SOFTWARE IMPLEMENTATIONS OF QOS SCHEDULING ALGORITHMS FOR HIGH SPEED NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

AYDIN PEHLIVANLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2015

Approval of the thesis:

SOFTWARE IMPLEMENTATIONS OF QOS SCHEDULING ALGORITHMS FOR HIGH SPEED NETWORKS

submitted by **AYDIN PEHLIVANLI** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Ece Güran Schmidt _____
Supervisor, **Electrical Electronics Engineering Department, METU**

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı _____
Electrical Electronics Engineering Department, METU

Assoc. Prof. Dr. Ece Güran Schmidt _____
Electrical Electronics Engineering Department, METU

Assoc. Prof. Dr. Ilkay Ulusoy _____
Electrical Electronics Engineering Department, METU

Assoc. Prof. Dr. Halit Oğuztüzün _____
Computer Engineering Department, METU

Ömer Lütfi Nuzumlalı _____
Aselsan INC.

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: AYDIN PEHLIVANLI

Signature :

ABSTRACT

SOFTWARE IMPLEMENTATIONS OF QOS SCHEDULING ALGORITHMS FOR HIGH SPEED NETWORKS

Pehlivanlı, Aydın

M.S., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Ece Güran Schmidt

January 2015, 91 pages

The end to end Quality of Service (QoS) support for the dominating multimedia traffic in the contemporary computer networks is achieved by implementing schedulers in the routers and deploying traffic shapers. To this end, realistic modeling and simulation of these components is essential for network performance evaluation.

The first contribution of this thesis is the design and implementation of a C++ simulator QueST (Quality of Service simulaTor) for this task. QueST is a modular cycle accurate simulator with a detailed modeling of the traffic flows, shapers and schedulers. The traffic generators and the schedulers of QueST are verified by comparison to the respective analytical models.

The QoS schedulers are data plane components in routers which have to operate at 10s of Gbps rates. Hence, the increasing scheduling complexity with the number of flows is an important problem. This problem can be alleviated by reducing the number of flows by traffic aggregation.

The second contribution of this thesis is the evaluation of previously developed Window Based Fair Aggregator (WBFA) in QueST under a large number of case studies to investigate its features and benefits as well as optimal parameter selection.

Keywords: Quality of Service Schedulers, High speed networks, Flow aggregation, Software simulator

ÖZ

YÜKSEK HIZLI AĞLAR İÇİN AĞ İLETİŞİMİ HİZMET KALİTESİ ÇİZELGELEYİCİ ALGORİTMALARININ YAZILIMSAL GERÇEKLENMESİ

Pehlivanlı, Aydın
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Ece Güran Schmidt

Ocak 2015 , 91 sayfa

Modern bilgisayar ağlarına hakim multimedya trafikler için uçtan uca servis kalitesi (QoS) desteği, ağ yönlendiricilerinde çizelgeleyicilerin gerçeklenmesi ve trafik şekillendiricilerin uygulanmasıyla sağlanır. Bu amaçla, ağ bileşenlerinin gerçek şekilde modellenmesi ve simule edilmesi ağ performanslarının değerlendirilmesi için gereklidir.

Bu tezin ilk katkısı QueST C++ simulatörünün tasarımı ve gerçeklenmesidir. QueST trafik akışlarını, şekillendiricilerini ve çizelgeleyicilerini detaylı modelleyen modüler kesin bir simulatördür. QueST simulatöründeki trafik üreticileri ve çizelgeleyicileri ilgili analitik modellerle kıyaslanarak doğrulanmıştır.

Yönlendiricilerde bulunan servis kalitesi çizelgeleyicileri onlarca Gbps hızında işlem yapması gereken veri düzlemi bileşenleridir. Bu yüzden, akış sayısı ile birlikte artan çizelgeleyici karmaşıklığı önemli bir problemdir. Bu problem trafik birleştirme

yöntemi ile akış sayısını azaltarak bastırılabilir.

Bu tezin ikinci katkısı daha önceden geliştirilen pencere tabanlı adil birleşticinin (WBFA) özellikleri ve faydalarının yanısıra optimum parametre seçimlerini araştırmak için büyük sayıda durum çalışmaları altında QueST simülatöründe değerlendirilmesidir.

Anahtar Kelimeler: Servis kalitesi çizelgeleyici, Yüksek hızlı ağ, Akış birleştirme, Benzetim yazılımı

To my wife and my family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis advisor Assoc.Prof. Dr. Ece Güran Schmidt for her guidance, advice, support, understanding and encouragement throughout the completion of this thesis study.

I am grateful to Aselsan for the time and resources that I use for this thesis.

I am grateful to my manager Mustafa Özgür Ateşoğlu for his tolerance during this study.

I would like to thank Mustafa Sanlı for his advice and help on completing this thesis.

My special thanks go to Murat Yılmaz, Kerem Furkan Çiçek, Halit Gölcük and Murat Vural for reviewing this thesis.

I would like to thank my friend Mehmet Sami Büyüksarıkulak for his help on writing this thesis.

I would like to thank my colleagues Serkan Avgören, Ömer Lütfi Nuzumlalı and Mustafa Karakurt for their encouragement and friendship.

I owe my wife Ebru Pehlivanlı a debt of gratitude for her love, support and understanding throughout this study.

Finally, I would like to thank my parents Adem and Zeliha Pehlivanlı and my brother Ayhan Pehlivanlı for their continuous support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

QoS	Quality of Service
$WBFA$	Window Based Fair Aggregator
ISP	Internet Service Provider
$VOIP$	Voice over IP
NGN	Next Generation Network
ATM	Asynchronous Transfer Mode
$IntServ$	Internet Integrated Service
$DiffServ$	Internet Differentiated Service
GPS	Generalized Processor Sharing
PFQ	Packet Fair Queuing
WFQ	Weighted Fair Queuing
WF^2Q	Worst-case Weighted Fair Queuing
WF^2Q+	Worst-case Weighted Fair Queuing Plus
$FAbS$	Flow Aggregate Based Services
IP	Internet Protocol
$WBFA$	Window Based Fair Aggregator
IC	Integrated Circuit
RIP	Routing Information Protocol
$OSPF$	Open Shortest Path First
TTL	Time to Live
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TG	Traffic Generator
$MGEN$	Multi-Generator
NRL	Naval Research Laboratory
$RUDE$	Real-Time UDP Data Emitter
$CRUDE$	Collector for RUDE
ABR	Available Bit Rate

ATM	Asynchronous Transfer Mode
FTP	File Transfer Protocol
ns	Network simulator
OTcl	Object-oriented Tool Command Language
NED	Network Description
QueST	Qos SimulaTor
WFI	Worst case Fairness Index
OOP	Object Oriented Programming
CDF	Cumulative Distribution Function
GUI	Graphical User Interface

CHAPTER 1

INTRODUCTION

The fraction of multimedia traffic carried over Internet Protocol (IP) networks increases rapidly with the popularity of video and Voice over IP (VOIP) applications. These applications require end to end Quality of Service (QoS) support at the network level such as bounded delay, guaranteed bandwidth and high throughput. Therefore, network routers and switches store the incoming packets in the per flow queues and apply QoS scheduling to decide the service order among these queues. Resulting differential service allocates the available bandwidth to these per flow queues with respect to their reserved service rate.

The desired properties of a scheduling algorithm is achieving low delay bounds and high throughput. Furthermore the flows should get fair service that is proportional to their reserved rates.

The QoS schedulers are data plane components in the routers which process each packet that goes through. Running such process under the ever increasing data rates of the Internet requires low complexity scheduling algorithm implementations. Furthermore, the complexity increases with the number of flows that are scheduled. The ubiquitous connectivity with computers, smart phones or any other devices as expected for Internet of Things increases the number of flows in the network at a high rate. One solution to alleviate this problem is aggregating the flows without degrading the service quality received by each individual flow.

The focus of this thesis is the modeling and performance evaluation of the network layer QoS support by traffic shapers and schedulers. To this end, the first contribution

of the thesis is a custom software simulator QueST (Quality of Service simulaTor) implemented in C++ that modularly implements traffic generators, shapers, per flow queues and QoS schedulers. QueST is a cycle accurate simulator which enables the user to investigate the packet by packet, cycle by cycle behavior of these network components. The widely used open source network simulators such as ns-2 and OMNeT++ do not provide such detailed implementation for the QoS schedulers. QueST enables users to modify and extend the simulator easily due to its well defined module interfaces.

The current traffic generator module in QueST implements the most common Poisson, Bursty and Pareto models. The current traffic shaper in QueST implements the well-known Token Bucket Shaper algorithm. The selected QoS Scheduler for implementation is Weighted Fair Queuing Plus (WF^2Q+) [11] algorithm which provides the best delay bound and fairness with a low complexity. The correctness of the implemented modules is verified by comparison to their analytical models.

The second contribution of this thesis is the detailed performance evaluation of the previously proposed Window Based Fair Aggregator (WBFA) [60]. The aggregator preserves the delay bounds of the constituent flows if the aggregate flows are scheduled by a scheduler with certain properties. WF^2Q+ algorithm satisfies these properties.

Extensive performance evaluations are carried out for WBFA-WF^2Q+ in QueST to demonstrate their features and benefits. Furthermore, the optimal value for the important window size parameter of WBFA is investigated experimentally using QueST. The results show that WBFA can achieve the performance as analytically derived in [60]. Furthermore the decreased simulation times under flow aggregation quantitatively demonstrate the reduced complexity of the flow aggregation.

The remainder of the thesis is organized as follows:

Chapter 2 presents the literature survey on QoS components as well as network simulator.

In Chapter 3, software modules and submodules of the QueST which is implemented by use of C++ language are introduced in detail with UML models. Assumptions

made during development process of QueST are given. Run time analysis is presented for each module. Verification experiments for each module and submodule are explained and presented in detail.

In Chapter 4, WBFA is explained and the Integration of WBFA to QueST is given detail. Verification experiment for WBFA module in QueST is performed. Optimum window size for WBFA is given. Run time analysis is presented for WBFA module. Effect of WBFA on WF^2Q+ scheduler module is analyzed by performing experiments.

In Chapter 5, the conclusion of the thesis work is discussed with obtained results.

CHAPTER 2

RELATED WORK: QUALITY OF SERVICE SUPPORT IN HIGH SPEED NETWORKS

The popularity of the Internet has been growing each year for the last decades. Therefore, the Internet traffic has become more crowded. New applications such as Voice Over IP (VOIP) and high definition video communication have created a need for an increase in the network bandwidth. In order to provide these services, end to end Quality of Service (QoS) support is needed at network layer. In this chapter, related works in the literature are presented about network routers, QoS requirements and schedulers, traffic profiles, traffic shapers, flow aggregations and network simulators.

2.1 High Speed Network Routers

The number of Internet Protocol (IP) network users increases day by day [3]. The growth of video traffic and multimedia streaming have also interesting statistic. %30 of European IP traffic is multimedia traffic [1]. Furthermore, real-time network applications such as IP telephony and video conference become important for network area. All of these services are formed by customer requirements and require QoS support [57]. The routers of the Internet Service Providers (ISP)'s have a quite important role in providing these services. This rapid increase in network traffic creates new challenges for router design.

[7] states that routers have been implemented in software conventionally. Therefore, performance of network routers depends on the quality of software processor and its code. However, rapid growth in the requirement of high speed routers has created the

need for wire speed routing which demands high performance processors and large memory units. This situation increases the cost and complexity of architectural design. Recent developments in technology have changed the viewpoint for implementing high speed network routers. Designing a single integrated circuit (IC) with silicon capability, embedded memories and microprocessors can be shown as an example for these technologies. These technologies make building of cost effective, single-chip routing solutions possible. Technological improvements in routing architecture which includes specialized hardware, switch fabrics, efficient routing algorithms have created a new family of routers which enable to send packets at multigigabit rates [43].

2.1.1 Network Router Functionalities

Network routers consist of network interface cards, processing modules, buffering modules and switch fabrics and send packets among the end users by using these modules [14]. Typically, packets are received by inbound interface cards, processed by processing module and queued in the buffering module. Then, they are sent to next hop by the help of outbound interface modules. In other words, the main task of network routers is to sent packets from source node to destination node [43].

Figure 2.1 shows the generic router architecture which is composed of input and output line cards, router processor or Central Processing Unit (CPU) and switch fabric. Responsibilities of these components are as follows [2]:

- *Line cards* are entry and exit points of routers. They enable the routers to connect external network. A variety of datalink technologies such as synchronization and frame processing are employed in the line cards.
- *Switch fabric* is used as the interconnection between input line cards, CPU and output line cards. In other words, it performs task switching. After that the packets that are queued at the buffers of output line cards. Then, the packets selected by QoS scheduler are transmitted.
- *CPU* is used to run routing protocols, update and maintain routing table. CPU provides basic management operations such as creation of routing table for the output line cards and link management.

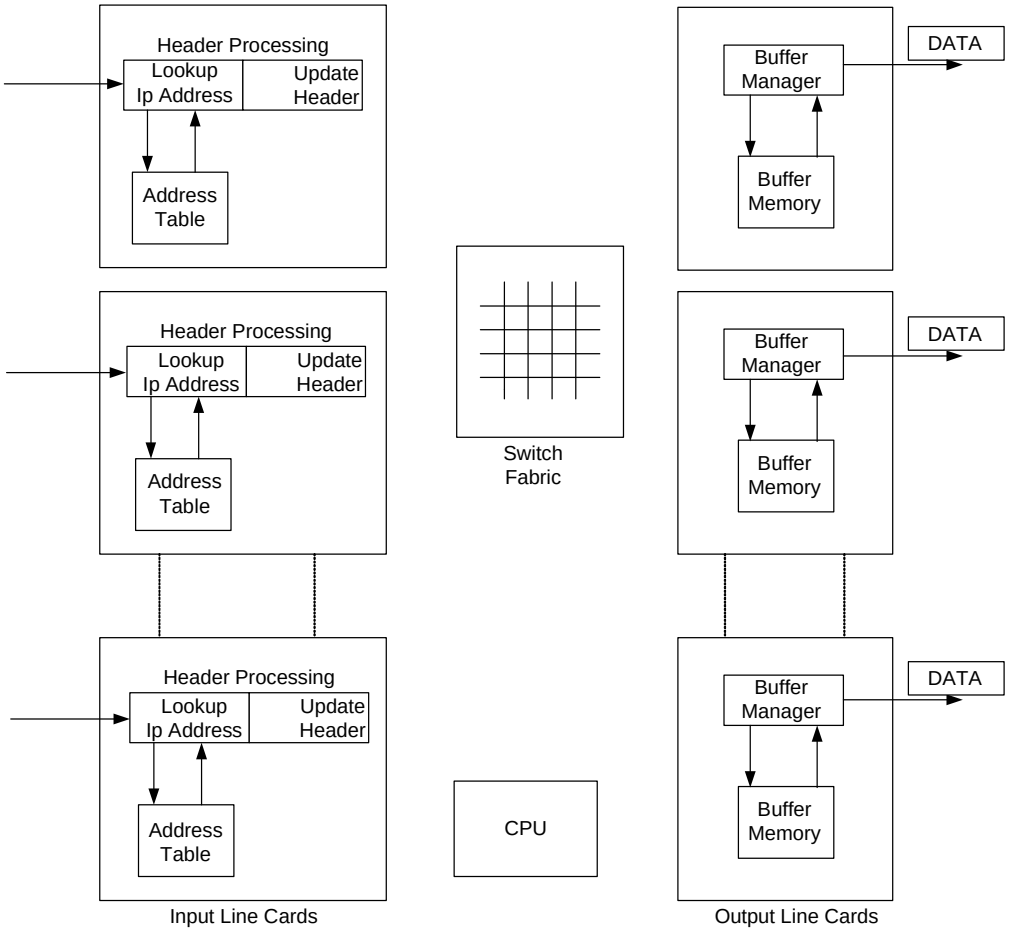

Figure 2.1: Generic Router Architecture

2.2 Quality of Service

The volume of IP network traffic increases per year [3], and this growth is expected to go on for many years. IP-TV and multimedia traffic which become 30 percent of European IP traffic have main effect on this growth [1]. According to [59], services for real-time applications such as IP telephony and video, virtual environments and global or local information centers are the motivation behind the Next Generation Network (NGN) paradigm.

The above-mentioned services need the end to end QoS support [17]. QoS can be defined by parameters such as obtained bandwidth, delay, and throughput [73]. Sorting the incoming packets into related flows on node, storing them in per flow queues and

applying a QoS scheduler to assign the available bandwidth resources to these flow queues with respect to their allocated rates are necessary to obtain end to end QoS support.

[17] states that some parameters such as number of network users, required bandwidth, types of QoS requirements and IP multicasting applications have negative effect on QoS guarantees. Dynamic routing protocols such as open shortest path forwarding (OSPF) [47] and congestion control are inefficient to provide QoS support. Network applications have different priorities for their transmission processes. While some of them require error-free data transmission, the others may require real time data transmission by tolerating data loss. Therefore, IP network elements are needed to be managed in a way that these requirements are provided. Networks which can dynamically adopt the QoS can provide custom support for different network users [73]. In order to measure QoS support, the following parameters are used [61, 46]:

- **Delay :** Time spent for a packet to travel from source to destination.
- **Jitter :** Variation in the latency of a packet.
- **Bandwidth :** Maximum data transfer rate or used capacity of the network link.
- **Throughput :** Ratio of total number of packets successfully transferred to total number of packets.

In order to provide good QoS support and maximize the network resource utilization, traffic management which is used to prioritize traffic flows according to their data contents should be provided [17]. In order to apply traffic management, each element in IP network must contain new logical QoS supporting facilities and functionalities [28]. There fundamental functionalities are traffic policing/shaping, packet scheduling and buffer management [66].

2.2.1 Traffic Profiles and Traffic Shaping

Over the last years, both academic and industrial research have been done to understand and characterize the behavior of computer network due to increasing bandwidth

and the number of new Internet applications. Network elements such as routers and switches are obliged to develop to meet new requirements. Therefore, these network elements need to be tested and evaluated during development process. The complexity of network topologies and their traffic characteristics make simulation technique a promising tool [6].

Traffic generation which is one of the challenges in network simulator design is required to test and evaluate network performance, new components and algorithms under specified load conditions [58]. Traffic generator generates packet with respect to a traffic pattern. A traffic generator must be controllable, scalable and realistic to have satisfied evaluation of network performance [6].

Increasing network bandwidth requirements results in new academic and industrial studies on the design and development of router architecture, QoS algorithms and queue management techniques. Queue management and scheduling of the queue have a great impact on performance metrics such as delay and throughput. In order to evaluate these metrics, different traffic arrivals such as Poisson, Pareto, and Bursty are needed. Therefore, traffic generators which generate packets with respect to these arrivals can simply present network performance evaluation results when they are implemented in software.

There is no one single model that can be used effectively for modeling traffic in all kinds of networks. In the literature, there are many number of traffic profiles proposed for analyzing the traffic characteristics of networks [45]. However, Poisson and Markov-Modulated On/Off traffic profiles are the mostly used traffic patterns in software-based traffic generators [30, 44, 56, 24] and Pareto traffic is the most suitable traffic profile to model high speed networks and IP networks [4, 35]. Therefore, in the simulator developed in this thesis, packets are generated with respect to these three traffic profiles which are Poisson traffic, Markov-Modulated On/Off traffic and Pareto traffic.

Poisson distribution is one of the most widely used and oldest traffic model [45]. [71] states that in many simulation scenarios the application of Poisson traffic is very reasonable because of its well defined conditions for simulations. Poisson traffic is suggested for traffic modelling in available bit rate (ABR) services in asynchronous

transfer mode (ATM) Networks [36].

Although the Poisson model is no longer suitable to accurately describe the bursty behavior of real traffic, poisson traffic is still used to model Internet traffic [71]. According to [39], backbone network traffics appears to be well described by the Poisson packet arrivals. In [53], it is shown that under some mild conditions,the feedback flow of customers returning to the back of the queue converges to a Poisson process as the feedback delay distribution is scaled up.

Markov-modulated on–off traffic model is one of the most widely used traffic models like Poisson Model. [75] states that, markov-modulated on–off traffic model emits packet when the source is ON state and in the OFF state no packet is emitted. In [75], Markov-modulated on-off model is suggested to simulate the aggregate traffic with great accuracy. Markov-modulated on-off model is also suggested for persistent TCP connection in [33]. Furthermore, the traffic in data centers exhibits on–off behavior [12].

The Pareto distribution model is used to produce independent and identically distributed inter-arrival times [4]. According to [74], when network packets are transmitted to the destination node, Pareto distribution can be used to estimate inter-arrival times of the generated packets. In [9], pareto distribution is suggested to model "heavy-tailed" traffic trains. Pareto distribution is also suggested in case of high speed networks with unexpected demand on packet transfers due to its correctness on the packet arrival times [4]. [35] says that if current IP traffic is needed to model, Pareto distribution model can be applied instead of exponential distribution. Pareto distribution which is used to perform the severity distribution in a context of catastrophe reinsurance can be used to model of an individual as stated in [29].

Smoothing traffic at the input of network edge is applied to increase the schedulable region of the network and reduce the traffic stream's peak rate and rate variance [64]. In order to smoothen traffic, shaping is used. Traffic shaping is applied to check traffic to ensure QoS, low latency, and increase bandwidth [8]. Traffic shaping controls burst data and rate to ensure conformance to a traffic contract. If a traffic stream does not obey the traffic contract, shaping algorithm delays that traffic until they conform to the profile [17]. Queues are used to delay traffic streams by buffering traffic packets

[8]. However, since the traffic shaper has finite size queues, some packets may be discarded when the queue does not have enough size to buffer delayed packets [17].

Traffic shaping is applied at the source prior to ingress of network or within the network. While traffic shaping which is done at the source applies self regulation to conform traffic contract, traffic shaping within the network does not need traffic contract [13]. According to [55], shaped traffic will be exposed to less queueing delay and jitter since it yields small size queues. In order to delay traffic streams to conform contract, traffic shaping uses buffering technique. Therefore, some queueing delay is observed by the traffic streams. However, since shaped traffic goes quickly along the remaining path, network performance will be improved [26].

2.2.2 Packet Scheduling

Network users are allowed to share network resources such as bandwidth and packet queues. However, since the number of network users is increasing, sharing these resources among network users will become more problematic inevitably [17]. Therefore, a packet scheduling discipline is necessary to determine which packet goes next when a number of network users share the same link. In other words, packet schedulers are needed to prioritize network users to provide QoS support by making use of network resources [16].

Packet scheduling determines the order of queued packet transmission and addresses different requirements of queued packets which are contending for the same outgoing interface [16]. Since many packets in a network node may be transmitted from the same outgoing link, packet scheduling also defines a set of rules in bandwidth usage. While defining a set of rules for network users, packet scheduling should consider the QoS requirements of each flow [13]. For example, while data traffic is loss sensitive and has low priority, real-time video traffic is delay sensitive and has the highest priority. Therefore, the privileged video traffic is transmitted first while data traffic is delayed. Figure 2.2 shows a packet scheduler which can be located at the processor of a router.

