                   from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                       // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

Algorithm 4.4: Dijkstra Algorithm: Short terms and Pseudocode[58]

4.3 Summary

In this chapter, the proposed ERED approach is presented and all steps of the proposed ERED approach using algorithms and diagrams are described. An overview of the basic RED protocol is presented, diagram and detailed algorithm are shown. Then this proposed ERED is described, the diagrams and algorithms of basic queue and secondary parallel queue are shown. Finally, Dijkstra algorthim is presented, it used to select shortest path.

In the next chapter, the experiments carried out to realize and evaluate the proposed ERED will be presented and discussed.

5 CHAPTER 5: RESULTS AND DISCUSSION

In this chapter, the experimental results are presented and analyzed to provide evidence that this proposed ERED approach can enhance the affective of RED protocol, making less dropping packet and avoid congestion. The enhancement makes the ERED constrain the oscillation and become more robust and adaptable. The chapter is ordered as following. Firstly, an overview of the basic ERED protocol is presented. Secondly, experimental environment and the implementation of ERED are shown and the basic queue and secondary parallel queue are presented. Also, the experimental results, discussion and comparison this proposed approach with RED protocol are presented. Finally, the evaluation quality of ERED approach is presented.

5.1 Overview of Enhanced Random Early Detection (ERED)

Our proposed enhanced random early detection (ERED) gateways for congestion avoidance in packet switched networks. The gateway detects incipient congestion by computing the average queue size after converting to secondary parallel queue. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or by selecting alternative path routing. When the average queue size exceeds a preset threshold, the gateway drops or marks each arriving packet with a certain probability, this in secondary parallel queue, where the exact probability is a function of the average queue size. ERED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the gateway. ERED gateways are designed to accompany a transport-layer congestion control protocol such as TCP. The ERED gateway has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their window at the same time. Simulator omnett++ 4.2.2 are used to illustrate the performance of ERED gateways.

5.2 Experimental Environment

Proposed ERED protocol is implemented using OMNET++ program of version 4.2.2. The OMNeT++ 4.x Integrated Development Environment is based on the Eclipse platform, and extends it with new editors, views, wizards, and additional functionality, etc.

We performed all programming and coding ERED on a personal computer with a platform of Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz (4 CPUs), ~2.5GHz, the kind of Windows edition is Windows 7 Ultimate with system type 64-bit operating system.

5.3 Implementation of ERED

The efficient implementations of ERED gateways is considered . We show that the ERED gateway algorithm can be implemented efficiently. In addition, the ERED gateway algorithm is not tightly coupled to packet forwarding and its computations do not have to be made in the time-critical packet forwarding path. Much of the work of the ERED gateway algorithm, such as the computation of the average queue size and of the packet dropping probability Pa, could be performed in parallel with packet forwarding, or could be computed by the gateway as a lower priority task as time permits. This means that the ERED gateway algorithm need not impair the gateway's ability to process packets, and the ERED gateway algorithm can be adapted to increasingly-high-speed output lines.

If the RED gateway's method of marking packets is to set a congestion indication bit in the packet header, rather than dropping the arriving packet, then setting the congestion indication bit itself adds overhead to the gateway algorithm. However, because RED gateways are designed to mark as few packets as possible, the overhead of setting the congestion indication bit is kept to a minimum. This is unlike DECbit gateways, for example, which set the congestion indication bit in every packet that arrives at the gateway when the average queue size exceeds the threshold [4].

