# Optimised Queuing Strategies for Multi-level QoS

by

Yan Ming Leong

Chief Supervisor Dr. C. K. Li

Department of Electronic and Information Engineering

The Hong Kong Polytechnic University,

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF PHILOSOPHY TO THE HONG KONG POLYTECHNIC UNIVERSITY

## CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

______________________________(Signed)

YAN MING LEONG (Name of Student)

# Abstract

A variety of network applications, such as FTP, HTTP, video on-demand, IP phone, have different requirements on traffic parameters, such as bandwidth, loss rate, delay bound and delay jitter. In order to provide the guaranteed service on above four traffic parameters, the concept of Quality of Service (QoS) and traffic management is essential to apply on the end-to-end paths of network applications.

Most QoS scheduling algorithms applies a single-parameter approach; they are based on either bandwidth or delay. In order to support more than one QoS parameter, the decoupling of bandwidth and delay in queuing can be deployed and this has been proved a difficult problem. Although recent research has tried to use rate regulator and priority-based scheduler to guarantee bandwidth and delay criteria respectively, the outgoing performance is insufficiently acceptable in order to be implemented in the present network environments.

In this research, a new scheduling algorithm called Delay-Differentiable Fair Queuing ($D^2FQ$) algorithm is proposed. This algorithm is a two-parameter approach that decouples the bandwidth and delay into a single model. Although $D^2FQ$ consists of the functions of rate regulator and delay scheduler, it has a low time complexity, $O(\log(L))$, where $L$ is a key component in $D^2FQ$ scheduler. This scheduling algorithm is also practicable for industrial applications. This thesis will also show the fairness analysis of $D^2FQ$ where the fairness on the bandwidth allocation is the basic criterion for QoS. The computation complexity and the delay bound will also be analyzed in detail.

For the purpose of simulation discussed in this thesis, a single-node network topology and multi-node network topology will be applied. Single-node network topology can simplify the network environment to facilitate the performance analysis of the scheduler in respect to fairness of bandwidth allocation. In this thesis, a single-node network will be applied on connectionless traffic analysis, which includes the throughput fairness and differentiate delay fairness on different offered loads and different traffic conditions. Some famous schedulers, such as WFQ, DRR and CSFQ, are used for performance comparison with $D^2$FQ. The two traffic parameters of $D^2$FQ, throughput weight and delay weight, produce a more complex traffic condition and provide a more useful analysis on $D^2$FQ.

A single-node network is generally used for connectionless traffic analysis. Most of the packet schedulers only use a single-node network topology. In addition to the propositions stated above, multi-node network topology will also be applied on connection-oriental traffic analysis in this thesis. Analysis on connection-oriented flow is useful because most of the traffics in present network are connection-oriented flow and the multi-node network topology can increase the increase the credibility of performance on $D^2$FQ. The analysis for simulation on multi-node network topology includes drop rate and delay on the condition of overloading. The simulation results show that $D^2$FQ has a good performance on throughput analysis and flow isolation, which conclude that $D^2$FQ can fairly decouple throughput and delay.

# Acknowledgements

I would like to express my gratitude to my supervisor Dr C. K. Li for his invaluable support, advice and help, not only on this thesis but also on all academic and personal issues. His kindness, trust and support has given me another chance to further expose myself to the excitement of networking and Electronic Engineering.

I would also like to specially thank Dr C. K. Leung and Mr. Wong Yun Lam for their advice; Dr Leung and Mr Wong have given me much help in the direction of my research and have guided me to explore my interest in this research area and its relations to the real world. Last but not least, I would like to thank my family whose love and encouragement has supported me to pursue my post-graduate studies; with special highlight to my mother's prayers which have been an invaluable source of support for all my academic achievements. Without any of the above people, this research project and my thesis would never have come into reality.

# Contents

# List of Figures

# List of Table

# Chapter 1. Introduction

In the 1970's, technology and scale of network were not sophisticated and thus the best-effect approach in network node can only fulfill a limited extent of the requirements of most applications such as text file transfer, email, etc. The performance of these applications is only measured in terms of bandwidth only. Until recent years, the best-effort internet cannot meet the rapid growth in the development and deployment of new network applications in business, entertainment industry or medicine area, for example, multimedia teleconferencing, video-on-demand and video with high-resolution, and imaging applications for medical operations. The performance of these applications requires low end-to-end delay bound. In order to support these multimedia traffics, the network must be able to guarantee performance bounds to meet the required Quality-of-Service(QoS). Traditional best-effort network models can no longer meet these QoS requirements. The scheduling in network node is one of key factors for guaranteeing the performance of QoS enabled network.

Bandwidth capacity and buffer size are the important resources of the packet switching networks to provide performance guarantees to network applications. A scheduling algorithm acts as a key factor affecting the allocation of these resources. Scheduling in network node is a scheme to select a queuing packet among multiple traffic flows and deliver it to the outgoing port. The delivery order of queuing packet affects the performance of throughput and delay of a traffic flow. Moreover, buffer management affects the packet loss rate in a congested network. If a packet of conforming traffic flow arrives at a congested node, buffer management should provide buffer space to the conforming packet and avoid any unfair packet drop in a congested node. In the current best-effort internet node, there is no guarantee for the delay bound as well as the drop rate. In general, most of these internet nodes employ a first-come-first-served mode (FCFS) for incoming packet treatment and packet drop if their buffer is full. FCFS model is simple and can support a large number of traffic flows but it cannot allocate bandwidth in a fair manner.

In a Defense of Service (DoS) attack network, a non-conforming flow congests one or more network nodes and dominates most of buffer space. All flows across the congested node would experience unfair drop rate. Such unfair buffer management scheme could produce a long queue that would induce a long queuing delay. These non-guaranteed drop rate and delay bound among flows would cause a significant deterioration in the performance of network applications, especially with respect to responsive traffic like real-time traffic. A QoS enabled node should resist to non-conforming flows at any congested node and provide fair bandwidth allocation and delay bound for all conforming flows. A QoS scheduling algorithm does not only provide a fair bandwidth allocation and delay distribution, but also should the buffer management of scheduler provide a protection on conforming flows which can affect a QoS parameter and lost rate of flows.

The Integrated Service or Intserv [34], and the Differentiated Service or Diffserv[35] are two different architectures in QoS. In the 1990’s, Integrated Service is the main research topic in QoS, which can support the quality to each traffic flow. Clients can rent a lease line to fulfill their quality requirement in their respective network applications. The versatility and scalability however, limit the variety of applications.

A number of researchers have moved their research direction from Intserv to Diffserv. Differentiated Service classifies all traffics into a number of classes, normally less than 10. Then all receive QoS according to their respective classes. The role of scheduling algorithm becomes less important because each traffic would be assigned a class and the total numbers of classes are bounded in a Diffserv system.

Although the development of the Diffserv system can completely solve the scalability limitation arising in Intserv, another new problem arises. Diffserv can support different QoS among classes but the fairness of traffic flows in the same class cannot be assured. Moreover, the number of classes in Diffserv needs to be kept within a small range to avoid the complexity problem in implementation, for example, scheduling algorithm. It is difficult for such a narrow range of classes to support different requirements of various network applications. Then researcher proposed some method, such as IntServ over Diffserv [28], but those methods still involve the resource allocation among classes. Therefore scheduling algorithm with QoS is still an important component in network architecture whichever the network architecture is Intserv or Diffserv.

Throughput, delay and loss rate are the essential metrics to quantify the performance of the network link. This research in scheduling algorithm mainly focuses on the treatment of the above three parameters, especially with emphasis in throughput and delay.

Bandwidth and latency bound are both important criteria of QoS and they are the natural perception for the user on the network performance. However, according to the traditional queueing theory, the characteristic of delay depends on the offered load and the pattern of source traffic. Therefore, most of scheduling algorithm can only support multi-class QoS in one dimension. The decoupling of the throughput and delay is a well-known limitation in most network scheduling research. Many researchers always query about the possibility to decouple the delay and throughput. Some algorithms [9, 10, 15] were proposed to solve this decoupling problem. Nevertheless, the scalability always limited the number of classes in the implementation.

In this research a scheduling algorithm called "Delay-Differentiable Fair Queuing ($D^2FQ$)" which is focus on decoupling of the throughput and delay on multi-class QoS, is proposed. To achieve the differentiated delay allocation and bandwidth allocation, this algorithm is applied to a dual parameter approach. The two parameters used are the delay weight and throughput weight. The proposed $D^2FQ$ consists of flow buffer and delay-slots. Each flow has state information of QoS, delay weight and throughput weight. Delay-slots in $D^2FQ$ are used to arrange the departure order of flows to differentiate the mean delay among flows. Besides the decoupling of throughput and delay, the implement cost is also considered. Generally, the time complexity of the implementation of scheduling algorithm depends on number of class such as O(n). The proposed algorithm will be shown that its implementation cost is independent of the number of class or flow.

The rest of the thesis is organized as follow. Chapter 2 presents the scheduling discipline and background of scheduling in general. In Chapter 3, a detail description of Delay Differentiable Fair Queueing ($D^2FQ$) is presented. Chapter 4 presents analytical results on the efficiency, fairness and performance characteristics of $D^2FQ$. Chapter 5 and Chapter 6 show simulation-based evaluation of the performance of $D^2FQ$ scheduler with other well-known scheduler in single-node topology and multi-node topology respectively. Finally, Chapter 7 gives a summary of this thesis, together with some conclusions.

# Chapter 2. Overview of Scheduling Algorithm

Scheduling in network node is a scheme to select the queuing packet among multiple traffic flows and deliver it to the outgoing port; the order of delivery of queuing packet affects the performance of the throughput and delay of traffic flow. In the traditional network model, the FIFO scheduling scheme cannot meet most application requirements and many fair scheduling algorithms have been proposed to satisfy the application requirements.

## 2.1 Real World Traffic in IP Network

A traditional voice network is a cooperative network; this means that the control and routing decisions for a customer are made with the total network in mind; individual performance objectives for each call is not a major factor. However, Internet is a non-cooperative network, which is characterized by multiple users' traffic. This multi-service network must be able to accommodate different requirements and maintain the independency of service among flows. There are many different kinds of network applications in the real world; these applications have different requirements including quality of service (QoS). In the networking layer, QoS can be defined as a set of techniques to manage bandwidth, delay, jitter, and packet loss of incoming and outgoing flows. QoS schemes in network layer are manage these four characteristics; for instance, Internet phone requires low-bandwidth and low-delay. FTP prefers large bandwidth only and is non-sensitive on delay and delay jitter.

In recent years, real-time applications such as multimedia have started to become popular. The corresponding applications also appear in Internet such as Voice over IP (VoIP), Video on Demand (VoD) and Video Conferencing. Although the bandwidth of the network link is increasing gradually, it is unable to fulfill many real time applications. Table 1 show the traffic requirement of different applications.

| **Application Types** | **Traffic requirements** | | | | |
|---|---|---|---|---|---|
| | **Elastic** | **Bandwidth** | **Delay** | **Jitter** | **Loss rate** |
| EMail | Yes | Low to Moderate | - | - | - |
| File Transfer | Yes | High Burst | - | - | - |
| Telnet | Yes | Low Burst | Moderate | - | - |
| Streaming Media (VoD) | Yes | Moderate | Sensitive | Sensitive | Sensitive |
| Video conferencing | No | Sustained High | Critical | Critical | Sensitive |
| Voice over IP | No | Low | Critical | Critical | Sensitive |

Table 1 Traffic requirement of diffent application

The conventional data applications do not usually have any delay criteria on quality of service; for instance, Email service can suffer a large delay variation and certain packet loss rate that they can work well under the best-effort model. On the other hand, some applications require real-time response or require receiving data continuously. Obviously, the major characteristics of these real-time applications are delay sensitive and packet loss sensitive. The requirement of bandwidth of VoIP is low however, the end-to-end delay bound is critical as the delay affects the perception of user. When two users are conversing with each other through VoIP, the delay in response time between both users is the criterion of quality of communication. Moreover, the delay jitter is also an important criterion of real time applications. Therefore, an ideal quality of service should fulfill all requirements of different network applications.

When the Internet becomes a part of life, troubles may occur in the Internet world. A non-conforming user or an attacker, may “flood” a network and prevent a legitimate user to access its network resource. In February, 2000 [13], the Yahoo!, Amazon.com, CNN.com, and other major Web sites were attacked by Distributed Denial of Service, DDoS. It was estimated that a 4-hour attack on above popular sites caused a total of $1 billion in economic impact. Although the flood packet may be blocked by a firewall at destination host, the normal user cannot access the service due to the packet drop at the congested node. DDoS not only affects the target host but also congest the node, which is along the path to the target host. Although many research and technology are trying to avoid the DDoS, in recent years a credit card firm was also attacked by DDoS in 2004[16]. Therefore, a QoS supporting network that provides a fair network resource allocation is a method to prevent the unfair packet drop. QoS embedded scheduler could act as one of key role in the network.

## 2.2 Scheduling Discipline

A scheduler has to ensure that the network resources are scheduled fairly among its contending users and scheduling disciplines are important because they are responsible for protecting one user's traffic from another and hence are key to fair sharing of network resources. By choosing the service order, the scheduler can allocate different mean delays to the packets belonging to different flows. In addition, it can allocate different bandwidths by serving a certain minimum number of packets from a particular flow in a given time interval. A scheduling algorithm that supports fair resource allocation and supports these performance bounds in order to serve the performance critical applications such as Voice Over IP (VoIP) and other interactive multimedia applications is needed. Fair scheduling becomes especially critical in access networks where the resource capacity constraints tend to be significantly limiting to the high-bandwidth multimedia applications today. Even with the growth of the bandwidth on the Internet, fairness in scheduling is essential to protect flows from other non-conforming flows triggered by deliberate misuse or malfunctioning software on routers or end-systems. Fairness in the management of resources is also helpful in countering certain kinds of denial-of-service (DoS) attacks. Some of the most desirable properties of a scheduling discipline include:

1. Fairness on resource allocation: The available link bandwidth must be distributed among the flows sharing the link in a fair manner. The classic notion of fairness given by the max-min fair share policy [24] is explained in detail in the following section. In general, it is desirable that the scheduler serves the connections proportional to their reservations and distributes the unused bandwidth left behind by the idle sessions proportionally among the active ones.

In addition, flows should not be penalized for the excess bandwidth they received while other flows were inactive. Fairness is also desirable for good performance, since unfair treatment of some traffic flows in the network can easily lead to unnecessary bottlenecks.

2. Decoupling among performance criteria: The scheduling algorithm should decouple the output performances of a set of performance criteria. When a network application requires guarantee service in a small bandwidth and a small latency, scheduling algorithm should avoid reserving a larger bandwidth to achieve the latency requirement and waste the over-claimed bandwidth reservation. Decoupling among performance criteria can increase the utility of network resource.

3. Predicable delay of serving flow: It is desirable that the scheduling discipline provides an end-to-end delay guarantee to individual flows. For guaranteed-rate services, the latency should be measured as the length of time, it takes a new flow to begin receiving service at the guaranteed rate. Low delay bounds imply low buffer requirements for guaranteeing no packet loss. Thus, the latency of a scheduler has a direct effect on the cost of implementation in terms of the required memory. The latency is also directly related to the amount of playback buffering required at the receiver for real-time communication applications.

4. Isolation among flow: The scheduling algorithm must isolation a flow and its performance should not affect the other flows. This will ensure that the QoS guarantee for a flow will be maintained even in the presence of other non-conforming flows. Note that, isolation among flows is necessary even when traffic policing strategies are used for traffic shaping at the edge of the

network, since the flows may become increasingly bursty as they traverse through the network [21]. Isolation among flows also results in a more predictable performance for end user applications.

5. Complexity of scheduling implementation: In addition to providing performance bounds and being fair, it is also important that a scheduler be easily implemented. A scheduler should require as few simple operations as possible to make a scheduling decision. In particular, the number of operations should be independent of the number of flows that are to be scheduled as possible. Thus, if *n* is the total number of queues or traffic flows to be scheduled by a scheduler, then a scheduler that has O(1) time complexity is preferred in comparison to the one that has O(*n*) time complexity. This property is especially desired in high-speed networks and in routers where the number of flows can be in thousands as in the Internet core.

## 2.3 Current Scheduling Algorithm

Throughput, delay and loss rate are the essential metrics to quantify the performance of the network link. The research of scheduling mainly focuses in the treatment of the throughput or delay. Rate-based algorithm [1, 2, 14, 25] and Round Robin based scheduler [19, 20, 26, 27] mainly focuses on the bandwidth allocation and deadline-based algorithm [7, 11], which is used to control the delay of traffic.

WFQ[1, 2, 25] is a well-known rate-based algorithm which can provide the nearly perfect fairness in bandwidth allocation. WFQ is the packetized version of Generalized Processor Sharing, GPS [3, 4]. GPS is an ideal fluid system and it can provide prefect fairness in throughput if the traffic is infinitely divisible. In WFQ, the queuing delay depends on the throughput-weight; the flow with larger throughput-weight would have shorter delay; this is the normal phenomenon on the rate-base algorithm. Frame-based or Round Robin based scheduler is also a common in the research of bandwidth allocation. These schedulers provides service opportunities to the backlogged flows in particular order and during each service opportunity; the intent is to provide the flow an amount of service proportional to its fair share of bandwidth. Deficit Round Robin[19, 20] is a modified Round-Robin scheduler which can provided a fair bandwidth allocation in IP network and the implementation complexity in time of DRR is as low as O(1) which mean independent of number of active. However, frame-based scheduler cannot provide shorter delay by increasing the throughput-weight.

In general, deadline-based scheduler classifies a packet into different priorities and the departure order is mainly dependent on their priority[11]. Priority queue can provide shortest queuing but which cannot guarantee the service for the flows in lower priority. Some proposed deadline-based scheduler such as EDD and its derivation [7] assign a timestamp to each incoming packet and server will sort the backlogged packets by its timestamp to decide the departure order. It needs more complex computation in sorting of the out-going packet. In addition, delay based scheduler also needs an extra rate regulator to achieve the bandwidth allocation which may introduce additional delay in a rate regulator.

### 2.3.1 Generalized processor sharing

In the max-min fair share allocation in bandwidth, Generalized processor sharing (GPS) [3, 4] is an ideal scheduler that it can provides exact max-min fairness. In GPS, a connection is called backlogged when it has data in the queue. GPS is based on an idealized fluid-flow model which can be able to serve all backlogged sessions instantaneously and that the capacity of the outgoing link can be split infinitesimally and allocated to these sessions.

Suppose that there are N flows being served by a server with service rate *R* and the *i*th flow is assigned a weight $\phi_i$, and let $W_i(\tau, t)$ be the amount of data serviced for flow *i* during an interval (τ, t). In GPS, for any backlogged flow *i* and for any other flow *j*, in (τ, t), the relation is:

$$\frac{W_i(\tau,\mathrm{t})}{\phi_i} \geq \frac{W_j(\tau,\mathrm{t})}{\phi_j} \tag{2.1}$$

In the interval (τ, t) the flow *i* receives a minimum fair share proportional to its weight

$$\frac{\phi_i}{\sum_{j=1}^{N}\phi_j} \times R \tag{2.2}$$

where *R* is the set of backlogged flow. However, GPS is an ideal fair scheduler, since serving in infinitesimal amount of data is impracticable and it cannot be used in the simulation study.

### 2.3.2 Weighted Fair Queuing

Demers, Keshav, and Shenker[1] propose a fair queueing to addressed fairness problem in real world. Faire queuing arrange competitive traffic flows to their bounded queues and the well-behaved traffic is protected form the ill-behaved traffic. Weighted Fair Queuing also known as Packet-by-Packet Generalized Processor Sharing (PGPS) [1, 2, 25], try to emulate the ideal GPS scheme by time-stamping each arriving packet with the virtual start time, which is the expected completion time that a packet would have had if it were scheduled by the GPS scheduler. The WFQ then serves the packets in the increasing order of the virtual finish time. Hence, this requires computation of the virtual finish time for every packet and then sorting among these time-stamps to determine the relative order in which the packets are to be served.

The following are Equations of WFQ to calculate the virtual start time (VST) $S_i^k$ and the virtual finish time (VFT) $F_i^k$.

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \tag{2.3}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i} \tag{2.4}$$

where *k*th session with packet *i* arrives at time $a_i^k$ with the packet size $L_i^k$. $\phi_i$ is the weighted share of the flow. V ($a_i^k$) is the system virtual time function which approximates the GPS clock. The Virtual function is illustrated below:

A GPS server serving N sessions is characterized by N positive real numbers, $\phi_1, \phi_2, \phi_3, \cdots, \phi_N$. The server operates at fixed rate r and the server rate for flow *i* is $r_i$:

$$r_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} \times r \tag{2.5}$$

The virtual function is given by:

$$\begin{aligned} &V(0) = 0 \\ &V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{j=1}^{N} \phi_j} \\ &\tau \le t_j - t_{j-1},\ j = 2,3,....... \end{aligned} \qquad (2.6)$$

*j* is the session of a specific flow.

WFQ de-queue the flow which gets the smallest virtual finish time among all the active flows which backlogged with packets. The WFQ FQS needs large computational effort and suffers from scalability problem.

### 2.3.3 Deficit Round Robin

WFQ and its derivative scheduler do not avoid the O(log n) complexity associated with sorting among the timestamps. Deficit Round Robin (DRR) [20], a less fair but more efficient scheduling discipline with an O(1) per packet work complexity, was proposed by Shreedhar and Varghese in 1996. DRR is not a timestamp-based algorithm, and therefore, avoids the associated computational complexity. DRR achieves O(1) time-complexity because it serves the active flows in a strict round robin order [19, 20]. It succeeds in eliminating the unfairness due to different packet size. This is done by keeping a state, associated with each queue called a deficit count (DC) to measure the past unfairness. A Quantum is assigned to each of the queues and when a flow is picked for service, its DC is incremented by the quantum value for that flow. A packet is served from a queue only if the packet size at the head is less or equal to the sum of the quantum and the deficit counter value; otherwise, the scheduler begins serving the next flow in the round robin sequence. When a packet is transmitted, the DC corresponding to that flow is decremented by the size of the transmitted packet. In DRR, in order that the per-packet work complexity is O(1), one has to make sure that the quantum value chosen is no smaller than the size of the largest packet that may potentially arrive at the scheduler [20]. Otherwise the per-packet work complexity increases to O(n) since one may encounter a situation, in which, even after visiting each of the n flows and examining the respective DC values, no packet is eligible for transmission. A per-packet work complexity of O(1) is ensured if there is at least one packet transmitted from each active flow during each round. This ensures that if the quantum is no smaller than the size of the largest possible packet, since this guarantees that the packet size at the head of each queue at the start of its service opportunity will always be less than the sum of the DC value and the quantum value of the flow. In order to achieve a per-packet work-complexity of O(1), therefore, the DRR scheduler requires knowledge of the upper bound on the size of a packet. DRR, thus, is not ideally suitable for wormhole networks since it requires the knowledge of the size of a packet before making a decision on transmitting it, and in addition requires an upper bound on the size of a packet.

### 2.3.4 Priority Queueing

Priority Queueing [11] is a simple scheduling algorithm that serves the highest-priority, non-empty queue to exhaustion and then moves on to the next-highest priority queue. This process repeats for each successively lower-priority queue; this scheduling function ensures that the highest-priority queue has the least loss, delay, and delay variation. Consequently, low-priority queues may experience significant delay variation, delays, as well as loss.

### 2.3.5 Jitter Earliest-Due-Date

Jitter Earliest-Due-Date (Jitter-EDD) [12] extends Delay-EDD, which have been described previously in this chapter. Jitter-EDD provides delay-jitter bound which is a bound on minimum and maximum delays. The expected deadline calculation which ensures the maximum delay bound is exactly the same as Delay-EDD. In Figure 2-1, the *PreAhead* time stamp labeling concept is added to provide the minimum delay bound. Each packet leaves the server will label a *PreAhead* timestamp and it is the difference between packet deadline and actual finishing time; if the packet is transmitting too fast, the regulator at the entrance of the next hop will hold the packet for *PreAhead* seconds that have been specified in the time stamp before it is eligible to be scheduled. This mechanism provides the minimum delay bound for the Jitter-EDD.

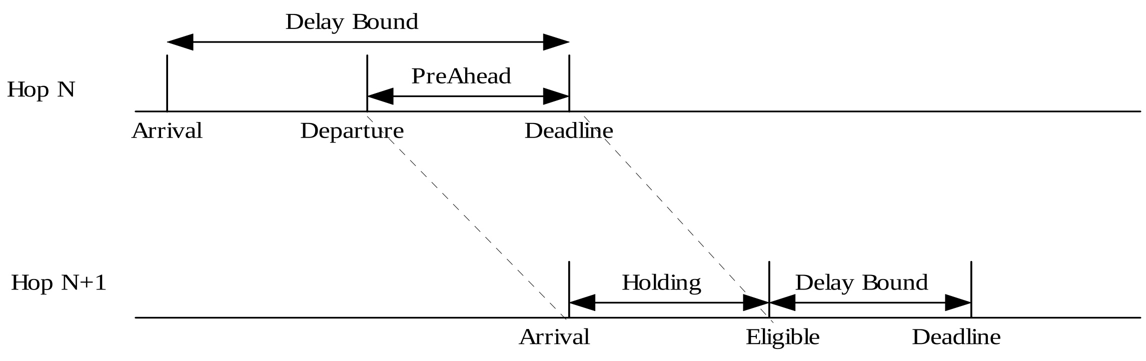


Figure 2-1 Packet Service in Jitter-EDD

### 2.3.6 Core-Stateless Fair Queueing Algorithm (CSFQ)

CSFQ [36] is an architecture of network rather than a simple scheduler in a node. In the proposed architecture in CSFQ, the routers in a network are classified to core routers and edge router. Edge routers will maintain the flow’s state and estimate flow rates and label the packets in their headers. Core routers perform probabilistic dropping on input based on these labels and estimation of fair share rate. This architecture can achieve approximately fair bandwidth allocation without complex implementation.