In today's routers, since both number of network users and QoS requirements in-

crease, packet scheduling has become more significant [59]. Therefore, performance metrics of packet scheduling is used to evaluate scheduler. In general, the following metrics can be listed for packet scheduler [54]:

- **Fairness :** The packet scheduler must isolate flows competing for the same link between each other. In other words, each flow should get its resource share and this share should not be penalizes by the misbehaviour of other flows.
- **Delay Bound :** Interactive applications such as VoIP, video conferencing and IP telephony require bounded delay. Since the packet scheduler determines the order in which packets are transmitted on outgoing link, the end to end delay is affected by packet scheduler.
- **Complexity :** Since the number of network users and supported line rates have been increasing day by day, the complexity which is related with computational resources required for the applying scheduling algorithm is also increasing. The packet scheduler must have low level of complexity.

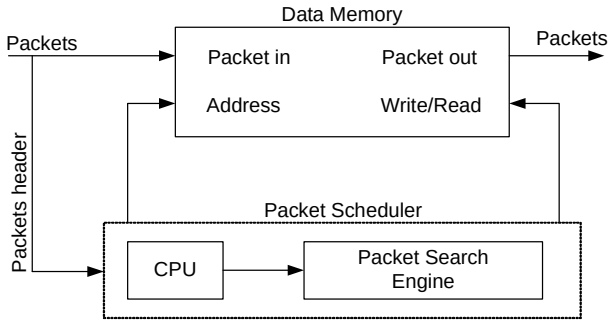

Figure 2.2: Packet Scheduler

In the literature, there is a lot of previous works on scheduling algorithms. Generalized Processor Sharing (GPS) [52] which is called the ideal scheduling algorithm provides perfect fairness and protection among network users [25]. However, GPS scheduling discipline which based on fluid flow model is not implementable due to its packet transmission procedure [54]. In GPS scheduling algorithm, network traffic flows are divided and many flows can take services proportional with their weight simultaneously. According to [23, 31], GPS scheduling discipline provides network

delay bound for leaky bucket constrained traffic. However, GPS can not be implemented in real world. Nevertheless, GPS acts as a benchmark for other scheduling algorithms due to its fairness and low delay.

In order to emulate GPS behaviour and get as close as possible to GPS, a number of packet fair queuing (PFQ) algorithms are proposed [32, 70, 67, 10]. In all PFQ algorithms, behaviour of GPS scheduling process is tracked by a function which is called virtual time. After calculating virtual time, finish time which gives the time that the packet is transmitted under GPS scheduler is calculated for each head of line (HOL) packets. Then HOL packets are served with the increasing order of finish times [59].

Weighted Fair Queuing (WFQ) [25] uses above method in order to compute the finish time of a HOL packet. [10] proves that, the delay bound provided by WFQ is within one packet transmission time and WFQ can be ahead of GPS. Although WFQ discipline provides advance delay bound and fairness, implementation of WFQ is difficult. The complexity of WFQ is $O(N)$ where N is the number of competing flows [54]. The complexity comes from computing the virtual time and finish time, and searching minimum finish time among HOL packets. Therefore, realization of WFQ discipline in network is difficult due to its high complexity [52].

Although WFQ provides low delay bound, it shows short term unfairness which is addressed by the Worst-case Weighted Fair Queuing (WF^2Q) [10] algorithm. WF^2Q algorithm develops WFQ by performing on eligibility test in order to select the transmitted packet. Therefore, WF^2Q scheduling discipline achieves worst-case fairness [10]. WF^2Q emulates GPS better than WFQ and the maximum service difference is one packet transmission time. Similar to WFQ, although (WF^2Q) has good delay bound and fairness, emulation of GPS results in high complexity. The complexity of WF^2Q is also $O(N)$ where N is the number of competing flows [54]. Worst-case Weighted Fair Queuing Plus (WF^2Q+) [11] is an advanced version of WF^2Q algorithm. WF^2Q+ computes virtual time function without emulating GPS and achieves worst-case fairness by using simpler calculation methods. Therefore, it has lower complexity than other PFQ algorithms.

Self-Clocked Fair Queuing [32] and Virtual Clock [76] are another PFQ schedulers.

They compute virtual times more efficiently in order to simplify the emulation of GPS server and sort the packets in increasing order of their finish times. Therefore, their time complexity is still $O(logN)$. However, their delay bounds are very bad.

Implementation of all PFQ algorithms is difficult due to their complexity which is related with the number of network users. Calculation of virtual time and searching minimum finish time become more difficult with the increasing number of network users. As a result, the scheduling algorithms could not realize their proper operations with increasing number of network users. In order to support high number of network users and decrease time complexity, flow aggregation method is proposed [18, 60].

2.2.3 Flow Aggregation

The complexity of PFQ algorithms increases with the QoS requirements which are delay bound and fairness. Since the schedulers have to run at wire speed, number of network users have significant effect on QoS support. When the number of network users exceeds a certain limit, calculation of virtual time and foundation of minimum finish time among flows will be more difficult. Therefore, scheduling algorithms could not realize their proper operations with the increasing number of network users.

Flow aggregation is proposed as a solution to decrease complexity by supporting high number of network users within network. Maintaining QoS guarantee of the input flows in the aggregator is a challenging issue for the flow aggregation method, because some greedy flows induce decreased delay and fairness for other constituent flows. [18] proposed a model which consists of aggregators and schedulers to overcome this problem. According to [18], if the flow aggregation method is fair and packet scheduler is a start time scheduler, then the end to end delay guarantee is protected with respect to case that flow aggregation is not performed. In this work, two different design approach for fair aggregators which are basic fair aggregator and greedy fair aggregator are proposed. While service rate is the sum of reserved rates of aggregated flows in basic fair aggregator, service rate is relaxed only if all flows have greater load than their reserved rate in greedy fair aggregator.

In [69], the end to end delay performance of the guaranteed rate schedulers is ob-

served. According to this work, it is stated that end to end delay bounds in rate schedulers is still preserved in the case of flow aggregation. In addition, it is pointed that delay bound obtained with flow aggregation is more successful.

[38] explored the effect of flow aggregation method on QoS support and stated that the average delay does not increase under flow aggregation. In this work, a new QoS architecture which is called Flow Aggregate Based Services (FAbS) is proposed. FAbS tries to avoid congestion by using flow aggregation method.

In [41, 40], effect of flow aggregation on fairness among TCP flows is examined in DiffServ network. According to these works, the performance obtained by network end users varies significantly when flow aggregation is used. Although flow aggregation with more flows has better throughput than flow aggregation with fewer flow, fairness is still a problem for flow aggregation with more flows.

When fair flow aggregator concept is introduced in [18], lots of works are performed to develop behaviour of fair aggregators. These work studied on the independence of delay on flow rates and the utilization of bandwidth properties. According to these studies, a flow aggregator must provide independent rate delay and work conservation. Although the fair aggregator proposed in [18] provides rate independent delay, it is not work conservative aggregator. In contrast, the fair aggregators presented in [68, 21, 22] are work conservative but they do not provide rate independent delay. However, the fair aggregator proposed in [19] provides both rate independent delay and work conservation at the expense of restriction on packet size and data rate.

[20] also proposes a new work conserving fair aggregation technique which supports rate independent delay. In this technique, a time tag which represents virtual finish time of the packet in the aggregator is inserted to input packet. The aggregated packets are transmitted with respect to these tag values. However, clock synchronization is the main disadvantage of this technique. Also in this technique, all packets are assumed as they have a fixed size.

2.3 Network Simulators

In network research area, network simulation is used to model the behaviour of network by calculating the relation between network nodes and presenting the observations. Therefore, the behaviour of network elements can be observed in a test lab environment.

Network simulators are software tools that represent behaviour of network elements [62]. Network devices, link and applications are used to model computer network and analyse the performance in simulators. Simulators enable users to customize the simulator to perform the specific analysis.

There are many open source and proprietary network simulators in the literature. However, a few of them are mostly used in the research paper. Network simulator (ns), OPNET and OMNET++ can be shown as an example of these simulators. OPNET is a high level event based network simulator which is constructed from C/C++ source code blocks. OPNET can be used as a research or network design tool which consists of high level user interface [50]. OMNeT++ is a modular and extensible network simulator which is constructed from C++ libraries. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools [49]. ns is an open source simulator. ns is a name for series of discrete event network simulators, specifically ns-1, ns-2 and ns-3 which are all discrete event network simulators [48].

2.3.1 Open Source Network Simulators

In this section, brief information is given about open source network simulators. The most widely used open source network simulators are OMNeT++, ns-2 and ns-3.

2.3.1.1 ns-2

ns-2 is Lawrence Berkeley National Laboratory's network simulator [42]. This simulator is an object oriented simulator which is written in C++ language. It uses Object-

oriented Tool Command Language (OTcl) as a configuration and command interface. The simulator supports class hierarchies in C++ and OTcl interpreter. There is one-to-one correspondence between these hierarchies from the user's perspective and the root of these hierarchies is the class TclObject [27].

Figure 2.3 shows the basic ns-2 structure. There are two key programming languages in ns-2 which are C++ and OTcl. While C++ language is used to create detailed network protocol implementations, OTcl is used to set up simulation configuration by assembling and configuring objects as well as scheduling discrete events. TclClass is used to link C++ and OTcl languages together [27]. After the simulation is finished, the processed data can be graphed using tools like XGRAPH.

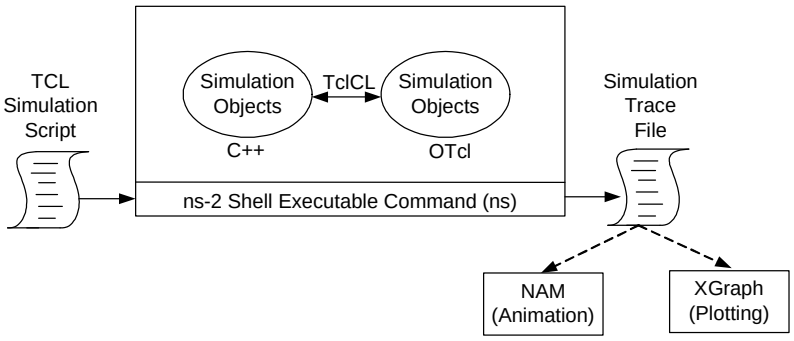

Figure 2.3: Basic ns-2 Architecture

Queue Management part of ns-2 includes First In First Out (FIFO), Round-Robin (RR), Deficit Round Robin (DRR), Fair Queueing (FQ) and Stochastic Fair Queueing (SFQ) scheduling algorithms [27]. Although,in our simulator, we implemented WF^2Q+ scheduling algorithm, ns-2 does not support WF^2Q+ scheduling algorithm. However, [5] proposes a patch for WF^2Q+ scheduling algorithm.

Traffic generator part of ns-2 consists of four traffic types which are exponential on/off traffic, pareto traffic, constant bit rate (CBR) traffic and trace traffic [27]. In our simulator, we implemented exponential on/off traffic, pareto traffic and poisson traffic. Although poisson traffic is not implemented in ns-2 simulator, exponential on/off traffic can replace Poisson process by setting parameters.

In order to implement our WF^2Q+ scheduling algorithm and WBFA algorithm to

ns-2, we have to prepare C++ code of WBFA and WF^2Q+ algorithms as a subclass of "Queue" class in ns-2. The "Queue" class is derived from a "Connector" base class and provides base class used by particular types of queue classes. This base class has virtaul functions called "deque" and "enque". Particular queues which are derived from "Queue" have to implement these two functions. Therefore, WBFA and WF^2Q+ algorithms have to be implemented within these functions. Furthermore, the linkage between the C++ code and OTcl code has to be defined in C++ code. According to [37], created C++ source file is put inside the Queue folder in ns-2 directory and folder information is added to "Makefile.in" folder. After editing the file entry, these procedures are followed:

- Go to the ns-2 location
- Type "./configure" in terminal in order to replace Makefile with modified one
- Type "make" in order to make all objects which are missing

If the created code has no error, then the compilation will success. After successful compilation, OTcl script is written with respect to different test scenarios.

2.3.1.2 ns-3

ns-3 simulator is a discrete event-based network simulator which is intended for research and educational use for network area [34]. The ns-3 is an open source project which is started in 2006. ns-3 which is licensed under the GNU GPLv2 license will bank on progressing contributions of the society to advance new model, maintains existing models [63, 65, 72]. ns-3 is not a extended version of ns-2. The main difference between ns-3 and ns-2 can be listed as [51]:

- *Separete software kernel* : While the kernel of ns-3 is written with Python scripting language, the kernel of ns-2 is written with OTcl language.
- *Attention to realism* : Protocols used in ns-3 is more close to real computers.
- *Software integration* : ns-3 supports the incorporation of the open source networking software and reduces the need to rewrite the simulation.

- *Support for Virtualization* : ns-3 uses lightweight virtual machines.
- *Tracing architecture* : ns-3 is developing a tracing and statistics gathering framework trying to enable customization of the output without rebuilding the simulation core.

2.3.1.3 OMNeT++

Similar to ns-2 and ns-3, OMNeT++ is an open source network simulator. OMNeT++ is a component-based network simulator with GUI support. Communication network is the main application area of OMNET++. Since OMNET++ has generic and flexible architecture, other areas such as IT systems, queuing networks and hardware architectures can be successfully simulated in OMNET++. INET package of OMNET++ enables OMNET++ to provide comprehensive collection of Internet protocol models. In OMNET++, modules are coded in C++ and assembled into larger models by using network description (NED) language [63, 65, 72].

Currently, OMNET++ is the mostly used in academic area for its extensibility and open source documentation. Moreover, OMNET++ is also used in industrial applications. OMNET++ is licensed under its own licence which is called Academic Public License.

CHAPTER 3

QUEST: QOS SIMULATOR

Over the last years, both academic and industrial researches have been made to understand and characterize the behavior of computer network due to increasing bandwidth requirements. Network routers have to be developed to respond to these requirements. In order to achieve good development, these network elements have to be tested and evaluated during development process. Therefore, simulation modelling of these elements is required to evaluate performance of network elements. Simulation enables developers to analyze network elements under different conditions.

In this thesis, we designed and implemented a QoS simulator which is called QueST to evaluate QoS performance of network routers. QueST is implemented by use of C++ language. The main reason of selecting C++ is to conduct large number of tests with different parameters. C++ software also enables us to collect any parameter which we want to analyze. The network packets used in QueST are analyzed with respect to delay which is a QoS metric. Delay of network packets is computed with clock cycle. Therefore, QueST is cycle based simulator. Also QueST is able to generate network packets with fixed or exponentially distributed packet size.

In design process of QueST, we have made some assumptions. Firstly, the transition time between QueST modules such as traffic generator, traffic shaper, scheduler is assumed zero. Secondly, searching all flow queues inside scheduler module is assumed to consume no time. Thirdly, the size of flow queues inside scheduler is assumed to has infinite size. Finally, the size of packet buffers in each module is assumed as infinite length.

Figure 3.1 shows software module diagram of QueST. As can be seen in Figure 3.1, QueST is made up of three software modules which are traffic generator module, traffic shaper module and packet scheduler module.

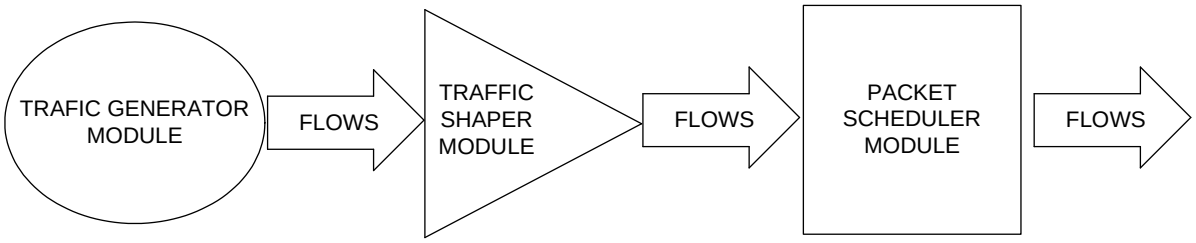

Figure 3.1: Software Modules of QueST

3.1 Modules

QueST is software simulator which is used to evaluate performance of packet scheduling in network routers. QueST gives chance to developers to modify and extend the simulator easily. When one module of QueST changes, the entire structure does not need to change. In addition, adding new modules to QueST does not affect the structure of QueST. Figure 3.2 shows Unified Modelling Language (UML) diagram of QueST. As can be seen in Figure 3.2, QueST consists of three modules which are traffic generator module, traffic shaper module and packet scheduler module.

Traffic generator module is used to generate traffic flows. The traffic generator module is able to generate variable and fixed size packets with Poisson, Bursty and Pareto arrivals. Characteristics of each traffic flow such as load, burst and traffic type are specified in this module. This module also designates the first destination address of generated flows. After generation of each flow, the traffic flows are sent to traffic shaper module.

After traffic generation module, each generated packets enter to traffic shaper module. Traffic shaper modules delays network flows which do not obey the traffic contract. Traffic shaper modules delay traffic flows by storing each flow in a buffer. After shaping each flow, the traffic flows are sent to packet scheduler module or fair aggregator module with respect to their destination address.

Packet scheduler module stores incoming packets in per flow queues when traffic flows are received from input links. Then this module determines the order of queued packet transmission with respect to used scheduling algorithm. After determination of transmission order, this module designates destination address of each flow and transmits these flows from outgoing link.

In QueST, packet scheduler modules can be cascaded to each other. In order to cascade these modules, destination addresses of a module's output flow queues must address to the input flow queues of other modules. By applying this procedure, many number of scheduler and aggregator modules can be cascaded to each other.

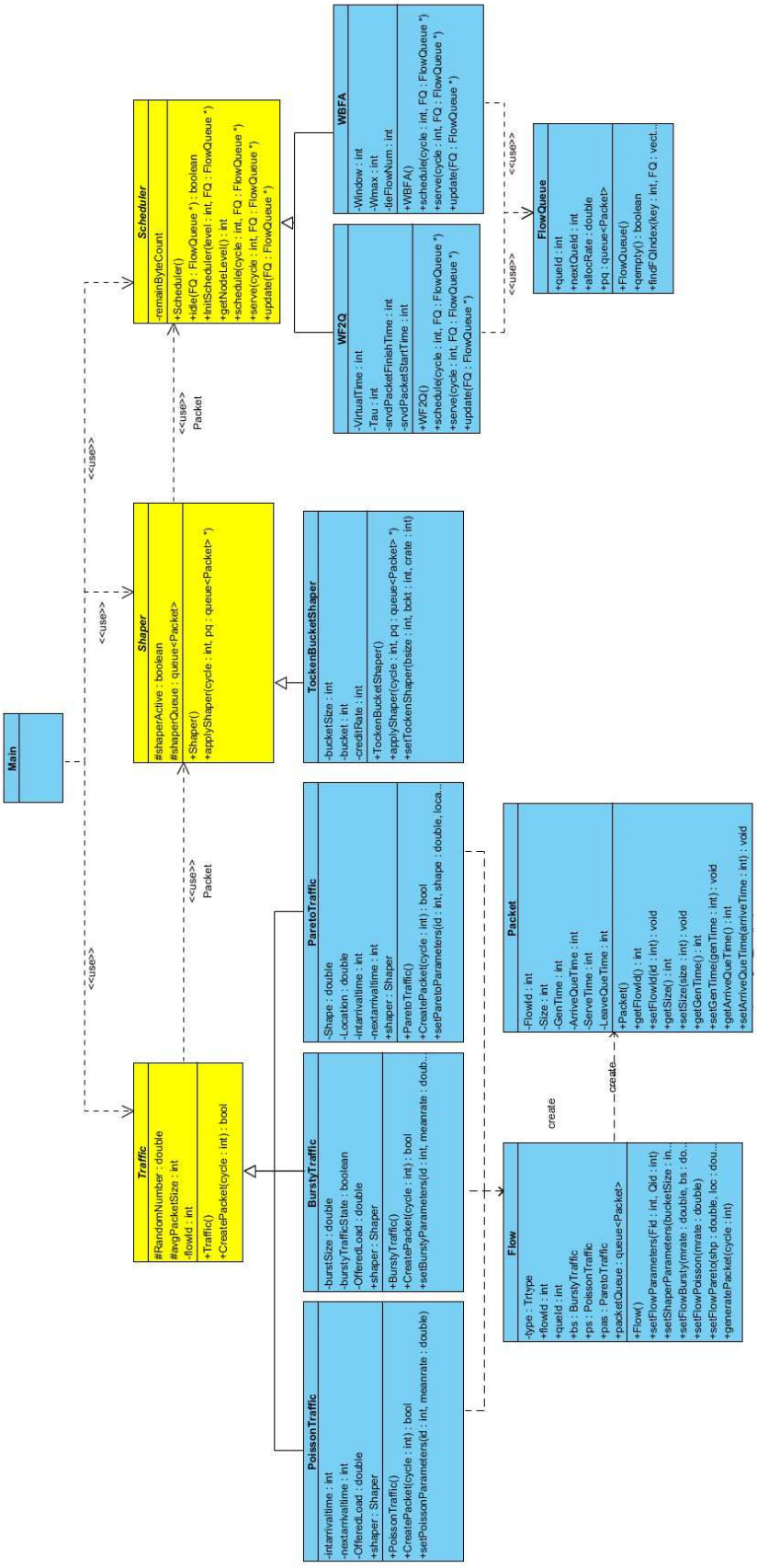

Figure 3.2: UML Model of QueST

3.1.1 Traffic Generator

Traffic generator module in QueST is used to generate traffic flows with respect to specified traffic patterns. Figure 3.3 shows class diagram of traffic generator module of QueST. As seen in Figure 3.3, this module can produce Poisson traffic, Markov modulated on/off (Bursty) traffic and Pareto traffic. The generated packets have fixed sizes or exponentially distributed sizes.

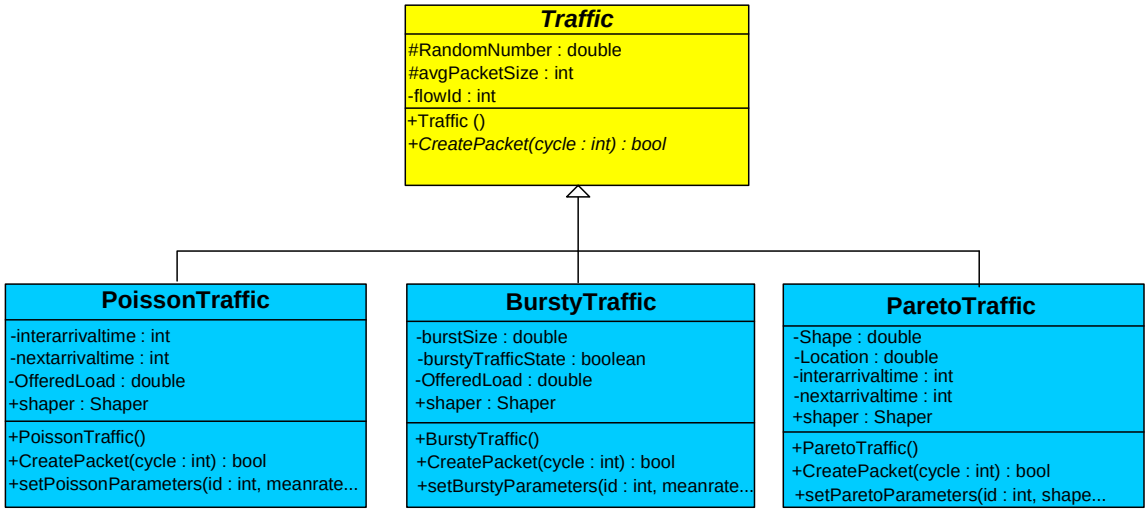

Figure 3.3: Class Diagram of Traffic Generator Module

In QueST, traffic generator module uses inheritance property of object oriented programming (OOP) language to produce traffics. The diagrams in Figure 3.3 use italicized text for `Traffic` class and `CreatePacket` operation. This indicates that the `Traffic` class is an abstract class and the `CreatePacket` operation is an abstract operation. In other words, the `Traffic` class provides the abstract operation signature of `CreatePacket` and the three child classes which are `PoissonTraffic`, `ParetoTraffic` and `BurstyTraffic` each implement their own version of that operation.