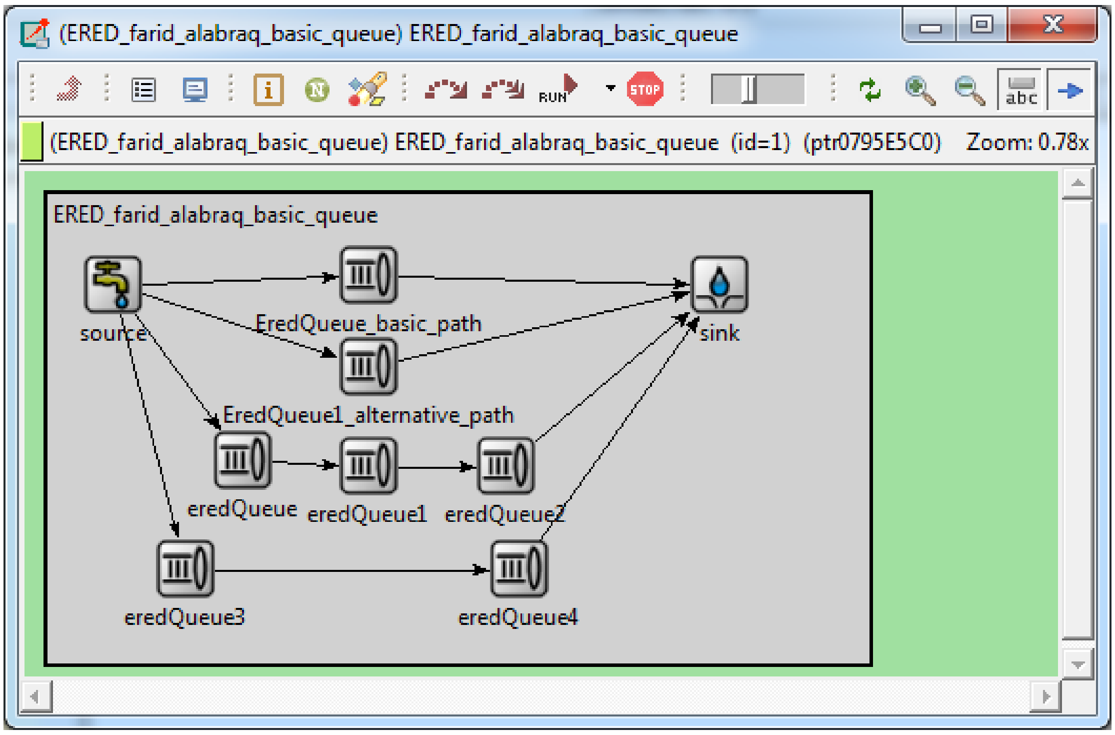
(ERED_farid_alabraq_basic_queue) ERED_farid_alabraq_basic_queue
RUN
STOP
abc
(ERED_farid_alabraq_basic_queue) ERED_farid_alabraq_basic_queue (id=1) (ptr0795E5C0) Zoom: 0.78x
ERED_farid_alabraq_basic_queue
source
EredQueue_basic_path
sink
EredQueue1_alternative_path
eredQueue
eredQueue1
eredQueue2
eredQueue3
eredQueue4

Figure 5.1: The proposed ERED system in omnet++

The proposed ERED system as in Figure 5.1 contains the source , sink as destination, queues (the basic queue with parallel queue) and alternative paths, before build our system, we build EREDqueue.cc and EREDqueue.h in the package of basic queue and EREDqueue.ned, it described below.

EREDqueue.ned: NED language topology description(s) (.ned files) that describe the module structure with parameters, gates, messages, etc. NED files can be written using any text editor, but the OMNeT++ IDE provides excellent support for two-way graphical and text editing. The parameters we use it in our proposed system are w_q =0.002 , minth =5, maxth = 15, maxp = 0.02 and packet rate = 150 arrivals per sec.

Message definitions (.msg files). We can define various message types and add data fields to them. OMNeT++ will translate message definitions into full-fledged C++ classes.

EREDqueue.cc and EREDqueue.h are simple module sources. They are C++ files, with .h/.cc suffix. Take in your account, the simulation system provides the following components: Simulation kernel. This contains the code that manages the simulation and the simulation class library. It is written in C++, compiled into a shared or static library and User interfaces: OMNeT++ user interfaces are used in simulation execution, to facilitate debugging, demonstration, or batch execution of simulations. They are written in C++, compiled into libraries.

Simulation programs are built from the above components. First, .msg files are translated into C++ code using the opp_msgc. program. Then all C++ sources are compiled and linked with the simulation kernel and a user interface library to form a simulation executable or shared library. NED files are loaded dynamically in their original text forms when the simulation program starts.