## 2.4 Queueing Delay

From the above mentioned scheduling algorithm, Scheduler with bandwidth treatment or delay treatment can easily achieve the QoS criteria. Compared with delay treatment, throughput treatment is a mechanism on the bandwidth allocation in physical link that may be implemented by using many counters to indicator the consumed bandwidth flow. Delay treatment is difficult to satisfy all flows’ requirement due to the delivery order of packet would affect experienced delay of all flows. Moreover, the packet arrival cannot be predicted thus the stochastic arrival time also affect the queuing delay of flows. The follow sections will discuss some other characteristics of queuing theory on delay that includes the conservation law and the coupling between throughput and delay in congestion condition

### 2.4.1 The Conservation Law

The classical conservation law of queuing theory states that the sum of the mean queuing delays received by the set of multiplexed connections, weighted by their fair share of the link’s load, is independent of the scheduling discipline. This law also reflects no scheduling algorithm and can achieve lower total mean delay to the other algorithm if there is no packet drop. If a scheduler wants to reduce the mean delay of one class, the other classes will suffer longer mean delay.

### 2.4.2 Tradeoff between throughput and delay in Congested Network

Two basic measures define the degree of congestion experienced – throughput and delay. The file transfer and voice/video applications represent extremes of application requirements for throughput and delay. Throughput is the data transfer rate actually achieved by the end application. For example, if a FTP application loses a packet, then it must retransmit that packet, and frequently in many implementations, all of the packets sent after it. Useful throughput is only those packets actually sequentially delivered to the end application without errors. Some applications like FTP, accept variable throughput to work acceptably.

Delay requirements differ by application types. Real-time traffic must be delivered within a fraction of a second, while for non-real-time applications that perform retransmission, delay takes on an additional dimension. When a protocol retransmits unsuccessfully delivered packets, the resulting delay is the time elapsed between the first unsuccessful transmission and the final successful reception of the packet at the destination.

Loss is another consideration in congestion control. Some application, like video, can adapt their transmission rate and still deliver good performance if the network congestion control minimizes loss. Other activities, like web-surfing, file transfer, and E-mail, recover from loss via transmission, usually with little user impact.

Note the throughput, delay and loss for some application is identical to that of the underlying IP or ATM network. For example, voice or video coded to operate acceptably under loss conditions is not retransmitted, and hence experiences the same throughput and delay as the underlying IP or ATM network. In practice, voice and video coding accept loss or delay up to a critical value; after which point the subjective perception of the image, or audio playback, becomes unacceptable.

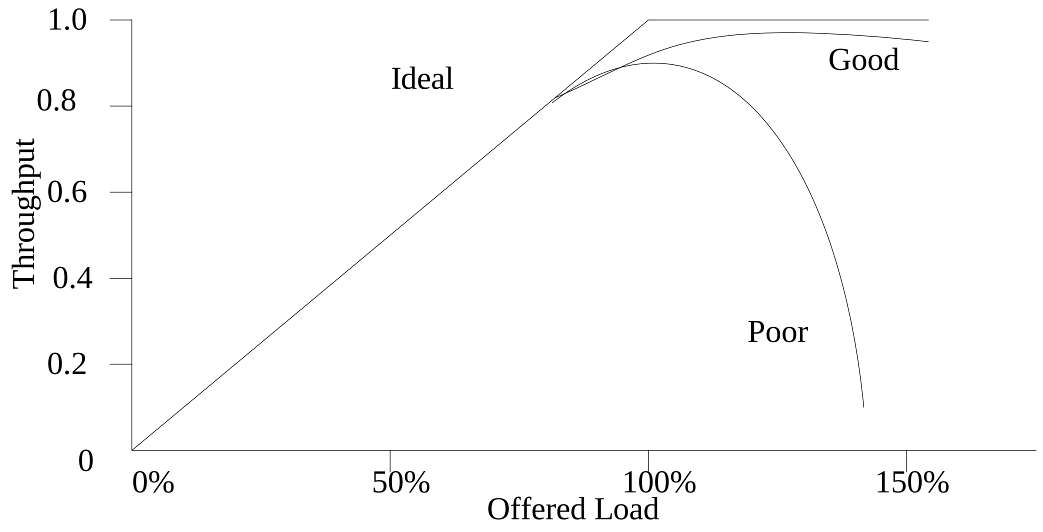


Figure 2-2 Congestion control for scheduler

Figure 2-2 shows the effective throughput versus offered load. An ideal scheduler has throughput that increases linearly until the offered load reaches 100 percent of the bottleneck resource. A good scheduler can approximate the ideal curve. A good scheduler with poor congestion control scheme exhibits the phenomenon called congestion collapse [22]. As the offered load increases by 100 percent, throughput increase to a maximum value and then decreases markedly due to user's application retransmissions; this causes packet loss or excessive delay and is identified as throughput collapses at the onset of congestion..

## 2.5 Research direction

In the previous section, the five disciplines for scheduler, which include fairness on resource allocation, decoupling among performance criteria, predicable delay of serving flow, isolation among flows and low complexity of scheduling implementation have been mentioned. The present schedulers, however, cannot fulfill all disciplines; some of scheduler, i.e. WFQ, can provide prefect fairness on bandwidth allocation and the isolation among flows but its computation complexity is high. DRR also provide acceptable fairness on bandwidth allocation and the low computation complexity but it cannot guarantee the isolation among flows. There are very few schedulers which can focus on decoupling among these performance criteria, especially the throughput and delay. Throughput and delay are two most important criteria for users. Users always consider two questions, they are “How many information can I get?” and “How long can I get the information?”. Bandwidth allocation in scheduler can answer the first question and predictable delay of serving flow could answer the second question.

Conservation law of queuing theory state that the scheduling order cannot reduce the overall mean queuing delay. Although the overall mean queuing delay cannot been reduced, the departure order can be adjusted to reduce delay of some flow and the other flow suffer longer delay. The adjustment of the departure order for delay differentiation would be one of the major directions on this research.

From the analysis on trade-off between throughput and delay on previous section, the overall delay depends on the offered load of the system. When a node is overloaded and the overall delay is rapidly increased, there is some non-conforming flow congested the node. The punishment on the non-conforming is necessary. In general, the punishment is packet drop on the non-conforming. Yet, the flow should not been over-punishment because over-punishment would decrease the performance of some adaptive network protocol such as TCP.   The non-conforming flow is not only punished by packet drop but also suffered non-guaranteed delay. The objective of this treatment on non-conforming flow is to protect the other conforming flow and ensure the conforming flow would not suffer long delay.

The scalability is very important consideration in scheduler design. The scalability of scheduler depends on the computation time. If the computation complexity is too large, the scheduler is not practical for implementation such as WFQ that is only implemented in low-speed routines of CISCO product series. Therefore, this research pays attention on computation complexity on the scheduler design. Round-Robin based scheduling has a low computation complexity. The performance of bandwidth allocation is also close to the performance of WFQ. Thus, this research is based on the concept of Round-Robin based scheduling in scheduling algorithm design for low computation complexity. A new mechanism for delay differentiation is proposed to decouple the throughput and delay requirements and provide a fair isolation of conforming flow and non-conforming.

# Chapter 3. Delay Differentiable Fair Queueing

In this section, the overall structure of proposed D$^2$FQ will be presented. D$^2$FQ scheduling algorithm supports two QoS criteria. They are the bandwidth allocation and the differentiable mean delay. A state vector of two elements is assigned to each flow. They are the throughput-weight and the delay-weight, which are used for the control of the throughput and the mean delay. They are also used for buffer management of all flows.

The traffic characteristics of real-time application are constant bit rate, CBR or light burst rate. They are always sensitive to delay. On the other hand, the traffic characteristics of data application, TCP application, are having variable-bit rate and heavy burst rate. They are not sensitive to the delay. When these two classes of traffic compete for the resource in a network node, the flows should follow their contracted traffic characteristics such as bandwidth and burstness duration to ensure that all traffic flows are scheduled fairly. Therefore, punishment on the "non-conforming flow" is a main concept in D$^2$FQ. When a heavily burst traffic arrives into D$^2$FQ node and its average arrival rate exceeds its contracted bandwidth limitation, the burst flow will be considered as a "non-conforming flow". The bandwidth allocation, packet departure mechanism and buffer management of D$^2$FQ will protect other conforming flows in teams of delay and bandwidth. The non-conforming flow will be served without QoS in the mean delay.

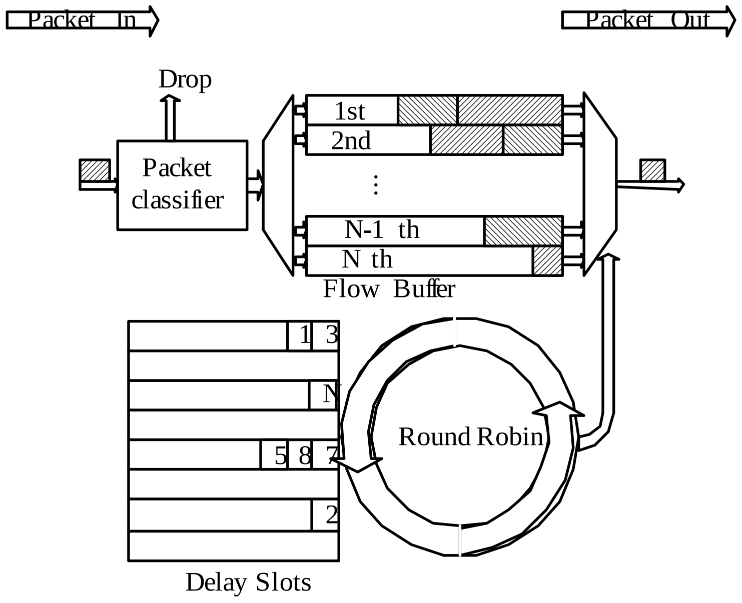


Figure 3-1 Structure of $D^2FQ$

## 3.1 Algorithm Description

During maintaining most of round-robin based scheduler, each flow consists of *active* and *idle* state. In general, a flow is in *active* state when a packet belonging to its flow is in the middle of being de-queued by the scheduler, or when the queue corresponding to the flow is not empty. Otherwise the flow is in an idle state which is shown in Figure 3-2

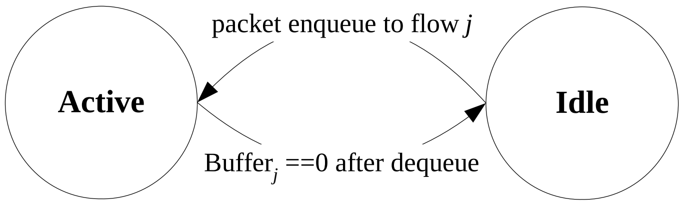


Figure 3-2 State diagram of Round Robin

In general round-robin based scheduler, all active flows are maintained by a linked list, called the *Active list*. When a new packet belongs to a flow whose queue was previously empty, the flow is inserted to the tail of *Active list*. Figure 3-3 illustrates the definition of round.

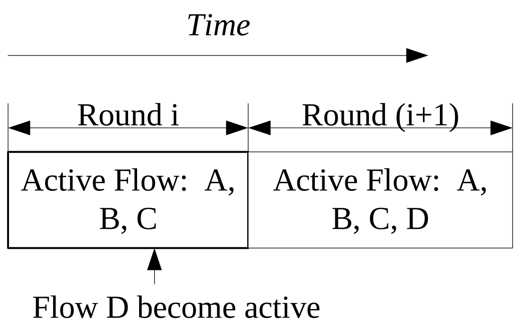


Figure 3-3 Example of Round Robin Scheduling

The new active flow cannot be served in the current round and it needs to wait until all reminding active flows are served in the current round; the delay of the first packet of a new flow needs to wait for a long frame time. This arrangement of new active flow can guarantee the fairness in bandwidth allocation for all active flows, however, the tradeoff is the extended delay of head packet in the flow buffer .

The round *i* is the time used to de-queue the packet in active flow A, B and C. Let flow D becoming active during the de-queuing packet at round *i*. The flow will be pushed to the end of the active link list. Although the flow D becomes *active* at round *i,* flow D can cannot be de-queue in round *i* and it need to wait to the next round *i+1* to de-queue the packet. This policy can confirm that each flow cannot be activated more than one time within a round. Therefore, one can imagine that the round boundary of round-robin scheduler is independent of the insertion time of any new actives flow but depends on the remainding active flows of the previous round. This de-queuing mechanism ensures that every active flow has equal opportunity to de-queue its packet and the fairness of bandwidth allocation among active flows can be guaranteed. However, a new active flow may experience a long queuing delay when a new flow is inserted into the active list at the time of a round beginning.

Different from the traditional round-robin based scheduler, which has a global starting time of round for all flows, however, the starting time of a round in D$^2$FQ is different for all flows. Therefore, the new state information of each flow, *Earning-Slot,* is introduced which is used to distinguish boundary of old round and new round of a flow. Moreover, when a flow is served and its corresponding buffer is empty, the flow cannot be trace as an idle flow immediately and it needs to wait unit the flow completed a full round time.

Figure 3.4 shows the state diagram of D$^2$FQ of flow *j*. Each flow in D$^2$FQ scheduler has three states: *idle*, *semi-idle* and *active*. When scheduler starts initialization, all flows are in *idle* as same as general round robin based scheduler. When a packet is asssigned to a flow at an idle state, the flow will be pushed to a non-empty state. Semi-idle state is a special condition of active flows; when D$^2$FQ scheduler has served a packet and its flow become empty, the flow will be pushed to the semi-idle state. That means the flow is in the middle of its round time but there is no packet in its flow buffer. It needs to wait until the current round time is finished. If an arrival packet is classified to a flow at semi-idle state, the flow will be pushed to the non-empty flow waiting for the service of the system. This additional semi-idle state is used to maintain the throughput fairness apart from all *non-idle* flows in the D$^2$FQ scheduler to avoid some flow activated more than one time within a round.

A pseudo-code implementation of the D$^2$FQ scheduling algorithm is shown in Figure 3-5. It consists of Initialize, Enqueue, Dequeue and some sub-routines;

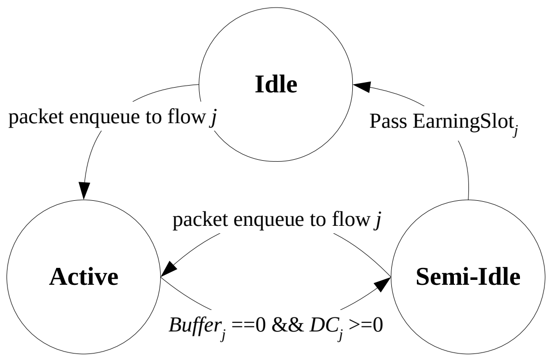


Figure 3-4 State diagram of $D^2FQ$

```
Initialize:
   PollingSlot =0 ;
Set all flows in idle state;


Enqueue: (Invoked when a packet, pkt arrives)
   f_index=classify(pkt);
   if (State[f_index ==Idle) then
            push pkt into Buffer[f_index];
            State[f_index] = Active;
            DC[f_index]=0;
            earning-slot[f_index] = PollingSlot;
            DelaySlotHopping(f_index);
   else
         if (State[f_index] ==Semi-Idle) then
                  push pkt into Buffer[f_index];
                  State[f_index] = Activated;
                  DelaySlotHopping(f_index);
         else
                  push pkt into Buffer[f_index];
         end if;
   end if;
```

```
De-queue:
  while(ture) do
    if (DelaySlot[PollingSlot]==EMPTY) then
        PollingSlot = Next_non-empty_slot();
    end if;
    f_index = GetHeadIndex(DelaySlot[PollingSlot]);
    if (Buffer[f_index] is not empty) then
            pkt = PullPakce(Buffer[f_index])t
            de-queueing packet(pkt)
            DC[f_index] -= sizeof(pkt);
    end if;
    DelaySlotHopping(f_index);
end while;


IsPassEarningSlot(oldslot, newslot, earningslot)
    if (oldslot <= earningslot) and ( earningslot < newslot) then
        return TRUE;
    else
        return FALSE;
    end if;


Update-deficitcounter(flow)
  if (DC[flow] >0) and (DC[flow]> totalsize(Buffer[flow]) then
    DC[flow] = totalsize(Buffer[flow]) + TW[flow]*Q;
  else
    DC[flow] += TW[flow]*Q;
  end
```

```
DelaySlotHopping(flow)
    nextslot = PollingSlot + DW[flow];
    if ( PollingSlot > earning-slot[flow]) then
        nextearningslot = earning-slot[flow]+L;
    else
        nextearningslot = earning-slot[flow];
    end if;
    if(DC[f_index] < 0) then
        // too many packet
        nextslot = Max( nextslot, nextearningslot +1);
        update-deficitcounter(flow);
    else
        if (State[flow] ==Semi-Idle) then
            State[flow] = Idle;
            exit;
        end if;
        if (Buffer[flow] is empty) then
            State[flow] =Semi-Idle;
            nextslot = Max( nextslot, nextearningslot);
        else
            if ( IsPassEarningSlot(PollingSlot, nextslot, nextearningslot)
            ==TRUE) then
                update-deficitcounter(flow)
            end if;
        end if;
    end if;
    nextdelayslot   = modulus (nextdelayslot,L);
    PushFlowToDelaySlot(flow, nextdelayslot);
```

Figure 3-5 A pseudo-code of $D^2$FQ scheduling algorithm

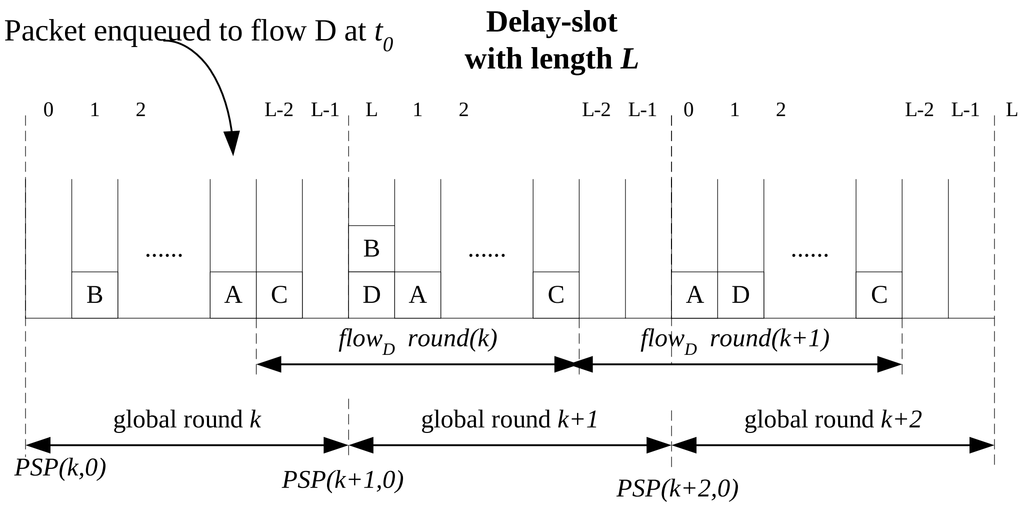


Figure 3-6 Definition of flow round of $D^2FQ$

In the $D^2FQ$ scheduler, the flow buffer only stores packet and the reordering is implemented through the use of a delay-slot. Delay-slot is a list of slots and each slot is used to store queue of flow-index. Flow index is an identification of flow buffer. In any time, each flow-index cannot queue more than one slot. *Polling-Slot-Pointer, PSP*, is used that works in round robin manner on the delay-slot. When the *Polling-Slot-Pointer* enquires the head index of slot, *s*, the corresponding flow buffer of enquired flow-index will be served and there is only one packet be served. After serving the flow buffer, its flow-index will be hopped to other slot to wait next service. The hopping mechanism will be discussed later. The *PSP* will try to enquire the same slot with other queued flow-indexes until the slot is empty and *PSP* will enquire next slot *s+1* cyclically.

Let *PSP*($k$,$s$) denoted the instance of time when *Polling Slot Pointer* of scheduler starts to serve slot $s$ at $k$-th time and $s \in (0, L\text{-}1)$ where $L$ is the number of delay-slot. In $D^2FQ$, global round is used to indicate the round of own scheduler. In general, of course, the scheduler is first initialized at $t_0$, then round $k = k_0$ and $s = 0$. e.g.

*$PSP(k_0,0) = t_0$*. In D$^2$FQ scheduler, the round of own scheduler is said as global round which is different from the flow round of each flow. Global round boundary starts from *0*-st slot to (*L-1*)*th* slot, and has total *L* slots. A flow round is also have L slots but the starting slot is depended on the *PSP*'s pointing location when the flow is being activated from idle state. Figure 3-6 illustrates this definition of global round and flow round.

Consider a packet enqueuing flow *D are* in idle state at time $t_1$. At $t_1$, and *PSP* enquires (*s-1*)-*th* slot in *k-th* global round and the head index of (*s-1*)-*th* is being served. $Round_D(k)$ is denoted as a flow round boundary time of flow *j* which is started from *PSP*(*k,s*) to *PSP* (*k+1,s*) and total size of slot is also *L*. $Round_D(k)$ may be across the boundary, e.g. *PSP*(*k+1,0*), between global round *k* and *k+1*.

Due to the *flow D* is previously idle, then flow state will be change from idle state to active state and a flow state information *Earning-Slot*, $ES_D$ is set to *s*, which indicates a boundary of flow round. Hence the flow round can be defined as [$PSP(k,ES_D)$, $PSP(k+1, ES_D)$] when the flow is not an idle state.

The arrival packet of flow *D* would queue in the $Buffer_D$ and the *flow-index, D,* hops to *delay-slot.* The bandwidth allocation of D$^2$FQ is using the advantage of DRR in which a *Quantum* is assigned to each of the flow and each flow is also associated a *deficit count(DC)* to measure the past unfairness. In D$^2$FQ, deficit counter is borrowed and DC is incremented by the quantum value whenever flow-index hopping is passing its *Earning-Slot.* Due to the different definition of round between D$^2$FQ and DRR, the increment of *DC* in D$^2$FQ is slightly different from the DDR in order to maintain the fairness of among flows,.

Let $Served_i(k,s)$ denoted the data served form flow *i* in the delay-slot *s* at global round *k*. Also, let $DC_i(k, s)$ denoted the deficit count of flow *i* when the flow-index *i* is hopping across delay-slot *s* at global round *k*. Note that delay-slot is a constant size and then the slot L-th is exactly equal to the slot 0-th in the next round and the slot (-1)-th is exactly equal to the slot (L-1) of previous round. Also, this deficit count is carried over to the first slot of the subsequent global round. Hence,

$$DC_i(k, L) = DC_i(k+1, 0) \tag{3.1}$$

and ,

$$DC_i(k, -1) = DC_i(k-1, L-1) \tag{3.2}$$

Note that, $DC_i(k, -1)$ is used to represent the value of deficit count of flow *i* when *PSP* is pointing at the end of delay-slot , slot 0-th, in (*k-1*)-th round and $DC_i(k, L)$ is used to represent the value of deficit count of flow *i* when *PSP* is pointing at the starting of delay-slot at (*k+1*)-th round.

When *s* is not the earning-slot of flow *i*, then

$$Served_i(k,s) = DC(k,s) - DC(k,s+1) \tag{3.3}$$

When *PSP* is across the earning-slot of a flow, the quantum of corresponding flow will be earned. During updating the *DC* in earning-slot, $D^2$FQ uses the minimum value between deficit count value and the total size of packets in *Buffer j*. This limitation is used to avoid some flows accumulated the unused deficit count and affect the fairness when the flow suddenly becomes a heavy burst traffic pattern. In $D^2$FQ, all flows have at least once opportunity to dequeue their packet. The minimum service of all flows is the minimum size of Packet. Let *Min* be the minimum size of packet for all flows. Therefore, in the worst case, flow can carry out quantum (*Q-Min*) to the next flow round and the new round have quantum 2Q –Min to obtain the service. If the last flow round have not used up all quanta, i.e.

*DC*(*k*,*ER*) *>0,*

$$\begin{aligned} Served_i(k, ER_i) = Min\{&DC_i(k, ER_i), SizeOf(Buffer_j)\} \\ &-DC_i(k, ER_i + 1) + Q \end{aligned} \tag{3.4}$$

When the flow have not over-claimed the bandwidth requirement, then no quanta would be trimmed off and then,

$$Served_i(k, ER_i) = DC_i(k, ER_i) - DC_i(k, ER_i + 1) + Q \tag{3.5}$$

Unlike DDR, D$^2$FQ will not check the packet size and *DC* before servicing the packet, D$^2$FQ must serve the head packet of in the Buffer$_D$ but DDR will only service the packet when its *DC* value is larger then the size packet. When *DC* value is negative, the flow index will be hopped to the next earning-slot to compensate the borrowed quantum. Therefore the boundary of the *DC* is

$$-M \quad \le \quad DC_i(k, s) \le \quad 2Q\text{-}Min \tag{3.6}$$

where *M* in the maximum packet size.

Hence, when flow *D* just becomes the active state from an idle state, the previous *DC* value will be set to zero and it can earn *DC* by quantum. The hopping of *flow-index D* is depended on the flow's delay-weight *dw* where $dw \in N$,( *where N* is set of nature number). Delay-weight is the amount of slots between each opportunity of a flow service. When flow *D becomes* active from the idle state, its flow-index *D* will hop to the slot offset from the current location of *PSP, (s-1)$^{th}$ slot,* to $\{dw_D + (s\text{-}1)\}^{th}$ *slot.*

D$^2$FQ server is continually serving the backlogged flow, *A*, *B*, *C* until flow *D* becomes the head of index at the slot of *PSP,* then flow *D* has an opportunity to dequeue one packet. The *DC* decrement of D$^2$FQ is the same as that of DDR, after dequeuing a packet from flow buffer, the packet length is subtracted from the deficit counter, and the flow-index will be hopped to the delay-slot again.