3.1.1.1 QueST Poisson Traffic Generator

Conceptual Design: Poisson model is one of the most widely used and oldest network traffic model. Poisson process is applied in many simulation scenarios because of its nice and well defined analytical properties.

In a Poisson model, the packet inter-arrival times are exponentially distributed with a rate parameter λ where $P\{A_n \leq t\} = 1 - exp(-\lambda t)$. Therefore, the Poisson model has a mean and variance equal to the parameter λ.

The poisson model can be seen as a limiting form of a widely used binomial process. According to Burke's Theorem [15], aggregation of independent Poisson process results in a new Poisson process whose rate is sum of independent Poisson arrival rates. In order to calculate inter-arrival times for Poisson process, the probability distribution function of Poisson process was used in our study. The probability distribution function and density function of Poisson process as follows:

$$Distribution\ \ Function : F(t) = 1 - e^{-\lambda t} \tag{3.1}$$

$$Density\ \ Function : f(t) = \lambda e^{-\lambda t} \tag{3.2}$$

Software Design: Poisson traffic generator module is implemented under `PoissonTraffic` class which is inherited from `Traffic` class. As seen in Figure 3.3, the `PoissonTraffic` class has two operations, four attributes and one constructor.

In OOP, the constructor method is a special function which is used to create an object of the class. The constructor is an instance method that usually has the same name as the class and can be used to set class attributes. In `PoissonTraffic` constructor, the attributes are initialized.

The `setPoissonParameters(id : int, meanrate : double)` operation sets load of traffic and flow id whose attributes names are `OfferedLoad` and `flowId` respectively. This operation takes two inputs. The first input is used to set the `flowId`. The second input is used to calculate the `OfferedLoad` which is calculated as : `OfferedLoad = meanrate/avgPacketSize`. This function is called at the initialization part of the QueST under `main` class.

The `CreatePacket(cycle : int)` operation creates packets and transmits those packets to the traffic shaper module. This operation takes one input, `cycle`, which is the cycle number of the scheduler. This operation is called each cycle and if the the packet is generated, the operation returns true. Otherwise, the operation returns false. As seen in Figure 3.4, flow diagram of the operation is as follows:

1. A packet is created by using `Packet` class.

2. Flow id of created packet is set to `flowId`. Packet size is calculated and saturated between 0 and 100 bytes. Generation time of the packet is set to `cycle`.

3. A random number between 0 and 1 is calculated and `RandomNumber` is set to obtained random number.

4. From Equation 3.1, the inter-arrival time is calculated as follows:

$$Interarrival\ \ Time = -\frac{1}{\lambda} ln(1 - F(t)), \tag{3.3}$$

   where $F(t)$ is a random number and λ is load of traffic. Therefore, from Equation 3.3;

$$interarrivaltime = (-1/OfferedLoad) * ln(1 - RandomNumber). \tag{3.4}$$

5. `nextarrivaltime` is updated by adding obtained `interarrivaltime`.

6. The created packet is sent to the traffic shaper module.

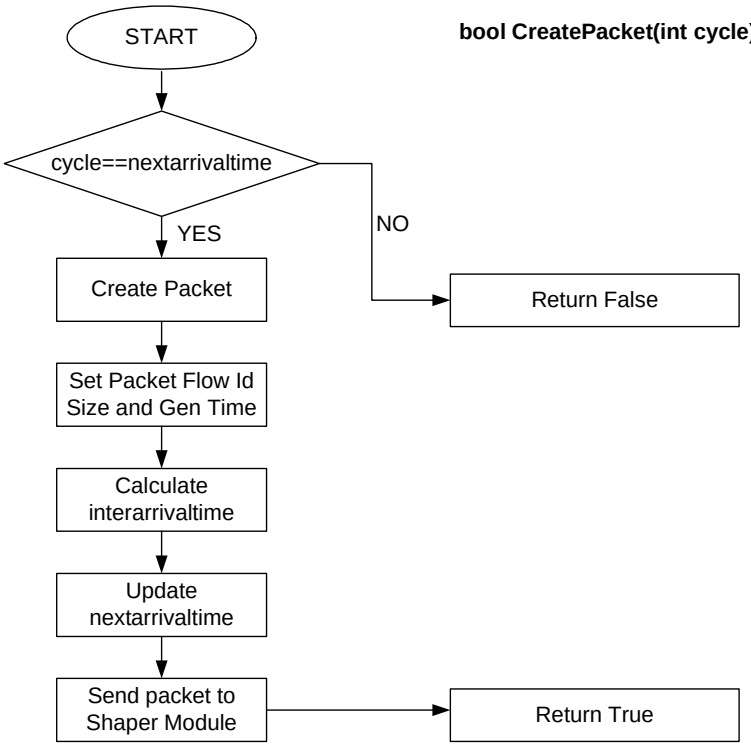

Figure 3.4: Flow Chart of CreatePacket Function in Poisson Traffic Generator Module

3.1.1.2 QueST Bursty Traffic Generator

Conceptual Design: Bursty traffic model is another widely used traffic model. In order to get the scaling behaviour of network traffic, the Bursty traffic model is frequently used. For example, evaluation of IP traffic is mostly accomplished by using Bursty traffic model.

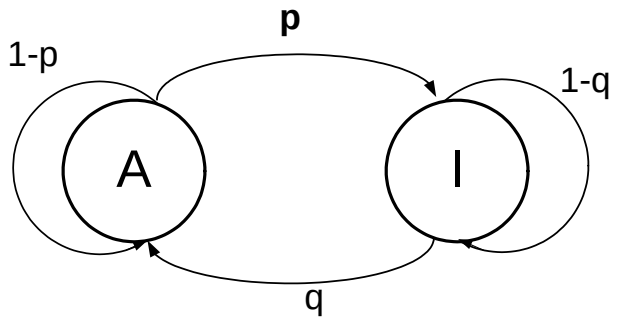

Figure 3.5: The state transition of Bursty traffic generator with probabilities

In QueST, Bursty model has two states which are active and idle states and changes its state between active and idle states periodically. In active state, Bursty traffic model generates packet continuously. On the contrary, the Bursty model does not generate any packet in idle state. The time spent in these states changes geometrically. Figure 3.5 shows state transition diagram of Bursty traffic generator with probabilities. As seen in Figure 3.5, the probability of changing state to the other state is constant. p is the probability of leaving active state and q is the probability of leaving idle state.

The probability of being active state in i packet times is:

$$Pr\{Active\ \ period = i\ \ packet\ \ times\} = p(1-p)^{i-1} \tag{3.5}$$

The probability of being idle state in j packet times is:

$$Pr\{Idle\ \ period = j\ \ packet\ \ times\} = q(1-q)^{j-1} \tag{3.6}$$

From equation 3.5, mean burst length is :

$$\beta = \sum_{i=1}^{\infty} p(1-p)^{i-1} i = \frac{1}{p} \tag{3.7}$$

From equation 3.6, mean idle period is :

$$mean\ \ idle\ \ period = \sum_{j=1}^{\infty} q(1-q)^{j-1} j = \frac{1}{q} \tag{3.8}$$

Offered load is the ratio of mean burst length to total time. Let ρ be the offered load of Bursty traffic generator. From equation 3.7 and equation 3.8 :

Offered load = mean burst length / (mean burst length + mean idle period)

$$\rho = \frac{\frac{1}{p}}{\frac{1}{p} + \frac{1}{q}} \tag{3.9}$$

From given mean burst length β and offered load ρ, the probabilities of state transitions can be calculated as follows :

$$p = \frac{1}{\beta}\ \ and\ \ q = \frac{\rho}{\beta(1-\rho)} \tag{3.10}$$

Software Design: `BurstyTraffic` class which is child class of `Traffic` class creates Bursty traffic generator module. As seen in Figure 3.3, the `BurstyTraffic`

class has two operations, four attributes and one constructor. The constructor of `BurstyTraffic` class which is also called `BurstyTraffic` creates an object of the class. In the constructor, the attributes of the class are initialized. Initially, the bursty traffic state, `burstyTrafficState`, is set to active.

The `setBurstyParameters(id : int, meanrate : double, burstsize : double)` operation sets load of traffic, flow id and burst size whose attributes names are `OfferedLoad`, `flowId` and `burstSize` respectively. This operation takes three inputs. The first input is used to set `flowId`. The second input is used to calculate `OfferedLoad` which is calculated as : `OfferedLoad = meanrate/avgPacketSize`. The third input is used to calculate `burstSize` which is calculated as : `burstSize = burstsize/avgPacketSize`. This function is called at the initialization part of the QueST under `main` class.

Since the `CreatePacket(cycle : int)` operation is abstract operation, each traffic generator modules implement this operation with respect to their own algorithms. Therefore, in all traffic generator modules, although main idea of this operation is same, algorithms of this operation changes for each module. Figure 3.6 shows flow chart of this operation in `BurstyTraffic` class. Flow diagram of the operation in `BurstyTraffic` is as follows:

1. Probability of transition from active to idle state, `p`, is calculated by using Equation 3.10.
2. Probability of transition from idle to active state, `q`, is calculated by using Equation 3.10.
3. If state of the class is active, then:
   (a) A packet is created.
   (b) Flow id of created packet is set to `flowId`. Packet size is calculated and saturated between 0 and 100 bytes. Generation time of the packet is set to `cycle`.
   (c) The created packet is sent to the traffic shaper module.
   (d) If the probability of staying active state is higher than `RandomNumber`,

state of the class becomes active. Otherwise, state of the class becomes idle.

4. If state of the class is idle, then the probability of staying idle state is compared with `RandomNumber`. If the probability is higher than `RandomNumber`, state of the class becomes idle. Otherwise, state of the class becomes active.

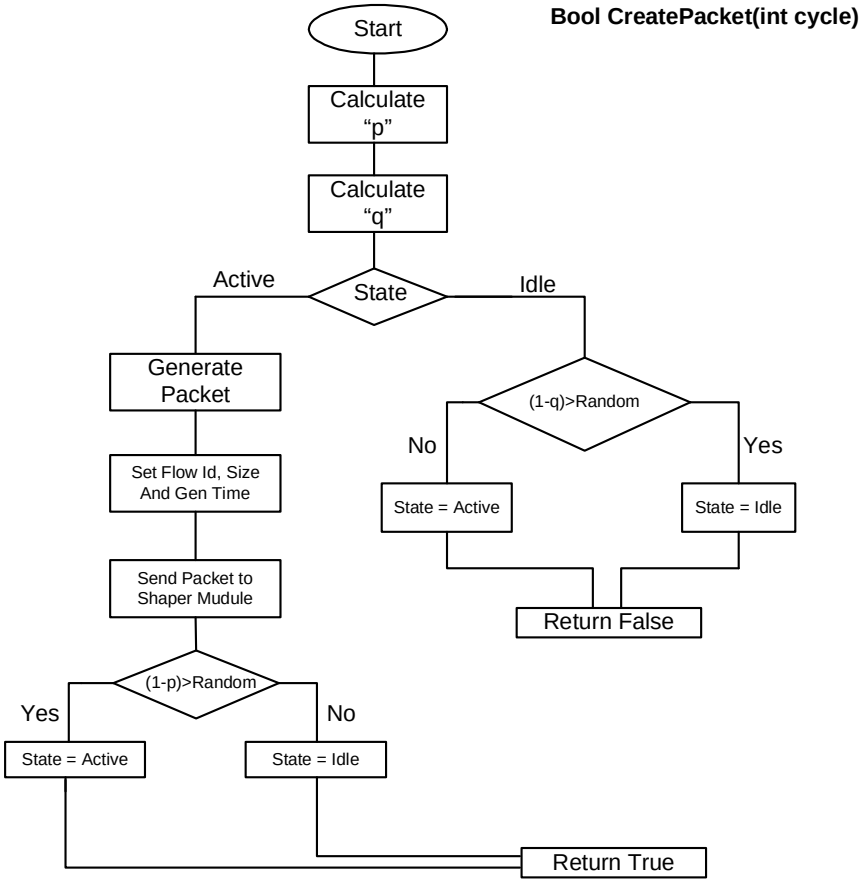

Figure 3.6: Flow Chart of CreatePacket Function in Bursty Traffic Generator Module

3.1.1.3 QueST Pareto Traffic Generator

Conceptual Design: Pareto traffic model which is suggested to model "heavy-tailed" traffic trains is used to produce independent and identically distributed inter-arrival times. The Pareto traffic model can also be used to model current IP traffic instead of Bursty traffic. It is also referred as power law distribution.

In general, if X is a random variable with a Pareto distribution, the probability that X is greater than any number x is:

$$P(X > x) = (x/x_m)^{-k} \quad for \ \ all \ \ x >= x_m \tag{3.11}$$

where k is a positive number and x_m is the minimum possible value of X_i. In QueST, the inter-arrival times of generated packets for Pareto model is calculated by using the probability distribution function of Pareto distribution. The probability distribution function and density function of Pareto process as follows :

$$Distribution \ \ Function : F(t) = 1 - (\alpha/t)^{\beta} \tag{3.12}$$

$$Density \ \ Function : f(t) = \beta\alpha^{\beta}t^{-\beta-1} \tag{3.13}$$

The parameters α and β which are used in Equation 3.12 are location and shape parameters respectively. When $\beta > 2$, the Pareto model has infinite variance and when $\beta < 1$, the Pareto model has infinite mean.

Software Design: `ParetoTraffic` class which is inherited by `Traffic` class creates Pareto traffic generator module. As seen in Figure 3.3, the `ParetoTraffic` class has two operations, five attributes and one constructor. The constructor of `ParetoTraffic` class creates an object of the that class. In the constructor, the attributes of the class are initialized.

The `setParetoParameters(id : int, shape : double, location : double)` operation takes three inputs. These inputs are used to set flow id, shape and location parameters whose attributes names are `flowId`, `_shape` and `_location` respectively. The first input is used to set `flowId`. The second input is used to set `_shape`. The third input is used to set `_location`. This function is called at the initialization part of the QueST under `main` class.

Similar to the other child classes of `Traffic` class, the `CreatePacket(cycle : int)` operation in `ParetoTraffic` class is used to create network packets by applying Pareto traffic algorithm. If the operation can not create packet, it returns false. Otherwise, it returns true. Figure 3.7 shows flow chart of this operation in `ParetoTraffic` class. Flow diagram of the operation in `ParetoTraffic` is as follows:

1. A packet is created by using `Packet` class.

2. Flow id of created packet is set to `flowId`. Packet size is calculated and saturated between 0 and 100 bytes. Generation time of the packet is set to `cycle`.

3. A random number between 0 and 1 is calculated and `RandomNumber` is set to obtained random number.

4. From Equation 3.12, the inter-arrival time is calculated as follows:

$$Interarrival\ \ Time = \frac{\alpha}{(1 - F(t))^{\frac{1}{\beta}}}, \tag{3.14}$$

   where $F(t)$ is a random number, α is location and β is shape. Therefore, from Equation 3.14;

$$interarrivaltime = \_location/(1 - RandomNumber)^{(1/\_shape)} \tag{3.15}$$

5. `nextarrivaltime` is updated by adding obtained `interarrivaltime`.

6. The created packet is sent to the traffic shaper module.

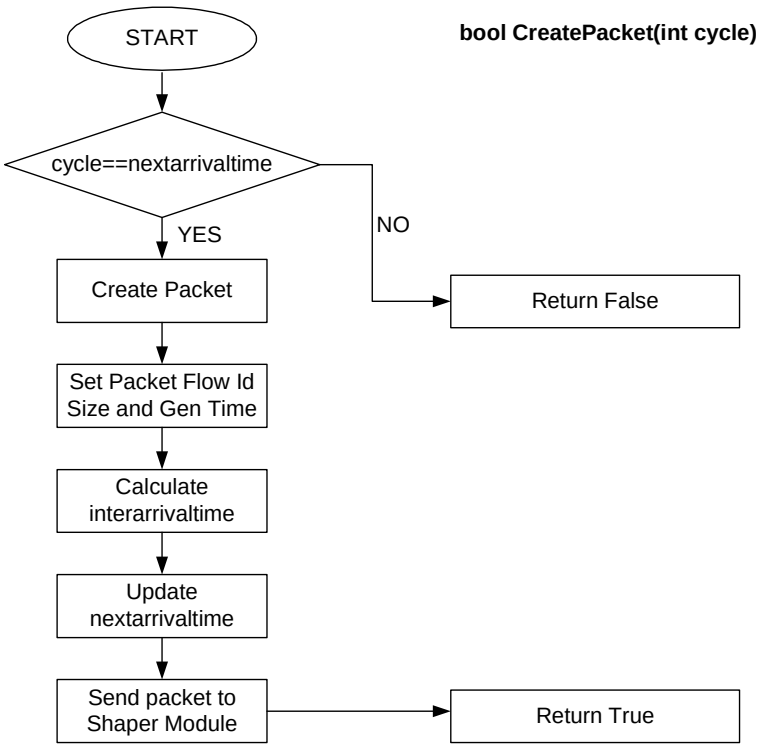

Figure 3.7: Flow Chart of CreatePacket Function in Pareto Traffic Generator Module

3.1.2 Traffic Shaper

Traffic shaping is performed in IP networks. Traffic shaping module controls rates and bursts of traffic flows generated by traffic generator module to ensure conformance to a traffic contract. If a traffic flow does not obey the traffic contract, then the traffic shaper module delays traffic flows by storing them in buffers until traffic flows obey the contract. If a shaper has a finite size buffer, some packets may be discarded. However, in QueST, the traffic shaper module has infinite size buffer so the packets are only delayed.

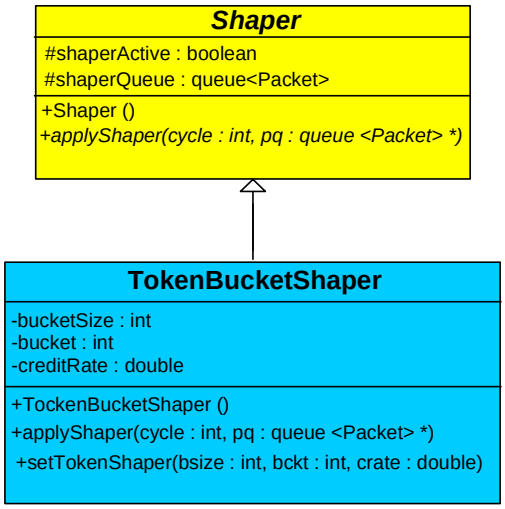

Figure 3.8: Class Diagram of Traffic Shaper Module

Traffic shaper module in QueST is software-based traffic shaper which is implemented in C++. Figure 3.8 shows class diagram of traffic shaper module. As seen in Figure 3.8, this module implements Token Bucket Shaper. In QueST, the shaper module is applied when the user enables shaper module. The shaped packets are sent to packet scheduling module.

In QueST, traffic shaper module also uses inheritance property of OOP language to implement shapers. As seen in Figure 3.8, the `Shaper` class is abstract class and the `applyShaper` operation is abstract operation. Therefore, the child class `TockenBucketShaper` implements its own shaper algorithm to the `applyShaper` operation.

3.1.2.1 QueST Token Bucket Shaper

Conceptual Design: Token bucket shaper is used in IP networks to control average flow rate. In the token bucket shaper, tokens are inserted to the bucket at a certain rate. If the bucket is full of tokens, newly incoming tokens are discarded. If the bucket does not have enough tokens to send a packet, the packets are waited until the bucket has enough tokens. In order to transmit a packet, the token bucket shaper removes tokens whose number is equal to the packet size from the bucket.

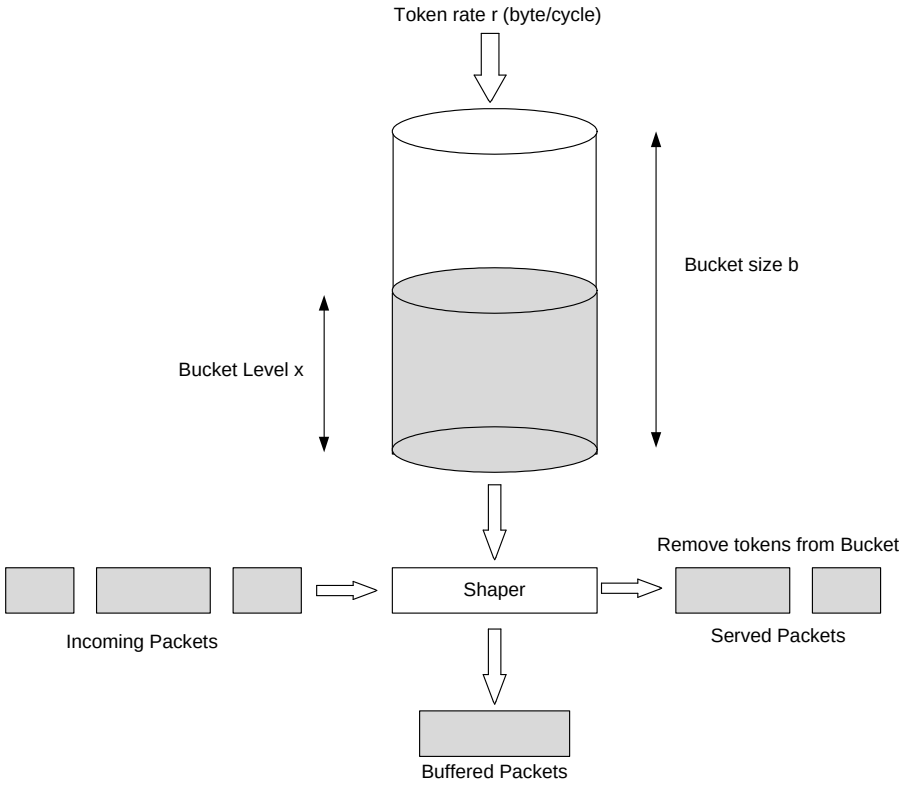

Figure 3.9: Operation of Token Bucket Shaper in QueST

Figure 3.9 shows token bucket shaper operation in QueST. As can be seen in Figure 3.9, the token bucket shaper has three components:

- *Token Rate r :* It is measured in bytes per cycle on average permitted by the token bucket.
- *Bucket Size b :* It shows total bucket size in bytes

- *Bucket Level x :* It shows the current bucket size in bytes.

The token bucket shaper module adds tokens to bucket at a r $byte/cycle$ rate. If incoming packet size L is equal or lower than bucket level x, $L \leq x$, then the packet is transmitted and L bytes from bucket level x is removed. Otherwise, the packet does not conform the contract and it is buffered. Therefore, the bucket level does not change. The amount of data transmitted over time interval t, $D(t)$, is described as follows:

$$D(t) \leq rt + b \tag{3.16}$$

From Equation 3.16, the actual average rate, $A(t)$, can be described as $A(t) = D(t)/t = r + b/t$. Therefore, as $t \Rightarrow \infty$, the actual rate $A(t)$ goes token rate r.