5.4 Experimental Result , Discussion and Comparison with RED

This section evaluates the effectiveness and performance of the enhanced RED algorithm by simulation. Omnet++ is still used as simulator. The control parameters are w_q =0.002 , min_{th} =5, max_{th} = 15, max_p = 0.02 and packet rate = 150 arrivals per sec.

The evolution of the instantaneous queue size and average queue size controlled by enhanced RED. The ERED constrains the queue size oscillations of RED, which may result in unacceptably large queue lengths and hence a large variance in delays for the flows going through. The Figure 5.2 and 5.3 depict the traces of the drop probability, which reflect the relationship between the drop probability and the average queue size (the initial transient process is omitted), just as above analysis, the standard RED stiffly adjusts the dropping probability, but the enhanced RED has more flexibility in choosing the dropping probability to adapt the unpredictable network condition. The standard RED only strictly comply with the restriction forced by the simple and stiff control profile. Figure 5.3 show the relationship between drop probability and the average queue length and Pa it appear as linear. From Figure 5.3, drop packets will start at ~7.8 until the average queue size reach to 15.

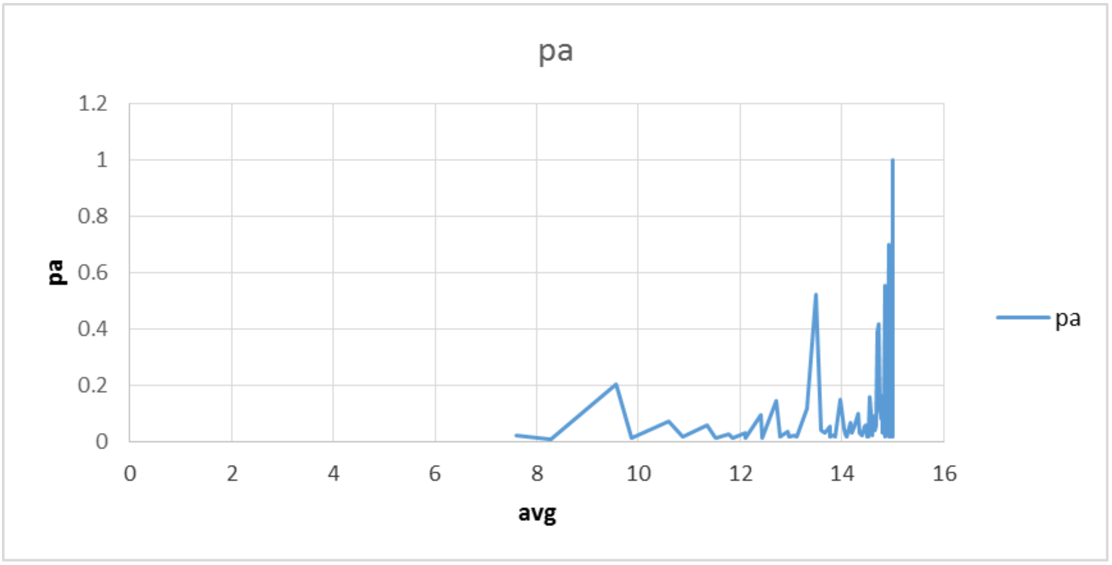

Figure 5.2: Relationship between the drop probability and the average queue size

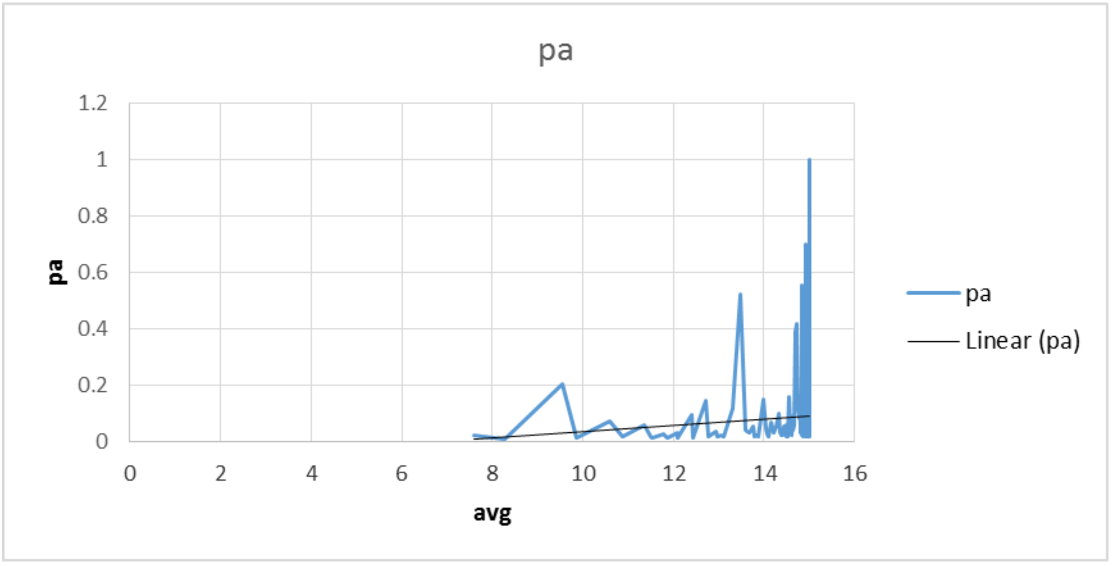

Figure 5.3: Relationship between the drop probability and the average queue size with linear Pa property

When we run our application, we get some reselt and relations. We fixed parameters as we takled above, however , Figure 5.4 show the relationship between the average queue size and time, it appear that the average queue size increasing with increasing time. The avg is between min_{th} and max_{th}.

By controlling the average queue size before the gateway queue overflows, RED gateways could be particularly useful in networks where it is undesirable to drop packets at the gateway.

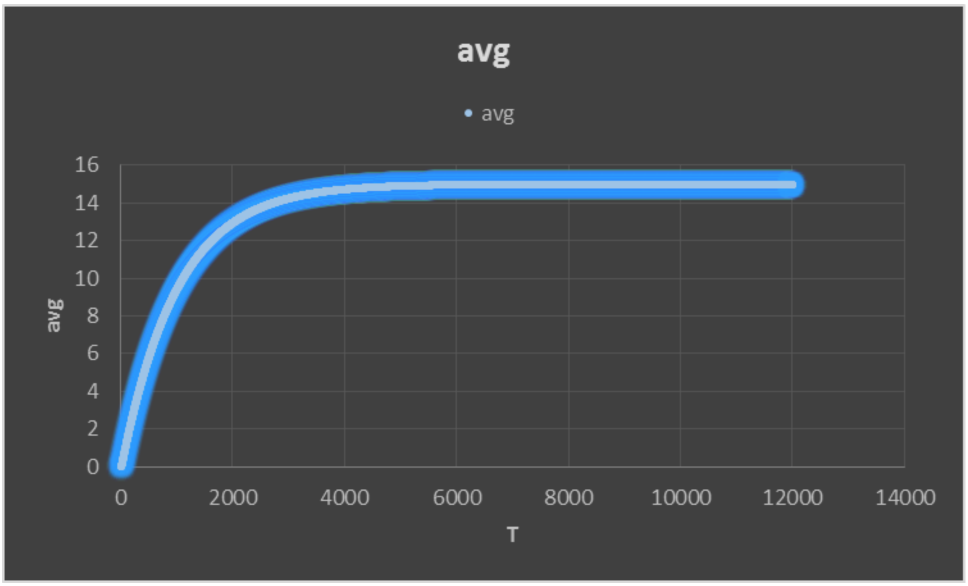

Figure 5.4: Relationship between the average queue size and time

Figures 5.5 and 5.6 show the relationship between the drop probability and time, it appear that the drop probability increasing with increasing time. The Pa is between 0 and 1.