When a flow is not a fresh active, the destination slot of hopping index is depended on the status of flow includes the *DC value and* the packet in the flow Buffer. When the *DC* of a flow is negative and its means that the flow has used up all its allocated bandwidth in this round and has borrowed deficit in the next round, the flow-index cannot receive any service in this flow round again and needs to hop to the next *Earning-slot, ES*, to earn more quantum. If the *DC* is larger than zero, that means the flow has not used up its allocated bandwidth in this round and it will have one and more opportunity to obtain the service. When there is some packet backlogged in its flow buffer and then the flow-index will hop to slot which has an offset *dw* slot from the current slot, i.e. the polling slot of PSP. When there is no packet backlogged in the flow buffer and the *DC* is larger then zero, the flow can still obtain service in the current round. If no packet arrives, the flow-index will be changed from an active state to a semi-idle state which still keeps the value of *DC*, but the flow-index will hop to the starting slot of the next flow round. When there is an arrival packet belongs to the flow before the flow round complete, then the flow will be activated and the flow-index will hop through *dw*'s slots from the current polling slot of *PSP*. Of course, the *DC* of flow would not been updated when the flow changed from semi-idle state to active state. When the server tries to provide service to a flow that in a semi-idle state, the flow that has not used up all its allocated bandwidth in the last flow round will enter a new round. The flow is considered in an idle state and its flow-index will not be pushed into the delay-slot again until the next flow's packet is reached.

## 3.2 Bandwidth allocation

This section shows proposed D$^2$FQ can be easily adapted for scheduling guarantee rate connections and presents the methodology of bandwidth allocation in D$^2$FQ scheduler. Consider a D$^2$FQ scheduler that has an output link of transmission rate *r*, Let *n* be the number of flows and Let $\rho_j$ be the smallest of the reserved rates. Note that since all flows share the same output link, a necessary constraint is that the sum of the reserved rates must be no more than the transmission rate of the output link. Hence,

$$\sum_{j=1}^{n} \rho_j \leq r \tag{3.7}$$

In order that each flow receives service proportional to its guarantee rate, the D$^2$FQ scheduler assigns a weight to each flow. The through-weight assigned to flow *j*, $tw_j$, is given by,

$$tw_j = \frac{\rho_j}{\rho_{\min}} \tag{3.8}$$

Note that for any flow, *j*, $tw_j \geq 1$

Actually, the throughput-weight supported D$^2$FQ only needs to modify the deficit count increment process when a flow is across its new starting slot of a round. Therefore $tw_j$ **Q* is used instead of *Q* only. The deficit count increment Equation (3.4), will become,

$$\begin{aligned} Served_i(k, ER_i) = Min\{DC_i(k, ER_i), SizeOf(Buffer_j)\} \\ -DC_i(k, ER_i + 1) + tw_i * Q \end{aligned} \tag{3.9}$$

Equation (3.5) will become,

$$Served_i(k, ER_i) = DC_i(k, ER_i) - DC_i(k, ER_i + 1) + tw_i * Q \qquad (3.10)$$

The boundary of deficit count also will change to,

$$-M \quad \leq \quad DC_i(k, s) \leq \quad 2(tw_i * Q)\text{-}Min \qquad (3.11)$$

## 3.3 Delay Distribution

This section will show the delay differentiation to all active flow.

If one considers the total n flows are in active state during the time period (*PSP(k,s), PSP(k+1,s))*. Then all flows have same throughput weight and same packet size, the service time of all packets is therefore the same.

## 3.4 Delay-Slot

From the above mechanism of flow-index hopping, a flow has at most $\lceil L/dw \rceil$ time opportunity to obtain service where *L* is the size of delay-slot. Moreover, each service opportunity only can de-queue one packet. Hence, $D^2FQ$ should guarantee each flow has enough opportunity to use up all its quanta in a flow round. Although different flows have different requirements in packet size, delay-weight and throughput-weight, the length of delay-slots, *l*, should be long enough to let all flows, n, using up its quanta.

Considering a flow *j* with delay-weight $dw_j$, and throughput-weight, $tw_j$, have a minimum packet $MinSize_j$. Then, the deficit count can be earned in each round is $tw_j$ *Q. In the extreme case, all packets in flow *j* are minimum packet size, then the number of opportunity is *($tw_j$*Q/MinSize$_j$)*. Therefore, the minimum length of delay-slot, $L_j$ ,for the flow *j* is

$$L_j = \frac{dw_j * tw_j * Q}{MinSize_j} \tag{3.12}$$

If delay-slot is shared to all flows, then the minimum size of delay-slot is

$$L \geq \underset{j \in G}{Max}\left\{\frac{dw_j * tw_j * Q}{MinSize_j}\right\} \tag{3.13}$$

where *G* is a set of all flows

## 3.5 Buffer Management

In most scheduling algorithms, there are two major strategies in buffer management. They are shared buffer management and separated buffer management. Shared buffer management can provide a higher utilization of output link [18], when some of the flows inject long burst traffic pattern. On the other hand, separated buffer management can provide a better protection for the conforming flow against the non-conforming flow. Here, a buffer management in D$^2$FQ is proposed. The buffer management in D$^2$FQ shares the whole buffer, *B*, and is proportional to flow's throughput weight $tw_i$ to achieve the higher utilization.

$$fb_i = \frac{w_i}{\sum_{j \in N} w_j} \times B \tag{3.14}$$

where $fb_i$ is the flow buffer size and *B is the overall buffer.*

Buffer allocation of D$^2$FQ is not carried out strictly when the buffer is not fully utilized. Moreover, there are some procedures to enhance flexibility of buffer management between fairness and utilization. Before an arrival packet en-queue to the buffer, the flow buffer loading is calculated

$$\eta_i = \frac{size(p_k) + bs_i}{fb_i} \tag{3.15}$$

where $\eta_i$ is the loading of the flow, $p_k$ is arrival packet and $bs_i$ is the size of queuing packets in the flow.

If the flow buffer loading is less than 1, then the packet can be en-queued safely. Otherwise, the overall buffer utilization is further calculated sing the following criterion.

```
If $\frac{\sum bs_j}{B} < \alpha$ and $\eta_i \le \beta$ then,
    enqueue the packet;
Else
    drop front packet of flow, en-queue $p_k$;
End if
```

Where $\alpha$ is the upper threshold of the overall buffer loading and $\beta$ is the upper threshold of each flow buffer loading. When the overall buffer utilization is below the threshold $\alpha$, and the flow buffer utilization is below the threshold $\beta$, the arrival packet can be en-queued to the buffer. Otherwise the flow is regarded as a non-conforming flow and the front packet of the corresponding flow will be dropped until the buffer size is large enough to en-queue the arrival packet. The drop of the front packet of flow can reduce the queuing delay of the flow.

The flow buffer loading indicates the usage of the flow buffer is used to reserve the capacity of sharing buffer. Moreover, the two thresholds, $\alpha$ and $\beta$ are used to increase the flexibility of the flow with short-term burst traffic pattern. This adaptive buffer management can ensure the performance in throughput for the short-team burst traffic but the delay will be increased due to the increase in buffer size.

In the simulation shown on Chapter 6 and Chapter 7, this buffer management in the $D^2$FQ server is used. Every server holds two threshold values $\alpha$ and $\beta$ are optimized by most of the simulation analysis. Both values are set to 0.8 and 2.5 respectively during the simulation.

# Chapter 4. Performance Analysis of $D^2FQ$

In this chapter, a detail analysis of the performance characteristics of the $D^2FQ$ discipline will be presented. The performance of $D^2FQ$ based on the three important properties that are fairness, latency and efficiency.

**Fairness**: The available link bandwidth must be distributed among the flows sharing the link in a fair manner. This ensures that the performance achieved by a flow is not affected when a possibly non-conforming flow tries to transmit packets at a rate faster than its fair share. A well-known and widely used metric, known as the relative fairness bound [23] is used to measure fairness.

**Latency**: An appropriate measure of packet schedulers in this regard, especially for schedulers seeking to provide guaranteed services is the upper bound on the length of time it takes a new flow to begin receiving service at the guaranteed rate [8]. The latency bound is directly related to the amount of playback buffering required at the receiver.

**Efficiency**: The efficiency of a scheduling discipline is measured in terms of the order of work complexity associated with the enqueuing and de-queuing operations, with respect to n, and the number of active flows. In high-speed networks with large numbers of active flows, the time available for a scheduler to make its scheduling decision is very small. Hence, it is desirable that the time to enqueue a received packet or to de-queue a packet for transmission is as independent as possible of the number of flows sharing the output link. A per-packet work complexity of O(1) is most desirable.

## 4.1 Fairness Analysis

The fairness of a scheduling discipline is best measured in comparison to the GPS scheduling algorithm. The quantity, known as the Absolute Fairness Bound (AFB) of a scheduler *S*, is defined as the upper bound on the difference between the service received by a flow under *S* and that under GPS over all possible intervals of time. This bound is often difficult to derive analytically. It has been shown in [29] that the AFB is related by a simple equation to another popular fairness measure known as the Relative Fairness Bound (RFB) first proposed in [23]. The RFB is also much easier to evaluate as compared to the AFB. Therefore, the RFB is used in the fairness analysis. the metric is identical to the one used in [20]. The RFB is defined as the maximum difference in the service received by any two flows over all possible intervals of time. In the following, a flow is considered as active during an interval of time, that mean the queue is never empty of packets.

The active flows only be considered to measure the fairness because it makes no sense in comparing a non-active flow because a non-active flow does not receive any service. Thus, in $D^2$FQ, the flows will only be considered active when flow is in an Active State. The flows which is in Semi-Idle or Idle state is treated as non-active.

**Definition 4.1.1** Let $Sent_i(t_1, t_2)$ be the number of bits transmitted by flow i during the time interval between $t_1$ *and* $t_2$. Given an interval $(t_1,t_2)$, the Relative Fairness, $RF(t_1, t_2)$ for this interval was defined as the maximum value of $|Sent_i(t_1, t_2) - Sent_j(t_1, t_2)|$ over all pairs of flows *i* and *j* that are active during this interval. Define the relative fairness bound (RFB) as the maximum of $RF(t_1, t_2)$ for all possible time intervals *$(t_1, t_2)$*.

It is desirable that RFB is a small constant. The smaller the RFB, the closer the scheduler emulates the GPS scheduler, which is considered an ideal fair scheduling algorithm.

**Definition 4.1.2** Define *M* as the size in bit of the largest packet that is served during the execution of a scheduling algorithm.

**Lemma 4.1.1** For any flow *i* which is in active state in whole round *k*, then the boundary of the deficit count at round *k+1* is

$$-M \leq DC_i\ (k+1, s) \leq Q \tag{4.1}$$

*Proof:* The lower bound is same as the Equation (3.6). In $D^2FQ$, when a flow always have backlogged packet in it flow buffer, the flow could used up all quantum in a round and it could further transmit once more packet before its flow-index is hopped to the earning round. Therefore, the lower bound of Equation (4.1) is the maximum size of packet. Moreover, there is no quantum be carried from the previous round to the current round when the flow is always active. The upper bound of deficit count is *Q* from the summing of *Q* and thus, old deficit count and the statement of lemma is proved.

**Theorem 4.1.1** For any execute of the $D^2FQ$ scheduling discipline, FM < *(3Q+2M )*

*Proof:* Considering all time intervals bounded by the time instants that coincide with the starting or ending of service to a flow. the statement of the theorem is proved by using the time interval between instants $t_1$ and $t_2$ where both $t_1$ and $t_2$ are the time instants at which $D^2FQ$ scheduler ends serving on flow and begins serving another. Refer to Figure 4.1, it shows the corresponded round and the slot of *PSP* for time instants $t_1$ and $t_2$. Thus,

$$PSP(r_1,s_1) \leq t_1 < PSP(r_1,s_1+1)$$

$$PSP(r_2,s_2-1) \leq t_2 < PSP(r_2,s_2)$$

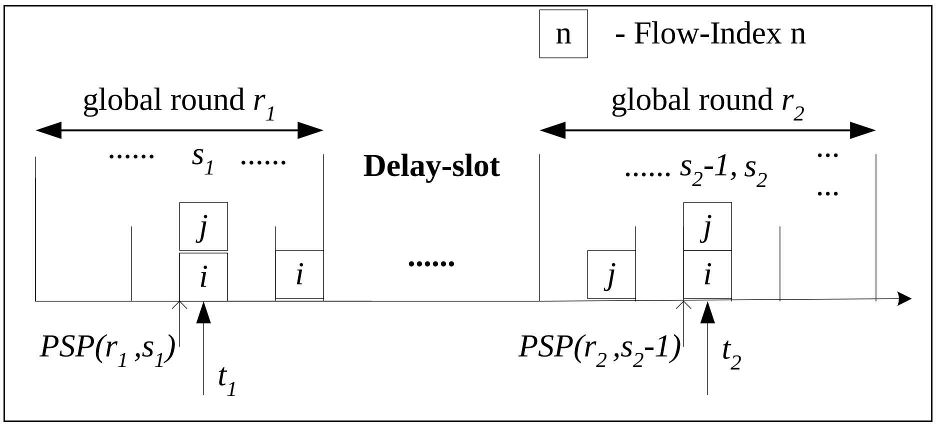


Figure 4-1 Fairness analysis of two flows *i* and *j*

Then, the sent bit in the interval $(t_1,t_2)$ of flow *i* is,

$$Sent_i(t_1,t_2) = \sum_{s=s1}^{L-1}\{Served_i(r_1,s)\} + \sum_{r_1-1}^{r_2-1}\sum_{s=0}^{L-1}\{Served_i(r_1,s)\} + \sum_{s=0}^{s_2-1}\{Served_i(r_2,s)\} \tag{4.4}$$

Flow *i* is always active at the interval $(t_1,t_2)$, and there is no quantum be trimmed off when the flow index is across the earning-slot. Combining Equation (4.1) into Equation (4.4),

$$Sent_i(t_1, t_2) = DC_i(r_1,s_1) + R_i{*}Q - DC_i(r_2,s_2) \tag{4.5}$$

where $R$ is number of time of flow $i$ hopped across the earning-slot.

Each flow has its earning-slot in D$^2$FQ scheduler, and the follow cases are consided:

Case 1: $(s_1 < s_2) \&\&(s_1 \leq ES_i < s_2)$

In the case, flow $i$ have can hop across the earning slot between time $(PSP(r_1,s_1), PSP(r_1,s_2))$ and it has $(r_2-r_1)$ times hopped across the earning slot. Thus,

$$R_i = (r_2-r_1+1)$$

Case 2: $(s_1 < s_2) \&\& (ES_i < s_1 \mid\mid s_2 \leq ES_i)$

$$R_i = (r_2-r_1)$$

Case 3: $s_2 < s_1 \&\& s_2 \leq ES_i < s_1$

$$R_i = (r_2-r_1-1)$$

Case 4: $s_2 < s_1 \&\& (ES_i \leq s_2 \mid\mid s_1 \leq ES_i)$

$$R_i = (r_2-r_1)$$

Now considering another flow $j$ which has received less service than flow $i$ between time interval $t_1$ and $t_2$. From Equation (4.5)

$$Sent_j(t_1, t_2) = DC_j(r_1,s_1) + R_j{*}Q - DC_j(r_2,s_2) \tag{4.6}$$

Then combining Equation (4.5) and Equation (4.6) and using Equation (4.1), consider the $s_1$ th slot less than $s_2$ th slot and in the worse case, $R_i = (r_2-r_1+1)$ and $R_j = (r_2-r_1)$

$$Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq \{DC_i(r_1,s_1) + (r_2-r_1+1)\,Q - DC_i(r_2,s_2)\}$$
$$-\{DC_j(r_1,s_1) + (r_2-r_1)\,Q - DC_j(r_2,s_2)\}$$

$$Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq (3Q+2M)$$

Consider the $s_1$ th slot less than $s_2$ th slot and in the worse case, $R_i = (r_2\text{-}r_1)$ and $R_j = (r_2\text{-}r_1\text{-}1)$, the equation becomes:

$$Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq \{DC_i(r_1,s_1) + (r_2-r_1)\, Q - DC_i\,(r_2,s_2)\}$$

$$-\{\, DC_j\,(r_1,s_1) + (r_2-r_1-1)\, Q - DC_j\,(r_2,s_2)\}$$

$$Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq (3Q+2M)$$

The above two cases illustrate that the worst-case fairness between two fairness is the same, i.e. $(3Q+2M)$. The statement of the theorem is proved. •

If the two flows have different throughput weight, and use Equation (3.11) to replace Equation(4.1), then

$$-M \leq DC_i\,(k+1, s) \leq w_iQ \tag{4.7}$$

The fairness bound can calculate by the above procedure.

## 4.2 Latency bound analysis of D²FQ

In this section, the latency bound of the D²FQ scheduler is analyzed. The concept of Latency Rate servers was first introduced in [8]. The Latency Rate (*LR*) theory provides a means to describe the worst-case behavior of a broad range of scheduling algorithms in a simple manner. The two key parameters that determine the behavior of a *LR* server are the latency and the reserved rate of each flow. The latency of a *LR* server is a measure of the cumulative time that a flow has to wait until it begins receiving service at its guaranteed rate. The latency of a particular scheduling algorithm may depend on a number of factors such as internal parameters of the scheduling discipline, the reserved rates of the other flows multiplexed on the same output link and the transmission rate of the flow on the output link. It has been shown in [8] that several well-known scheduling disciplines such as Weighted Fair Queuing (WFQ), Self-Clocked Fair Queuing (SCFQ), Virtual Clock and Deficit Round Robin (DRR) belong to the class of *LR* servers. The detail definition of *LR* server is presented in Appendix A.

**Theorem** The D$^2$FQ scheduler belongs to the class of *LR* servers, with an upper bound on the latency for flow i given by,

$$\Theta i \leq \frac{1}{r}\left((n-1)M + \left(\frac{W}{w_i} - 1\right)(2Qw_i)\right) \tag{4.8}$$

where *n* is the total number of active flows, *W* is the sum of the throughput weights of all active flows and *r* is the transmission rate of the output link.

*Proof:* Since the latency of an *LR* server can be estimated based on its behavior in the flow active period, the theorem was proved by showing that,

$$\Theta i \leq \frac{1}{r}\left((n-1)M + \left(\frac{W}{w_i} - 1\right)(2Qw_i)\right)$$

Let flow *i* become active at time instant $\tau_i$. For deriving an upper bound on the latency of D$^2$FQ, a time interval ($\tau_i$, *t*) is considered during which flow *i* is continuously active. Then we obtain the lower bound on the total service received by flow *i* during the time interval under consideration. the lower bound is expressed lately in the form of Equation (3.10) to derive the latency bound.

The active periods ($\tau_i$, *t*) is consider which satisfy the following requirements:

1. $\tau_i$ coincides with the start of service opportunity of some flow..

2. Time instant $t$ belongs to a subset of all possible time instants at which the scheduler begins serving flow $i$.

Let $\tau_i(r,s)$ be the time instant marking the start of service of flow $i$ when flow $i$ is at the head queue of delay-slot $DS_s$ in globe round $r$. In the other words, this time instant represents the start of the service opportunity of flow $i$ when its flow-index is at the delay-slot $DS(r,s)$. Note that $\tau_i(r,s)$ belongs to the set of time instants when the scheduler begins serving flow $i$. Therefore, in order to determine the latency bound of the D$^2$FQ, time intervals $(\tau_i,\ \tau_i(r,s))$ is considered for all $(r,\ s)$ in which flow $i$ receives service.

Firstly, the latency bound to divide suitable time interval $(\tau_i,\ \tau_i(r,s))$ is analyzed such that the length of time interval is maximized. Note that, the time instant $\tau_i$, may or may not coincide with the start of new global round. Let $k_0$ be the global round which is in progress at time instant $\tau_i$ or which starts exactly at time instant $\tau_i$,. In either case, flow $i$ will receive an opportunity to transmit $Q_i$ worth of data the $k_0$-th flow round $k_0$. Let the time instant $t_h$ mark the start of delay-slot $ES_i$ of global round $k_0$, i.e., $(\tau_i,\ \tau_i(r,s))>t_0$. In the case the time interval $(t_0,\ \tau_i,)$ will be excluded from the time interval $(\tau_i,\ \tau_i(r,s))$ is maximal. Therefore the $\tau_i$, coincides is assumed with the start of the $k_0$-th flow round. Figure 4.1 illustrates the time interval under consideration assuming that $(r,s)$ is equal to $(k_0+k,\ v)$.

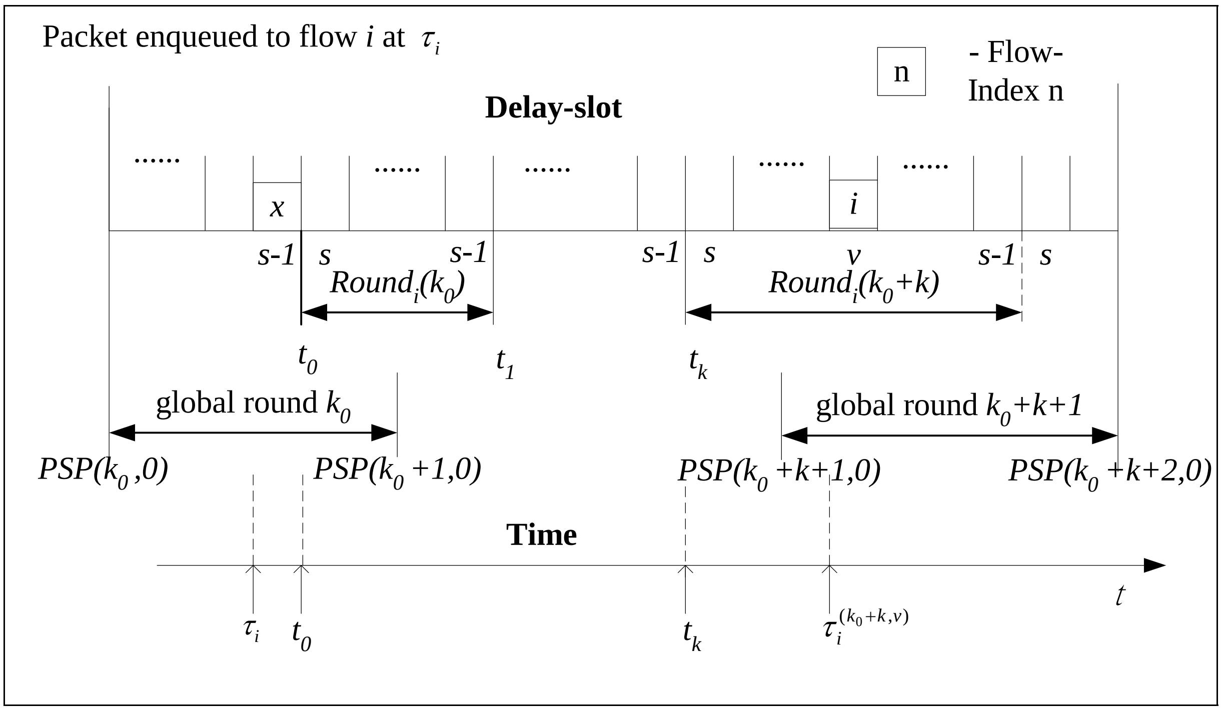


Figure 4-2 Three sub-interval for latency analysis

The time interval *($\tau_i$, $\tau_i(r,s)$)* can be split into three sub-interval by above consideration, that is *($\tau_i$, $t_0$), ($t_0$, $t_k$) and ($t_k$, $\tau_i^{(k_0+k,v)}$)*

1. *($\tau_i$, $t_0$)*: This sub-interval includes the part of the service at the slots, the time interval (*PSP($k_0$, s-1), PSP($k_0$, s-1)*) coincides the start of service of flow *i* which be active at the time instance $\tau_i$, *thus*

$$PSP(k_0, s-1) \le \tau_i < PSP(k_0, s)$$

Then consider the service in *(ER-1)*th slot. note that slot *s* is the earning-slot of flow *i*.

$$t_0 - \tau_i \le \frac{1}{r} \sum_{\substack{j=1 \\ j \ne i}}^{n} \left\{ Served_j\left(k_0, ER_i - 1\right) \right\} \qquad (4.9)$$

Thus,

$$t_0 - \tau_i \le \frac{1}{r}\sum_{\substack{j=1\\ j\ne i}}^{n}\left\{DC_j\left(k_0, ES_i - 1\right) - DC_j\left(k_0, ES_i\right)\right\} \tag{4.10}$$

2. *(t₀, tₖ)*: This sub-interval includes $k$ flow rounds of execution of the D²FQ scheduler start at flow round $k_0$. Consider the time interval *($t_h$, $t_h$+1)* when flow round *($k_0$ +h)* is in progress. Then starting slot of flow round is *($k_0$,$ER_i$)* and the last slot the flow round is *($k_0$+h+1, $ER_i$)*. The equation of *n* flow is following,

$$t_{h+1} - t_h = \frac{1}{r}\sum_{j=1}^{n}\left\{\sum_{s=ER_i}^{L-1} Served_j\left(h, s\right) + \sum_{s=0}^{ER_i-1} Served_j\left(h+1, s\right)\right\} \tag{4.11}$$

By Equation (3.10)

$$t_{h+1} - t_h = \frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0 + h\ , ES_j\right) + w_j Q - DC_j\left(k_0 + h + 1, ES_j\right)\right\}$$

$$t_{h+1} - t_h = \frac{WQ}{r} + \frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0 + h, ES_j\right) - DC_j\left(k_0 + h + 1, ES_j\right)\right\} \tag{4.12}$$

After summing the above *k* rounds,

$$t_k - t_0 = \frac{WQ}{r}k + \frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0, ES_j\right) - DC_j\left(k_0 + k, ES_j\right)\right\} \tag{4.13}$$