Software Design: `TokenBucketShaper` class which is inherited by `Shaper` class shapes incoming packets by using token bucket shaper algorithm. Figure 3.8 shows that the `TokenBucketShaper` class has two operations, three attributes and one constructor. In the constructor, the attributes of the class are initialized and the state of shaper is set to passive state.

The `setTokenShaper(bsize : double, bckt : double, crate : double)` takes three inputs. The first input is used to set bucket size, `bucketSize`. The second input is used to set current bucket level, `bucket`. The third input is used to set token rate, `creditRate`. This function is called at the initialization part of the QueST under `main` class. If this operation is called, shaper becomes active and packets are shaped. Otherwise, shaper becomes passive and shaping algorithm is not applied.

The `applyShaper(cycle : int, pq : queue<Packet>*)` operation shapes incoming packets and sends those packets to the next module. Since this operation is an abstract operation, token bucket shaper module implements the operation with respect to its own algorithm. The operation takes two inputs which are `cycle` and `pq`. The first input is the cycle number of scheduler and the second input is the buffer of shaper module. Figure 3.10 shows flow chart of this operation in `TokenBucketShaper` class.

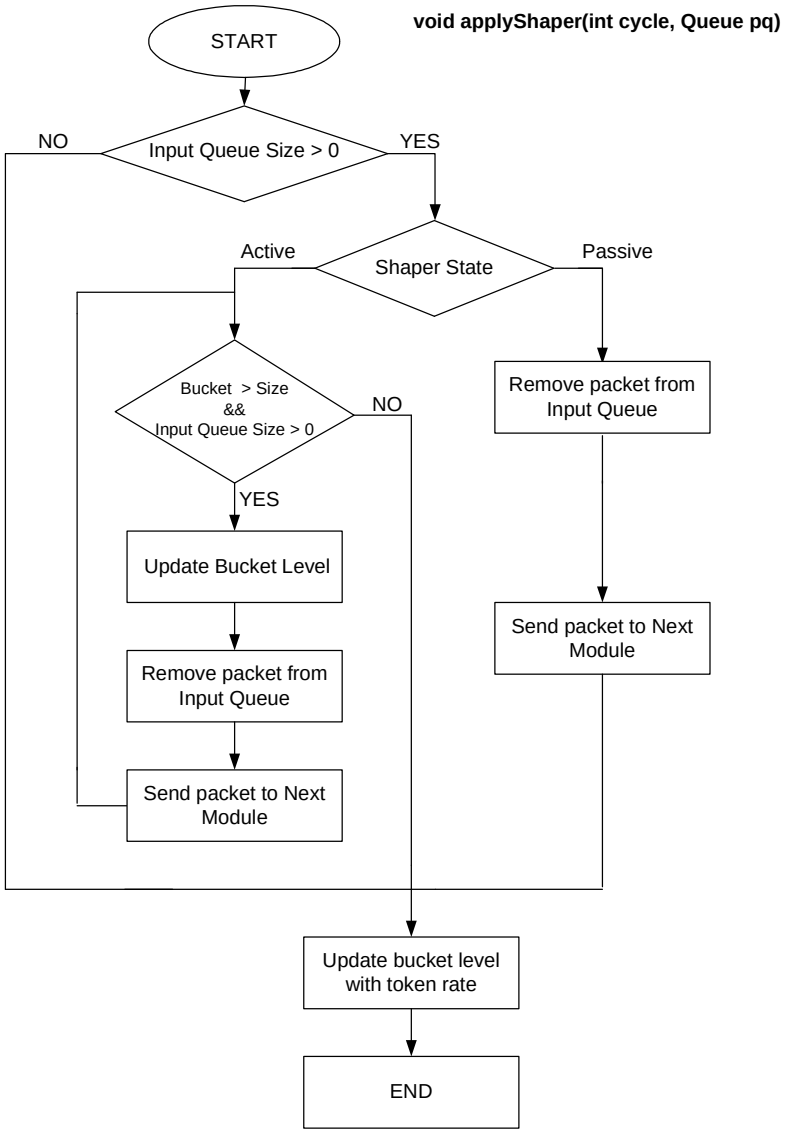

Figure 3.10: Flow Chart of applyShaper Function in Token Bucket Shaper Module

Flow diagram of the operation in `TokenBucketShaper` is as follows:

1. If there is not any packet in the input buffer of shaper, bucket level, `bckt`, is updated with token rate, `creditRate`, and the operation ends.
2. If the input buffer of shaper has some packets and shaper is not active, the packets are removed from input buffer of shaper and sent directly to the next module.
3. If the input buffer of shaper has some packets and shaper is active, the following

processes are performed:

(a) While the current bucket level is greater than the size of HOL packet in the input buffer, HOL packet is removed from buffer and sent to next module.

(b) Bucket level is updated by subtracting the size of HOL packet from bucket level.

(c) Leaving time from shaper of the removed packet is set to `cycle`.

4. Bucket level is updated by adding token rate.

3.1.3 Packet Scheduler

The packet scheduling algorithms which are implemented in the packet schedulers of the routers have significant effect on providing QoS support to the IP networks. Packet scheduler is used to determine the order of queued packets transmission. Since these packets are transmitted from same outgoing link, the packet scheduler defines a set of rules for these packets by considering their QoS requirements. In other words, sophisticated scheduling algorithms are required to prioritize network users by meeting QoS requirement for each network users.

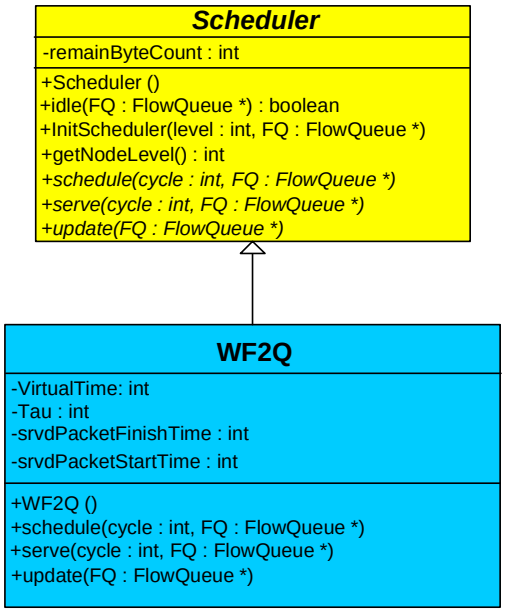

Figure 3.11: Class Diagram of Packet Scheduler Module

Packet scheduler module in QueST is implemented in C++. Figure 3.11 shows class diagram of the packet scheduler module. As seen in Figure 3.11, the packet scheduler module implements WF^2Q+ scheduling algorithm. QueST enables users to cascade many number of packet scheduler modules to each other. If the packets served by packet scheduler module is not addressed another packet scheduler module, these packets are stored in a file to analyze simulation results.

Inheritance property of OOP language is also used in Packet scheduler module. As seen in Figure 3.11, the `Scheduler` class is abstract class. The `Scheduler` class has three common operations and three virtual operations. The common operations which are `idle`, `getNodeLevel` and `InitScheduler` are common for all child classes of `Scheduler` class. The first common function `idle` determines whether a scheduler is idle or not. If a scheduler has no packet in the server and all flow queues of scheduler are empty, then the scheduler is idle. The second common function `getNodeLevel` is used to obtain id of scheduler. The third common function `InitScheduler` initializes the scheduler and sets id of scheduler. Each child classes of `Scheduler` class implement the virtual (abstract) functions which are `scheduler`, `serve` and `update` with respect to their own algorithms.

3.1.3.1 QueST Worst-case Weighted Fair Queuing Plus Scheduler

Conceptual Design: WF^2Q+ scheduling algorithm is one of the common PFQ algorithms. The WF^2Q+ scheduler provides the tightest delay bound and has the smallest worst case fairness index (WFI) value among all PFQ schedulers. It should be noted that although both WF^2Q+ and WF^2Q algorithms have same WFI and end to end delay bound, the complexity of WF^2Q+ algorithm is lower than the complexity of WF^2Q algorithm. Therefore, in QueST, we implemented WF^2Q+ scheduler. Virtual time function, $V(t)$, of the WF^2Q+ scheduler is calculated as:

$$V(t+\tau) = max\{\ \ V(t)+\tau,\ \ min_{i\in B(t)}\{S_i(t)\}\ \ \}, \tag{3.17}$$

where $B(t)$ is the set of flow queues which are backlogged at time t, and $S_i(t)$ is the virtual start time of HOL packet of backlogged flow queue i. The complexity of $V(t)$ comes from searching for the minimum start time value among all backlogged flow queues.

In order to emulate the GPS behaviour, WF^2Q+ scheduler has system virtual time $V(t)$, a virtual start time $S_i(t)$ and a virtual finish time $F_i(t)$ for each flow queue i. Start time $S_i(t)$ and finish time $F_i(t)$ are calculated when a new packet becomes HOL packet for related flow queue. A packet becomes HOL packet in either two cases:

1. When a packet comes to an empty flow queue, it becomes HOL packet.
2. When a packet depart from a flow queue, the previous packet in that queue becomes HOL packet.

In case 2, departure of packet and arrival of new packet happens at the same time. Therefore;

$$S_i(t) = \begin{cases} max\{V(t), F_i(t^-)\}, & For\ \ case\ \ 1 \\ F_i(t^-), & For\ \ case\ \ 2 \end{cases} \tag{3.18}$$

$$F_i(t) = S_i(t) + \frac{L_i^{HOL}}{r_i}, \tag{3.19}$$

where, $F_i(t^-)$ is the finish time of flow queue i before the departure, and L_i^{HOL} is the size of HOL packet for the flow queue i. In WF^2Q+ scheduler, a flow queue i is eligible if only start time $S_i(t)$ is lower and equal than $V(t)$. When the WF^2Q+ scheduler is in idle position in which no packet is transmitted and all the flow queues are not empty at that time, the scheduler selects eligible flow queue with minimum $F_i(t)$ and transmits HOL packet of that queue.

Software Design: `WF2Q` class which is inherited by `Scheduler` class selects eligible packet among HOL packet of per flow queues and serves selected packet to the destination address. In order to select eligible packet, the `WF2Q` class uses WF^2Q+ scheduling algorithm. As shown in 3.11, the `WF2Q` class has three operations, four attributes and one constructor. In the constructor, the attributes of the class are initialized.

The `schedule(cycle : int, FQ : FlowQueue*)` operation selects eligible packet among flow queues and sends selected packet to the server of scheduler. Since this function is a abstract function, the `WF2Q` class implements this function with respect

to WF^2Q+ scheduling algorithm. The operation takes two inputs which are `cycle` and `FQ`. The first input is the cycle number of scheduler and the second input is the per flow queues of scheduler module. Figure 3.12 shows flow chart of this operation. Flow diagram of the operation in `WF2Q` is as follows:

1. If there is a newly arriving HOL packet in the flow queues, virtual start and finish time of the HOL packet is computed. From Equation 3.18 case 1, virtual start time is calculated as:

$$FQ[i].startTime = MAX(FQ[i].prevFinishTime, VirtualTime), \tag{3.20}$$

   From Equation 3.19, virtual finish time is calculated as:

$$FQ[j].finishTime = FQ[j].startTime + \frac{FQ[j].HOLPacketSize}{FQ[j].AllocRate} \tag{3.21}$$

2. If the scheduler is idle state, the operation ends.

3. The function tries to find flow queue with minimum finish time.

4. The start and finish time of the HOL packet in the selected flow queue is saved.

5. The HOL packet in the selected flow queue is sent to the server.

6. If selected flow queue still has a HOL packet, then virtual start and finish time of the HOL packet is computed. From Equation 3.18 case 2, virtual start time is calculated as:

$$FQ[i].startTime = FQ[i].prevFinishTime, \tag{3.22}$$

   Virtual finish time is calculated in the same way with Equation 3.21.

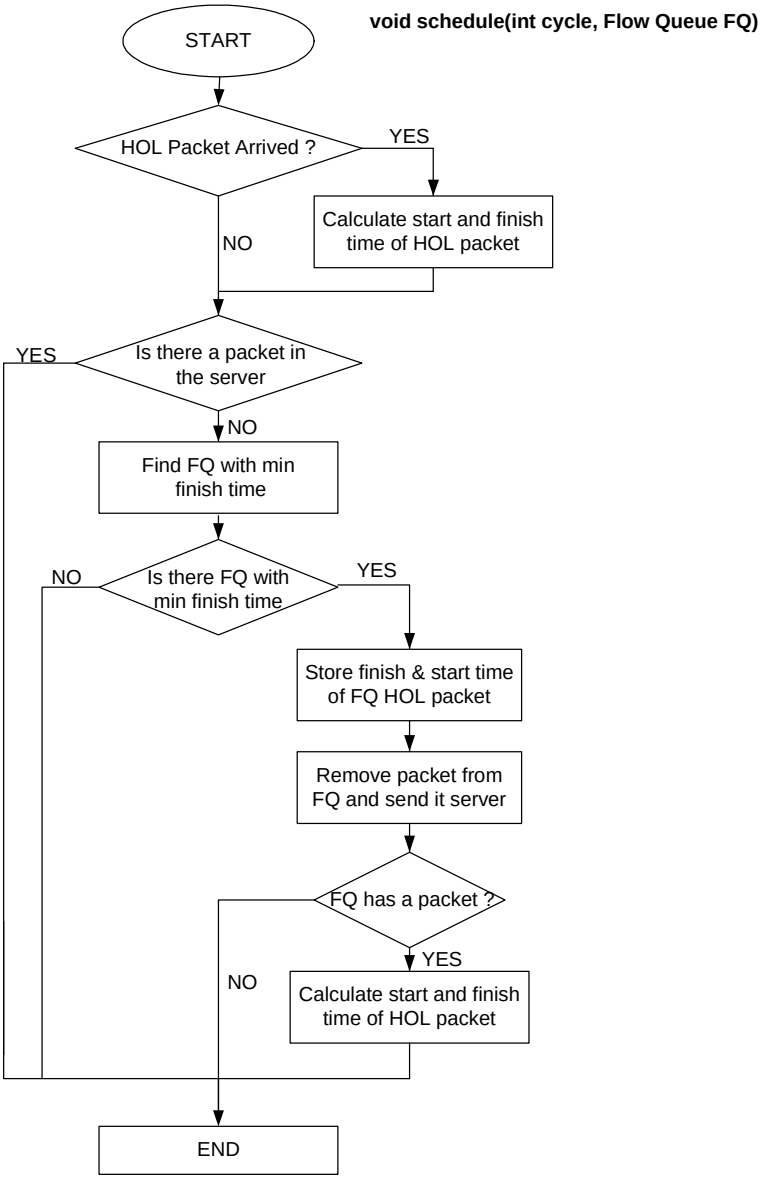

Figure 3.12: Flow Chart of schedule Function in WF2Q Scheduler Module

The `serve(cycle : int, FQ : FlowQueue*)` transmits one byte in each cycle. The operation sends transmitted packet to the destination address. Therefore, cascading different scheduler modules can be applied in this function. If the transmitted packet does not have any destination address, it is stored in a file to analyze. The operation takes two inputs which are `cycle` and `FQ`. The first input is the cycle number of scheduler and the second input is the per flow queues of scheduler modules. As seen in Figure 3.13, flow diagram of the operation in `WF2Q` is as follows:

1. If there is not any packet in the server, the function ends. Otherwise, function goes on with following processes.
2. If there is packet in the server, packet is served one byte at a cycle.
3. After one byte of data transfer, If packet is not completely transferred, the function ends. Otherwise, function goes on with following processes.
4. Serving time of the packet is set to `cycle`.
5. If served packet has a destination address, then the packet is sent to flow queues of destination module. Otherwise, the packet is stored in a file to analyze.

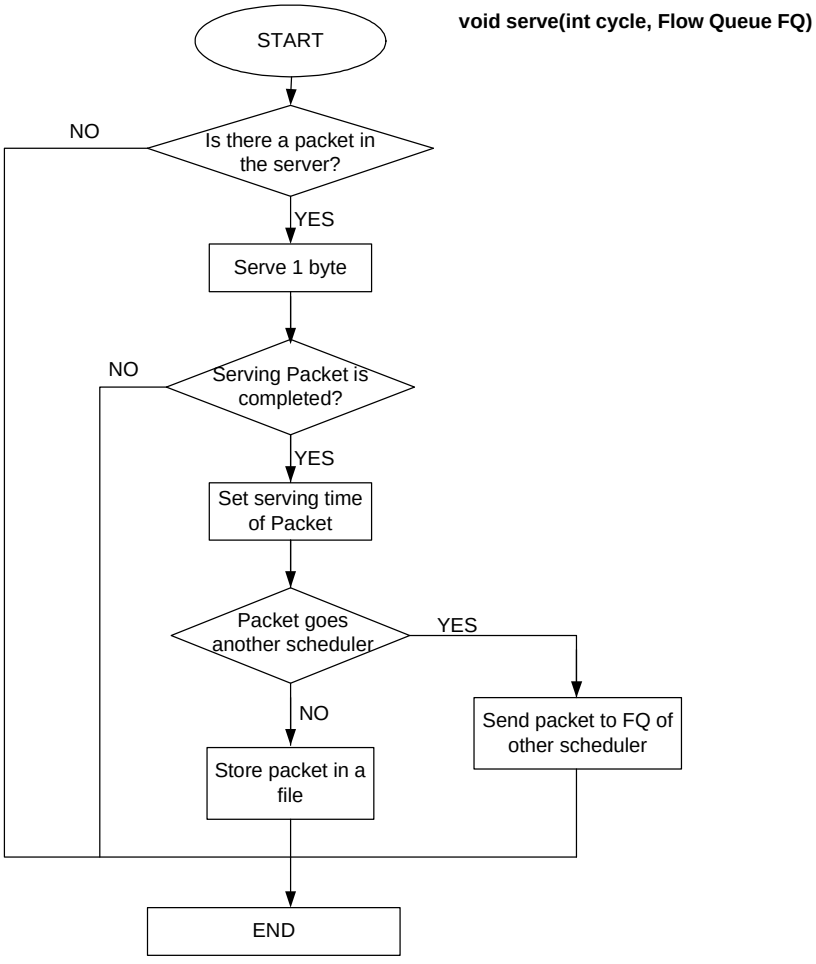

Figure 3.13: Flow Chart of serve Function in WF2Q Scheduler Module

In order to emulate GPS, WF^2Q+ scheduling algorithm uses virtual time function. The `update(FQ : FlowQueue*)` updates virtual time in each cycle with respect to

WF^2Q+ scheduling algorithm. If the scheduler module is in idle position, virtual time is set to zero. The function takes one input which is `FQ`. The input is the per flow queues of scheduler modules. Figure 3.14 shows flow chart of this operation. Flow diagram of the operation in `WF2Q` is as follows:

1. If the scheduler is in idle state, the virtual time, `VirtualTime`, finish time of previously served packet, `prevFinishTime`, and start time of previously served packet, `srvdPacketStartTime`, are set to zero. Otherwise, the function goes with following processes.

2. The function finds minimum start time in the scheduler by comparing start times of each packets in the scheduler.

3. After finding minimum start time, the function updates virtual time by using Equation 3.17:

$$VirtualTime = MAX(VirtualTime + Tau, minStartTime); \tag{3.23}$$

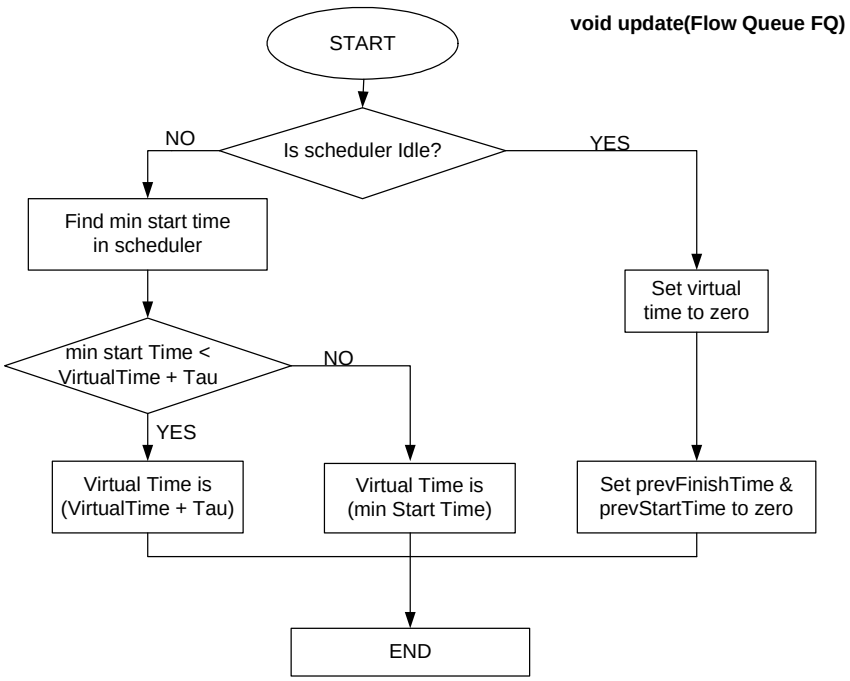

Figure 3.14: Flow Chart of update Function in WF2Q Scheduler Module

3.2 Scalability and Complexity

In this section, we discuss time complexity of QueST and modules of QueST. The time complexity of a software implies the amount of time under different input conditions. The time complexity is expressed by using big O notation. We analyze the time complexity of each QueST module by performing different experiments.

Firstly, we analyze how run time of QueST changes with generated traffic types. In QueST, we have three software traffic generator modules which are Poisson traffic generator module, Pareto traffic generator module and Bursty traffic generator module. Therefore, we perform three experiments to compare run times of each traffic generator module by collecting a set of data under different traffic loads. We execute our experiments by generating 40 flows which are equally allocated. In these experiments, traffic shaper module is not used. For each experiments one WF^2Q+ scheduler module is used and 25K packets are collected. In Figure 3.15, the experiment results are presented.

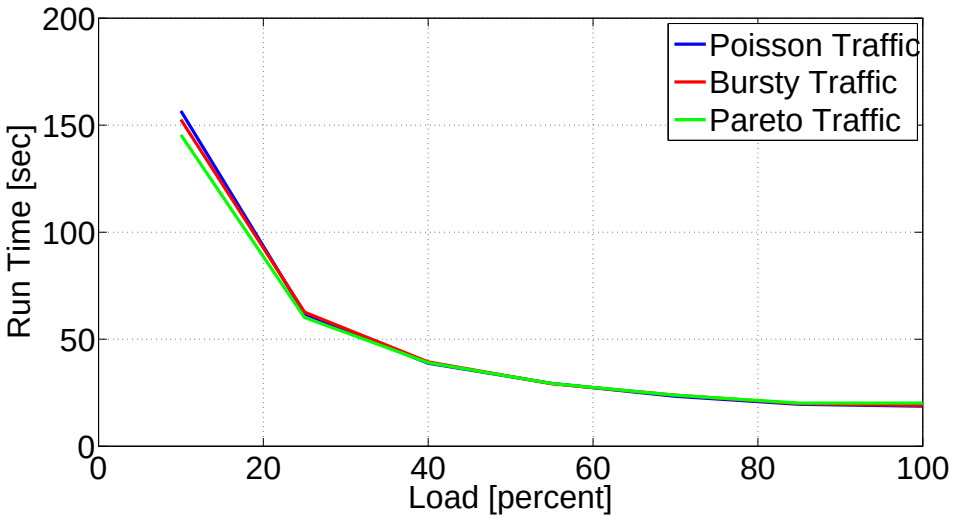

Figure 3.15: Run times of QueST under different traffic generator modules

As can be seen in Figure 3.15, run times are very close to each other under same load for each traffic generator module. This is because all the traffic generator modules behave similarly. They all generate packet and compute next generation time in each

cycle. Therefore, all the traffic generator modules in QueST have the same run times.