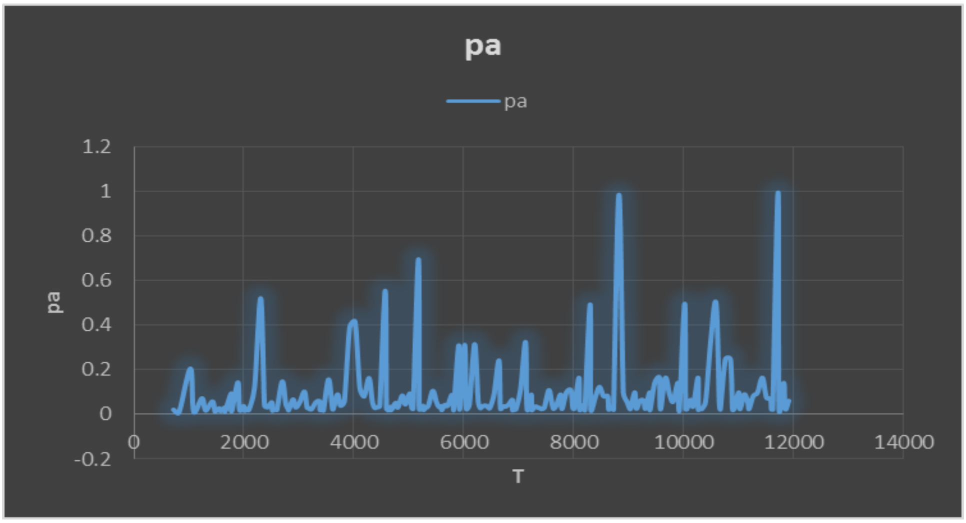

Figure 5.5: Relationship between the drop probability and time(continuas line)

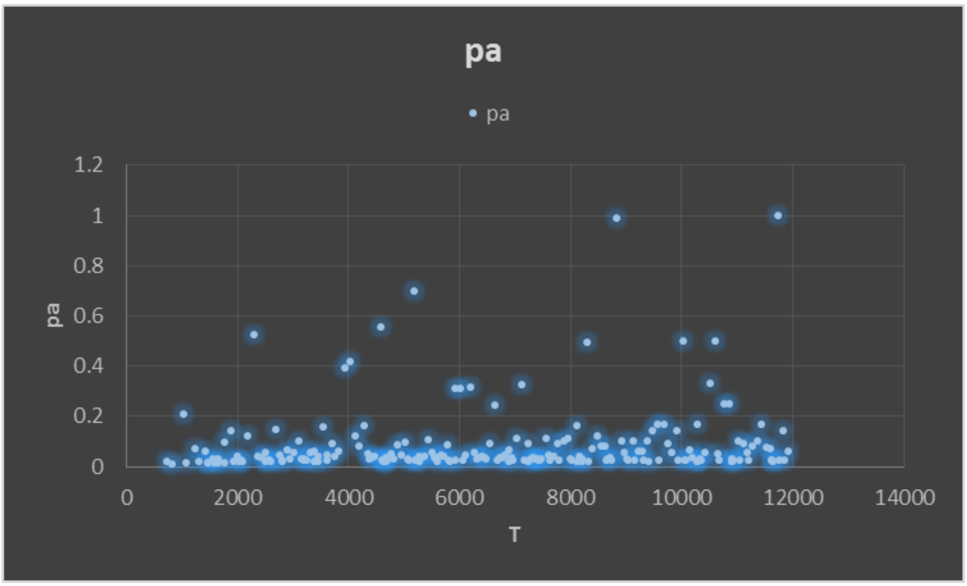

Figure 5.6: Relationship between the drop probability and time(discrete)

5.5 Evaluation Quality of ERED

In addition to the design goals for the congestion avoidance schemes [44], [59] described how our goals have been met by ERED gateways.

Congestion Avoidance – ERED allows for queue congestion to be managed before a critical overflow point is reached, parallel queue or secondary parallel queue help us to avoid congestion. Also, keeping the queue size lower decreases delay for those packets that are not dropped. Select another route or alternative path makes congestion avoidance property realized.

Simplicity - The ERED gateway algorithm could be implemented with moderate overhead in current networks using omnetpp simulator and simple to design and implementation using background on java and C++.