3. *($t_k$, $\tau_i^{(k_0+k,v)}$):* This sub-interval includes the part of the *($k_0$+ k)*th round prior to the start of the service of flow when it is at the head delay-slot *$DS_v$* . In the worst case, flow *i* will be the last flow to receive service among all the flows which may be present in. In this case, during the sub-interval under consideration, the service received by flow *i* equal *$Sent_i$($k_0$+k, v-l)*

The following two cases is considered,

Case 1 $\tau_i^{(k_0+k,v)}$ in *(k*$_0$*+k )* th global round:

$$\tau_i^{(k_0+k,v)} - t_k = \frac{1}{r}\sum_{j=1}^{n}\left\{\sum_{s=ER_i}^{v} Served_j\left(k_0+k,s\right)\right\} \tag{4.14}$$

Case 2 $\tau_i^{(k_0+k,v)}$ in *(k*$_0$*+k+1)*th global round:

$$\begin{aligned}\tau_i^{(k_0+k,v)} - t_k &= \frac{1}{r}\sum_{j=1}^{n}\left\{\sum_{s=ER_i}^{L-1} Served_j\left(k_0+k,s\right) + \sum_{s=0}^{v} Served_j\left(k_0+k+1,s\right)\right\}\\ &\le \frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0+k,ES_i\right)+w_jQ-DC_j\left(k_0+k+1,v\right)\right\}\\ &\le \frac{WQ}{r}+\frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0+k,ES_i\right)-DC_j\left(k_0+k+1,v\right)\right\}\end{aligned} \tag{4.15}$$

After combining three sub-time intervals, i.e. (4.10), (4.13) and (4.15), it becomes:

$$\tau_i^{(k_0+k,v)} - \tau_i \le (k+1)\frac{WQ}{r}+\frac{1}{r}\sum_{j=1}^{n}\left\{DC_j\left(k_0,ES_i-1\right)-DC_j\left(k_0+k+1,v\right)\right\} \tag{4.16}$$

Since flow *i* becomes active at $\tau_i$, its deficit count at time instance *PSP(k*$_0$*, ER*$_i$ *)* is zero, i.e. $DC_i\left(k_0,ES_i-1\right)=0$, and using equation (4.1) substitute into equation (4.16)

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i \le (k+1)\frac{WQ}{r}&+\frac{\left(W-w_i\right)Q+\left(n-1\right)M}{r}\\ &-\frac{DC_i\left(k_0+k+1,v\right)}{r}\end{aligned} \tag{4.17}$$

Solving *k*

$$(k+1) \ge \frac{r\left(\tau_i^{(k_0+k,v)}-\tau_i\right)-(n-1)M-(W-w_i)Q+DC_i\left(k_0+k+1,v\right)}{WQ} \tag{4.18}$$

Over *k* rounds of servicing of flow *i*, the total data transmitted can be expressed as the following summation:

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)=Sent\left(\tau_i,t_0\right)+Sent\left(t_0,t_k\right)+Sent\left(t_k,\tau_i^{(k_0+k,v)}\right) \qquad (4.19)$$

There is no packet dequeue at time interval *(τ$_i$, t$_0$)* for flow *i*, i.e. $Sent\left(\tau_i,t_0\right)=0$

And flow *i* consume all quantum at time interval *(t$_0$, t$_k$)*, i.e. $Sent\left(t_0,t_k\right)=(k+1)w_iQ$,

thus

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)=(k+1)w_iQ-\left(DC_j\left(k_0+k+1,v\right)\right) \qquad (4.20)$$

Using Equation (4.18) to substitute for *(k+1)* in equation (4.20)

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)\geq\frac{r\left(\tau_i^{(k_0+k,v)}-\tau_i\right)-(n-1)M-(W-w_i)Q+DC_i\left(k_0+k+1,v\right)}{WQ}w_iQ$$
$$-\left(DC_j\left(k_0+k+1,v\right)\right)$$

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)\geq\frac{w_i r\left(\tau_i^{(k_0+k,v)}-\tau_i\right)}{W}-\frac{w_i(n-1)M}{W}$$
$$-\left(1-\frac{w_i}{W}\right)Qw_i-\left(1-\frac{w_i}{W}\right)DC_j\left(k_0+k+1,v\right)$$

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)\geq\rho_i\left(\tau_i^{(k_0+k,v)}-\tau_i\right)-\rho_i\frac{(n-1)M}{r}-\frac{W}{rw_i}\left(1-\frac{w_i}{W}\right)\left(Qw_i+DC_j\left(k_0+k+1,v\right)\right)$$

$$Sent\left(\tau_i,\tau_i^{(k_0+k,v)}\right)\geq\rho_i\left(\tau_i^{(k_0+k,v)}-\tau_i\right)-\rho_i\frac{(n-1)M}{r}-\rho_i\frac{1}{r}\left(\frac{W}{w_i}-1\right)\left(Qw_i+DC_j\left(k_0+k+1,v\right)\right)$$

Compare the above equation with Equation A-1 , the latency bound of D$^2$FQ scheduler is given by

$$\Theta i\leq\frac{(n-1)M}{r}+\frac{1}{r}\left(\frac{W}{w_i}-1\right)\left(Qw_i+DC_j\left(k_0+k+1,v\right)\right)$$

$$\Theta i \leq \frac{1}{r}\left( (n-1)M + \left( \frac{W}{w_i} - 1 \right)(2Qw_i) \right)$$

and the statement of theorem is proved. •

## 4.3 Work Complexity

The work complexity of a scheduling discipline is defined as follows

**Definition 4.3.1** Consider an execution of a scheduling discipline over n flows. We define the work complexity of the scheduler as the order of the time complexity, with respect to n, of enqueueing and then dequeueing a packet for transmission

**Theorem 4.3.1** The worst-cast work complexity of D$^2$FQ scheduler is O(log *L*) where L is the length of delay-slot.

*Proof:* The time complexity of enqueueing a packet is the same as the time complexity of the Enqueue routine in Figure 3.6, which is executed whenever a new packet arrives at a flow. Identifying the flow at which the packet arrives is an O(1) operation. Then the packet will be push to the flow buffer. If the flow is in an idle state, then the deficit count, *DC*, would be reset to zero. The state of flow will be marked as Active and the earning-slot will be set as current location of Polling-Slot-Pointer. If the state of the flow is semi-idle state, it only needs to push the packet in flow buffer and mark the flow as active state. If the flow is in the active state, then D$^2$FQ will do nothing after pushing the packet into the buffer. If the flow is in an idle state or a semi-state before the flow is reached, both states need to push the flow-index into the delay-slot.

Before a flow-index is hopping to a new slot, some procedure shown in Figure 3.10

need to be executed firstly. If the deficit count is not negative and the flow buffer do not have any backlogged packet, the flow will be marked as semi-idle state. This checking procedure is O(1). Of course, the above checking procedure is useless for the enqueuing procedure due to the flow's buffer must contain packet. This procedure, however, is also same for dequeuing procedure. And then the calculation of the destination slot is also O(1) that is the offset value from the current position of Polling Slot Pointer. When a flow-index is push to a slot, the flow-index will queue at the end of the slot. Actually, there are link-lists to maintain the queue flow-index, (not packet) in each slot. The push and pull operation of link-list is O(1) operation. Then $D^2FQ$ will check the flow-index hopping whether across the flow's earning-slot and the checking procedure is also O(1). The deficit count would be update when the flow-index hops across its earning-slot. The update process of DC is the sum of the current DC value and the weighted Quantum value. The overall complexity of enqueuing process is O(1).

Let us now consider the time complexity of de-queuing packet. $D^2FQ$ server will enquire the current slot, which is indicated by PSP. If there is no flow-index in that slot, then server will need to search the next non-empty slot. The searching operation for a element $L$ is O(log $L$). After finding a non-empty delay-slot, a head flow-index of the slot will be pull out. This pull out operation work in link-list operation with O(1). Then the flow-index indicated flow can serve a packet. The packet will push to the outgoing link and the packet length is subtracted from the deficit counter. After de-queuing packet, the flow-index will hop to the delay-slot again in O(1) which has been mentioned at the enqueuing procedure. Due to the searching of non-empty slot is O(log $L$), the overall complexity of de-queuing procedure is O(log $L$).

Note that the complexity of $D^2FQ$ is not depended on number of flow but the length of delay-slot. The length of delay-slot is fixed when the server is initialized. The complexity of the searching algorithm can be transfered from time domain to space domain. Due to the constant length of the delay-slot and the limited size, it is very easy to use hardware to implement the searching algorithm of non-empty slot. On the other hand, the number of flows competing for a link may be the order of tens of thousands of flows in core routers, n >> *L*. Thus, delay-slot is always occupied by flow-indexes and the next slot of *PSP* is almost non-empty and the work complexity of the $D^2FQ$ scheduler is always lower than its worst cast complexity, i.e. O(log *L*).

# Chapter 5. Simulation Study on Single Node Topology

The performance study of D$^2$FQ scheduling algorithm is based on the simulation by the Network Simulator 2 [30]. This simulation has used two different network topologies to verify the characteristic of D$^2$FQ. The two topologies are single node topology and multi-node topology.

Single node topology used in the analysis of delay and throughput of connectionless communication such as UDP. Connectionless communication does not have flow control, since connectionless services typically operate on a better effort strategy without any notion of congestion detection and control. Therefore, many DDoS used the connectionless flows to attack the network.

Multi-node topology used to study an end-to-end performance of a real world network. This study focus on the overall throughput and the end-to-end performance of a flow, thus the connection-oriented communication is major analysis target. The simulation study of multi-node topology is presented in Chapter 6.

## 5.1 Single-node Topology

In this chapter, single-node topology is used to study the performance of the different scheduling algorithms in a network node. Figure 5.1 is the single-node topology that is using 80 sources and 80 sinks and packets departed from the sources to the sinks through a single link, h0 to h1. This shared bandwidth of the link was 10Mb/s among the 80 flows. The traffic model had constant bit rate,CBR, with 500-byte packet size.

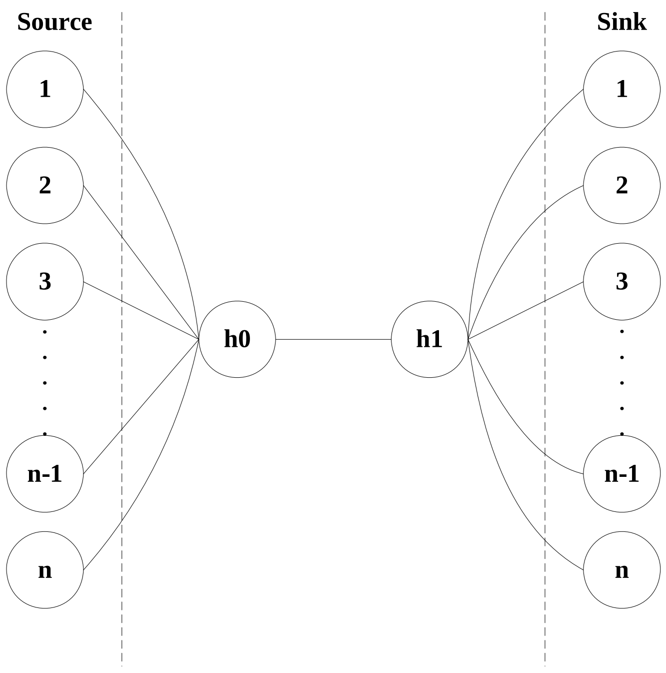


Figure 5-1 Simulation Network Topology

There are four simulation conditions used this topology to evaluate the performance of D$^2$FQ and compare with other scheduling algorithms such as DRR, WFQ. WFQ is the packetized version of Generalized Processor Sharing, where GPS provide prefect fairness in throughput if the traffic is infinitely divisible. DRR provide acceptable fairness in throughput with complexity O(1). This two scheduling algorithms are suited to fairness analysis on single node topology network. The aim of the evaluation includes the fairness between bandwidth allocation and differentiability of mean delay under different loading condition.

## 5.2 Bandwidth Allocation.

In this simulation, the performance of bandwidth allocation of D$^2$FQ is compared with that of weighted DRR and WFQ. The following shows the configuration of this simulation

- The 80 flows are divided into two classes, first class and second class, and each class have 40 flows. Each class is further divided into four groups. i.e. 1st group to 4th group for first class and 5th group to 8th group for second class. Each group has 10 flows.
- The flows in first class will get constant throughput and consider as a conforming traffic.
- The flows in second class will get varied throughput and is considered as non-conforming traffic.
- The throughput weight of 1st group to 4th group are 1, 2, 3 and 4 respectively and the throughput weight of 5th group to 8th group are 1, 2, 3 and 4 respectively
- The total throughput weight is (1+2+3+4)*20 = 200. Let 90% of offering load is the acceptable loading in general. Each throughput weight in 10Mbps channel would be (10M* 0.9 /200) = 45kbps and 45kbps is also set as reference level for throughput normalization.
- The arrival rate of first class traffic is according to the throughput weight of their flow. i.e. flow in 1st group is 45kbps and flow in 4th group is 180kbps
- The arrival rate of second class traffic is according to the throughput weight of their flow and arrival rate factor, $\lambda$ . i.e. the arrival rate from 5th group to 8th group are $\lambda$, $2\lambda$, $3\lambda$, $4\lambda$ respectively.
- The total arrival rate is $(45 +\lambda)* (1+2+3+4) *10 = 4.5\text{Mbps} + 100\lambda$, and $\lambda$ will

produce different loading condition.

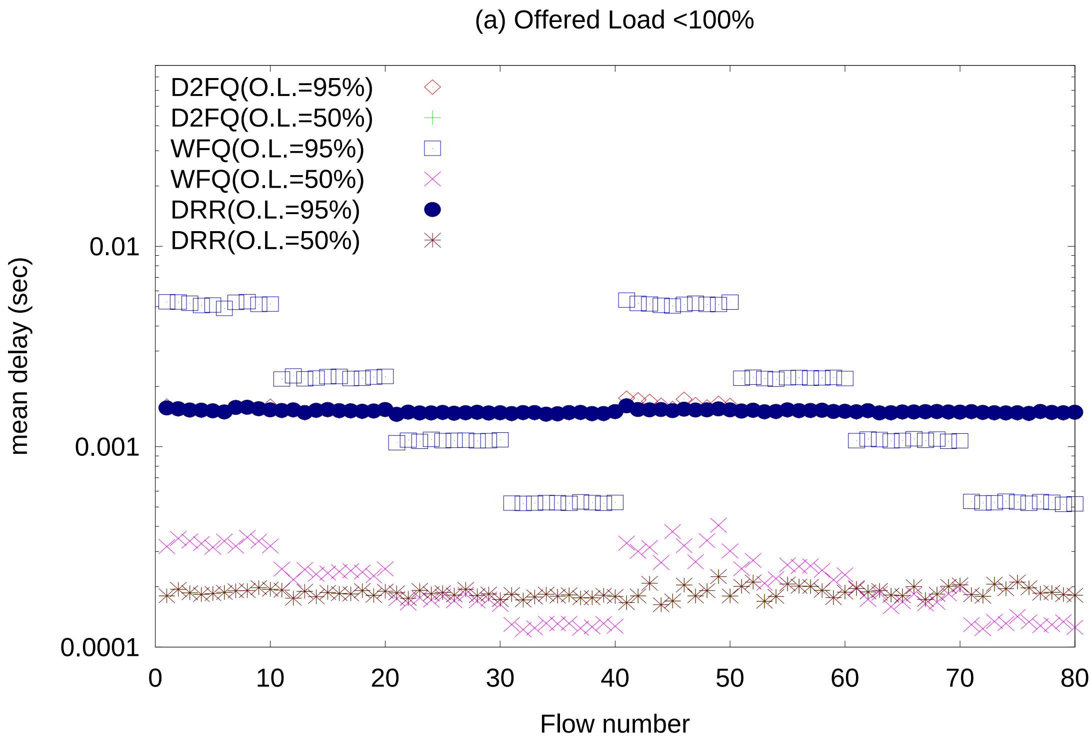
- In the simulation, the λ would set from 5kbps to 105kbps and thus the offered load is varied from 50% to 150% respectively.
- $D^2$FQ, DRR and WFQ algorithm are used in the node, h0, in this simulation.
- The simulation time is set to 100 seconds.

According to the simulation result, $D^2$FQ, DRR and WFQ schedulers provide the same overall throughput for all conforming flows, 1st to 40th flow, under any offered load. It implied that bandwidth allocation of $D^2$FQ scheduler is as same as the bandwidth allocation of WFQ and DRR schedulers that the bandwidth allocations among conforming flows have not been influenced under any offered load. For the non-conforming flows, 41st to 80th flow, no packet drop occurs when the offered load is below 100% and thus the overall throughput of non-conforming flows are the same in $D^2$FQ, WFQ and DRR schedulers. When the offered load is above 100%, $D^2$FQ, WFQ and DRR schedulers are started to drop packet from non-conforming flow. However, the result of the overall throughput among three schedulers is almost same and only has 0.1% difference. Three schedulers could also allocate the bandwidth to conforming flow and non-conforming flow and the allocated bandwidth of non-conforming is complied by the max-min fair share policy [24].

Although the performance of throughput in this simulation is same among $D^2$FQ, WFQ and DRR scheduler, the performance in term of queueing delay is very different. Figure 5-2 is the mean delay of all flows under different offered load. When the offered load is below 100%, the mean delay of all flows in $D^2$FQ and DRR scheduler are almost keep in the same range. The throughput weight does not incur variation on the mean delay of the flow. When the offered load is closed to 100 or excess 100%, the non-conforming flow's packet in $D^2FQ$ scheduler cannot be de-queued within a round time. Simultaneously, other incoming packets accumulate flow's buffer rapidly and $D^2FQ$ scheduler starts to drop packet to avoid buffer overflow. This packet drop process only applies on non-conforming flow and conforming flow can receive a relatively low delay.

For DRR scheduler, the received delay is same among all flows regardless conforming flow or non-conforming flow. The mean delay of DRR scheduler depends on the offered load of the scheduler and DoS could easily increase the queuing delay of other conforming flow.

According to the figure, the mean delay of all flows in WFQ depends on the flow's throughput weight regardless offered load condition. The high throughput weight 's flow can be served faster and induces a low mean delay. When the offered load is excess 100%, the non-conforming flow also influences the mean delay of conforming flow in WFQ scheduler. Table 4 is the normalized delay, which is referred to the average delay at 90% loading. Although the mean delay of conforming flow in WFQ scheduler is lower than non-conforming, the mean delay in WFQ scheduler is still longer than the mean delay of conforming flow in $D^2FQ$ scheduler. During the offered load is 150%, the range of conforming flow 's mean delay in $D^2FQ$ scheduler is varied from 2.8 times to 3 times and the range of conforming flow 's mean delay in $D^2FQ$ scheduler is varied from 5.5 times to 6.4 times. The variation of mean delay in WFQ scheduler is much larger than that in $D^2FQ$. Therefore, WFQ has no way to provide a Low-Delay Low-Throughput service because of the coupling effect of throughput and delay.



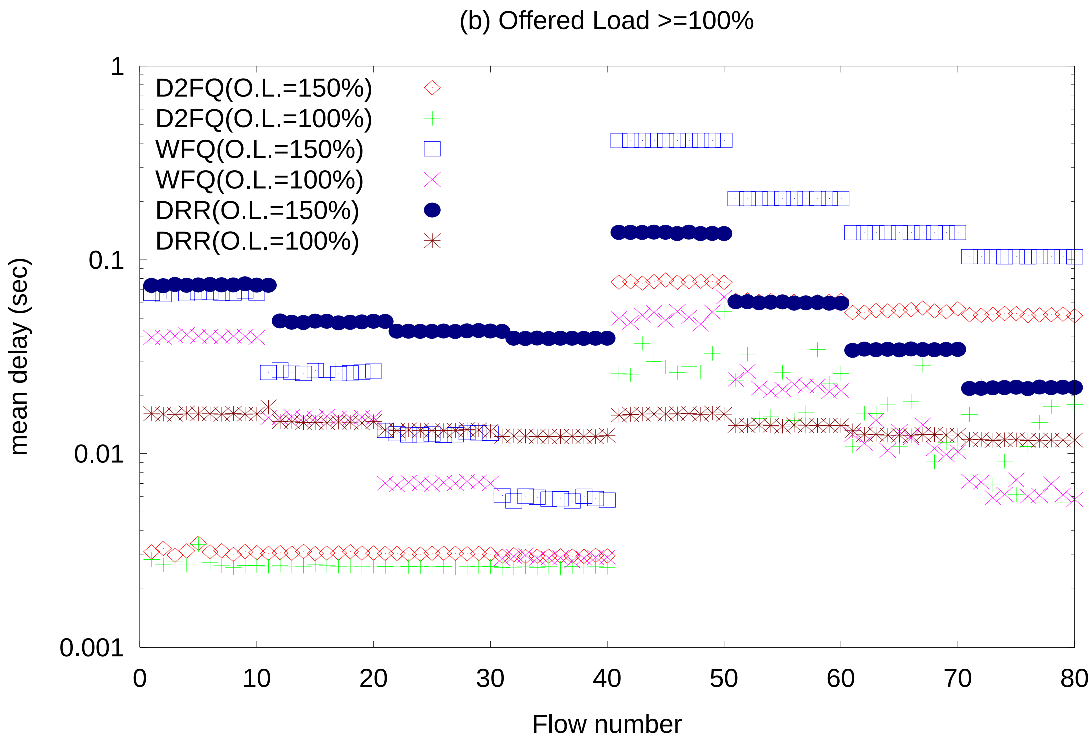


Figure 5-2 Mean delay of flows with different throughput weight. (a) Offered Load < 100%, (b) Offered Load >= 100%)

| **Algorithm Type** | **Throughput weight** | **Offered load (%)** | | | | |
|---|---|---|---|---|---|---|
| | | **50%** | **90%** | **95%** | **100%** | **150%** |
| $D^2$FQ conforming flow | 1 | 100.06 | 99.89 | 99.68 | 99.40 | 99.64 |
| | 2 | 99.92 | 100.04 | 99.68 | 99.84 | 100.00 |
| | 3 | 99.90 | 99.86 | 100.10 | 99.91 | 99.81 |
| | 4 | 99.85 | 99.96 | 99.80 | 99.96 | 99.96 |
| $D^2$FQ non-conforming flow | 1 | 11.08 | 99.66 | 111.42 | 121.47 | 122.22 |
| | 2 | 10.99 | 99.68 | 111.08 | 122.00 | 122.22 |
| | 3 | 11.13 | 99.64 | 110.68 | 121.87 | 122.22 |
| | 4 | 11.09 | 99.91 | 111.00 | 122.09 | 122.22 |
| WFQ conforming flow | 1 | 100.06 | 99.89 | 99.68 | 99.39 | 99.56 |
| | 2 | 99.92 | 100.04 | 99.68 | 99.83 | 99.99 |
| | 3 | 99.90 | 99.86 | 100.10 | 99.91 | 99.80 |
| | 4 | 99.85 | 99.96 | 99.80 | 99.96 | 99.96 |
| WFQ non-conforming flow | 5 | 11.08 | 99.66 | 111.42 | 121.87 | 122.28 |
| | 6 | 10.99 | 99.68 | 111.08 | 122.09 | 122.24 |
| | 7 | 11.13 | 99.63 | 110.68 | 121.93 | 122.24 |
| | 8 | 11.09 | 99.91 | 111.00 | 122.10 | 122.22 |
| DRR conforming flow | 1 | 100.06 | 99.89 | 99.68 | 99.40 | 99.64 |
| | 2 | 99.92 | 100.04 | 99.68 | 99.83 | 100.00 |
| | 3 | 99.90 | 99.86 | 100.10 | 99.90 | 99.81 |
| | 4 | 99.85 | 99.96 | 99.80 | 99.95 | 99.96 |
| DRR non-conforming flow | 5 | 11.08 | 99.66 | 111.42 | 121.92 | 122.22 |
| | 6 | 10.99 | 99.68 | 111.08 | 122.10 | 122.22 |
| | 7 | 11.13 | 99.64 | 110.68 | 121.94 | 122.22 |
| | 8 | 11.09 | 99.91 | 111.00 | 122.10 | 122.22 |

Table 2 Normalized throughput of flow with different throughput weight.

| **Algorithm Type** | **Throughput weight** | **Offered load (%)** | | | | |
|---|---|---|---|---|---|---|
| | | **50%** | **90%** | **95%** | **100%** | **150%** |
| D2FQ conforming flow | 1 | 17.91 | 102.54 | 146.07 | 260.92 | 297.26 |
| | 2 | 17.57 | 100.20 | 142.57 | 249.17 | 291.01 |
| | 3 | 17.41 | 97.78 | 139.01 | 246.66 | 288.11 |
| | 4 | 16.96 | 97.89 | 138.60 | 245.56 | 280.49 |
| D2FQ non-conforming flow | 1 | 17.68 | 104.01 | 156.04 | 2969.36 | 7289.59 |
| | 2 | 18.23 | 101.21 | 142.37 | 2158.57 | 5787.85 |
| | 3 | 17.99 | 98.92 | 140.48 | 1421.18 | 5177.01 |
| | 4 | 18.08 | 97.44 | 139.52 | 1199.50 | 4933.16 |
| WFQ conforming flow | 1 | 31.38 | 317.87 | 488.98 | 3805.60 | 6396.96 |
| | 2 | 22.34 | 144.97 | 209.63 | 1451.19 | 2497.93 |
| | 3 | 16.36 | 74.41 | 101.54 | 666.19 | 1199.23 |
| | 4 | 12.11 | 39.47 | 49.65 | 274.42 | 555.95 |
| WFQ non-conforming flow | 1 | 30.49 | 317.65 | 489.51 | 4929.70 | 39093.40 |
| | 2 | 22.68 | 146.01 | 208.37 | 2123.83 | 19596.68 |
| | 3 | 16.84 | 74.93 | 102.15 | 1125.37 | 13072.58 |
| | 4 | 12.41 | 39.53 | 49.80 | 612.64 | 9810.95 |
| DRR conforming flow | 1 | 17.91 | 102.54 | 145.48 | 1516.71 | 7020.08 |
| | 2 | 17.57 | 100.20 | 143.25 | 1400.75 | 4776.21 |
| | 3 | 17.41 | 97.78 | 139.63 | 1247.62 | 4104.62 |
| | 4 | 16.96 | 97.89 | 139.31 | 1162.58 | 3721.98 |
| DRR non-conforming flow | 1 | 17.68 | 104.01 | 145.38 | 1513.33 | 12098.17 |
| | 2 | 18.23 | 101.21 | 142.87 | 1322.36 | 5685.56 |
| | 3 | 17.99 | 98.92 | 141.12 | 1187.63 | 3256.51 |
| | 4 | 18.08 | 97.44 | 140.23 | 1112.11 | 2068.14 |

Table 3 Normalized delay of flow with different throughput weight

## 5.3 Differentiability of Delay

This simulation shows delay differentiability of $D^2FQ$ in different loading condition. This simulation topology is same to Figure 5.1 that used 80 sources and 80 sinks and packets departed from the sources to the sinks through a single link,. The following is the other configuration for simulating environment.