Secondly, we analyze how run time of QueST changes with the number of traffic flows. For this experiment, poisson traffic generator module is used to generate traffic flows without traffic shaper module. We collect run time of QueST under a set of traffic loads. For each experiment, WF^2Q+ scheduler module is used and 25K packets are collected. In Figure 3.16, the experiment result is presented.

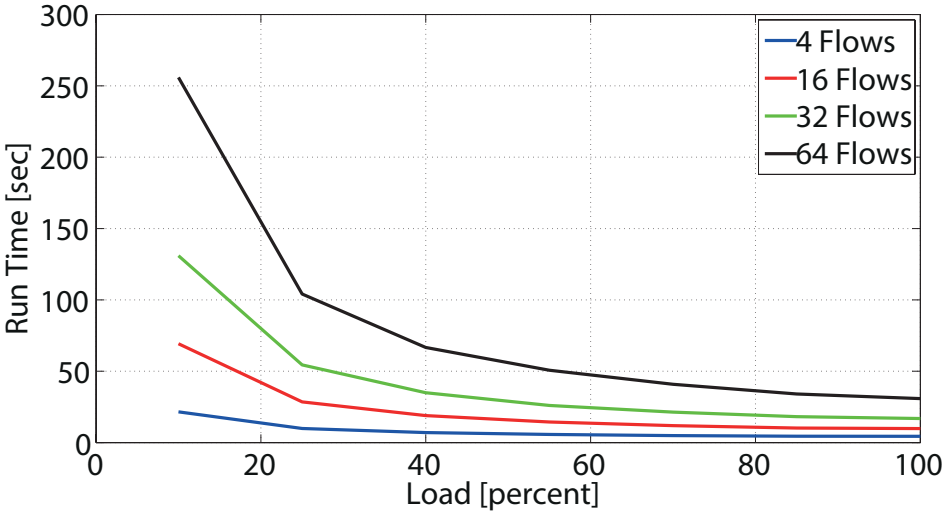

Figure 3.16: Run times of QueST under different number of traffic flows

As seen in Figure 3.16, run times of the experiments increase proportionally with the number of traffic flows under same load. This is because that when the number of traffic flows increases, the number of per flow queues in packet scheduler module also increases. Therefore, packet scheduler module spends more time to find eligible packet among per flow queues.

Finally, we analyze how run time of QueST changes with the number of cascaded packet scheduler module. Poisson traffic generator module is used to generate 32 traffic flows which are equally allocated. This experiment is composed of four step. At each step, we increase the number of WF^2Q+ scheduler module in QueST from one to four under different traffic loads and collect 25K packets. Figure 3.17 shows the experiment result.

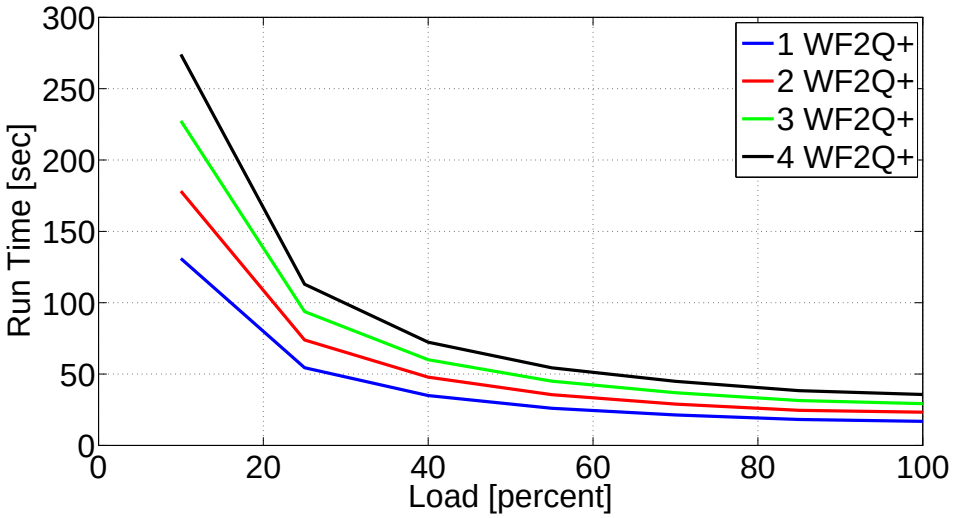

Figure 3.17: Run times of QueST under different number of WF^2Q+ scheduler modules

According to Figure 3.17, run times of the experiments increase with the number of cascaded WF^2Q+ scheduler module under same load. This increase comes from searching minimum start time among all n WF^2Q+ scheduler module.

3.3 Verification Tests

In this section, we perform verification tests for each module of QueST. We run QueST for each test and collect simulation parameters. MATLAB environment was used to evaluate collected simulation parameters.

3.3.1 Verification of Poisson Traffic Generator in QueST

The inter-arrival time between two consecutive packets for Poisson traffic generator module with load λ is exponentially distributed with a mean of $1/\lambda$. Therefore, we want to show that the inter-arrival time between two consecutive packets which are generated by poisson traffic generator module in QueST is meaningful with the desired inter-arrival times. To this end, we run 6 experiments with different load values. In our experiments, the packet sizes are exponentially distributed between 0 and 100

bytes. The average packet size is set to 10 bytes. In these experiments, we collect more than 25000 packets with their generation times. The load values are presented in Table 3.1

Table3.1: Poisson traffic experiment loads

Experiment i	Load (λ_i)
1	0.07
2	0.15
3	0.3
4	0.6
5	0.8
6	1.0

In this verification test, we demonstrate the cumulative distribution function (CDF) of the collected inter-arrival times in comparison to computed CDF value of Poisson process. The empirical CDF of poisson traffic is the ratio of inter-packet time measurements that is less than or equal to cycle to all measurements in experiment *i*.

The computed CDF value is defined as follows:

$$F_i(t_{i,j}) = 1 - e^{\lambda_i t_{i,j}}, \tag{3.24}$$

where $t_{i,j}$ is the sample inter-arrival time. In Figure 3.18, all of the experiment results can be seen.

As can be seen in Figure 3.18, the CDF is used to analyze the poisson traffic generator module. This is because CDF is the most used and meaningful statistical tool for a set of data. Therefore, when we compare empirical CDF and calculated CDF, we can see that the poisson traffic generator module in QueST can generate Poisson traffic which is very closely to the actual Poisson process.

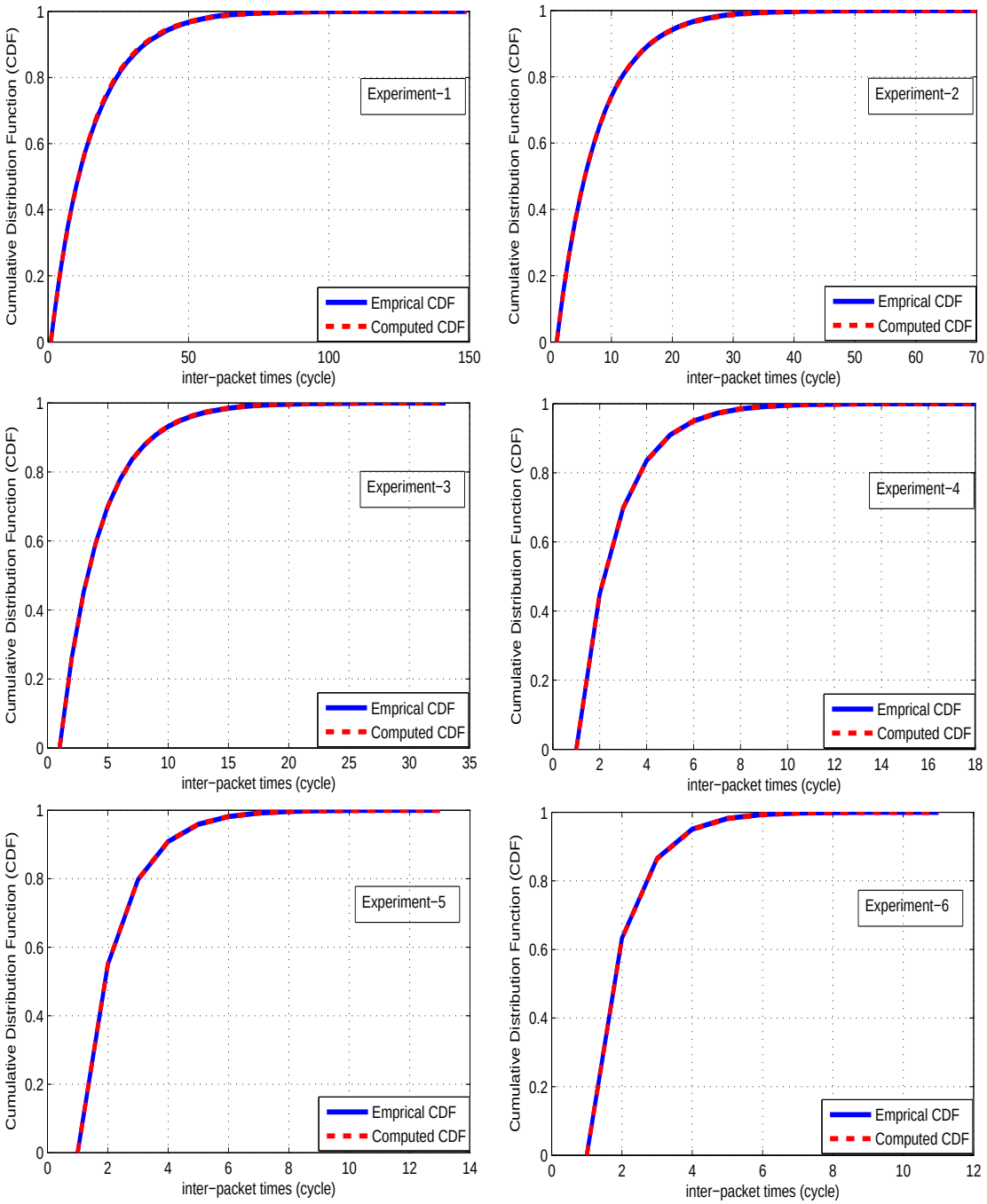

Figure 3.18: Experiment results for the comparison of poisson traffic distribution and the theoretical expectation

3.3.2 Verification of Bursty Traffic Generator in QueST

In this section, we want to show that our Bursty traffic generator module in QueST can generate Bursty traffic which is very close to Bursty process. To this end, we run 4 experiments. In each experiment, we collected more than 25000 packets. In the experiments, the packet sizes are set to 10 bytes and the burst sizes (β) are set to 10 packets. The load values are changed for each experiment. In table 3.2, the load values of experiments and the result of the experiment can be seen.

Table3.2: Bursty Traffic Generator Test Results

Experiments	Desired Values		Test Results	
	Burst Size β_i	Load ρ_i	Burst Size β_i	Load ρ_i
1	10	0.33	10.12	0.326
2	10	0.50	9.99	0.494
3	10	0.66	10.15	0.667
4	10	0.82	10.22	0.824

As can be seen in table 3.2, the Bursty traffic generator module in QueST can generate burst traffic which is very closely to the actual Markov modulated on/off process.

3.3.3 Verification of Pareto Traffic Generator in QueST

In this section, we want to show that the inter-arrival time between two consecutive packets which are generated by Pareto traffic generator module in QueST is meaningful with the desired inter-arrival times. To this end, we run 6 experiments. In each experiment, the location parameters of Pareto traffic is set to 1. The shape parameters are changed for each experiment. In these experiments, we collect more than 25000 packets with their generation times. The shape and location values are presented in Table 3.3.

Table3.3: Shape and location values of pareto traffic experiments

Experiment i	Shape (β_i)	Location (α_i)
1	0.6	1
2	1.0	1
3	1.2	1
4	1.5	1
5	1.8	1
6	2.0	1

In this verification test, we demonstrate the cumulative distribution function (CDF) of the collected inter-arrival times in comparison to computed CDF value of Pareto process. The empirical CDF of poisson traffic is the ratio of inter-packet time measurements that is less than or equal to cycle to all measurements in experiment i. The computed CDF value for Pareto traffic is defined as follows:

$$F_i(t_{i,j}) = 1 - (\frac{\alpha}{t_{i,j}})^{\beta}, \tag{3.25}$$

where $t_{i,j}$ is the sample inter-arrival time. In Figure 3.19, all of the experiment results can be seen.

As can be seen in Figure 3.19, when we compare calculated CDF and emprical CDF of Pareto traffic generator module in QueST, we can say that the Pareto traffic generator module in QueST can generate Pareto traffic which is very closely to the actual Pareto process.

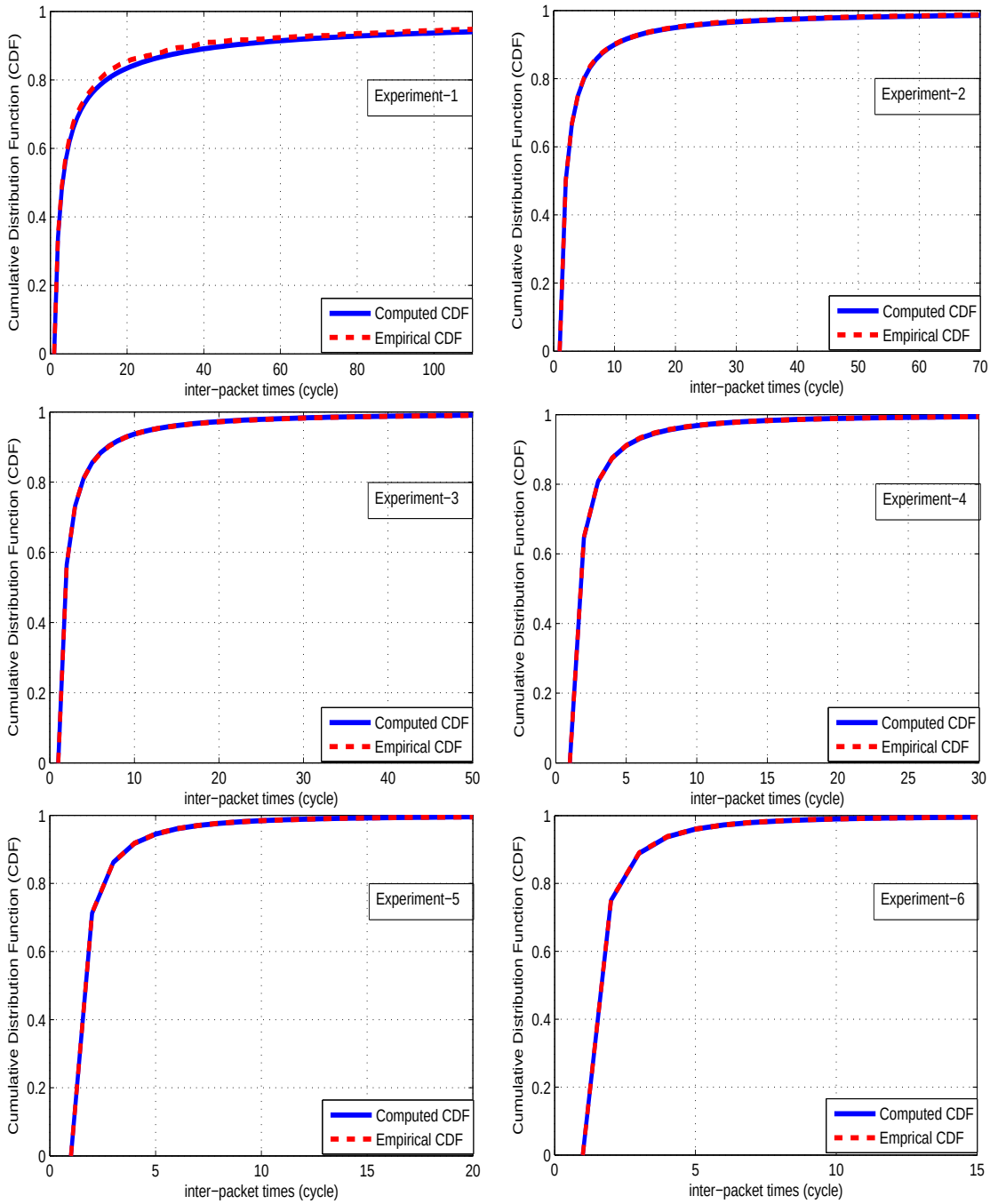

Figure 3.19: Experiment results for the comparison of pareto traffic distribution and the theoretical expectation

3.3.4 Verification of Token Bucket Shaper in QueST

In this section, we want to show that the shaper module in QueST shapes incoming packets with respect to token bucket shaper algorithm. To this end, we perform 4 experiments. In each experiment, the poisson traffic generator module with mean load 0.9 is used to generate packets. In the first two experiments, bucket sizes are assigned to 250 bytes. In the last two experiments, the bucket sizes are assigned to 300 bytes. Table 3.4 shows the used bucket sizes and token rates for each experiment.

Table3.4: Token Bucket Shaper Experiment Values

Experiment i	Token Rate i (byte/cycle)	Burst Size (byte)
1	5	250
2	10	250
3	15	300
4	20	300

In these experiments, firstly theoretical value of the amount of data which is sent in a specified time interval (300-3000 cycles) comparison to the calculated number of data is demonstrated. The theoretical value of the amount of data which is sent in a specified time interval can be calculated by using Equation 3.16. Furthermore, the actual rates of shaper comparison to token rates is demonstrated. In table 3.5, the result of the experiments can be seen.

Table3.5: Token Bucket Shaper Experiment Results

Experiments	Desired Values		Test Results	
	Allowed Length	Token Rate	Calculated Length	Actual Rate
1	13750 bytes	5 byte/cycle	13492 bytes	5.000 byte/cycle
2	27250 bytes	10 byte/cycle	26950 bytes	10.001 byte/cycle
3	40800 bytes	15 byte/cycle	40535 bytes	15.001 byte/cycle
4	54300 bytes	20 byte/cycle	54070 bytes	20.001 byte/cycle

As can be seen in table 3.5, the token bucket shaper module in QueST can shapes incoming packets which is very closely to token bucket shaper algorithm.

3.3.5 Verification of WF2Q+ Scheduler in QueST

In this section, implementation of WF^2Q+ scheduling algorithm in QueST is tested under Poisson and Bursty traffic with exponentially distributed packet sizes whose average packet size is 10 bytes. In order to evaluate WF^2Q+ scheduler in QueST, two experiments are performed. In the first experiment, we used one WF^2Q+ scheduler module. In the second experiment, we used cascaded WF^2Q+ scheduler modules.

3.3.5.1 Experiment 1: Verification of WF2Q+ Scheduler with one WF2Q+ module

In this experiment, average and maximum packet delay in the WF^2Q+ scheduler module are measured and then measured values are compared with the analytical delay bound that is defined below for flow f_j.

$$D_{max} = \frac{\sigma_j + L_{max}}{r_j} + \frac{L_{max}}{C}, \tag{3.26}$$

where σ_j is the maximum burst size for flow f_j. r_j is the allocated rate for f_j. C and L_{max} represent the server rate and maximum packet size respectively.

This experiment is performed under eight different scenarios. In the first four scenarios, the poisson traffic generator module is used to generate packets whose sizes are exponentially distributed between 0 and 100 bytes. The average packet size is 10 byte for each test. The total load of poisson traffic flows are %40, %60, %80 and %96. The mean per flow traffic generation rate is equal among all flows. The server rate of scheduler is 1 byte/cycle. In the second four scenarios, the Bursty traffic generator module is used to generate packets whose sizes are 10 bytes. The average burst sizes of all Bursty flows are assigned to 20 bytes. Total traffic load of these flows and allocation rates of flow queues are same as with the first four scenarios. The packets generated by the Bursty traffic generator module first enter shaper which constraints maximum burst size to 400 bytes and the average rate per flow to 0.1875 byte/cycle. Table 3.6 gives the experimental values for each scenarios.

Table3.6: Experiment-1 WF2Q+ scheduler test values for each scenarios

Scenarios i	Traffic Type	Each Flow Load	Burst Size	Total Load	Server Rate
1	Poisson	0.10 byte/cycle	-	%40	1 byte/cycle
2	Poisson	0.15 byte/cycle	-	%60	1 byte/cycle
3	Poisson	0.20 byte/cycle	-	%80	1 byte/cycle
4	Poisson	0.24 byte/cycle	-	%96	1 byte/cycle
5	Bursty	0.10 byte/cycle	20 byte	%40	1 byte/cycle
6	Bursty	0.15 byte/cycle	20 byte	%60	1 byte/cycle
7	Bursty	0.20 byte/cycle	20 byte	%80	1 byte/cycle
8	Bursty	0.24 byte/cycle	20 byte	%96	1 byte/cycle

The experiment results are presented in Table 3.7.

Table3.7: Experiment-1 WF2Q+ scheduler test results for each scenarios

Scenarios i	Total Load	Avg Queue Delay	Max Queue Delay	Delay Bound
1	%40	3.65 cycle	92 cycle	837 cycle
2	%60	6.08 cycle	110 cycle	900 cycle
3	%80	9.32 cycle	110 cycle	882 cycle
4	%96	18.75 cycle	158 cycle	900 cycle
5	%40	7.67 cycle	138 cycle	1004 cycle
6	%60	17.63 cycle	308 cycle	1114 cycle
7	%80	18.01 cycle	444 cycle	1017 cycle
8	%96	20.64 cycle	580 cycle	928 cycle

The maximum packet transmission delay for scenarios with poisson traffic is 100 cycles. This is because the maximum packet size is 100 bytes and server rate is 1 byte/cycle. If we add this maximum transmission delay to maximum queuing delay that is obtained in Table 3.7, we get a total maximum delay of 258 cycle. In this experiment, since we use equally allocated flows, allocated rate r_i is 0.25 byte/cycle for each flow. When we assume that the maximum burst size is 0 byte then from Equation 3.26, the maximum delay bound is 500 cycles which is much higher than measured end to end delay.

The maximum packet transmission delay for scenarios with Bursty traffic is 100 cycles. If we add this maximum transmission delay to maximum queuing delay that is

obtained in Table 3.7, we get a total maximum delay of 680 cycle. When we assume that the maximum burst size is 400 bytes which comes from maximum burst size of used shaper then from Equation 3.26, the maximum delay bound is 2100 cycles and much higher than measured queueing delay.

3.3.5.2 Experiment 2: Verification of WF2Q+ Scheduler with cascaded 4 WF2Q+ modules

In this experiment, average and maximum packet delay in the cascaded four WF^2Q+ scheduler modules are measured and then measured values are compared with the analytical delay bound that is defined below for flow f_j.

$$D_{max} = \frac{\sigma_j}{r_j} + n * \frac{L_{max}}{r_j} + \sum_{i=1}^{n} \frac{L_{max}}{C} \tag{3.27}$$

where n is the number of cascaded WF^2Q+ scheduler modules. σ_j is the maximum burst size of traffic flows. r_j is the allocated rates of traffic flows. C and L_{max} represent the server rate and maximum packet size respectively.