Appropriate time scales - After notifying a connection of congestion by marking a packet, it takes at least a roundtrip time for the gateway to see a reduction in the arrival rate. In ERED gateway the time scale for the detection of congestion roughly matches the time scale required for connections to respond the congestion. ERED gateways do not notify connections to reduce their windows as a result of transient congestion at the gateway.

No Global Synchronization - By marking packets for early discard, the number of consecutive drops can be reduced. Many Internet designers were concerned that consecutive drops when queues became full could cause global instability in the network as many queues signal their source to reduce their window at the same time.

Fairness – Reduces the bias against bursty traffic, as mentioned earlier. ERED will avoid a situation in which bursty traffic faces extreme packet loss compared to smooth traffic.

5.6 Summary

This chapter presented and analyzed the experimental results to provide evidence that this proposed ERED approach can enhance the affective of RED protocol. It makes less dropping packet and avoid congestion. The enhancement makes the ERED constrain the oscillation and become more robust and adaptable.

An overview of the basic ERED protocol is presented, then the experimental environment and implementation of ERED are shown. Finally, the experimental results, discussion and comparison this proposed approach with RED are presented.

In the next chapter, the results of this ERED protocol will be concluded.

6 CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In this thesis, ERED gateways could be particularly useful in networks where it is undesirable to drop packets at the gateway. The goal of this project is to avoid the congestion at the gateways. ERED gateways are designed to accompany a transport layer congestion control protocol such as TCP. The ERED gateways have no bias against bursty traffic and avoid the global synchronization of many connections decreasing their window at the same time.

The reasons that cause the queue size oscillation in the RED algorithm are investigated and analyzed. The dropping function is modified as the enhancement to the standard RED algorithm. The simulation experiments validate this modification. The enhanced RED algorithm can constrain the oscillation occurred in the steady state, moreover, the control parameter settings is more adaptable and robust, no longer too sensitive to the number of the active flows. The design and analysis included in this approach is based on control theory, It is likely that the early random drop is the most feasible congestion control mechanism in the current Internet, so all the problems is how to design a robust and effective controller to determine the dropping probability according to the network load condition, there are two aspects that deserve to be study in the future. The exponential weighted moving average algorithm used to sense the network load status is equivalent to the low pass filter, whose effectiveness is worthy to studying. Perhaps, there are other filters that are more suitable for observing the network load or designing the controller.

Using alternative path and secondary parallel queue make avoidance to the congestion at the gateway and make it more adaptable and robust.

6.2 FUTURE WORK

This project work implements a congestion avoidance algorithm for a higher level of congestion avoidance in gateways. In this work the threshold is calculated based on randomly generated values. Future work can be carried on optimizing the average q-size for maximizing the throughput with minimal delay. The discussion of the optimal average queue size for a particular traffic mix is left as a question for future research. The algorithm can be further enhanced with incorporation of traffic management algorithms. Our work is implemented by targeting TCP and further enhancements can be carried out to make it compatible with other protocols.

Although much research effort has been focused on understanding and utilizing RED algorithm to leverage the current network, some interesting research topics are yet to be investigated in more detail in future. For example, since it is widely accepted that Poisson model is not sufficient to characterize the traffic in current Internet, it is important to understand how RED and similar Active Queue Management (AQM) algorithm act when self-similar network traffic is applied. Further studies may produce more meaningful characterization of RED performance in the real-world network. RED algorithm does not avoid TCP synchronization [17].

REFERENCES

[1] Larry L. Peterson & Bruce S. Davie, 2007- Edtion 4, “Computer Networks a system approach”.

[2] Molinero-Fernandez, P., 2003, “Circuit Switching In The Internet, Dissertation, Stanford University”.

[3] Xiaole Bai, Marcin Matuszewski, Liu Shuping, Raimo Kantola, 2004, “A Novel Multi-Path Routing Protocol” .

[4] Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans Networking 1993;1(4): 397–413.

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

[6] Zhang, L., “A New Architecture for Packet Switching Network Protocols”, MIT/LCS/TR-455, Laboratory for Computer Science, Massachusetts Institute of Technology, August 1989.