- The 80 flows divide into two classes, 1st class and 2nd class, and each class has 40 flows. In the first class, the flows further divide into eight groups, i.e. 1st group to 8th group. In the second class, the flows also divided into eight groups, i.e. 9th group to 16th group. Therefore, each group has five flows.
- The flows in first class would get constant throughput, 100kbps, and consider as conforming traffic.
- The flows in second class would get different arrival rate, $\lambda$, and consider as non-conforming traffic.
- The delay weight of 1st group to 8th group are 1, 2, 4, $8, \cdots, 64$ and 128 respectively and the delay weight of 9th group to 16th group are 1, 2, 4, $8, \cdots, 64$ and 128 respectively.
- The throughput weight of all flows is same to 1 and the total throughput weight is 80. Let the normal offering load is 80%. Each throughput weight in 10Mbps channel would be shared 100kbps.
- The total arrival rate is $(100k+\lambda)* 40 = 4.0\text{Mbps} + 40\lambda$, and $\lambda$ would result different loading condition.
- In the simulation, the $\lambda$ would set from 50kbps to 200kbps to result 60% to

120% offered load.

- This simulation time is set to 600 seconds for each $\lambda$ value.

In Figure 5-3, as the loading increase from 0.6 to 1.2, the mean delay of conforming flows in first class increased slightly. Moreover, the mean delay of conforming first class flows was differentiable relative to its delay weight when the loading were increasing.

On the other hand, the distribution of mean delay of second class depends on the arrival rate. When $\lambda$ = 0.1Mbps and the loading was 0.8, the mean delay of second class was the same as that of first class. When arrival rate, $\lambda$, of flows continued to increase, the flow in second class became non-conforming flows. The mean delay started to increase exponentially when the link was nearly saturated. According to the figure, the mean delay of second class did not maintain the delay differentiability when the link was overloaded. In addition, the mean delay was slightly reduced due to the drop of front packet of the non-conforming flow.

According to Figure 5-4, non-conforming flows of lower delay weight, i.e. *dw*=1, in D$^2$FQ scheduler was punished starting from loading larger than 0.85, whereas those of higher delay weight group, i.e. *dw*=128, was punished starting from loading larger than 0.95. A longer delay was the penalty for both groups of non-conforming flows. Moreover, there was an interesting phenomenon in the first class. During increasing the load, the delay of conforming class was increasing slightly but there was a suddenly drop when the link was marginally saturated. The reason is due to the D$^2$FQ scheduler starts to drop the front packets of non-conforming flows and the conforming flows can reduce the queuing time relatively.

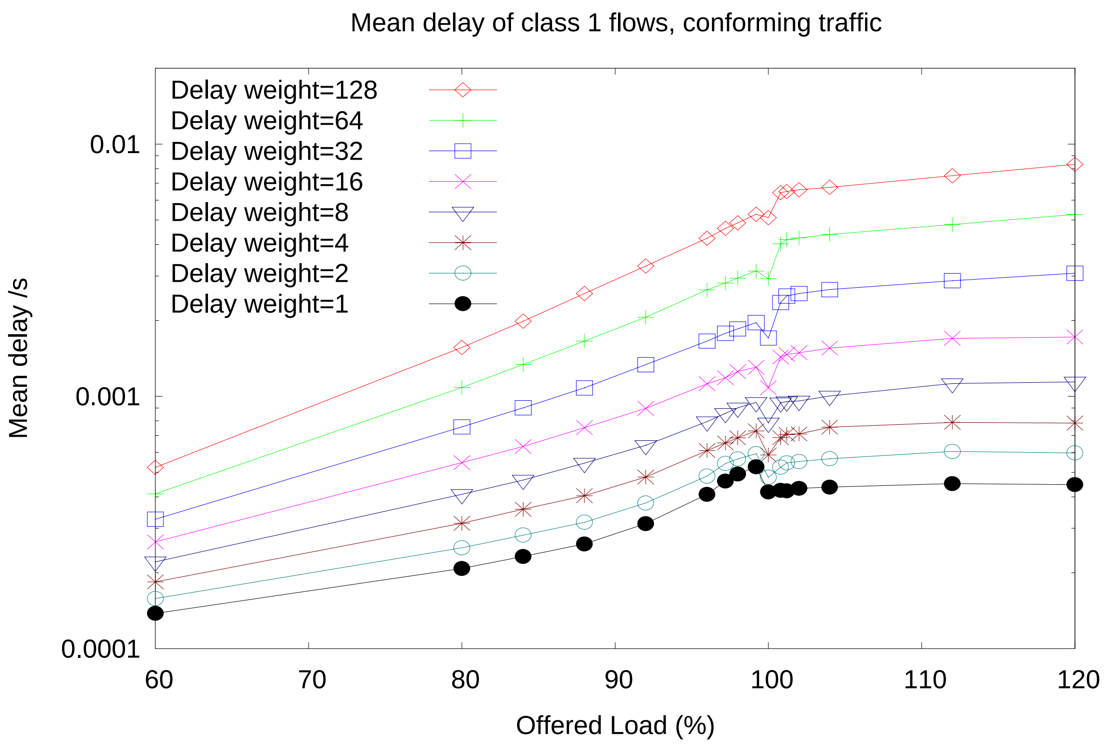


Figure 5-3 Mean delay of conforming traffic with different delay weights

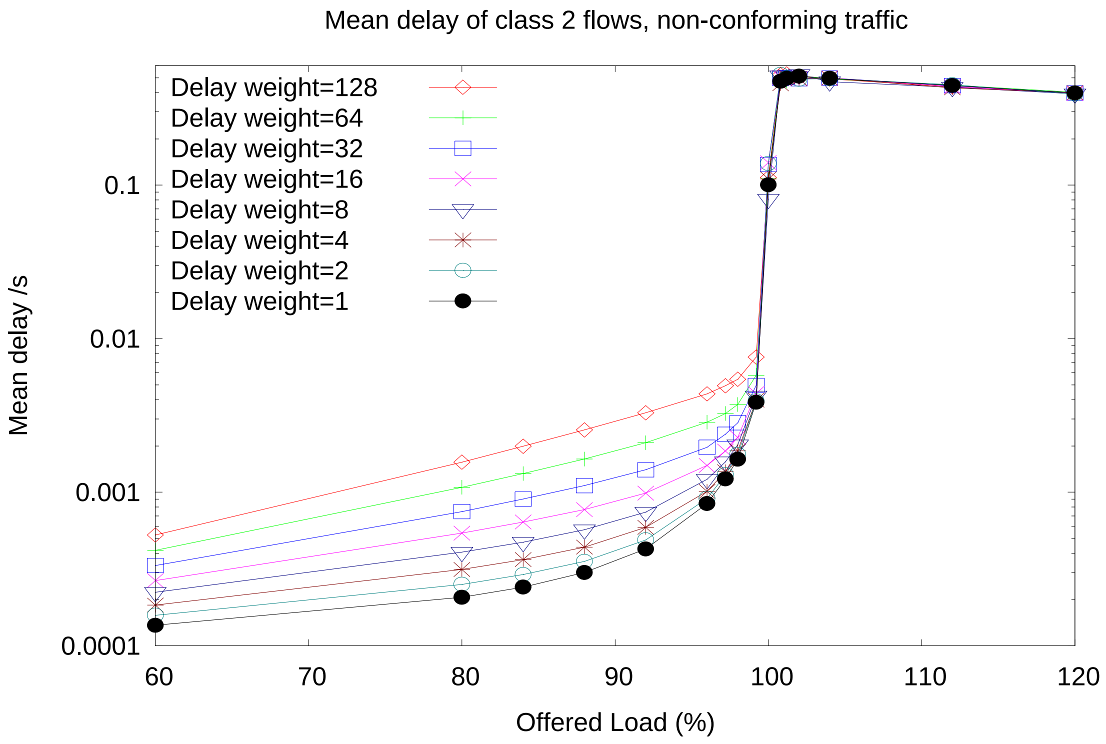


Figure 5-4 Mean delay of non-conforming traffic with different delay weights

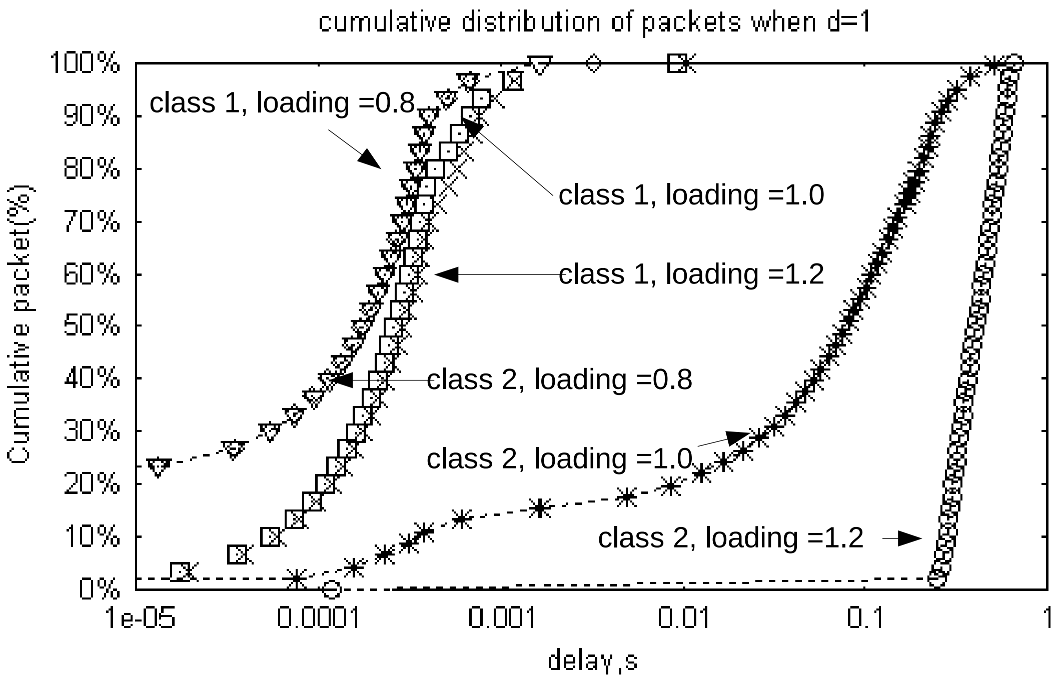


Figure 5-5 Cumulative distribution of packets when delay weight =1

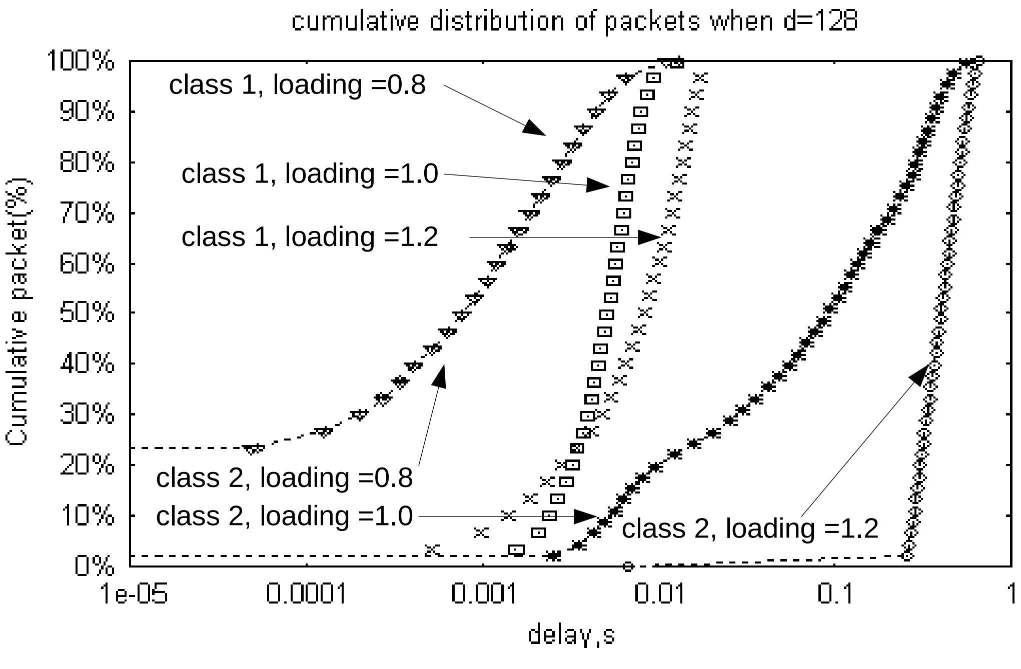


Figure 5-6 Cumulative distribution of packets when delay weight =128

We now evaluate the cumulative delay distribution of packets in the same simulation result. Figure 5-5 and Figure 5-6 show the cumulative delay distribution of two flows with *dw*=1 and *dw*=128 respectively. In each flow of first class traffic, the range of delay bound of 95% packets was small in different loading, especially *dw*=1. The delay bound of 95% packets was also differentiable to different delay weight. From the Figure 5.4, we can see that the worst case of delay has been bounded. The overall delay of conforming flows was bounded roughly by a round time of delay-slots and the delay of non-conforming flows was bounded by flow buffer length and their throughput weight.

## 5.4 Conforming Flow Compete with Non-Conforming Flow

When some of non-conforming traffic congest network such as DoS, almost all of the flows would induced longer delay time. A good scheduler should be able to protect the conforming traffic to keep the delay of conforming in the guaranteed level. This simulation would show the delay of conforming flow in different share of non-conforming traffic under an overloaded condition.

- The 80 flows divide into two classes, first class and second class. 1st class have *x* flows and 2nd class have (80-*x*) flows.
- The flows in first class would get constant throughput, 100kbps, and consider as a conforming traffic.
- The flows in second class would get different arrival rate, λ, and consider as non-conforming traffic.
- The delay weight and throughput weight among all flows is equal.

- The overall offered load in the simulation is keep to 120%.
- The total arrival rate is, 100kbps *$x$ + (80-$x$)* λ = 12Mbps, for when ,$0 < x < 80$..
- When $x = 80$, it means all flows are conforming flow and the offered loading is 80%. The received mean delay would be the reference value in simulation.

In this simulation, DRR, WFQ and D$^2$FQ is used to verify the output performance. In D$^2$FQ, two different delay weights setting is also used in the simulation. One sets the delay weight to 2 and one sets the delay weight to 16. Therefore, four simulation settings were used in this simulation.

The result of simulation shows that all conforming flows have no packet drop and hence the outgoing throughput of the conforming flows is same for four simulation settings. In this simulation, the outgoing throughput of non-conforming flows is not concern.

Figure 5-7 shows the mean delay of conforming flows in different percentage of conforming flows. The result shows that when all traffic is conforming flow, ie conforming ratio is 100%, four scheduler settings have the same mean delay (about 0.7ms) which is consistent with the conservation law. The mean delay, 0.7 ms, is used as the reference level and show the normalized delay of the Figure 5-8. The figure shows that when the network only existed 90% conforming flow, the delay of conforming flows of WFQ and DRR scheduler increased about 10 times. That implied 10% of non-conforming traffic could extend the delay of conforming flow. When the congested node is using D$^2$FQ scheduler, the delay of the conforming flow will only increased about 2 times which is much less than the flow delay of DRR and

WFQ scheduler. Moreover, the mean delay of flows in DRR and WFQ schedulers are gradually increasing during the percentage of conforming flow decreasing. The D$^2$FQ scheduler could keep the mean delay of conforming flow in an acceptable range and the worst case is about 300% of the reference delay. In addition, there is no obvious trend of the mean delay by increasing share of non-conforming. Even there is 90% of serving time occupied by non-conforming flow, the received mean delay of conforming flow is extended to 120% of reference delay for the flow with delay weight 2 and 160% of reference delay for the flow with delay weight 16. The received mean delay is independent on the share of conforming flow. D$^2$FQ scheduler can maintain a lower mean delay for conforming flow regardless the share of non-conforming flow.

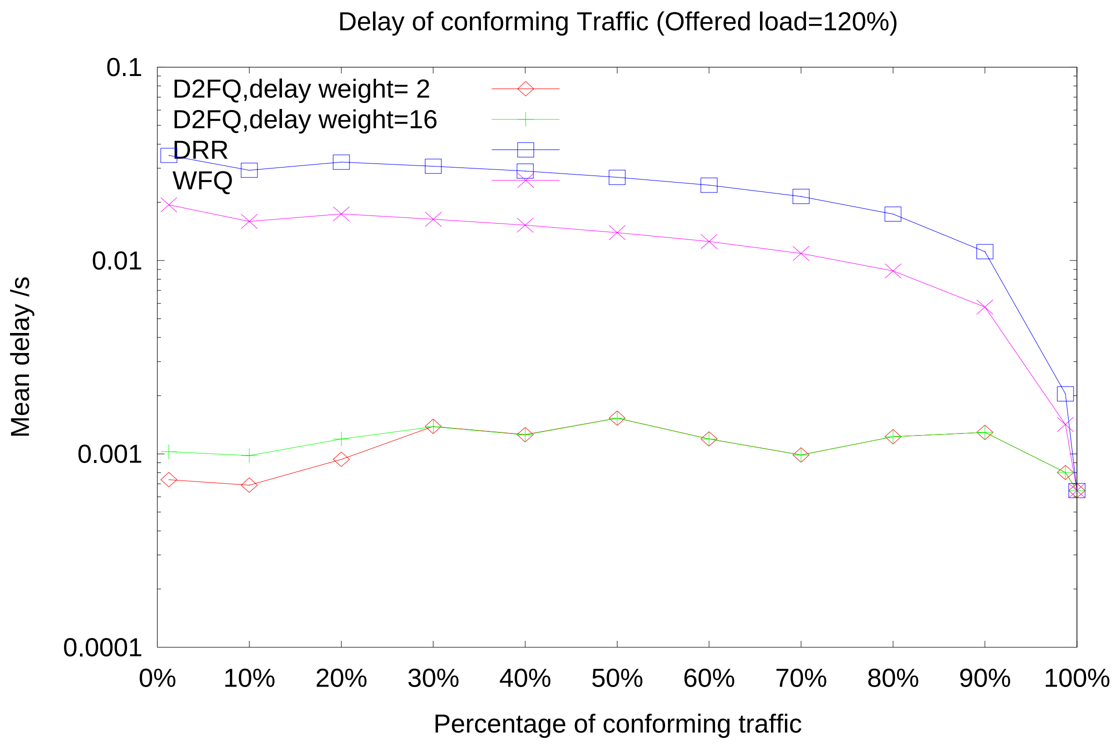


Figure 5-7 Delay of conforming traffic with varied portion of conforming traffic.

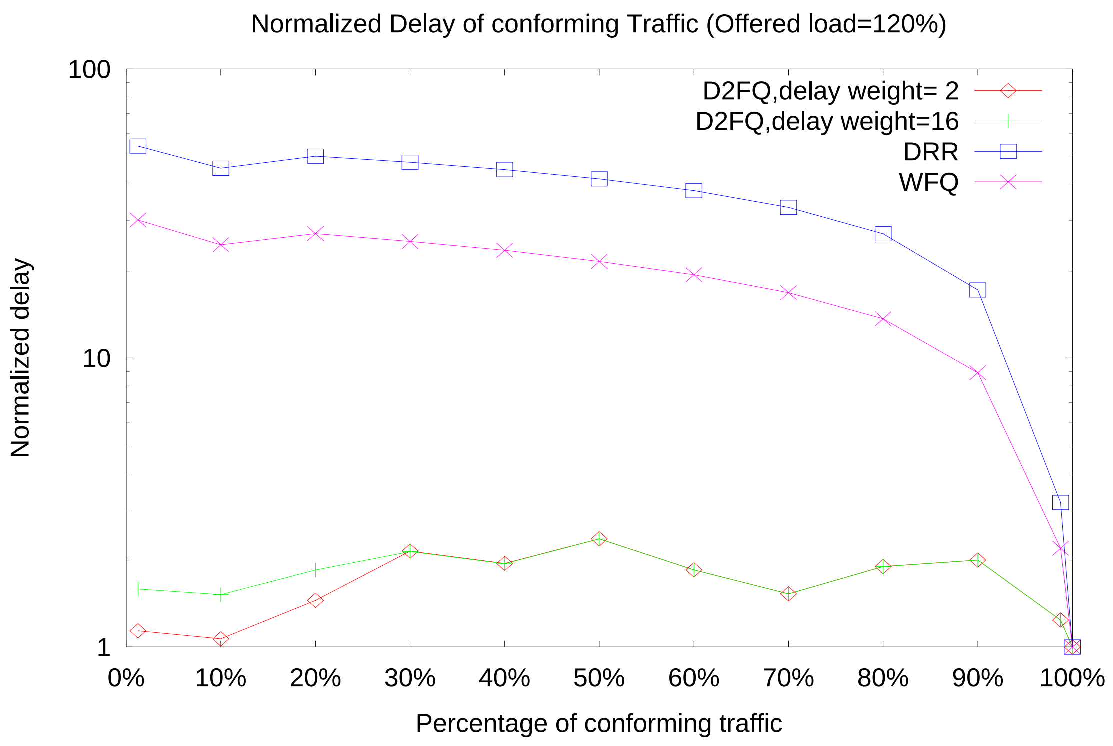


Figure 5-8 Normalized delay of conforming traffic.

## 5.5 The Fairness Delay For Flows with Vary Packet Size

In the common IP network, the packet size is not fixed and theoretically, the size of IP datagram can vary form 40 bytes to $2^{16}$ bytes. Although there is limitation of packet size in the physical layer such as Ethernet, the general packet size of IP network is varied from 40 bytes to 1500 bytes. Therefore, a good scheduler should also consider serving packet size in throughput. Some scheduler such as DRR is designed for fair bandwidth allocation among the flows with different packet size. Generally, most scheduler such as WFQ and DRR can provide a fair bandwidth allocation among flows with different packet size. However, this scheduler has no treatment on the fairness of delay distribution. It can imagine that when a scheduler is transferring a large packet, all backlogged packet would suffer long delay time.

In this simulation, the flows have been set to different combination in packet size and the influence of mean delay with different packet size have been observed under different offered load. The following are the detail conditions of this simulation.

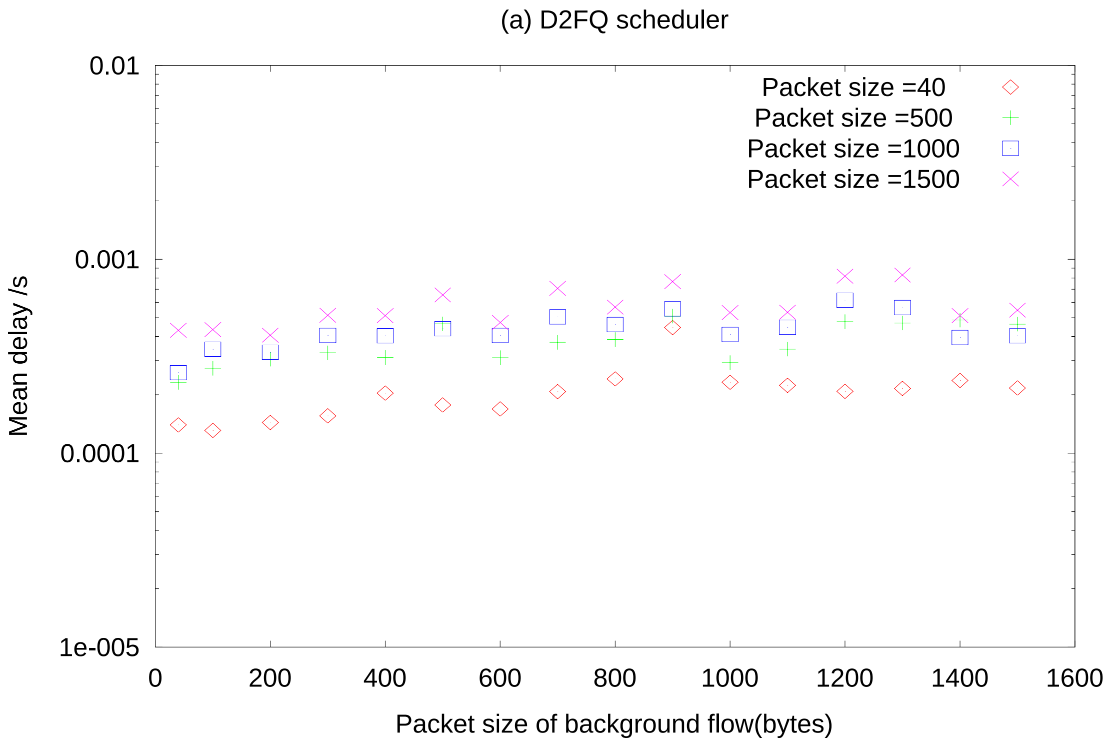
- The 80 flows divide into two classes, first class and second class. 1st class have 50 flows and 2nd class have 30 flows. The throughput weight among all flows is equal.
- The flows in first class would act as background traffic. It consists of thirty conforming flows and twenty non-conforming flows.
- The flows in second class would act as foreground traffic. The flows in this class consist of three different arrival rate, 51st to 60th's flow are light conforming traffic, 61st to 70th's flow are normal conforming traffic and 71st to 80th's flow are heavy non-conforming traffic.
- All flows have same throughput weight and delay weight. The total throughput weight is 80. Let 80% be the acceptable offering load in general. Each throughput weight in 10Mbps channel would be shared 100kbps.
- According to the Table 4, the overall arrival rate is 13.25Mbps and the offered load is 135.25%. The detail setting of arrival rate
- The packet size of the foreground and background traffic and the detail arrival rate of each simulation is shown on Table 4
- In this simulation, the mean delay of D$^2$FQ is compare with WFQ and DRR's mean delay.