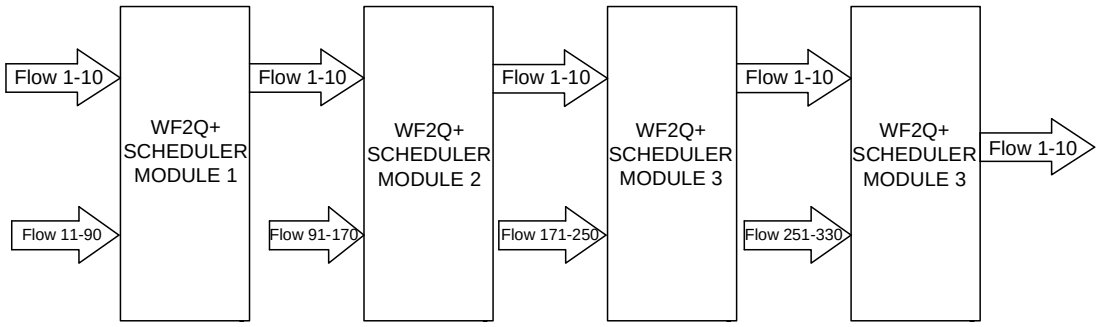

Figure 3.20: Flow diagram of experiment -2

In this experiment, we used 330 traffic flows which are generated by Bursty traffic generator module. The sizes of generated packets are exponentially distributed between 0 and 100 bytes. The average packet size is assigned to 10 bytes. We performed the experiment by using four WF^2Q+ scheduler modules. Each of WF^2Q+ scheduler modules has 90 flow queues which are allocated with equally. As seen in

figure 3.20, while flows from number 1 to 10 are transmitted through the four WF^2Q+ scheduler modules, the others are transmitted only one WF^2Q+ scheduler module. At the end of the experiment, we collected statistical values of flow from number 1 to 10. The experiment consists of four different test scenarios. The total load of all flows for each test scenarios are %40, %60 ,%80 and %90 respectively. The mean per flow traffic generation rate is equal among all flows.

When we assume that the maximum burst size is 0 byte then from equation 3.27, the maximum delay bound for four WF^2Q+ scheduler modules is 36763 cycle. As can be seen in table 3.8, although we assume the maximum burst size is 0 byte, the calculated maximum delay bound is much higher than maximum measured end to end delay.

The experiment results are presented in Table 3.8.

Table3.8: Experiment-2 WF2Q+ scheduler test results for each scenarios

Scenarios i	Total Load	Avg E2E Delay	Max E2E Delay	Delay Bound
1	%40	100.80 cycle	2485 cycle	39228 cycle
2	%60	119.9 cycle	3165 cycle	40512 cycle
3	%80	283.82 cycle	6768 cycle	44230 cycle
4	%96	702.25 cycle	10868 cycle	44582 cycle

CHAPTER 4

WINDOW BASED FAIR AGGREGATOR (WBFA) EVALUATION WITH QUEST

The QueST is a modular simulator so that adding new modules and creating new classes for QueST do not affect the existing structure and classes of QueST. Therefore, QueST has suitable structure for modifying and expanding. In other words, when a new class or module is added to QueST, the use of newly added module or class is achieved by creating its object in the `main` class of QueST without changing overall structure.

In this chapter, we integrated Window Based Fair Aggregator (WBFA) [60] to QueST. Since QueST enables users to cascade many number of modules, we cascaded WBFA module to the Packet scheduler module. We analyzed the performance of packet scheduler module by integrating WBFA module under different test scenarios. Furthermore, we proposed optimal maximum window size value for the WBFA.

4.1 Window Based Fair Aggregator (WBFA)

Increasing number of network users has negative affect on the provided QoS support for each user. Therefore, the complexity of the packet scheduling algorithms has been increasing with the number of traffic flows. This is an observed result in the experiment for run time evaluation of QueST under different number of flows in Figure 3.16. In the literature, many solutions are proposed to decrease negative effect of increasing number of flows on packet scheduling algorithms. Flow aggregation is one of these solutions. Supporting QoS to each constituent flows in the aggregator is the

most significant problem for the flow aggregation. If one of the constituent flows has greedy behaviour, the other flows may have low QoS support.

In order to provide QoS guarantee for each constituent flow, fair flow aggregation must be performed. [18] proposes a framework that a fair aggregator preserves end to end delay bounds of the constituent flows with respect to the case no flow aggregator is used. These delay bounds are not dependent to allocated rates of constituent flows. In other words, if a flow aggregator limits service rate of constituent flows, it achieves fairness among constituent flows. Therefore, service link is not fully used by greedy flow and average delays of constituent flows increase.

[60] proposes a fair aggregator which is called Window Based Fair Aggregator (WBFA). The delay bound for constituent flows of WBFA are preserved and independent of allocated rates of constituent flows. In WBFA, constituent flows use the full capacity of the output link until the difference between received services among flows reaches a limit which is called window function in WBFA. Therefore, the constituent flows use output link more efficiently than the aggregators proposed in [18] and have lower end to end delay.

4.1.1 WBFA Operation

In this section, we will mention about WBFA operation with respect to [18, 60]. We give preliminaries before we present WBFA operation.

4.1.1.1 Preliminaries

A computer network consists of cascaded packet schedulers and aggregators. In this part of the chapter, we will use following notations for each flow f and scheduler s.

- $R.f$: forwarding rate of flow f
- $p.f.i$: ith packet of flow f
- $S.s.f.i$: start time of packet i in flow f

- $E.s.f.i$: exit time of packet i in flow f
- $C.s$: output link capacity of scheduler s

The start time of a packet is the time at which packet is forwarded to channel. If flow f is only input flow, then a scheduler is start time scheduler if and only if;

$$E.s.f.i = S.s.f.i + \delta.s.f.i \tag{4.1}$$

where $\delta.s.f.i$ is a constant.

If flow f enters two cascaded schedulers which are t and s then

$$S.t.f.i = S.s.f.i + \Delta.s.f.i \tag{4.2}$$

where $\Delta.s.f.i$ is the maximum value of $\delta.s.f.x$ where $1 \leq x \leq i$.

If flow f traverses a sequence of schedulers from $t_1, ..., t_k$, then from Equation 4.1 and Equation 4.2, the end to end delay bound can be given as:

$$S.t_k.f.i \leq S.t_1.f.i + \sum_{x=1}^{k-1} \Delta.t_x.f.i \tag{4.3}$$

$$E.t_k.f.i \leq S.t_k.f.i + \delta.t_k.f.i \tag{4.4}$$

An aggregator is a scheduler. It receives a number of input flows and produces a single output by aggregating input flows. Figure 4.1 shows an aggregator A and a scheduler S. Aggregator A consists of two constituent flows which are f and h. Output of the aggregator A is a single flow g which enters scheduler S.

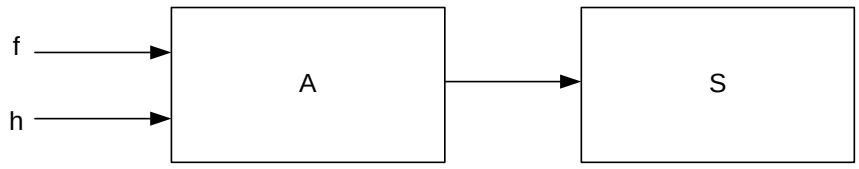

Figure 4.1: An aggregator and a scheduler

The aggregator A is defined as fair aggregator if and only if;

$$S.S.g.j \leq S.A.f.i + \lambda.A.f.i \tag{4.5}$$

where $\lambda.A.f.i$ is a constant and $p.g.j = p.f.i$.

According to [18], if the network is composed of fair aggregators and start time schedulers, the end to end delay bound is preserved. In [18], two aggregator methods are introduced which are basic fair aggregator and greedy fair aggregator. While the basic fair aggregator limits the output rate of aggregator to the sum of constituent flows ($R.g = R.f + R.h$), the greedy fair aggregator relaxes this limit as long as the arrival rate of constituent flows are greater than their reserved rates. Therefore, from Equation 4.3, there is a problem about link utilization in two aggregator method. However, in WBFA, the constituent flows use full capacity of the output link as long as the difference between received services does not exceed a limit.

4.1.1.2 Operations

Window Based Fair Aggregator (WBFA), [60], aggregates two inputs and produces one output which is composed of two input flows. Although WBFA is designed to aggregate two input flows, aggregation of many number of flows can be succeeded by cascading WBFAs to each other.

WBFA uses a window based counting algorithm to provide fairness among the constituent flows. In this algorithm, a window function $w(t)$ is designed to keep track of received services for each constituent flows. If $w(t)$ is within the predefined range, the input flows of aggregator use full capacity of the output link. Therefore, WBFA provides lower end to end delay than aggregators proposed in [18]. Furthermore, defining a range for $w(t)$ helps to preserve fairness.

Calculation of $w(t)$: Initially $w(t)$ is taken as zero. Let the two inputs of WBFA be flow f and flow h. When flow f and h receive services, the $w(t)$ is increased and decreased respectively. The $w(t)$ is limited between $-wmax$ and $wmax$ to preserve fairness.

If flow f receives service, $w(t)$ is updated as :

$$w(t) = w(t^{-}) + \frac{S_f(t, t^{-})}{R.f} \tag{4.6}$$

where $S_f(t, t^-)$ gives the amount of service received by f between t^- and t.

If flow h receives service, $w(t)$ is updated as :

$$w(t) = w(t^-) - \frac{S_h(t, t^-)}{R.h}, \tag{4.7}$$

where $S_h(t, t^-)$ gives the amount of service received by h between t^- and t.

As long as $w(t)$ is different than 0, $w(t)$ approaches 0 at a rate of $R.f + R.h$.

$$w(t2) = \begin{cases} w(t1) - (t2 - t1)(R.f + R.h), & when \;\; w(t) > 0 \\ w(t1) + (t2 - t1)(R.f + R.h), & when \;\; w(t) < 0 \end{cases} \tag{4.8}$$

where ∀ t, t2>t>t1.

Packet Transmission: The window function $w(t)$ keeps track of difference between received services for each constituent flows. $w(t)$ is limited between $wmax$ and $-wmax$. The packets received from input flows are stored in separate flow queues. If flow f has a HOL packet, the amount of service that is required to send HOL packet of f is compared with $w(t)$. If;

$$w(t) + \frac{P_f^{HOL}(t)}{R.f} \leq wmax, \tag{4.9}$$

where P_f^{HOL} is the size of HOL packet of flow f, then f is eligible to send packet. If flow h has a HOL packet, the amount of service that is required to send HOL packet of h is compared with $w(t)$. If;

$$w(t) - \frac{P_h^{HOL}(t)}{R.h} \geq -wmax, \tag{4.10}$$

where P_h^{HOL} is the size of HOL packet of flow h, then h is eligible to send packet. When the both HOL packets of input flows f and h are eligible to transmitted, one of them is arbitrarily selected. It must be noted that, when a packet becomes HOL packet for each flow, it is immediately transmitted as long as the $w(t)$ is within defined range.

4.1.2 WBFA Performance

Figure 4.2 shows the aggregators m and n which are cascaded with scheduler s. Here n is a WBFA aggregator and m is GPS based aggregator. Output capacity of m,$C.m$, is $R.f + R.h$ and output capacity of n is, C_n, greater than $R.f + R.h$. [60] calculates and presents following expression:

$$S.s.g.k \leq S.n.f.i + \beta.m.f.i + \lambda.m.f.i, \tag{4.11}$$

where $p.f.i = p.g.k$ and, $\beta.m.f.i$ and $\lambda.m.f.i$ are constants.

Equation 4.11 shows that, n is a fair aggregator and preserves delay bound for each constituent flows with respect to Equation 4.5.

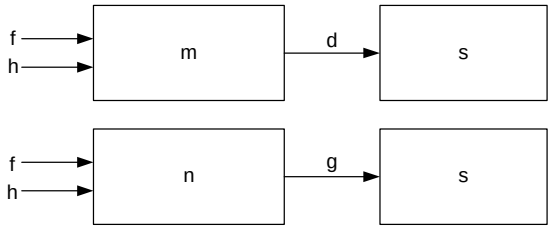

Figure 4.2: The aggregators m,n and a scheduler s

In order calculate end to end delay bound for WBFA, $\lambda.A.f.i$ from Equation 4.5 has to be calculated. [60] defines $\lambda.A.f.i$ which is defined as delay bound as follows:

$$Delay\ \ Bound = \lambda.A.f.i = \frac{L_{max}.f}{R.f} + \frac{(wmax_n)(R.f) + L_{max}.f}{R.f + R.h}. \tag{4.12}$$

4.2 Integration of WBFA to QueST

The WBFA can be defined as a scheduler which receives two input traffic flows and serves these input flows in a single output flow by aggregating packets of each flow. The output capacity of WBFA is modeled to be infinite. Therefore, packet transmission from WBFA does not spend any time. Figure 4.3 shows class diagram of WBFA

and the Packet Scheduler Module. As seen in Figure 4.3, since WBFA behaves like a scheduler, `WBFA` class is inherited from the `Scheduler` class. `WBFA` class can be cascaded with each other and `WF2Q` class. Therefore, many number of flows can be aggregated by cascading `WBFA` classes to each other.

As seen in Figure 4.3, the `WBFA` class has three operations, three attributes and one constructor. In the constructor, the attributes of the class are initialized. The first attribute of `WBFA` class is `window` which represents window function $w(t)$ of WBFA. The second attribute of `WBFA` class is `Wmax` which represents limit value *wmax* for window function. The last attribute of `WBFA` class is `tieFlowNum` which gives the id of flow in a line.

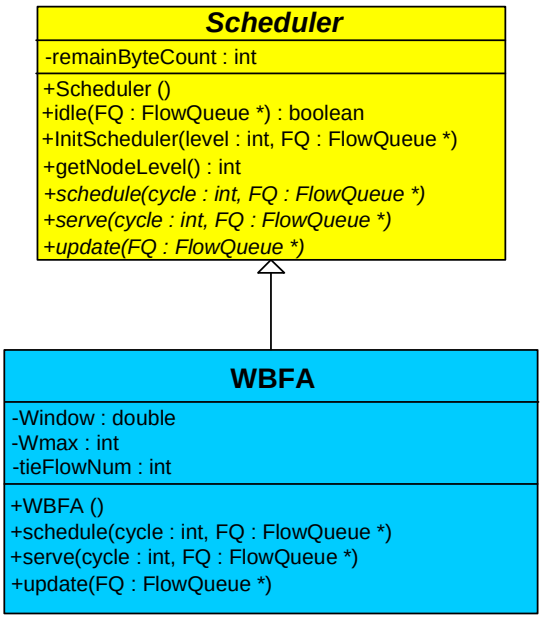

Figure 4.3: The Class Diagram of WBFA and Scheduler Module

The `schedule(cycle : int, FQ : FlowQueue*)` operation selects eligible packet among flow queues and sends selected packet to the output channel of WBFA. Since this function is an abstract function, the `WBFA` class implements the function with respect to *WBFA* algorithm. The operation takes two inputs which are `cycle` and `FQ`. The first input is the cycle number of scheduler and the second input is the per flow queues of scheduler module. Figure 4.4 shows flow chart of this operation. Flow diagram of the operation in `WBFA` is as follows:

1. If there is a packet in the line, window function is updated and the packet is sent to output channel of WBFA and the operation ends. Otherwise, function goes on with following processes.

2. HOL packets are searched in all flow queues. If there is not any HOL packet, the function ends. Otherwise, function goes on with following processes.

3. If HOL packet belongs to first flow, then;

   (a) Window value is computed by using Equation 4.6.

   (b) If window value is lower than `Wmax`, HOL packet is marked as eligible.

   (c) If window value is greater than `Wmax`, HOL packet is marked as not eligible.

4. If HOL packet belongs to second flow, then;

   (a) Window value is computed by using Equation 4.7.

   (b) If window value is greater than `-Wmax`, HOL packet is marked as eligible.

   (c) If window value is lower than `-Wmax`, HOL packet is marked as not eligible.

5. If both flows are eligible to send HOL packet, one of them is arbitrarily selected and the other is sent to line.

6. Window function is updated with selected flow window value.

7. Selected HOL packet is sent to output channel of WBFA and the operation ends.

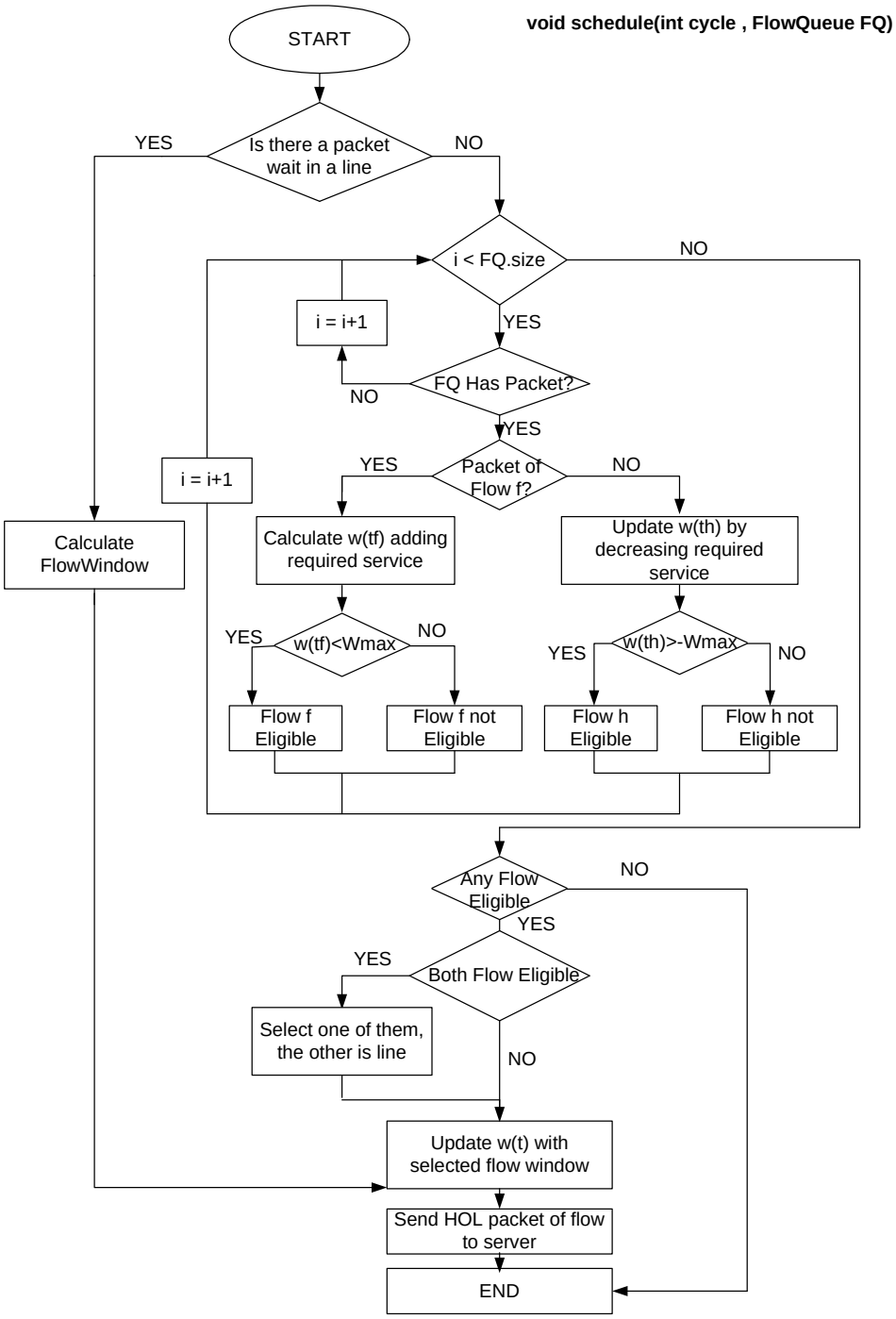

Figure 4.4: Flow Chart of schedule Function in WBFA Scheduler Module

The `serve(cycle : int, FQ : FlowQueue*)` operation transmits all packets in one cycle. The operation sends transmitted packets to the destination address. If the transmitted packet does not have any destination address, it is stored in a file to

analyze. The operation takes two inputs which are `cycle` and `FQ`. The first input is the cycle number of scheduler and the second input is the per flow queues of WBFA modules. As seen in Figure 4.5, flow diagram of the operation in `WBFA` is as follows:

1. If there is not any packet in the server, the function ends. Otherwise, function goes on with following processes.
2. If there is packet in the server, packet is totally served.
3. Serving time of the packet is set to `cycle`.
4. If served packet has a destination address, then the packet is sent to flow queues of destination module. Otherwise, the packet is stored in a file to analyze.

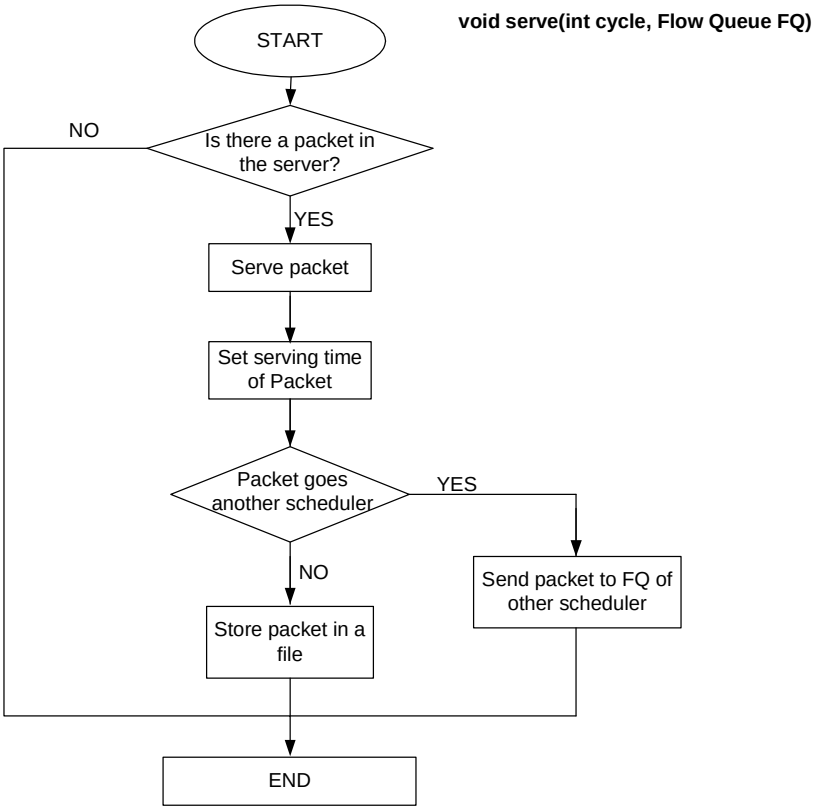

Figure 4.5: Flow Chart of serve Function in WBFA Scheduler Module

The `update(FQ : FlowQueue*)` operation approaches window function $w(t)$ to zero at each cycle. The function takes one input ,`FQ`, which is the per flow queues

of each flow. Figure 4.6 shows flow chart of the function. As seen in Figure 4.6, the window function $w(t)$ is updated by using Equation 4.8 at each cycle.