[7] Mankin, A. and Ramakrishnan, K. K., editors for the IETF Performance and Congestion Control Working Group, “Gateway congestion control survey”, RFC 1254, August 1991, p.21.

[8] Shenker, S., “Comments on the IETF performance and congestion control working group draft on gateway congestion control policies”, unpublished, 1989.

[9] T. Senthil Kumaran *, V. Sankaranarayanan, “Early congestion detection and adaptive routing in MANET”, Egyptian Informatics Journal (2011) 12, 165–175.

[10] Atsami Y, Kondoh E, Altintas O, Yoshida T. Improving fairness and stability in best effort service: a new congestion control algorithm for SACK-TCP. IEICE Trans Commun 1998;E81-B(11):2023–33.

[11] Braden B et al. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309; April 1998.

[12] Heinanen J, Baker F, Weiss W, Wroclawski J. Assured forwarding PHB. RFC 2597; June 1999.

[13] Chuah, M.; Zhang, Q.; Yue, O., Access priority schemes in UMTS MAC, IEEE Conference on Wireless Communications and Networking 1999, Vol. 2, Pages: 781 - 786.

[14] Toshimitsu, K.; Yamazato, T.; Katayama, M, Ogawa, A., A novel spread slotted ALOHA system with channel load sensing protocol, IEEE Journal on Selected Areas in Communications, Vol. 12, No. 4, May 1994, Pages 665 - 672.

[15] Omiyi, P. E.; O'Farrell, T., A New throughput analysis of a novel MAC protocol for CDMA-based wireless LANs, RAWCOM'98 Proceedings, 1998, Pages: 23 - 26.

[16] Koo J, Song B, Chung K, Lee H, Kahng H. MRED: a new approach to random early detection. IEEE Comput Soc 15th Int Conf Info Networking 2001:347–52.

[17] Bonald T, May M, Bolot J. Analytic evaluation of RED performance. Proceedings of IEEE Infocom 2000;3:1415–24.

[18] Suter B, Lakshman T, Stiliadis D, Choudhury A. Buffer management schemes for supporting TCP in gigabit routers with per-flow queuing. IEEE J Selected Areas Commun 1999;17:1159–69.

[19] Ott T, Lakshman T, Wong L. SRED: stabilized RED. IEEE INFOCOM 1999:1346–55.

[20] Feng W, Kandlur D, Saha D. Adaptive packet marking for maintaining end to end throughput in a differentiated service Internet. IEEE/ACM Trans Networking 1999;7(5):685–97.

[21] Feng G, Agarwal A, Jayaraman A, Siew C. Modified RED gateways under bursty traffic. IEEE Commun Lett 2004;8(5):323–5.

[22] Sun J, Ko K, Chen G, Chan S, Zukerman M. PD–RED: to improve the performance of RED. IEEE Commun Lett 2003;7(8):406–8.

[23] Christiansen M, Jeffay K, Ott D, Smith F. Tuning RED for web traffic. IEEE/ACM Trans Networking 2001;9(3):249–64.

[24] May M, Bolot J, Jean-Marie A, Diot C. Simple performance models of differentiated services schemes for the Internet. IEEE INFOCOM 1999;3:1385–94.

[25] Kim W, Lee B. The FB-RED algorithm for TCP over ATM. IEEE GLOBECOM 1998:551–5.

[26] Lin D, Morris R. Dynamics of random early detection. Proc ACM SIGCOMM 1997:127–37.

[27] Lakshman T, Neidhardt A, Ott T. The drop from front strategy in TCP and in TCP over ATM. IEEE INFOCOM 1996;3: 1242–50.

[28] Feng W, Kandlur D, Saha D, Kang G. Self-configuring RED gateway. IEEE INFOCOM 1999:1320–8.

[29] Parris M, Jeffay K, Smith F. Lightweight active router queue management for multimedia networking. Proc SPIE 1999:162–74.

[30] Firoiu V, Borden M. A study of active queue management for congestion control. IEEE INFOCOM 2000;3:1435–44.