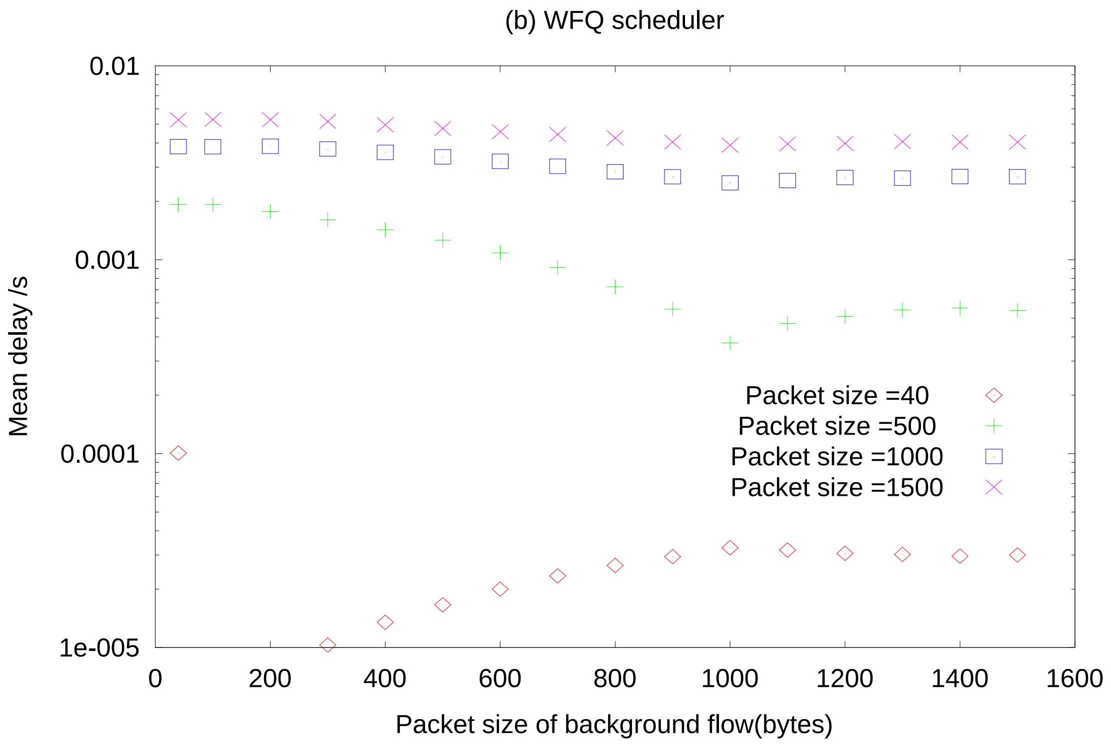
| **Flow number** | **Packet size (byte)** | **Arrival rate** | **Traffic** |
|---|---|---|---|
| 1st flow to 30th flow | (background traffic) 40, 100, 200, 300,⋯, 1400 and 1500 | 125kbps | Normal |
| 31st flow to 50th flow | | 300kbps | Heavy |
| 51st flow to 60th flow | (foreground traffic) 40, 500, 1000 and 1500 | 50kbps | Light |
| 61st flow to 70th flow | | 100kbps | Normal |
| 71st flow to 80th flow | | 200kbps | Heavy |

Table 4 Configuration of simulation of varied packet size.

Figure 5-9 to Figure 5-11 show the result of this simulation. For the light traffic flow, its arrival rate is half of its reserved bandwidth and no packet drop on three schedulers. In Figure 5-9, three schedulers also show that the smaller packet size of the flow has shorter mean delay compared with the larger packet size of the flow. That implied that small packet can get better service. For D$^2$FQ scheduler, the mean delay of 40-byte flow is 10 times less then the mean delay of 1500-byte flow but the mean delays of all packet size's flows are still shorter than 1ms. For WFQ scheduler, only 40-byte packet and 500-byte packet have a low mean delay shorter than 1 ms. The 1000-byte packet and 1500-byte packet have to suffer long mean delay and the different of mean delay between 40-byte flow and 1500-byte flow is larger than 50 times. Please be noticed that the throughput weight of all flows is same. The cause of the large different on mean delay is that WFQ scheduler tries to depart a packet with smallest virtual finish time, and small packet always gets small virtual finish time. Moreover, the result of WFQ also shows that when the size of background packet is increasing, the mean delay of the foreground packet is decreasing. The large background packet suffers longer delay than the small foreground. The different of mean delay in DRR between 40-byte packet and 1500-byte packet below 10 times that is smaller than the different of WFQ. However, the overall mean delay of DRR is usually longer than the mean delay of $D^2$FQ.

For the normal traffic, 61st to 70th flows, Figure 5-10, the mean delay of three schedulers are almost same to the mean delay of its light traffic. For the heavy traffic flows, 71st to 80th flows, these are considered as non-conforming flows because their arrival rate, 100kbps, are double to their reserves rate, 100kbps. Therefore, these flows will be punished and the mean delay is not guaranteed. According to the Figure 5-11, the mean delay of $D^2$FQ is increased to 10ms. The flows with varied packet size suffer a long mean delay and the range is from 4ms to 13ms. The mean delay of DRR is also the same as the mean delay of $D^2$FQ. We can state that the punishment of DRR and $D^2$FQ on the non-conforming is independent of the packet size. However, the non-conforming in WFQ is different. The 40-byte packet flow has lower mean delay compared with the 1500-byte flow. The mean delay of non-conforming 40-byte flow of WFQ, 2ms, is lower than the mean delay of conforming 1500-byte flow of WFQ. According to the simulation results, WFQ will not punish the non-heavy traffic fairly and the small packet flow will always receive a better server. Compare with mean delay of $D^2$FQ, the punishment of $D^2$FQ on the non-conforming is fairer on the delay distribution that is independent of the packet size.


(a) D2FQ scheduler
Mean delay /s
Packet size of background flow(bytes)
Packet size =40
Packet size =500
Packet size =1000
Packet size =1500
0.01
0.001
0.0001
1e-005
0
200
400
600
800
1000
1200
1400
1600


(b) WFQ scheduler
Mean delay /s
Packet size of background flow(bytes)
Packet size =40
Packet size =500
Packet size =1000
Packet size =1500
0.01
0.001
0.0001
1e-005
0
200
400
600
800
1000
1200
1400
1600

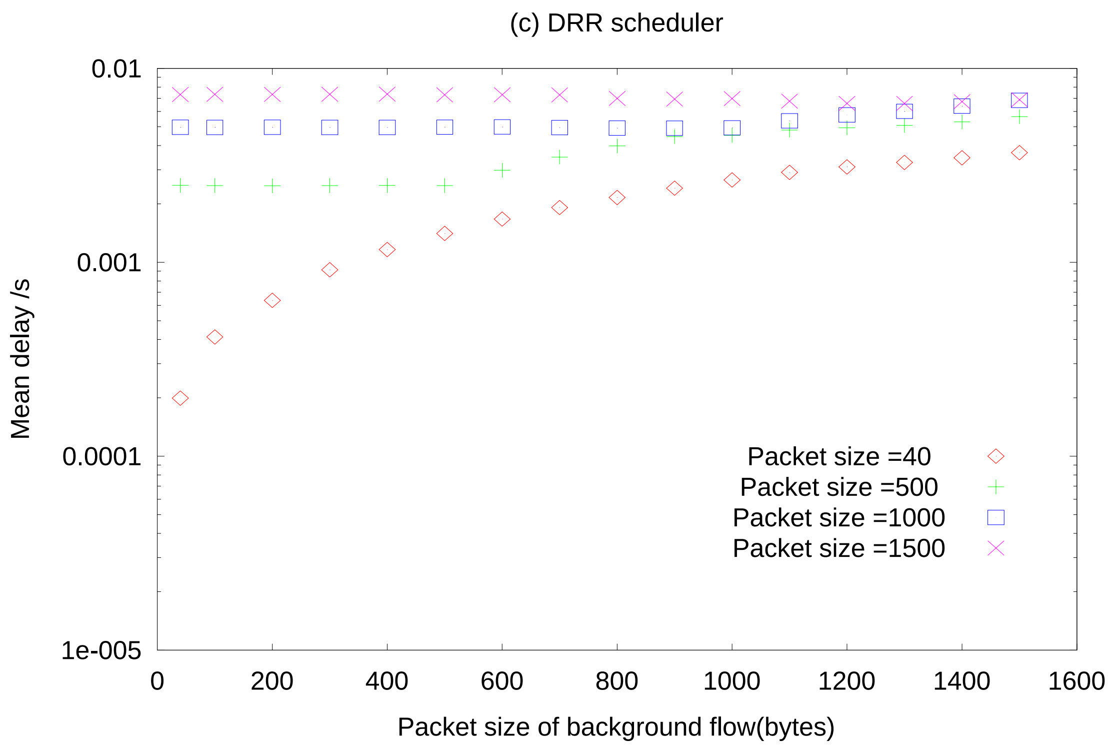
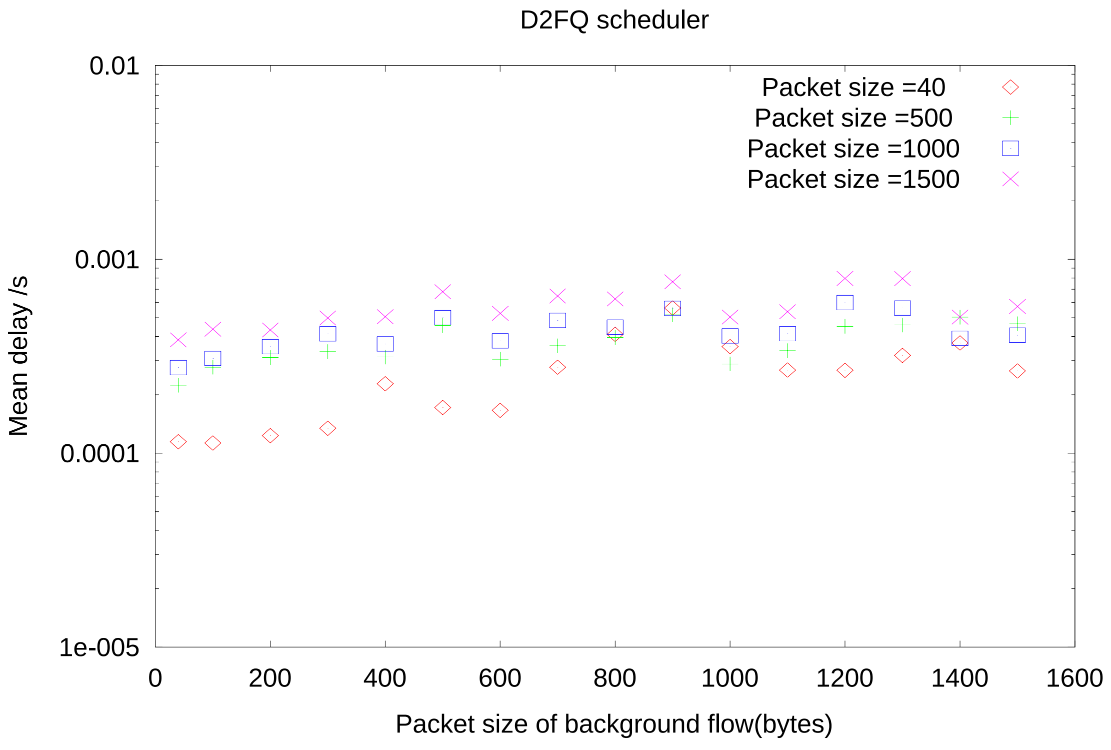
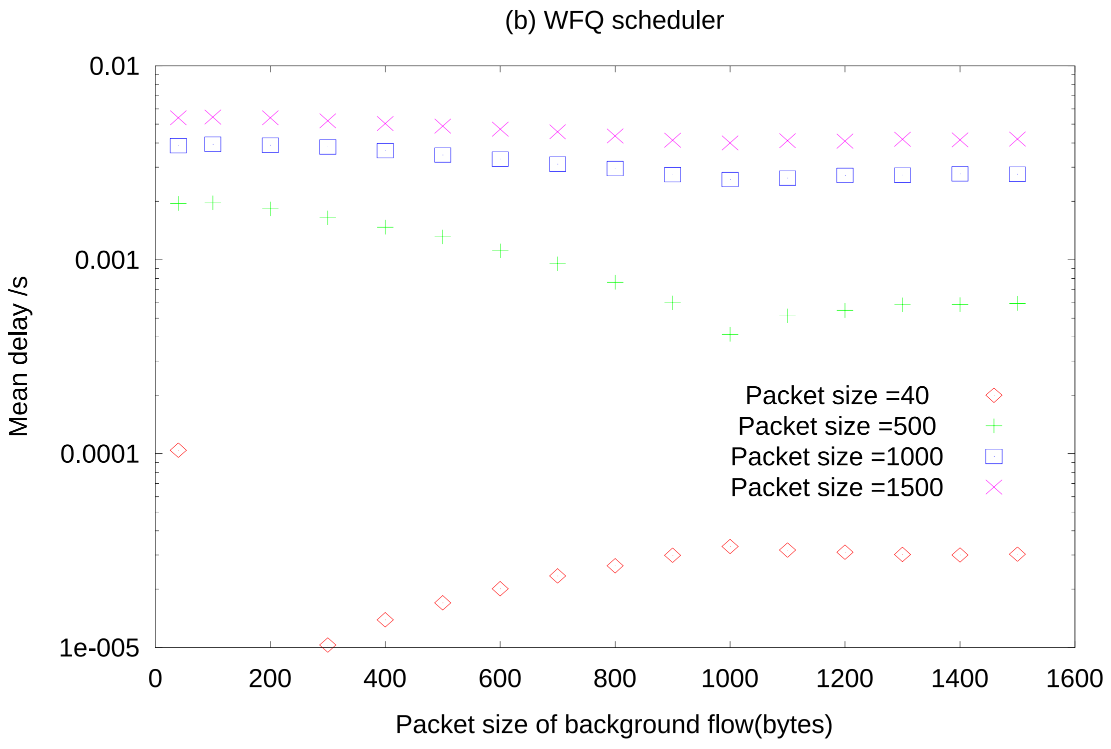


Figure 5-9 Mean delay of light traffic flows, 51st flow to 60th flow, with varied packet size. (a) $D^2FQ$ scheduler, (b) WFQ scheduler, (c) DRR scheduler.


D2FQ scheduler
Packet size =40 ◇
Packet size =500 +
Packet size =1000 □
Packet size =1500 ×
Mean delay /s
0.01
0.001
0.0001
1e-005
0
200
400
600
800
1000
1200
1400
1600
Packet size of background flow(bytes)


(b) WFQ scheduler
Packet size =40 ◇
Packet size =500 +
Packet size =1000 □
Packet size =1500 ×
Mean delay /s
0.01
0.001
0.0001
1e-005
0
200
400
600
800
1000
1200
1400
1600
Packet size of background flow(bytes)

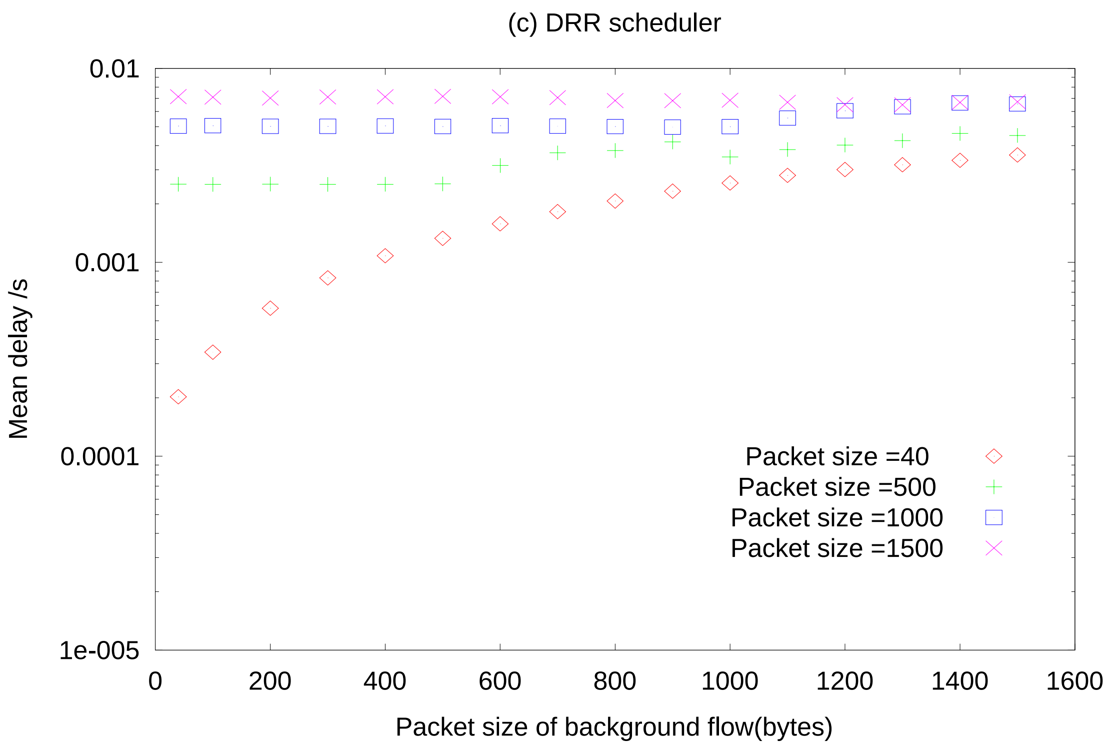
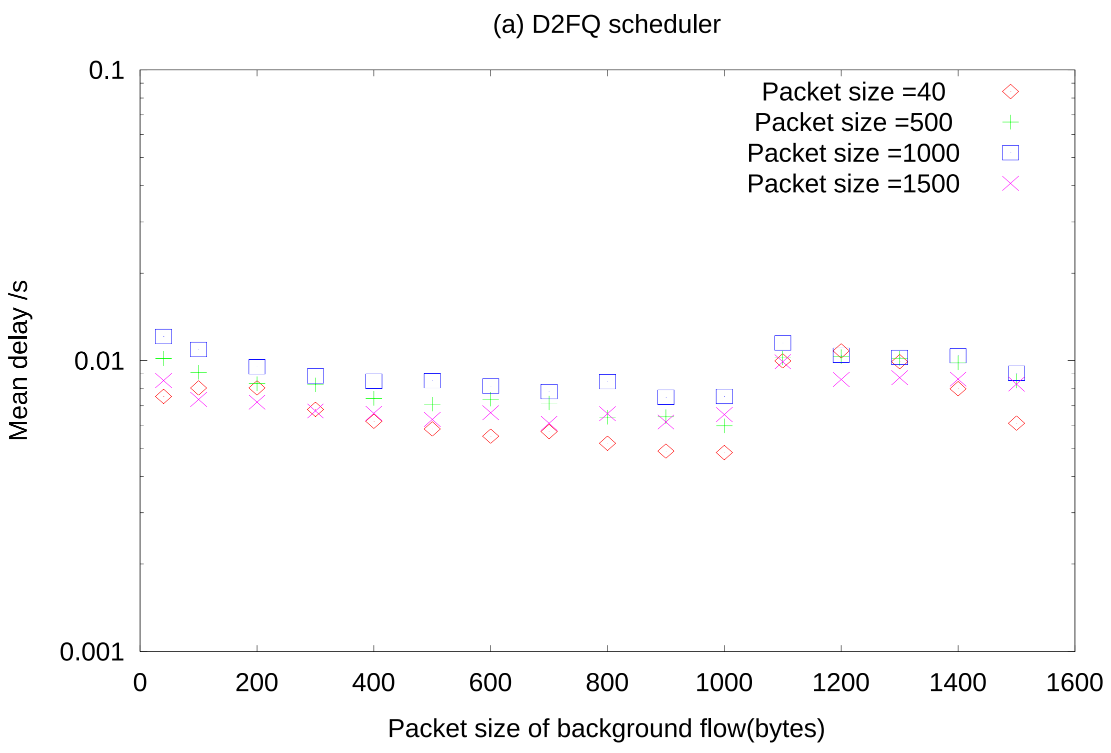
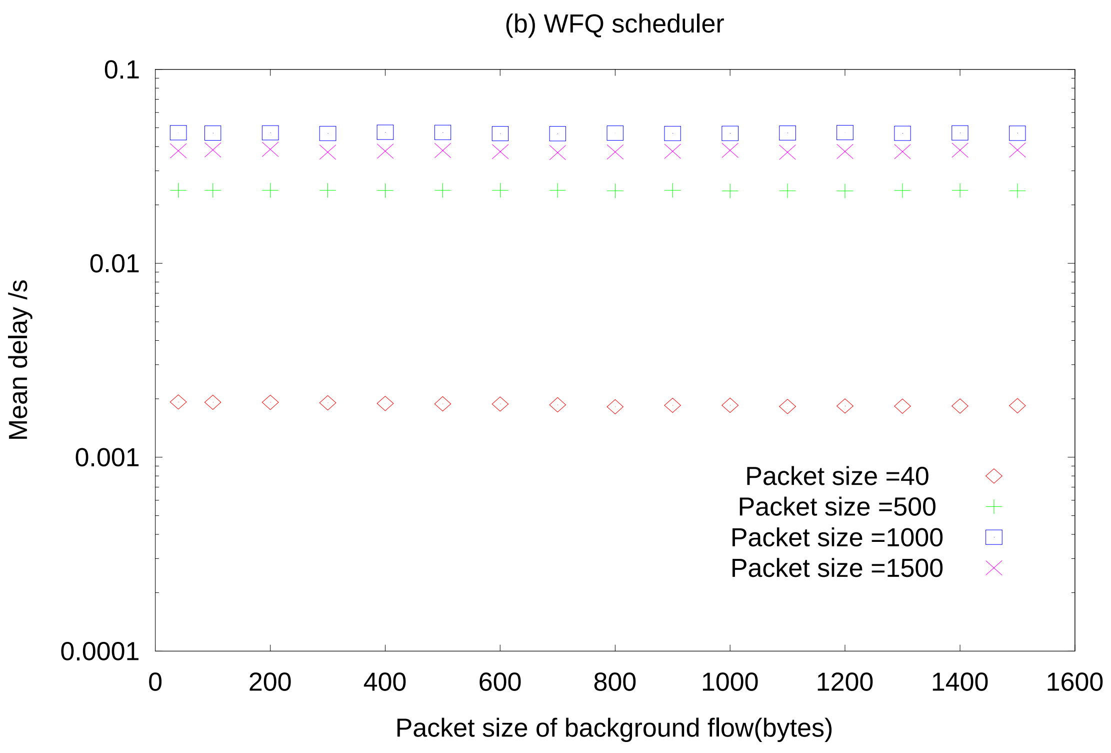


Figure 5-10 Mean delay of moderate traffic flows, 61st flow to 70th flow, with varied packet size. (a) $D^2FQ$ scheduler, (b) WFQ scheduler, (c) DRR scheduler.


(a) D2FQ scheduler
Packet size =40
Packet size =500
Packet size =1000
Packet size =1500
Mean delay /s
0.1
0.01
0.001
0
200
400
600
800
1000
1200
1400
1600
Packet size of background flow(bytes)


(b) WFQ scheduler
Packet size =40
Packet size =500
Packet size =1000
Packet size =1500
Mean delay /s
0.1
0.01
0.001
0.0001
0
200
400
600
800
1000
1200
1400
1600
Packet size of background flow(bytes)

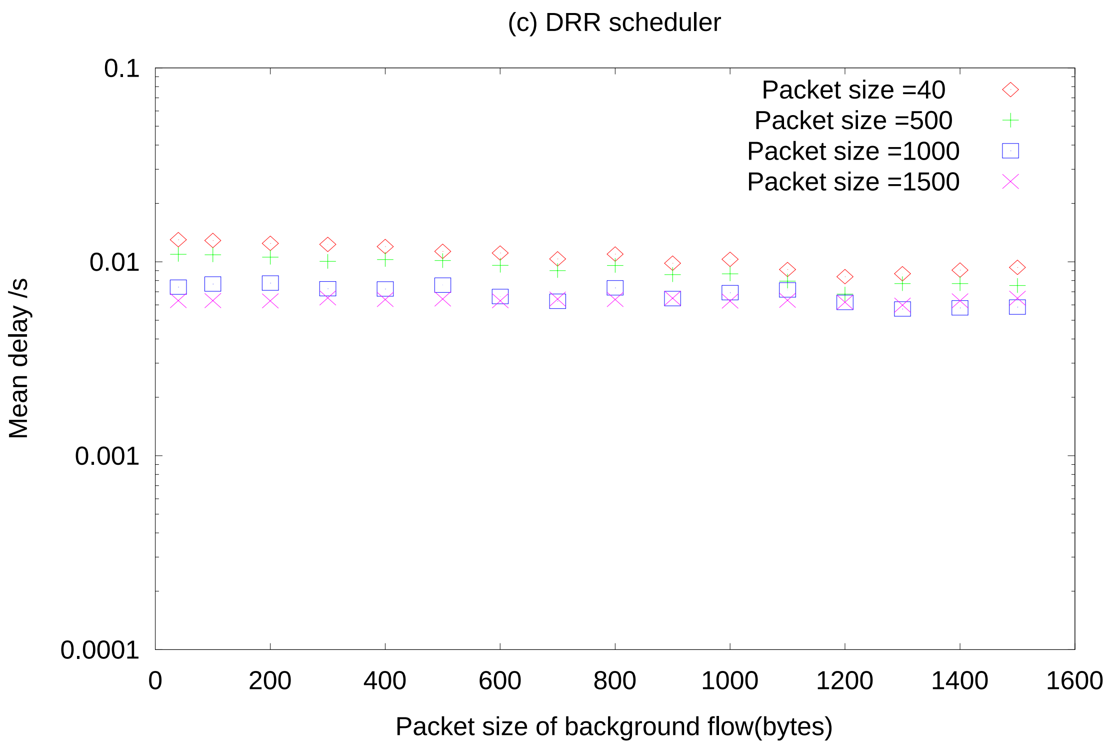


Figure 5-11 Mean delay of heavy traffic flows, 71st flow to 80th flow, with varied packet size. (a) D$^2$FQ scheduler, (b) WFQ scheduler, (c) DRR scheduler.