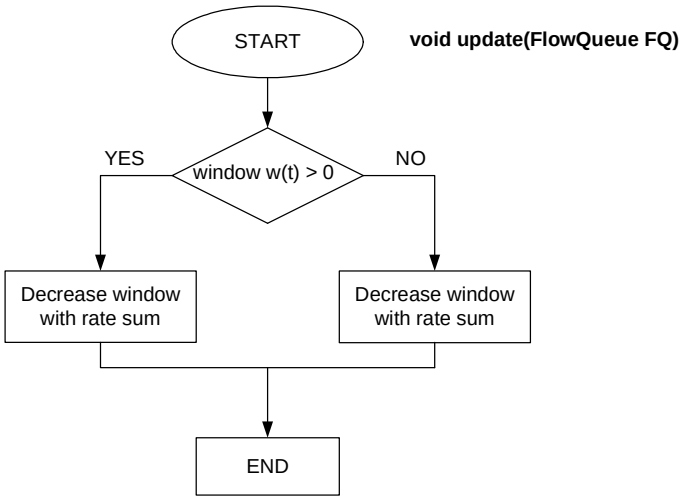

Figure 4.6: Flow Chart of update Function in WBFA Scheduler Module

4.3 Experiments and Results

In this section, the effect of WBFA on packet scheduler module and delay bound is analyzed by performing different experiments. In our experiments, Poisson traffic generator module and Bursty traffic generator module are used to generate traffic flows at adjustable loads.

Firstly, we perform verification test for WBFA by comparing our experiment results in [60]. Secondly, the run time analysis for WBFA module is presented by performing experiments with cascaded WBFA modules. Thirdly, the optimal maximum window size value,*wmax*, is calculated. Finally, performance and delay bound of packet scheduler module with WBFA is analyzed by performing different experiments.

4.3.1 Experiment 1: Verification of WBFA under Poisson Traffic Generator

In this experiment, the WBFA module in QueST is verified with the WBFA in [60] by comparing experiment results under Poisson traffic generator. Total traffic load is %75 of output scheduler rate. The size of each generated packet is exponentially

distributed between 0 and 100 bytes and the average packet size is 10 bytes. As seen in Figure 4.7, the experiment runs with 4 traffic flows which are F_1, F_2, F_3 and F_4 in two cases. In the reference case, WBFA module is not used and traffic flows enter directly to WF^2Q+ scheduler module without aggregation. Output rate of WF^2Q+ scheduler is $C.s$ which is equal to 1 byte/cycle. The reserved rates for all traffic flows are the same and equal to %25 of $C.s$. In the second case, F_1 and F_2 are aggregated by WBFA. The output of WBFA, F_{12}, is scheduled together with F_3 and F_4. While %50 of WF^2Q+ scheduler output rate is reserved for F_{12}, %25 of WF^2Q+ scheduler output rate is reserved for F_3 and F_4. *wmax* is set to 4000 which enables the flow with smallest reserved rate transmit 10 maximum sized packets.

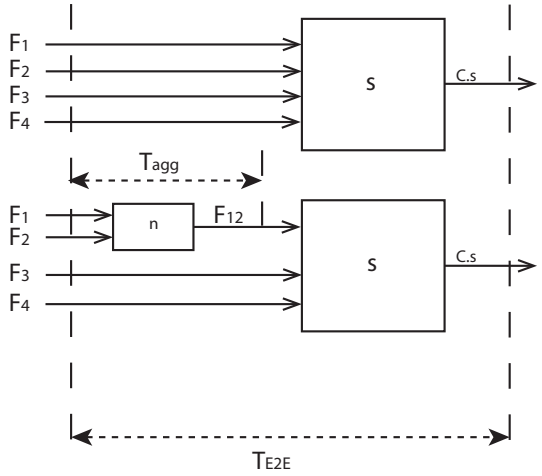

Figure 4.7: Experiment setup for verification of WBFA (a) Reference case: 4 flows scheduled with a WF2Q+ scheduler s without aggregation. (b) WF2Q+ scheduler s follows aggregator n.

The average end to end delay, T_{E2E}, and the average aggregation delay, T_{agg} is measured as indicated in Figure 4.7. The end to end delay for a packet is calculated by subtracting packet generation time from packet serving time. The aggregation delay for a packet is calculated by subtracting packet generation time from the time when packet leaves aggregator. If many number of WBFA modules are cascaded to each other, then the aggregation delay for a packet is calculated by subtracting packet generation time from the time when packet leaves the last level aggregator. In other words, the total aggregation delay for a packet in a system with cascaded WBFA mod-

ules is the sum of all aggregation delays of WBFA modules that the packet passes through. Since the WBFA has infinite output capacity, the packet transmission between WBFA and WF^2Q+ takes 1 clock cycle for any sized packet. We collect 25K packets for two cases. The unit of measurement is $P/C.s$ where P is the average packet size which is equal to 10 bytes. The end to end delay for the reference case without aggregation is denoted as T_{Ref}.

Table 4.1 gives the experiment results conducted in [60] and QueST. When we compare experiment results in QueST with the results in [60], we can see that the result values are very close to each other for both experiments. Trend of each experiment is similar after aggregation is applied. In both experiments, after the aggregation while F_1 and F_2 have lower delay than reference case, F_3 and F_4 have higher delay than reference case.

Table4.1: (a) Experimental Results Under Poisson Traffic Generator, %75 Load in [60] (b) Experimental Results Under Poisson Traffic Generator, %75 Load in QueST

(a)

	WBFA			
	F_1	F_2	F_3	F_4
T_{E2E}	3.20	3.22	3.64	3.62
T_{agg}	0.04	0.04	—	—
T_{Ref}	3.45	3.30	3.35	3.45

(b)

	WBFA			
	F_1	F_2	F_3	F_4
T_{E2E}	3.65	3.66	4.00	3.85
T_{agg}	0.001	0.001	—	—
T_{Ref}	3.73	3.76	3.78	3.64

4.3.2 Experiment 2: Verification of WBFA under Bursty Traffic Generator

Our second experiment aims to verify WBFA module in QueST with the WBFA in [60] by comparing experiment results under Bursty traffic generator. In this experiment, total traffic load is %85 of the output scheduler rate and size of each generated packet is set to 10 bytes. As seen in Figure 4.7, the experiment runs with 4 traffic flows which are F_1, F_2, F_3 and F_4 in two cases. In the reference case, WBFA module is not used and the traffic flows directly enter to WF^2Q+ scheduler module. In the second case, F_1 and F_2 are aggregated by WBFA and the output of aggregation is scheduled together with F_3 and F_4 by WF^2Q+ scheduler module. While burst size of F_1 is set to 200 bytes, burst sizes of the others are set to 20 bytes.

T_{E2E}, T_{agg} and T_{Ref} denote mean end to end delay with WBFA, aggregation delay and mean end to end delay without WBFA respectively as indicated in Figure 4.7. The unit of measurement is $P/C.s$ where P is the average packet size which is equal to 10 bytes, $C.s$ is the output rate of scheduler which is equal to 1 byte/cycle. Table 4.2 shows the experiment results.

Table4.2: (a) Experiment Result Under Bursty Traffic Generator, %85 Load in [60] (b) Experiment Result Under Bursty Traffic Generator, %85 Load in QueST

(a)

	WBFA			
	F_1	F_2	F_3	F_4
T_{E2E}	33.7	18.5	7.4	8.2
T_{agg}	3.1	0	—	—
T_{Ref}	47.1	4.8	5.0	5.6

(b)

	WBFA			
	F_1	F_2	F_3	F_4
T_{E2E}	38.4	23.8	8.03	8.09
T_{agg}	1.1	0.001	—	—
T_{Ref}	61.4	6.27	6.36	6.46

As seen in Table 4.2, the result values are similar to each other. In both experiments, while F_1 encounters lower delay with aggregation than reference case, the others encounter higher delay with aggregation than reference case. This is because F_1 uses full capacity of WBFA as long as $w(t)$ permits. Therefore, F_1 gains service advantages of WBFA and delay of the other flows increase.

4.3.3 Experiment 3: Run time analysis for WBFA

In this section, we analyze how run time of QueST changes with the number of cascaded WBFA modules to aggregate more flows. In this experiment, Poisson traffic generator module is used to generate 16 traffic flows which are equally allocated. This experiment is composed of four cases. At each case, we collect 25K packets under different traffic loads and increase the number of WBFA modules. Figure 4.8 shows the experiment setup.

One level WBFA which reduces the number of flows to half is applied to traffic flows in the first case. In the other cases, level of WBFA is increased one by one up to four. Therefore, at last case, we obtain one traffic flow as a result of aggregation.

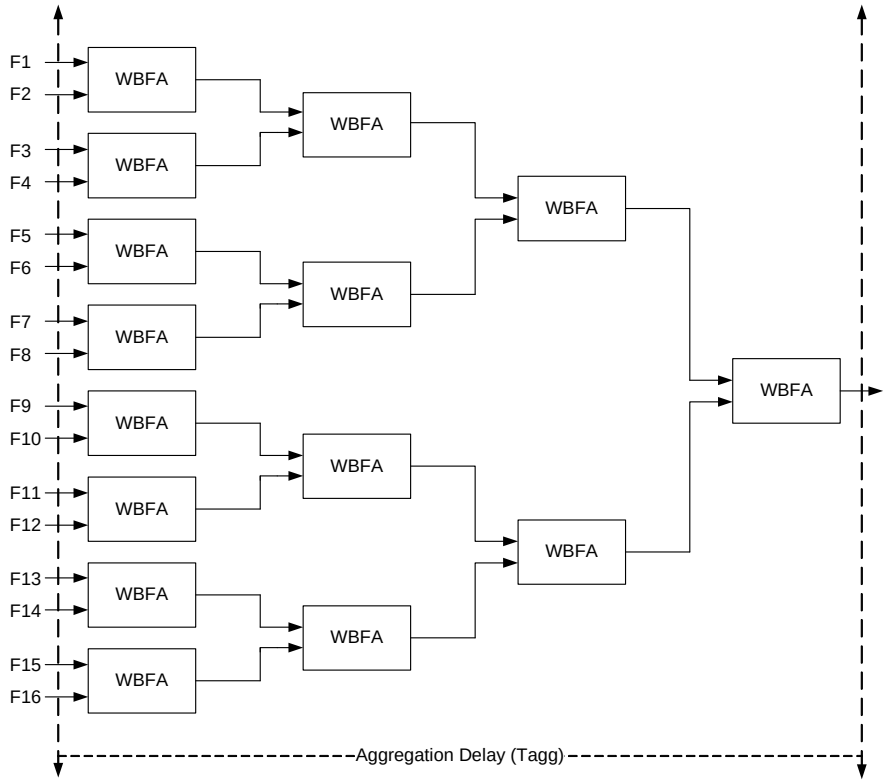

Figure 4.8: Experiment setup for complexity analysis of WBFA

According to Figure 4.9, run times of the experiments increase with the number of cascaded WBFA modules under same load. This increase comes from searching flow queues to find eligible flow.

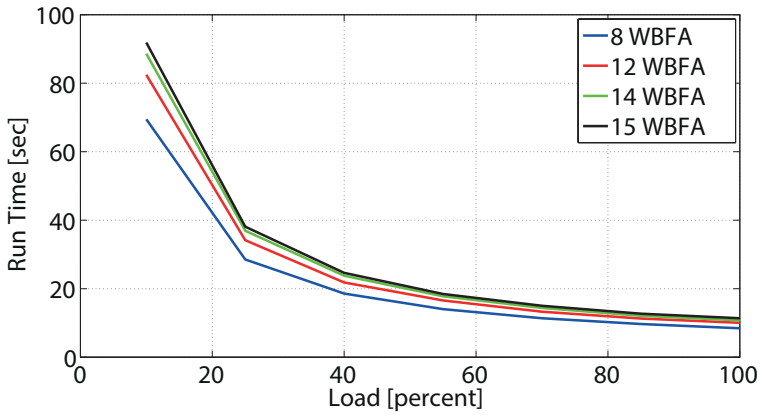

Figure 4.9: Run times of QueST under different number of WBFA modules

4.3.4 Experiment 4: Optimal Maximum Window Size

Although WBFA has infinite output capacity and the packets are transmitted without waiting by WBFA, packets may be delayed at some cases. For example, if one of the two constituent flows is greedy flow, at some point service received by greedy flow will reach maximum window value. Therefore, the greedy flow will be buffered and encounter aggregation delay until the window function stays in limit.

In this section, we investigate optimal maximum window size value for WBFA where the aggregation delay is minimum. In order to find optimal maximum window value, we perform experiment with two cases. In first case, while allocated rates of input flows are kept constant, load of input flows are changed. In second case, while load of input flows are kept constant, allocated rates of input flows are changed. In this experiment, Poisson traffic generator module is used to generate 2 traffic flows which directly enter WBFA module.

Table4.3: (a) Experiment Values for Case-1 (b) Experiment Values for Case-2

(a)

Exp i	ρ_f	ρ_h	R_f	R_h
1	0.45	0.45	0.5	0.5
2	0.3	0.3	0.5	0.5
3	0.2	0.2	0.5	0.5

(b)

Exp i	ρ_f	ρ_h	R_f	R_h
1	0.45	0.45	0.5	0.5
2	0.45	0.45	0.4	0.4
3	0.45	0.45	0.3	0.3

Table 4.3 gives the experiment values for each case where ρ_f and ρ_h show loads of input flows and R_f and R_h show allocated rates of input flows. The minimum aggregation delay indicates optimal maximum window size.

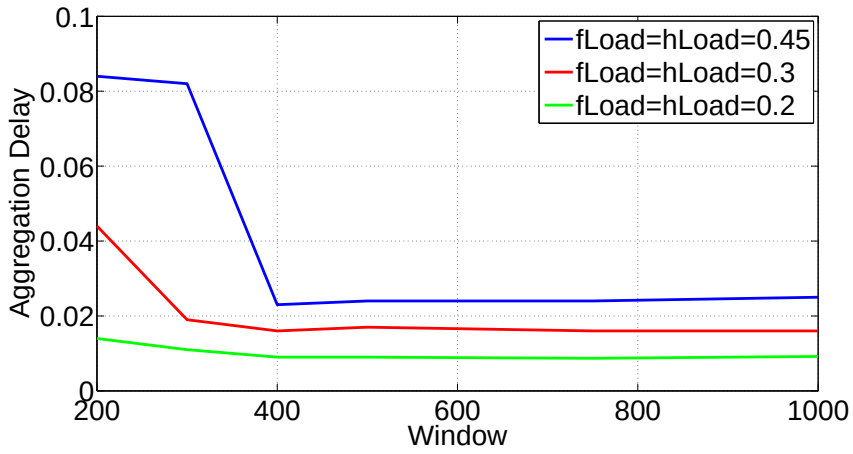

Figure 4.10: Experiment Results under same reserved rates and different traffic loads

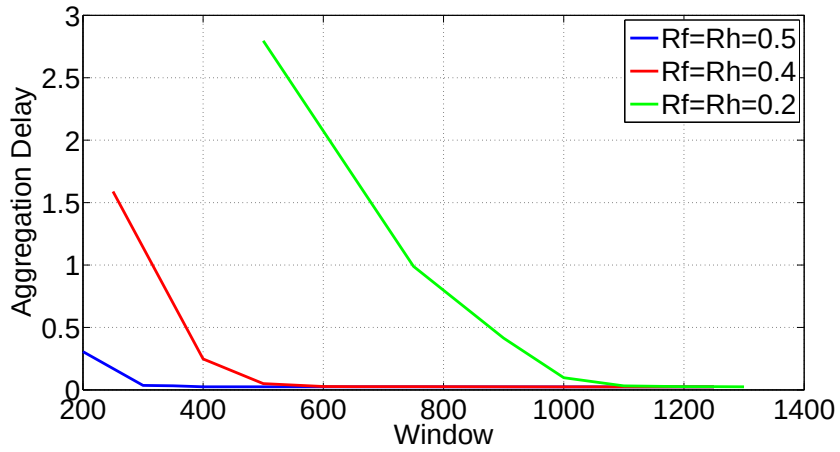

Figure 4.11: Experiment Results under same traffic loads and different reserved rates

As can be seen in Figure 4.10 and Figure 4.11, the optimal maximum window value does not change with respect to traffic loads. However, when the allocated rates of input flows is changed, the optimal maximum window value changes. For each maximum window value, we collect 10 samples and take average of aggregation delays for each sample. The reported averages are within the %0 and %1 confidence interval of the true mean with %99 probability. When we analyze the relationship between optimal maximum window value and allocated rates of input flows, the following expression for optimal maximum window value is found:

$$Optimal\ \ maximum\ \ Window\ \ Value = \frac{L_{max}}{R_f * R_h}, \tag{4.13}$$

where L_{max} is the maximum packet size which is equal to 100 byte for this experiment. If the maximum window value is selected larger than optimal maximum window value, the aggregation delay does not change significantly and stays constant. However, selecting very large maximum window value affects fairness property of WBFA among constituent flows negatively. If the maximum window value is selected lower than optimal maximum window value, the input flows of WBFA encounter significant delay. Furthermore, if the selected maximum window value is too small, then any packet from input flows can not be served by WBFA. Therefore, theoretical maximum window value, *wmax*, must be larger than following equation which is calculated as by using Equation 4.6, Equation 4.7, Equation 4.9 and Equation 4.10:

$$2 * wmax \geq \frac{S_h(t, t^-) * R.f + P_h^{HOL}(t) * R.f + S_f(t, t^-) * R.h + P_f^{HOL}(t) * R.h}{R.f * R.h}, \tag{4.14}$$

where t^- is the transmission time of last packet. Since Poisson traffic generator module in QueST generates network packets which have packet sizes between 0 and L_{max}, $S_h(t, t^-)$, $S_f(t, t^-)$, $P_f^{HOL}(t)$ and $P_h^{HOL}(t)$ can not be larger than L_{max}. Therefore, from Equation 4.14:

$$wmax \geq \frac{L_{max}(R.f + R.h)}{R.f * R.h}, \tag{4.15}$$

where $R.f$ and $R.h$ are normalized reserved rates between 0 and 1. In QueST, sum of normalized reserved rates of all input flows can not be larger than 1. Therefore, in Equation 4.15, $R.f + R.h$ can be maximum 1 and if we replace $R.f + R.h$ with 1, we get following expression:

$$wmax \geq \frac{L_{max}}{R.f * R.h}. \tag{4.16}$$

We can see that the experimental result for the optimal maximum window value has the same expression with the Equation 4.16. Therefore, the optimal maximum window value found in this experiment is verified with theoretical expression in Equation 4.16.

4.3.5 Experiment 5: Fairness Analysis for WBFA

In this section, we consider fairness of WBFA module in QueST and analyze how number of served packets for each flow changes with the allocated rates. In this experiment, Poisson traffic generator module is used to generate 3 traffic flows whose packets have different lengths. Each flow has the same load. This experiment is composed of two cases. In first case, WBFA module is not used and traffic flows directly enter WF^2Q+ scheduler module. In second case, while F_1 and F_2 enter WBFA module, F_3 directly enters WF^2Q+ scheduler module. Output of WBFA module, F_{12}, is connected to WF^2Q+ scheduler module. The maximum window size value for WBFA is selected as optimal maximum window size value which is calculated from Equation 4.13. 25K packets are collected for each case.

Table4.4: Experiment Values for both Case-1 and Case-2

Exp i	Each Flow Load	R_{f1}	R_{f2}	R_{f3}
1	1.0	0.4	0.3	0.3
2	1.0	0.6	0.2	0.2
3	1.0	0.8	0.1	0.1

As can be seen in Table 4.4, mean loads of generated flows are the same for two cases. The allocated rates for input flows are changed for each experiment. After 25K packets are collected at the output of WF^2Q+ scheduler module, the number of served packets for each flow is obtained. Table 4.5 shows the experiment results for each case.

As can be seen in Table 4.5, the served packet number for each flow is proportional with the allocated rate of that flow in both cases. Therefore, both WBFA and WF^2Q+ modules in QueST are fair schedulers.

Table4.5: (a) Number of served packets for Case-1 (b) Number of served packets for Case-2

(a)

Exp i	Served F_1	Served F_2	Served F_3
1	10101	7467	7432
2	14965	4957	5078
3	20016	2562	2422

(b)

Exp i	Served F_1	Served F_2	Served F_3
1	9709	7794	7497
2	14601	5461	4938
3	19624	2893	2483

4.3.6 Experiment 6: Effect of WBFA on WF2Q+ Scheduler

In this section, we consider the effect of WBFA on WF^2Q+ scheduler. According to [60], WBFA is fair aggregator and it preserves end to end delay bounds. [60] also states that WBFA decreases complexity of the scheduler. To this end, we perform experiments to analyze effect of WBFA on WF^2Q+ scheduler with respect to end to end delay bound and complexity under different traffic scenarios. Figure 4.12 shows experiments setup. As can be seen in Figure 4.12, each experiment is composed of two cases. In the reference case, WBFA is not used and three level WF^2Q+ schedulers are used. In the other case, different level WBFA modules are used before flows enter WF^2Q+ schedulers.

We perform two experiments to analyze how effect of WBFA on WF^2Q+ scheduler changes with respect to generated packet size. Therefore, in first experiment, network packets are generated with fixed sizes by Poisson and Bursty flows. In second experiment, network packets are generated with exponentially distributed sizes by Poisson flows. 320 traffic flows are generated and 25K packets are collected under different traffic loads at the output of last WF^2Q+ scheduler. Analytical delay bounds for all experiment, DB, are calculated by using Equation 3.27. The unit of measurements for all experiments is the required time to serve a packet with size P on the WF^2Q+ scheduler which is $P/C.s$. The maximum window size value for WBFA is selected as

optimal maximum window size value which is calculated from Equation 4.13.

In each experiment, $T_{E2E}Mean$, $T_{E2E}Max$ and T_{Run} for both cases are measured where,

- $T_{E2E}Mean$ is the average end to end delay which is obtained by subtracting flow generation time from flow serving time.
- $T_{E2E}Max$ is the maximum end to end delay which is obtained by subtracting flow generation time from flow serving time.
- T_{Run} is the run time of scheduler and unit of run time is second,sec.

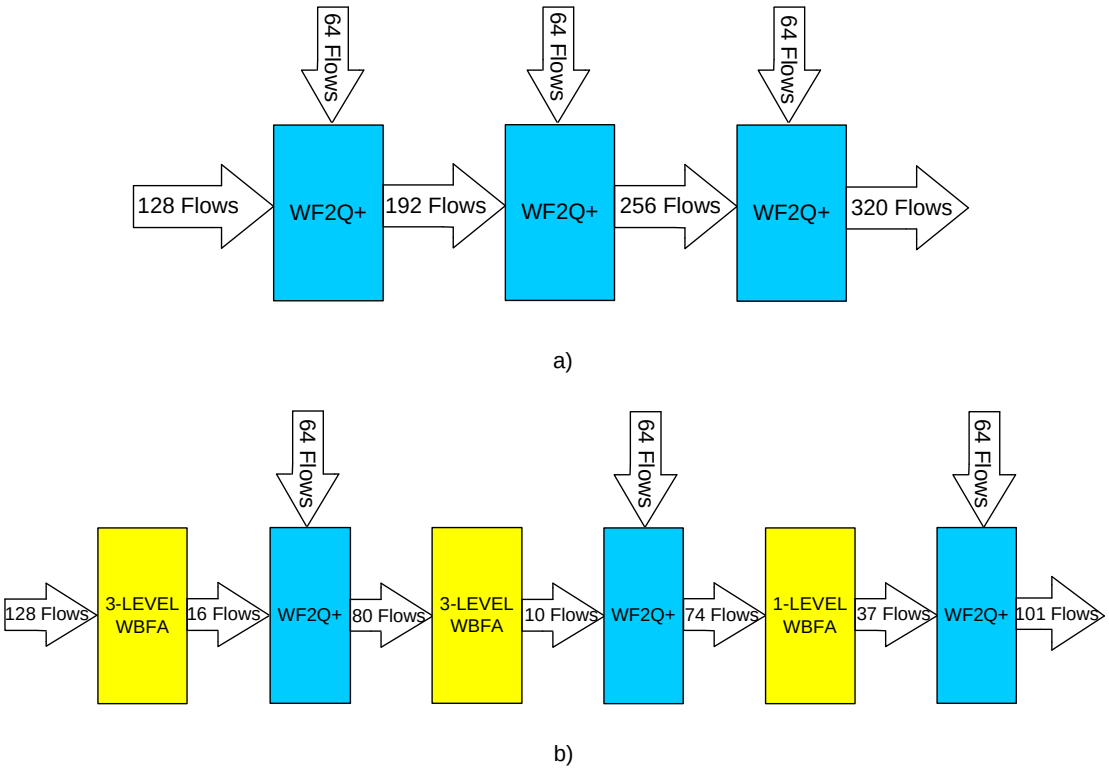

Figure 4.12: Experiment setup for the effect of WBFA on WF2Q+ a) shows reference case without WBFA b)shows WBFA case

4.3.6.1 Experiment 6-1: Effect of WBFA on WF2Q+ Scheduler under Fixed Size Packets

This experiment is composed of two traffic scenarios. In the first scenario Poisson traffic generator module is used to generate fixed size packets. The packet sizes are

set to P. The traffic flows are generated under three different traffic loads which are %30, %50 and %80. Each flow has same load. Table 4.6 shows the experiment result.