[31] Siva Ram Murthy C, Manoj BS. Ad hoc wireless networks architectures and protocols. Pearson Edu 2007.

[32] Lu Y, Wang W, Zhong Y, Bhargava B. Study of distance vector routing protocols for mobile ad hoc networks. Proc IEEE Intl Conf Pervasive Comput Commun (PerCom) 2003:187–94.

[33] http://en.wikipedia.org/wiki/Multipath_routing (Accessed 2015-03-11).

[34] Ron Banner, and Ariel Orda., 2007 “Multipath Routing Algorithms for Congestion Minimization”.

[35] S.Santhi, G.Sudha Sadasivam, 2011, “Performance Evaluation of Different Routing Protocols to Minimize Congestion in Heterogeneous Network”.

[36] Johnny Chen, June, 1999, “New Approaches to Routing for Large Scale Data Networks”,55-70.

[37] Martha Steenstrup. Routing in communications networks. Prentice-Hall, Fort Collins, CO., 1995.

[38] E. W. Djikstra. A note on two problems in connection with graphs. Numerische Mathematik, 1, 1959.

[39] D. M. Topkis. A k shortest path algorithm for adaptive routing in communications networks. IEEE Transactions on Communications, 36, 1988.

[40] R. E. Bellman. Dynamic Programming. Princeton University Press, Princeton, N.J., 1957.

[41] C. Cheng. A loop-free extended Bellman-Ford routing protocol without bouncing effect. ACM Computer Communication Review, 19(4):224–236, 1989.

[42] Atul Khanna and John Zinky. The revised ARPANET routing metric. In Proceedings of ACM SIGCOMM, pages 45–56, September 1989.

[43] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In Proceedings of ACM SIGCOMM, pages 1–12, 1989.

[44] Van Jacobson. Congestion avoidance and control. In Proceedings of ACM SIGCOMM, pages 314–329, August 1988.

[45] Vern Paxson. Measurements and analysis of end-to-end Internet dynamics, April 1997. Ph.D. dissertation, University of California, Berkeley.

[46] Sugih Jamin, Peter Danzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. IEEE/ACM Transactions on Networking, 5(1):56–70, February 1997.

[47] H. Saran, S. Keshav, and C. R. Kalmanek. A scheduling discipline and admission ontrol policy for Xunet 2. In Proceedings of NOSSDAV, November 1993.

[48] C. Parris, S. Keshav, and D. Ferrari. A framework for the study of pricing in integrated networks. ICSI Technical Report TR-92-016 and AT&T Bell Labs Technical Memorandum TM-920105-03, January 1992.

[49] S. Keshav and R. Sharma. Achieving quality of service through network performance management. In Proceedings of NOSSDAV, July 1998.

[50] S. Keshav and R. Sharma. Issues and trends in router design. IEEE Communications Magazine, May 1998.

[51] S. P. Morgan. Queuing disciplines and passive congestion control in byte stream networks. IEEE Transactions on Communications, 39(7):1097–1106, 1991.

[52] David Eppstein. Finding the k shortest paths. In Proc. 35th Symp. Foundations of Computer Science, pages 154–165. Inst. of Electrical & Electronics Engineers, November 1994.

[53] D. R. Shier. On algorithms for finding the k sortest paths in a network. Networks, 9:195–214, 1979.

[54] D. P. Bertsekas and J. N. Tsitsiklis. Parallel and distributed computation: Numerical methods. Prentice-Hall, Englewood Cliffs, N.J., 1989.

[55] Dimitri Berksekas and Robert Gallager. Data Networks, Second Edition. Prentice-Hall, Englewood Cliffs, N.J., 1992.

[56] J. N. Tsitsiklis and G. D. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. MIT LIDS-report LIDS-P-2238, May 1986.

[57] Andras Varga and OpenSim Ltd. OMNeT++ User Manual, Version 4.2.2. 2011.

[58] http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html (Accessed 2015-02-23).

[59] Jain, R., “Congestion Control in Computer Networks: Issues and Trends”, IEEE Network, May 1990, pp. 24-30.