# Chapter 6. Simulation Study on Multi-node Topology

In this chapter, multi-node topology is used to study the performance of connection-oriented communication through a chain of network node. In the above chapter, the performance of D$^2$FQ scheduler has been verified under different loading condition. When the offered load is below 90%, D$^2$FQ almost has same performance with other well-known scheduler such as WFQ. In the heavy loading condition, the congestion control in the source host will adapt the delivery rate of the packet and the congestion control will increase the complexity of the resource allocation of scheduler. Thus, end-to-end metric will be used to analysis the performance of scheduler.

## 6.1 Multi-node Topology

Three different simulations are conducted for D$^2$FQ performance study on multi-node topology. All simulations are based on the topology in Figure 6-1. On the first simulation which studies on the delay, throughput and drop rate of a TCP flow on varied number of congested node. A TCP flow will compete with other ten’s non-conforming flow in each congested node. Due to the congestion control mechanism of connection-oriented flow, the end-to-end delay and the drop rate of the flow will affect the transmission rate of the source. The bandwidth allocation of scheduler is an important factor for end-to-end performance. In this simulation, WFQ, DRR and CSFQ[28] will be used were used as comparison in this situation. Same as single node topology, WFQ and DRR were used to fairness comparison on bandwidth allocation. The architecture of CSFQ network will be applied in fairness study on CSFQ. Sources and Sink node will be classified to edge router and other internal node will be classified to core router. This complex architecture is favor to performance study on multi-node topology.

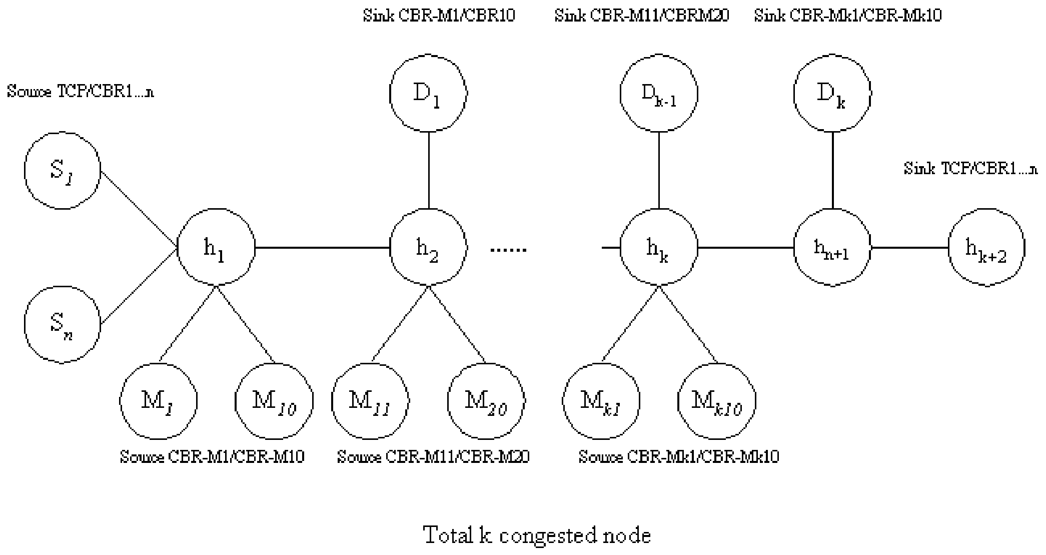


Figure 6-1 Multi-node network topology

Figure 6-1 is the simulation topology for multi-congested node. All links have a limited bandwidth 10Mb/s. In the chains of node, $h_1$ to $h_{k+2}$, non-conforming flow will delivery packet from node $M_i$ to node $D_i$ in CBR pattern. Each node, $h_k$, will be congested by 40 non-conforming flows. The node $h_1$ to $h_k$, therefore, will be totally congested all the time. Moreover, the testing traffic will delivery packet from source, $S_j$ to sink, $h_{k+2}$. These testing flows include TCP traffic flows and UDP traffic flows.

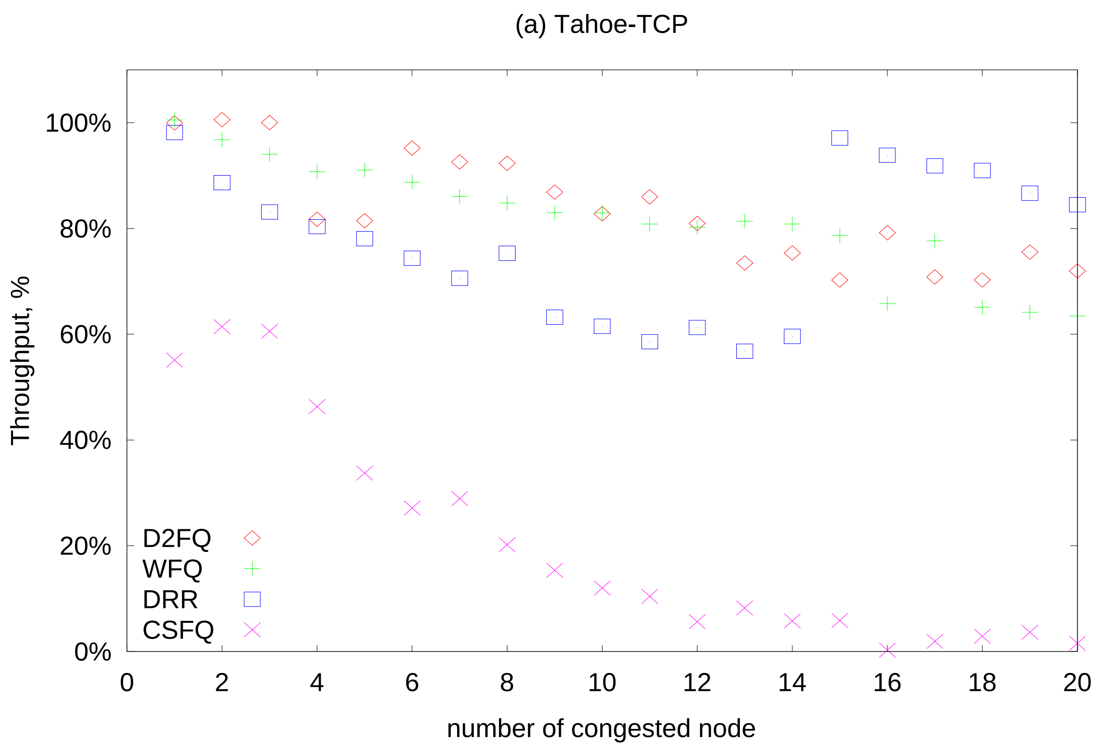
## 6.2 TCP Flow with Varied Number of Congested Node.

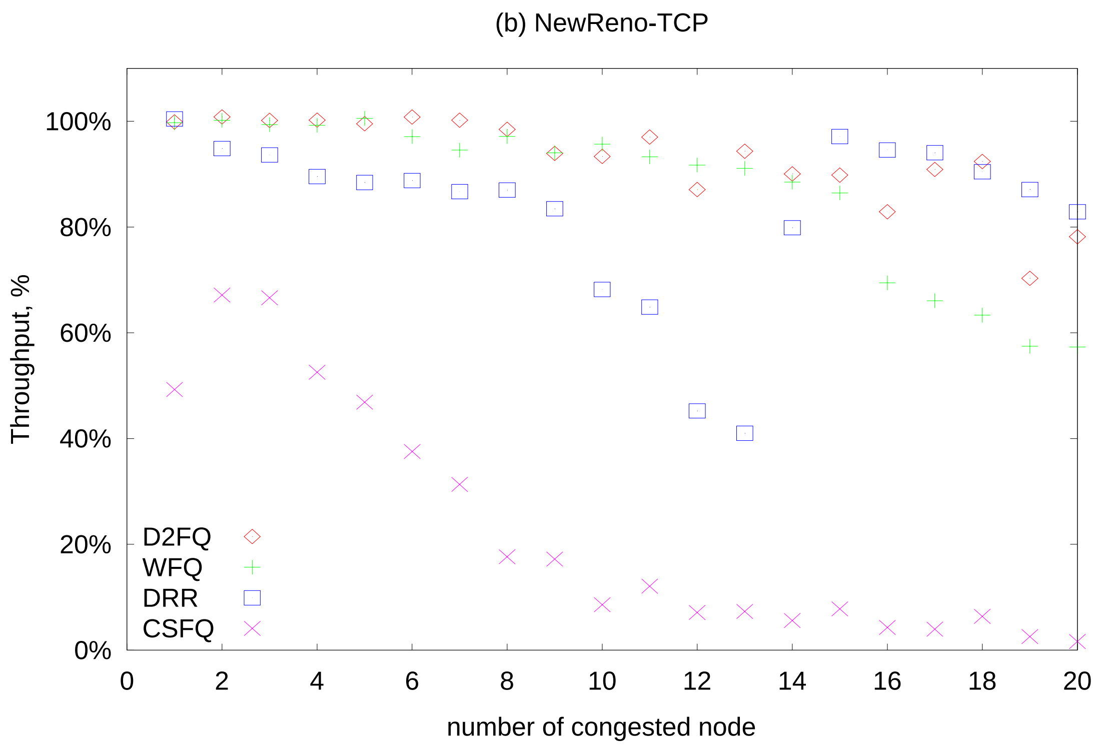
TCP is the most popular connection-oriental protocol. When a node is congested, the congestion control mechanism of the protocol would have its way to achieve a larger outgoing bandwidth. In TCP, Tahoe, Reno, New Reno[31], SACK [32, 33], and some modifications of improved TCP were proposed. Although they have better performance than previous TCP, no TCP type would get the best performance in all network conditions. Moreover, the cooperation between different types of TCP is also not well. Although scheduler can maintain the fairness between different connections, some TCP-unfriendly connections such as non-conforming UDP flow will be adverse to outgoing throughput of TCP flow.

In this simulation, varied number of congested nodes are used to evaluate the performance of $D^2$FQ on three TCP types, which are Tahoe, New Reno and Sack. Figure 6-2 shows the normalized throughput of a single TCP flow in varied scheduler. In general, the results show that the overall tendency of the normalized throughput is decreasing when the number of congested node is increasing. The four schedulers algorithm, $D^2$FQ, WFQ, DRR and CSFQ schedulers, also have this phenomenon. In the result of three TCP types, when the congested node is less than 15, the performance of TCP in $D^2$FQ and WFQ schedulers are obviously better than DRR and CSFQ schedulers.

Compare with WFQ, $D^2$FQ scheduler often achieved higher throughput for a single TCP flow. Especially in the Sack TCP connection, the throughput of $D^2$FQ scheduler is always higher than the throughput of WFQ scheduler. These three TCP types can show that $D^2$FQ scheduler is more efficient than WFQ scheduler in TCP flow protection in congested network. There is an interesting phenomenon for number of congested node is larger than 15, the normalized throughput of DRR scheduler increased rapidly even higher than the throughput of $D^2FQ$ and WFQ scheduler. The cause of this phenomenon is the buffer management of DRR is synchronized with the sliding window of TCP connection. For three TCP type, the performance of Sack’s TCP is the highest compared with the performance of new-Reno’s TCP and Tahoe’s TCP and this comparison result is also consisted to many research of TCP comparison[31, 32, 33].

Figure 6-3 is the drop rate of three TCP type with varied number of congested node. According to the results in Figures, the overall drop rate is decreasing when the number of congested node is increasing for $D^2FQ$, WFQ and DRR scheduler except CSFQ scheduler. The result shows that CSFQ scheduler encounters a relative high drop rate when number of node is larger than seven nodes. The large drop rate of CSFQ scheduler is the cause of low throughput. Although throughput of $D^2FQ$ scheduler is generally higher than throughput of WFQ scheduler, the drop rate of $D^2FQ$ scheduler is also higher than the drop rate of WFQ scheduler by two times. This high outgoing throughput and high drop rate characteristics implied the TCP-source is willing to delivery more packets to the destination even though there are many packet drops. When the bandwidth of connection can be recovery fast after some packet drop, the connection can be said as TCP friendly connection. The Figure 6-4 shows the End-to-End delay of TCP packet and the Figure 6-5 shows the average queueing delay per each node of TCP packet. These figures show the overall end-to-end delay of TCP flow in $D^2FQ$ scheduler is much shorter than other scheduling algorithms.


(a) Tahoe-TCP
Throughput, %
100%
80%
60%
40%
20%
0%
0
2
4
6
8
10
12
14
16
18
20
number of congested node
D2FQ ◇
WFQ +
DRR □
CSFQ ×


(b) NewReno-TCP
Throughput, %
100%
80%
60%
40%
20%
0%
0
2
4
6
8
10
12
14
16
18
20
number of congested node
D2FQ ◇
WFQ +
DRR □
CSFQ ×

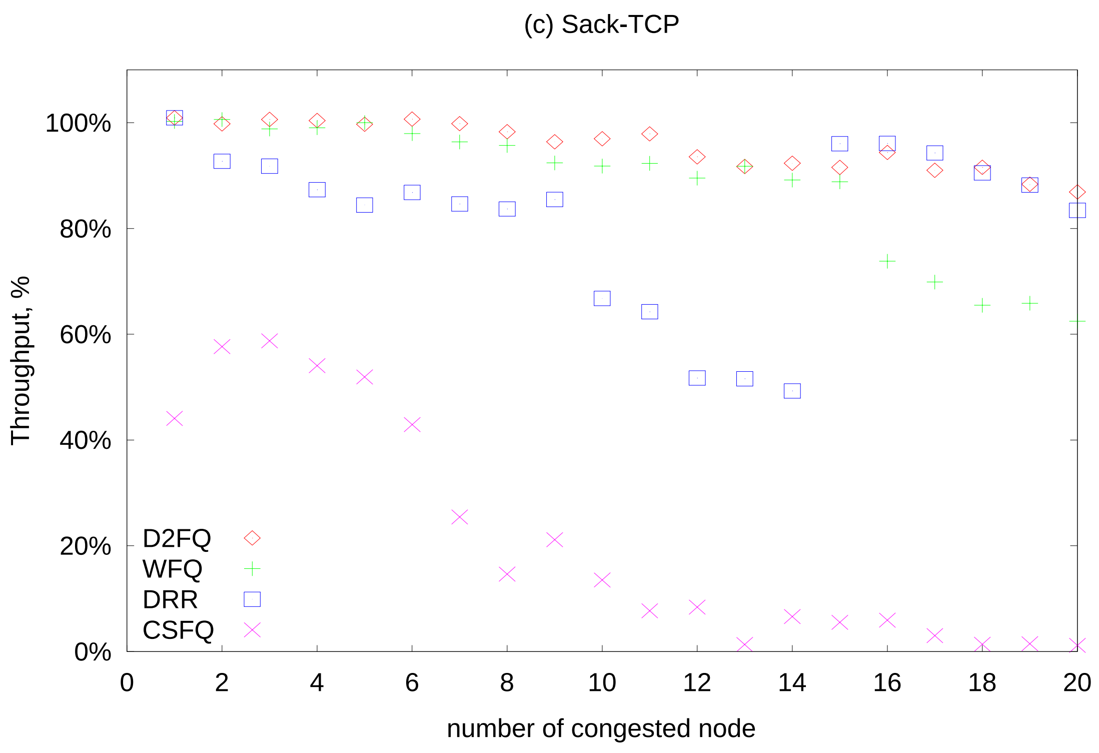
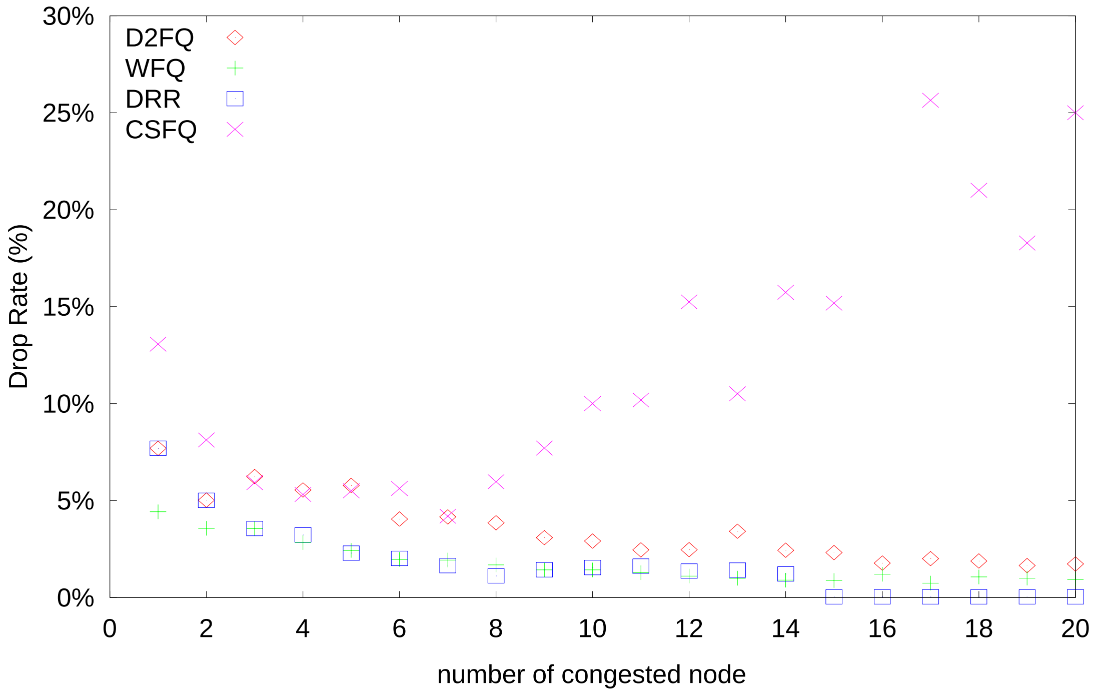
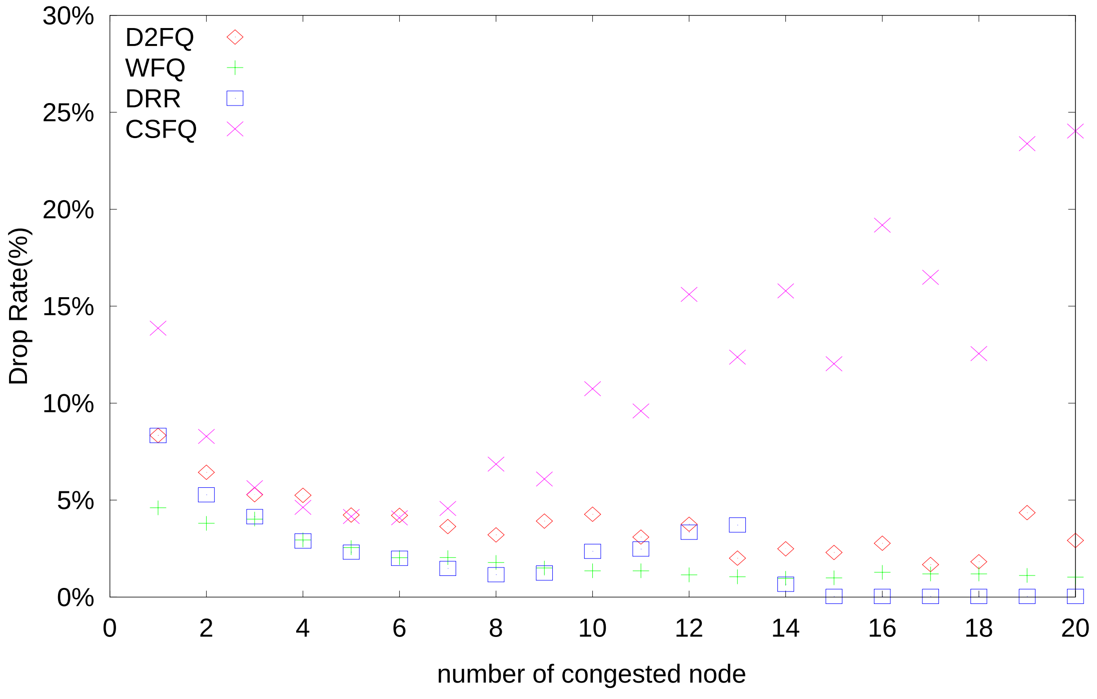


Figure 6-2 Thoughput of single TCP flow with varied congested node. (a) Tahoe TCP, (b) New Reno-TCP, (c) Sack-TCP)

(a) Tahoe-TCP



(b) NewReno-TCP



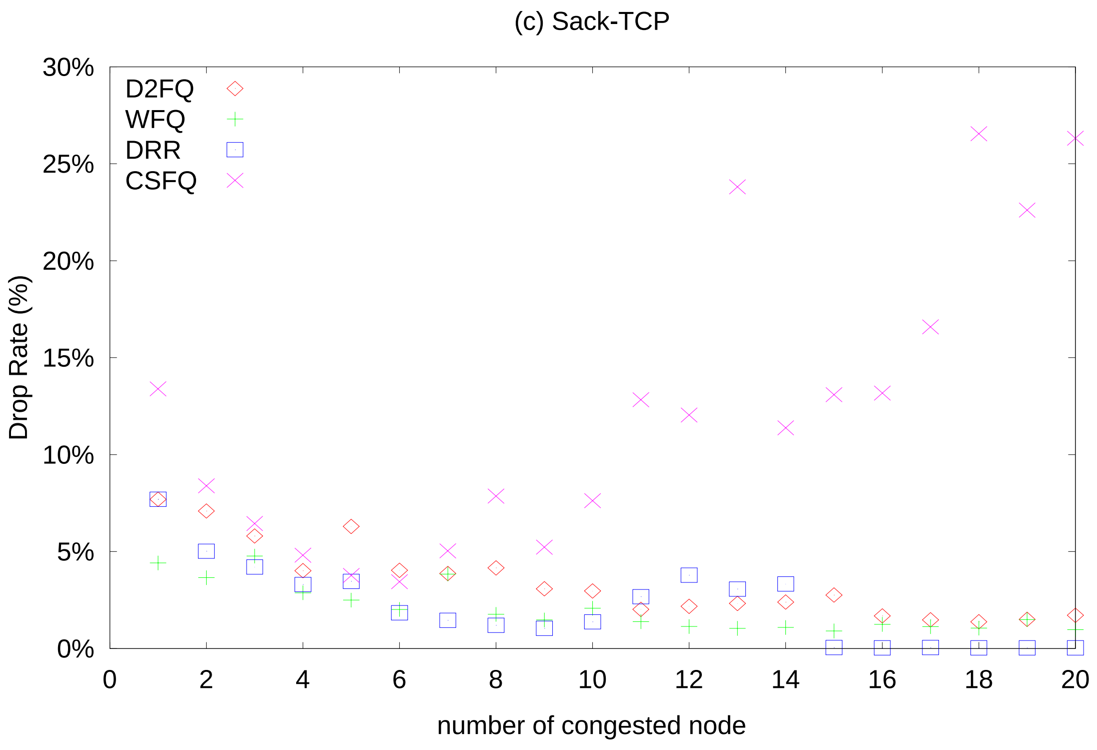
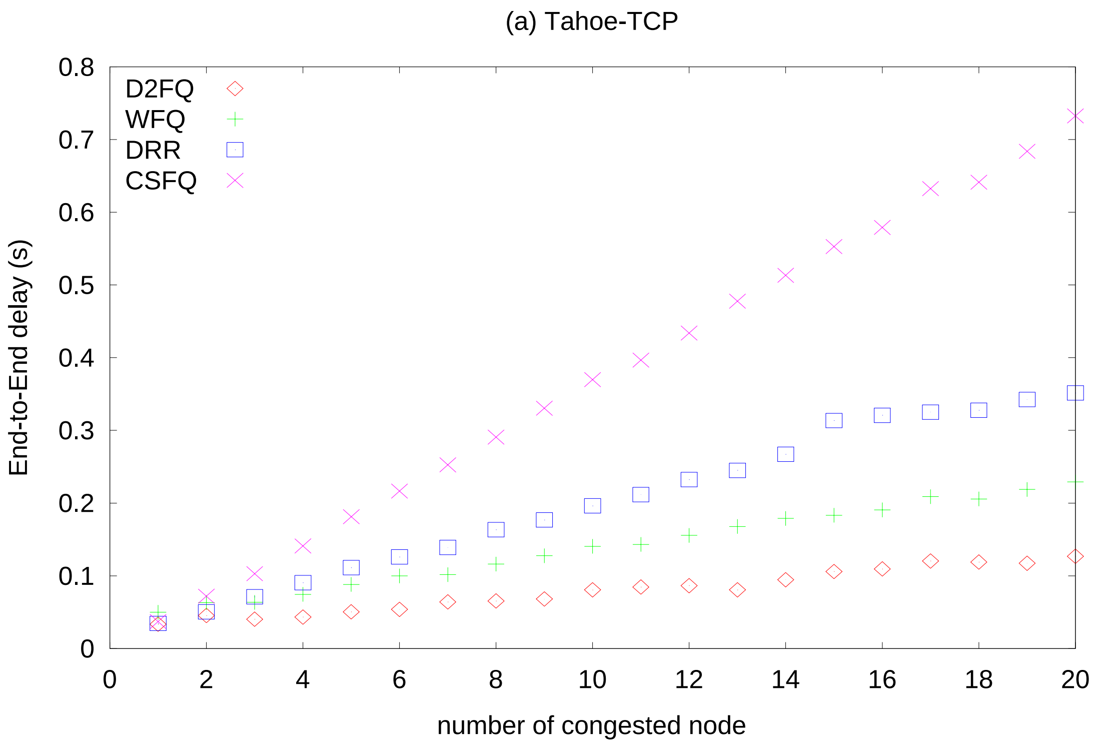
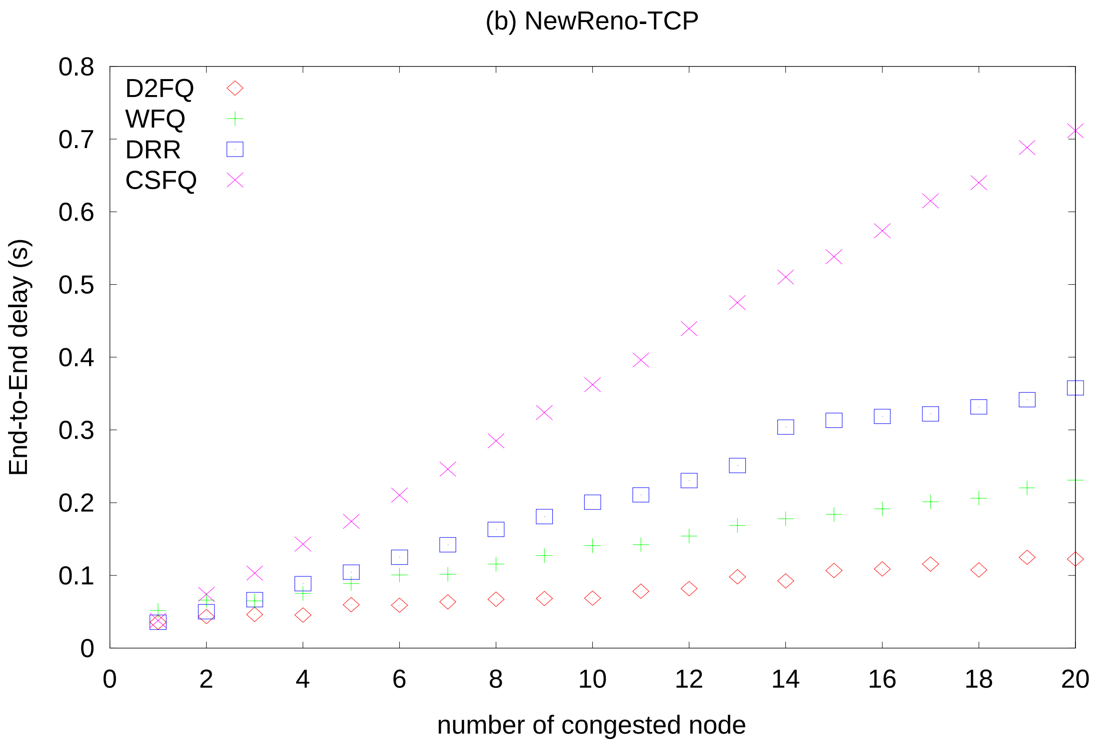


Figure 6-3 Drop rate of Sack-TCP with varied congested node. (a) Tahoe-TCP, (b) New-Reno, (c) SACK-TCP.