Table4.6: Experiment Results for Poisson Traffic Generator with Fixed Size Packets

Load	T_{Run}		$T_{E2E}Mean$		$T_{E2E}Max$	
	Ref	WBFA	Ref	WBFA	Ref	WBFA
%30	651	614	2.52	2.2	8	11
%50	399	370	2.81	2.5	25	18
%80	264	235	4.39	4.01	82	120

In the reference case, we performed the experiment by using three WF^2Q+ scheduler modules. We assume that the maximum burst size is equal to 0 byte. Maximum packet size is equal to average packet size which is equal to P. From Equation 3.27, the total analytical delay bound for reference case is 963 $P/C.s$. The maximum packet transmission delay is 1 $P/C.s$. As can be seen in Table 4.6, the maximum end to end delay is 120 $P/C.s$. If we add this maximum end to end delay to maximum transmission delay, we get a total maximum delay of 121 $P/C.s$. As a result, the delay bound is much higher than total maximum delay. In other words, WBFA preserves delay bound.

When we look at the run times for both cases, we see that run time of scheduler with WBFA is lower than run time of scheduler without WBFA. This is because that while WBFA preserves delay bounds, it also decreases complexity of the scheduler.

When we compare the mean end to end delays in both cases, we see that the WF^2Q+ scheduler with WBFA encounters lower delay than the case where WBFA is not added to WF^2Q+ scheduler.

In the second scenario Bursty traffic generator module is used to generate fixed size packets. The packet sizes are set to P. The burst size for Bursty traffic is set to P. The traffic flows are generated under three different traffic loads which are %30, %50 and %80. Each flow has same load. Table 4.7 shows the experiment result.

Table4.7: Experiment Results for Bursty Traffic Generator with Fixed Size Packets

Load	T_{Run}		$T_{E2E}Mean$		$T_{E2E}Max$	
	Ref	WBFA	Ref	WBFA	Ref	WBFA
%30	659	625	5.43	5.04	401	400
%50	399	376	6.84	6.6	511	520
%80	257	235	12.94	12.91	777	871

In Bursty traffic, the maximum burst size is set to P. Maximum packet size is equal to average packet size which is equal to P. From Equation 3.27, the total analytical delay bound for reference case is 1283 $P/C.s$. The maximum packet transmission delay is 1 $P/C.s$. As can be seen in Table 4.7, the maximum end to end delay is 871 $P/C.s$. If we add this maximum end to end delay to maximum transmission delay, we get a total maximum delay of 872 $P/C.s$. As can be seen, the delay bound is much higher than total maximum delay and so WBFA preserves delay bound. When we look at the run times for both cases, we see that run time of scheduler with WBFA is lower than run time of scheduler without WBFA due to fact that WBFA decreases complexity. When we compare the mean end to end delays in both cases, we see that the WF^2Q+ scheduler with WBFA encounters lower delay than the case where WBFA is not added to WF^2Q+ scheduler.

4.3.6.2 Experiment 6-2: Effect of WBFA on WF2Q+ Scheduler under Different Sized Packets

In this experiment Poisson traffic generator module is used to generate packets. The generated packet sizes are exponentially distributed between 0 and $10P$. The average packet size is P. The traffic flows are generated under three different traffic loads which are %30, %50 and %80. Each flow has same load. Table 4.8 shows the experiment result.

In this experiment, we assume that the maximum burst size is 0. Maximum packet size is equal to $10P$. From Equation 3.27, the total analytical delay bound for reference case is 9630 $P/C.s$. When we add the maximum packet transmission delay which is equal to $10P/C.s$ to maximum queueing delay which is obtained in Table

4.8, we get a total maximum delay of 172 $P/C.s$. As can be seen, WBFA preserves delay bound. If we compare run times of two cases, we see that the run time of scheduler with WBFA is lower than the case without WBFA due to decrease in complexity.

Table4.8: Experiment Results for Poisson Traffic Generator with Exponentially Distributed Sized Packets

Load	T_{Run}		$T_{E2E}Mean$		$T_{E2E}Max$	
	Ref	WBFA	Ref	WBFA	Ref	WBFA
%30	642	613	3.21	2.99	34	33
%50	396	373	3.91	3.90	50	37
%80	253	234	5.72	7.23	160	162

When we compare the mean end to end delays in both cases, we see that in low loads the WF^2Q+ scheduler with WBFA encounters lower delay than the case where WBFA is not added to WF^2Q+ scheduler. However, in high loads the WF^2Q+ scheduler with WBFA encounters higher delay. This can be because of the fact that the inter-arrival times between two consecutive packets is very low in high loads due to aggregation of traffic flows. However, in low loads inter-arrival times are bigger than the high load case. Therefore, a packet in the scheduler flow queues has to wait transmission of many predecessor packets in high load case. Furthermore, Poisson traffic is a very smooth traffic pattern and flows generated by Poisson traffic generator encounter lower delay in packet scheduler. However, if we aggregate two Poisson flows by using WBFA, the output flow is not a Poisson flow anymore. Therefore, obtained flow from aggregation may encounter higher delay than Poisson flow.

4.3.7 Experiment 7: Analyzing effects of WBFA on WF2Q+ scheduler under the conditions with more than 1000 Traffic Flows

In this section, we consider the effect of WBFA on WF^2Q+ scheduler by generating more than 1000 traffic flows. To this end, we perform one experiment by using Poisson traffic generator module to generate 1120 traffic flows. Figure 4.13 shows experiments setup. As can be seen in Figure 4.13, each experiment is composed of two

cases. In the reference case, WBFA is not used and three level WF^2Q+ schedulers are used. In the other case, different level WBFA modules are used before flows enter WF^2Q+ schedulers.

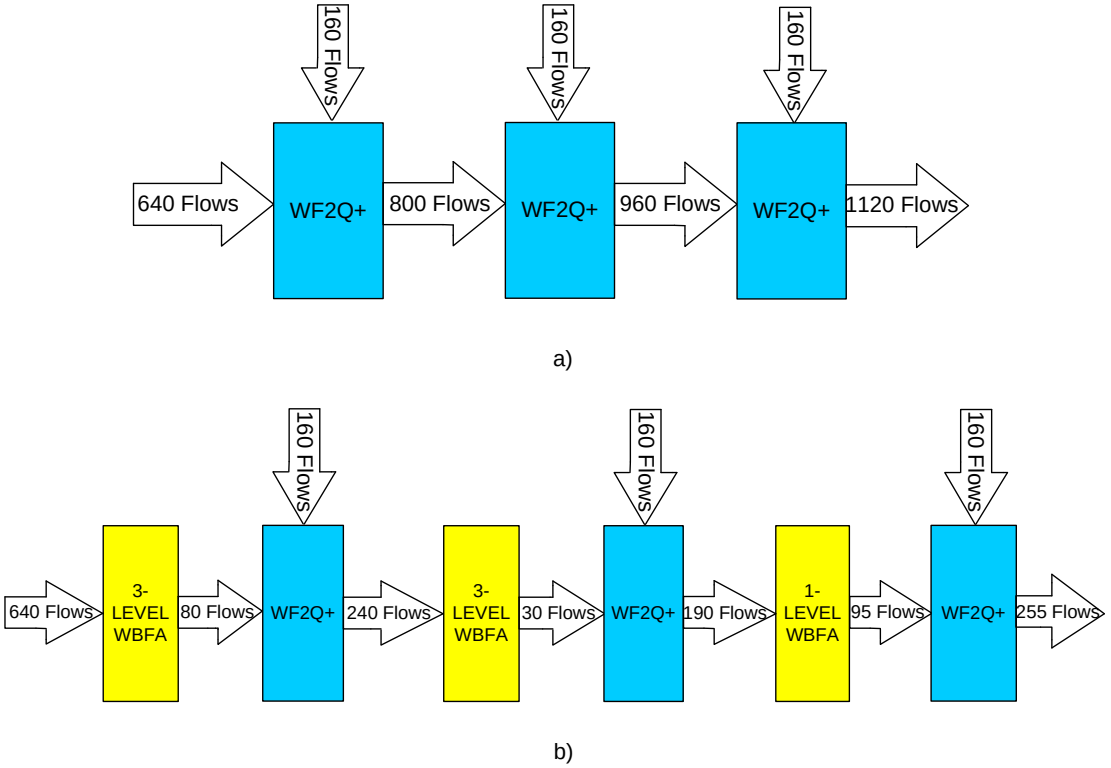

Figure 4.13: Experiment setup for the effect of WBFA on WF2Q+ under 1120 Traffic flows a) shows reference case without WBFA b)shows WBFA case

In the experiment, network packets have exponentially distributed packet sizes between 0 and $10P$, and average packet size is P. The traffic flows are generated under three different traffic loads which are %30, %50 and %80. Each flow has same load. 25K packets are collected at the output of last WF^2Q+ scheduler. The maximum window size value for WBFA is selected as optimal maximum window size value which is calculated from Equation 4.13. In the experiment, average end to end delay, $T_{E2E}Mean$, maximum end to end delay, $T_{E2E}Max$, and run time of the scheduler, T_{Run}, are measured and compared. The unit of measurement for $T_{E2E}Mean$ and $T_{E2E}Max$ is the required time to serve a packet with size P on the WF^2Q+ scheduler which is $P/C.s$ and the unit of measurement for T_{Run} is second, sec. Table 4.9 shows the experiment results.

Table4.9: Experiment results under 1120 Poisson traffic flows with different sized packets

Load	T_{Run}		$T_{E2E}Mean$		$T_{E2E}Max$	
	Ref	WBFA	Ref	WBFA	Ref	WBFA
%30	2334	2075	3.64	3.84	43	44
%50	1430	1246	4.51	4.94	77	46
%80	920	779	6.94	10.31	193	360

In this experiment, we assume that the maximum burst size is 0 and maximum packet size is equal to $10P$. From Equation 3.27, the total analytical delay bound for reference case is 33630 $P/C.s$. When we add the maximum packet transmission delay which is equal to $10P/C.s$ to maximum queueing delay which is obtained in Table 4.9, we get a total maximum delay of 370 $P/C.s$. As can be seen, WBFA preserves delay bound. If we compare run times of two cases, we see that the run time of scheduler with WBFA is lower than the case without WBFA due to decrease in complexity. When we compare the mean end to end delays in both cases, we see that the WF^2Q+ scheduler with WBFA encounters higher delay. Therefore we can say that although WBFA preserves delay bound and decreases complexity, it increases end to end delay for the traffic flows.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The focus of this thesis is the implementation and evaluation of QoS modules at the network layer by extensive simulation experiments. To this end, a new cycle and packet accurate software network simulator QueST is developed and implemented in C++. QueST has a modular, extendable structure and currently implements Poisson, Markov modulated on/off Bursty and Pareto traffic generators, Token Bucket Shaper and WF^2Q+ scheduler.

The correctness of the modules is verified by comparison to their respective analytical models. The runtime complexity of QueST is investigated as a function of the number of flows and scheduler modules. It is found that runtime of QueST is proportional to the number of flows. The runtime of the WF^2Q+ scheduler module is found to increase proportional to the logarithm of the number of flows which confirms with the analytical results where N is the number of the WF^2Q+ scheduler modules.

Windows Based Fair Aggregator (WBFA) proposed by [60] is implemented and integrated to packet scheduler module of QueST to experimentally investigate the benefits of flow aggregation in detail. WBFA module in QueST is verified by comparing its results with the results in [60] under Poisson and Bursty traffic generators. WBFA provides fairness with the help of window function. In this work, optimal maximum window value is found for WBFA module and the fairness of WBFA module is demonstrated by applying different experiments.

We present a number of experiment results that are performed by QueST for the effect of WBFA on WF^2Q+ scheduler module with respect to delay and complexity.

In these experiments, we see that the complexity of the scheduler decreases and experiments spend less time when WBFA is used. In the experiments with WBFA, it is shown that the end to end delay bounds are preserved. The experiments where WBFA is used show that the mean end to end delay performance of the scheduler is improved when generated traffic packets have fixed size. However, the mean end to end delay performance of the scheduler becomes worse when generated traffic packets have different sizes.

As a future work; how delay performance of the scheduler changes with respect to packet size when WBFA is used can be analyzed. Implementing other well-known packet scheduling algorithms can increase contribution of the QueST. Furthermore, QueST can be improved by designing a graphical user interface (GUI) for experiment setups.

REFERENCES

[1] Internet observatory. *<http://www.internetobservatory.net/>*.

[2] Internetworking elements, available at:. http://www.highteck.net/EN/Basic/Internetworking.htm

[3] Minnesota internet traffic studies (mintss). *<http://www.dtc.umn.edu/mints/home.php/>*.

[4] A. Adas. Traffic models in broadband networks. *IEEE Communications Magazine*, 1997.

[5] S. Ali. Implementation of worst-case weighted fair queueing (wf2q+). http://www.cs.cmu.edu/ cheeko/wf2q+/.

[6] S. Avallone, D. Emma, A. Pescape, and G. Ventre. High performance internet traffic generators. *The Journal of Supercomputing*, 2006.

[7] J. Aweya. Ip router architectures: An overview. *Nortel Networks*.

[8] A. Azim and Z. I. Awan. Network emulation, pattern based traffic shaping and kaunet evaluation. *Masters of Science Thesis*, 2008.

[9] X. Bai and A. Shami. Modeling self-similar traffic for network simulation. 2005.

[10] J. C. R. Bennett and H. Zhang. Wf2q: Worst-case fair weighted fair queueing. *in Proc. IEEE INFOCOM*, 1996.

[11] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Trans. Networking*, 1997.

[12] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM*, 40(1), 2010.

[13] H. Bidgoli. The handbook of computer networks. *Wiley*, 3, 2007.

[14] M. Bornhager. Router and routing basics. *Networking Academy*, 2011.

[15] P. J. Burke. The output of a queuing system. *Operations Research*, page 699–704, 1956.

[16] H. J. Chao and X. Guo. Quality of service control in high-speed networks. *Wiley*, 2002.

[17] H. J. Chao and B. Liu. High performance switch and routers.

[18] J. A. Cobb. Preserving quality of service guarantees in spite of flow aggregation. *IEEE/ACM Transactions on Networking*, 10(1), 2002.

[19] J. A. Cobb. Work conserving fair-aggregation with rate-independent delay. *in Proc. IEEE ICCCN*, 2008.

[20] J. A. Cobb. Rate-independent delay across state-reduced networks. *in Proc. Local Computer Networks*, 2009.

[21] J. A. Cobb and X. Zhe. Maintaining flow isolation in work-conserving flow aggregation. *in Proc. GLOBECOM*, 2005.

[22] J. A. Cobb and X. Zhe. Guaranteed throughput in work-conserving flow aggregation through deadline reuse. *in Proc. IEEE ICCCN*, 2006.

[23] R. L. Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Trans. Inform. Theory*, 3, 1991.

[24] D-ITG. Available online at:. http://www.grid.unina.it/software/ITG/.

[25] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *in Proc. ACM SIGCOMM*, 1989.

[26] A. Elwalid and D. Mitra. Traffic shaping at a network node: theory, optimum design, admission control. *In proc. of IEEE INFOCOM*, 1997.

[27] K. Fall and K. Varadhan. The ns manual. 2011.

[28] E.-B. Fgee, J. D. Kenney, W. J. Phillips, W. Robertson, and S. Sivakumar. Comparison of qos performance between ipv6 qos management model and intserv and diffserv qos models. *IEEE Magazine*, 2005.

[29] K. A. Froot, G. O'Connell, and G. Paul. On the pricing of intermediated risks: Theory and application to catastrophe reinsurance. *Journal of Banking*, 2008.

[30] T. Generator. Available online at:. http://www.postel.org/tg/tg.htm/.

[31] L. Georgiadis, R. Guerin, V. Peris, and R. Rajan. Efficient support of delay and rate guarantees in an internet. *in Proc. ACM SIGCOMM*, 1996.

[32] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. *in Proc. IEEE INFOCOM*, 1994.

[33] A. Gupta and V. Sharna. A unified approach for analyzing persistent, non-persistent and on-off tcp sessions in the internet. *Performance Evaluation*, 63(2), 2006.

[34] S. G. Gupta, M. M. Ghonge, P. D. Thakare, and D. P. M. Jawandhiya. Open-source network simulation tools: An overview. *International Journal of Advanced Research in Computer Engineering Technology (IJARCET)*, 2, 2013.

[35] A. Harrison, O. Florence, B. Benjamin, O. Cletus, and O. Folasade. The desirability of pareto distribution for modeling modern internet traffic characteristics. *IJNREAS*, 1(1), 2014.

[36] M. Hosamo. A study of the source traffic generator using poisson distribution for abr service. 2012.

[37] T. Issariyakul. Including your modules into ns 2. 2010.

[38] J. Joung, J. Song, and S. L. Soon. Flow-based qos management architectures for the next generation network. *ETRI Journal*, 30(2), 2008.

[39] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido. A nonstationary poisson view of internet traffic. *in Proc. INFOCOM*, 2004.

[40] V. Laatu, J. Harju, and P. Loula. Fairness comparisons of per-flow and aggregate marking schemes in diffserv networks. *in Proc. The 9th Open European Summer School and IFIP Workshop on Next Generation Networks*, 2003.

[41] V. Laatu, J. Harju, and P. Loula. The impacts of aggregation on the performance of tcp flows in ds networks. *in Proc. International Conference on Networking*, 2004.

[42] S. McCanne and S. Floyd. ns—network simulator. http://www-mash.cs.berkeley.edu/ns/.

[43] C. Metz. Ip routers: New tool for gigabit networking. *Cisco Systems*.

[44] M.-G. MGEN. Available online at:. http://cs.itd.nrl.navy.mil/work/mgen/index.php/.

[45] A. M. Mohammed and A. F. Agamy. A survey on the common network traffic sources models. *International Journal of Computer Networks(IJCN)*, 3, 2011.

[46] M. Moshin, W. Wong, and Y. Bhatt. Support for real-time traffic in the internet, and qos issues. 2002.

[47] J. T. Moy. Ospf: Anatomy of an internet routing protocol. 1998.

[48] NS-2. Available online at:. http://www.isi.edu/nsnam/ns/.

[49] OMNET++. Available online at:. http://http://www.omnetpp.org/.

[50] OPNET. Available online at:. http://www.riverbed.com/products/performance-management-control/opnet.html/.

[51] J. Pan. A survey of network simulation tools: Current status and future developments. *report*.

[52] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Trans. Networking*, 1, 1993.

[53] E. A. Pekoz and N. Joglekar. Poisson traffic flow in a general feedback queue. 2002.

[54] S. Ramabhadran, A. Bose, and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. *SIGCOMM*, 2003.

[55] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. Scalable architectures for integrated traffic shaping and link scheduling in high speed atm switches. *IEEE J. on Slc. Areas in Comm.*, 15(5), 1997.

[56] RUDE and CRUDE. Available online at:. http://rude.sourceforge.net/.

[57] J. L. Salina and P. Salina. Next generation networks: Perspectives and potentials. *Wiley*, 2008.

[58] M. Sanlı, E. G. Schmidt, and H. C. Güran. Fpgen: A fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks. *Elsevier*, 2011.

[59] M. Sanlı, E. G. Schmidt, and H. C. Güran. Hardware design and implementation of packet fair queuing algorithms for the quality of service support in the high-speed internet. *Elsevier*, 2012.

[60] M. Sanlı, E. G. Schmidt, and H. C. Güran. A flow aggregation method for the scalable and efficient quality of service support in next generation networks. *Globecom*, 2013.

[61] F. A. Sekib, S. McClellan, M. Singh, and S. Chakravarthy. End-to-end testing of ip qos mechanisms. *IEEE Magazine*, 2002.

[62] N. Simulation. Available online at:. http://en.wikipedia.org/wiki/Network-simulation/.

[63] S. Siraj, A. K. Gupta, and Rinku-Badgujar. Network simulation tools survey. *International Journal of Advanced Research in Computer and Communication Engineering*, 1, 2012.

[64] V. Sivaraman, F. M. Chiussi, and M. Gerla. Traffic shaping for end-to-end delay guarantees with edf scheduling. 2000.

[65] A. Sobeih, W.-P. Chen, J. C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang. J-sim: A simulation environment forwireless sensor networks. *Proceedings of the 38th Annual Simulation Symposium (ANSS'05)*, 2005.

[66] R. Stader. Qos provisioning for ip telephony networks by advanced bandwidth management. *Masters of Science Thesis*, 2001.

[67] D. Stiliadis and A. Varma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. *in Proc. ACM SIGMETRICS*, 1996.

[68] W. Sun and K. G. Shin. Coordinated aggregate scheduling for improving end-toend delay performance. *in Proc. IEEE IWQoS*, 2004.

[69] W. Sun and K. G. Shin. End-to-end delay bounds for traffic aggregates under guaranteed-rate scheduling algorithms. *IEEE/ACM Transactions on Networking*, 13(5), 2005.

[70] S. Suri, G. Varghese, and G. Chandranmenon. Leap forward virtual clock: A new fair queueing scheme with guaranteed delays and throughput fairness. *in Proc. IEEE INFOCOM*, 1997.

[71] G. Terdik and T. Gyires. Lévy flights and fractal modeling of internet traffic. *IEEE/ACM Transactions on Networking*, 17(1), 2009.

[72] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. *IEEE Communications*, 2009.

[73] J. Wroclavski and S. Shenker. Network element service specification template. *RFC 2216*, 1997.

[74] X. Yang and A. Petropulu. The extended alternating fractal renewal process for modeling traffic in high-speed communication networks. *IEEE Trans. Sig. Proc.*, 49(7), 2001.

[75] D. Zaragoza and C. Belo. Experimental validation of the on–off packet-level model for ip traffic. *Elsevier*, 2006.

[76] L. Zhang. Virtual clock: A new traffic control scheme for packet switching networks. *In ACM SIGCOMM*, 1990.