(a) Tahoe-TCP
End-to-End delay (s)
number of congested node
D2FQ
WFQ
DRR
CSFQ
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10
12
14
16
18
20


(b) NewReno-TCP
End-to-End delay (s)
number of congested node
D2FQ
WFQ
DRR
CSFQ
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10
12
14
16
18
20

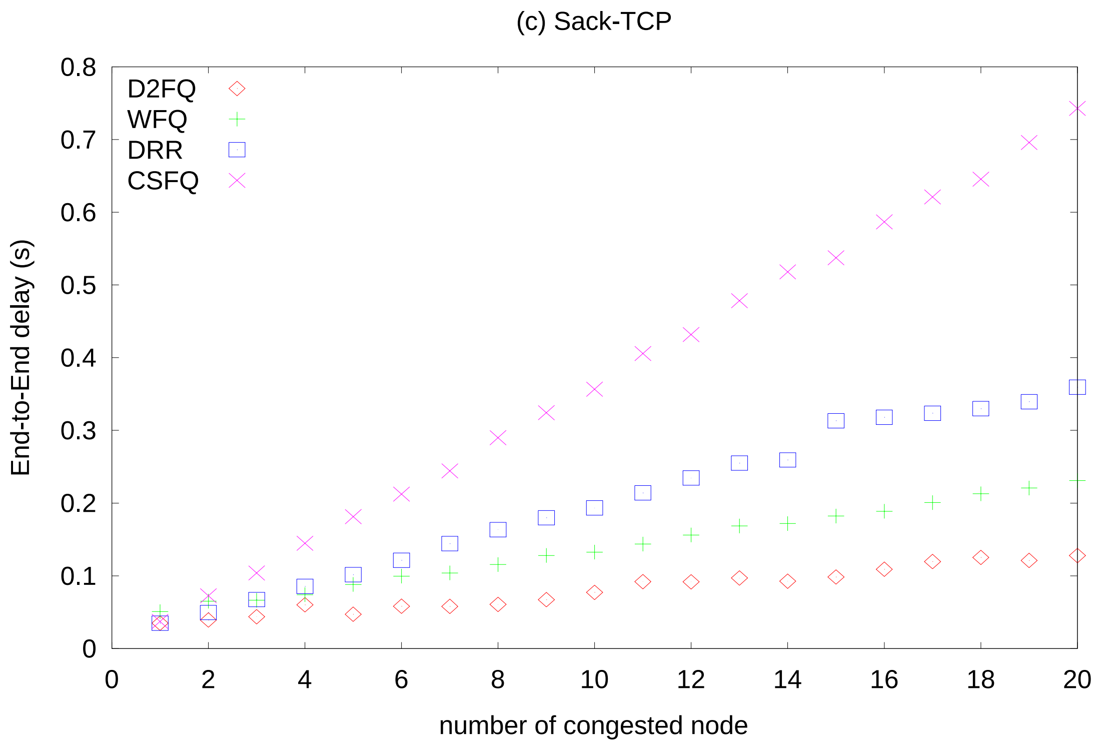
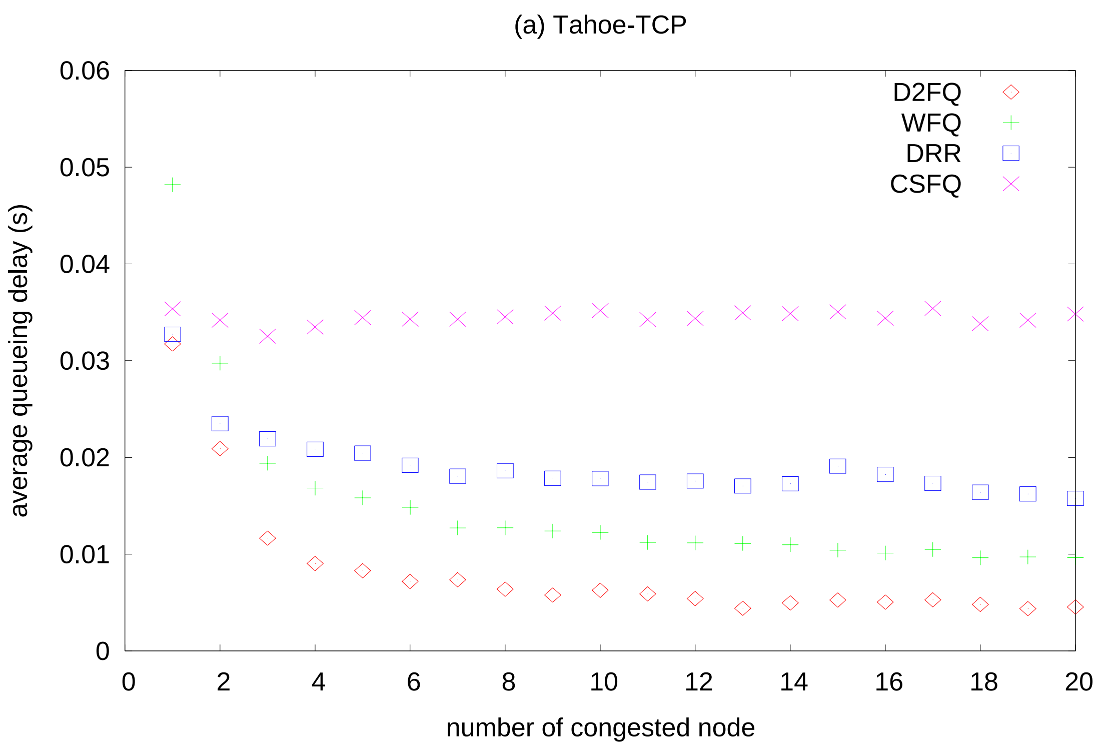
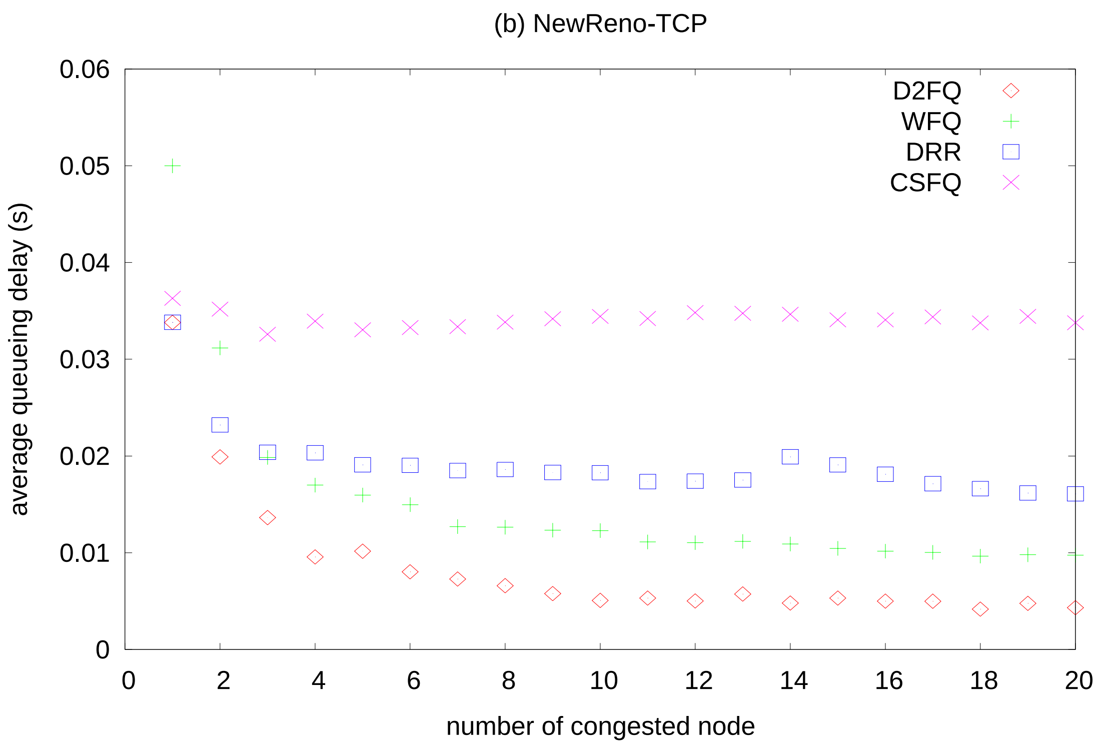


Figure 6-4 End-toEnd delay of single TCP with varied congested node. (a) Tahoe-TCP, (b) New-Reno, (c) SACK-TCP.


(a) Tahoe-TCP
average queueing delay (s)
number of congested node
D2FQ
WFQ
DRR
CSFQ
0
0.01
0.02
0.03
0.04
0.05
0.06
0
2
4
6
8
10
12
14
16
18
20


(b) NewReno-TCP
average queueing delay (s)
number of congested node
D2FQ
WFQ
DRR
CSFQ
0
0.01
0.02
0.03
0.04
0.05
0.06
0
2
4
6
8
10
12
14
16
18
20

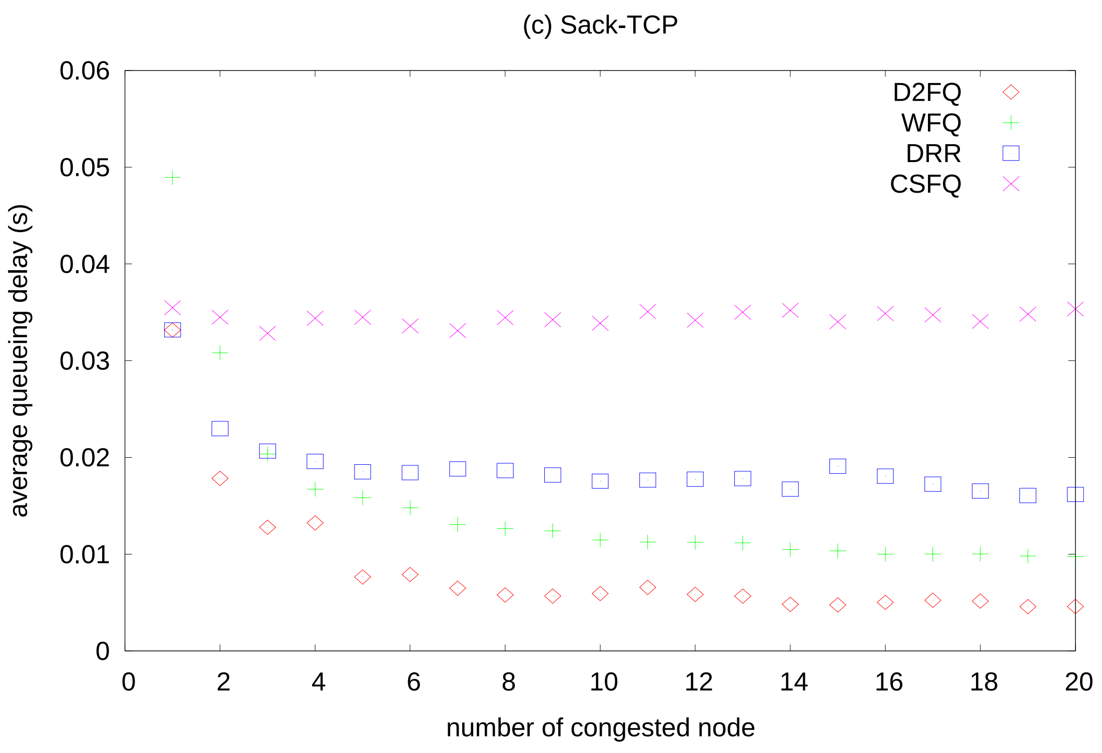


Figure 6-5 Average delay per node of Tahoe-TCP with varied congested node. (a) Tahoe-TCP, (b) New-Reno, (c) SACK-TCP.

## 6.3 Performance of TCP Flow with Different Throughput Weight

Generally, the competition among TCP flows for bandwidth allocation will easily get unfairness on distribution of outgoing throughput. In addition, some phenomena will be adverse to TCP connection because of buffer management of scheduler may be TCP-unawareness. This simulation shows the fairness of bandwidth allocation, drop rate and end-to-end delay for 40 TCP flows.

The detail configurations of this simulation is as follow,

- The number of contested node is fixed and there are five congested nodes, $h_1$ to $h_5$, between source and sink.
- The 40 new-Reno TCP flows delivery packets from source, $S_1$ ...$S_{40}$ to sink $h_7$

- These 40 flows divide into four groups, i.e. 1st group to 4th group and its throughput weight of 1st group to 4th group are 1, 2, 3 and 4 respectively.

- There is another 40 background flows delivery packet from node $M_i$ to node $D_i$

- There are three offered loads, which are resulted by 40 background flows. The three offered loads are light (20%), normal (60%), and heavy offered load, (400%) on five congested nodes. And the background traffic contribute 2Mbps, 6Mbps and 40Mbps in offered load respectively

Figure 6-6 shows the normalized throughput for 40 TCP flows in different offered load. The result shows that the normalized throughput of WFQ and $D^2$FQ are closed to 100%. That implied the bandwidth allocation of WFQ and $D^2$FQ is complied with pre-assigned throughput weight. The bandwidth allocation of TCP flow for DRR and CSFQ scheduler is unfair that its normalized throughput is not close to 100%. In the heavy or moderate background traffic condition, the unfairness of TCP flow in DRR and CSFQ scheduler is more serious but the bandwidth allocation of $D^2$FQ and WFQ scheduler can still complied with the flow’s throughput weight.

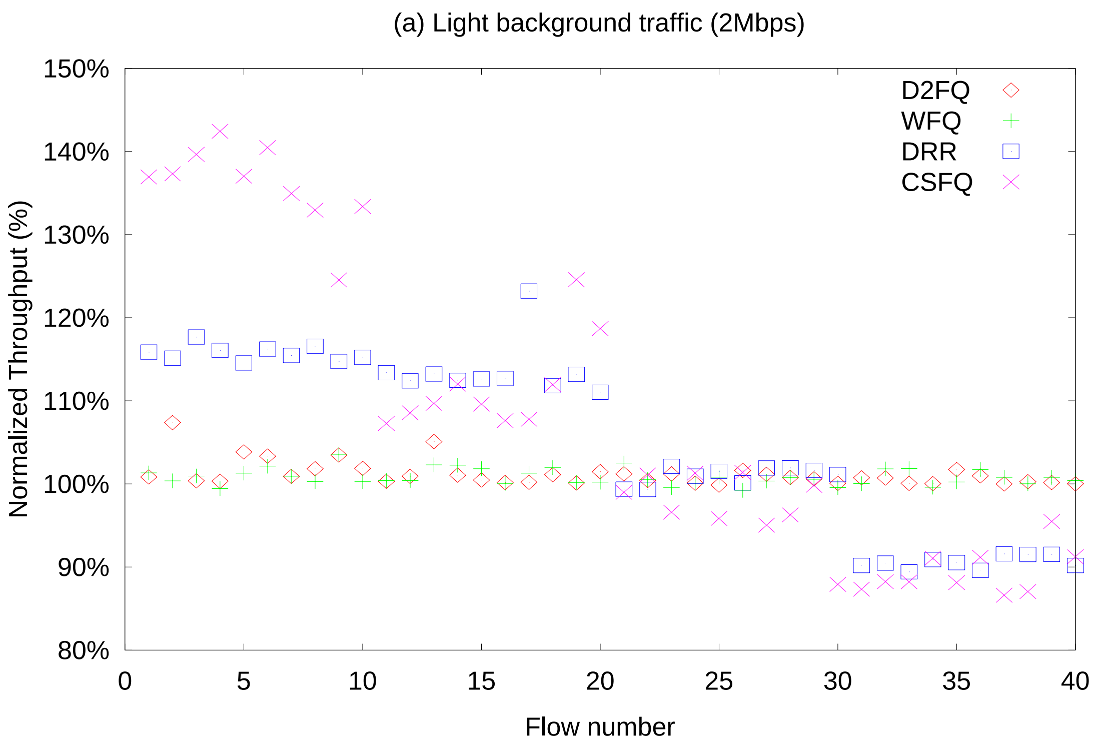
Although the normalized throughput of 2nd flow of $D^2$FQ in light background traffic is 108%, it also do not influence by the other flows which have low throughput. When the background traffic is light, the 40 TCP flows would try to share the reminder bandwidth. The bandwidth competition among TCP flows would incur slightly unfairness due to each TCP flow has its recovery time or retransmit time that is the essential parameter of TCP flow control. Therefore, the range of normalized throughput has slightly fluctuation even the flows with same throughput weight. If we carefully observe the normalized throughput of WFQ and $D^2$FQ, the normalized throughput of $D^2$FQ is much close to 100%.

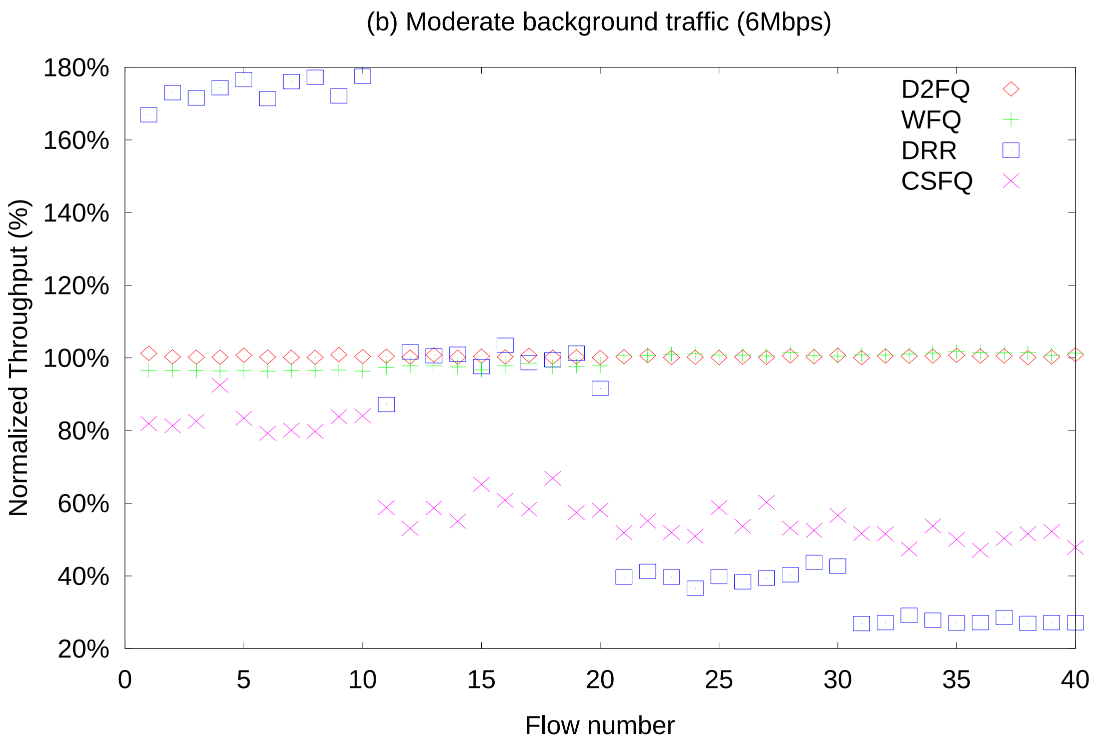
Generally, drop rate is also an important metric for the QoS of TCP connection. Figure 6-7 shows the drop rate of TCP flows. According to the simulation result, the outgoing drop rate of TCP flow depends on its throughput weight in $D^2$FQ scheduler. For the flow with throughput weight equal to 1, the drop rate of that flow in $D^2$FQ scheduler is about 10%. For the flow with throughput weight equal to 4, the drop rate of that flow is about 4%. The drop rate of a TCP flow is inverse proportional to its throughput weight in the $D^2$FQ scheduler. However, we cannot state that higher drop rate reflects the worse performance. For a general user, their perception of network performance is delay and throughput and does not consider the drop rate if the performance of throughput is complied with its reserved bandwidth. Actually, the high drop rate on the low throughout weight's flow can be seen as the mechanism of bandwidth allocation to maintain fairness of bandwidth allocation of TCP flow.

From the result of end-to-end delay in Figure 6-8(a), when the background traffic is light, the mean end-to-end delay of 40 TCP flows in $D^2$FQ is within a narrow range, i.e. 0.2 to 0.3 second. Compare with end-to-end delay of WFQ, the received delay of WFQ is seriously dependent on its throughput weight. The end-to-end delay is 0.12 second for the flow with throughput weight 4 and the end-to-end delay is 0.6 second for the flow of throughput weight 1. This is unfair for low throughput weight's flow suffering more delay.

According to the Figure 6-8(c), the heavy background traffic would incur a very long end-to-end delay on WFQ, DRR and CSFQ schedulers, i.e. 0.5 second for WFQ, 1 second for DRR and 0.9 second for CSFQ. The end-to-end delay of $D^2$FQ scheduler still maintains within 0.2 and 0.3 second, which is as same as the end-to-end delay of flow under light background traffic. That implied that the end-to-end delay of TCP flow in WFQ, DRR and CSFQ scheduler will be influenced by the amount of the background traffic. If the network is attacked by non-conforming traffic, TCP will still suffer very long end-to-end delay especially for flow with low throughput weight. Based on the above analysis, $D^2$FQ scheduler can support fair bandwidth allocation on TCP flows regardless to the offered load of background traffic. The buffer management of $D^2$FQ and its delay treatment can maintain the end-to-end delay in an acceptable level under any loading condition.

The relationship of outgoing throughput, end-to-end delay and the drop rate between $D^2$FQ and WFQ scheduler is very interesting. $D^2$FQ can maintain end-to-end delay within an acceptable range and achieve the fair bandwidth allocation that complied with its throughput weight. However, its drop rate is depended on the throughput weight. WFQ can provide fair bandwidth allocation for all flows and with same drop rate among all flows but the received delay is fully depended on the throughput weight, i.e. lower throughput weight but longer delay. As mentioned in above section, the guaranteed bandwidth and predictable end-to-end delay is more important than the stable drop rate.


(a) Light background traffic (2Mbps)
Normalized Throughput (%)
Flow number
D2FQ
WFQ
DRR
CSFQ
150%
140%
130%
120%
110%
100%
90%
80%
0
5
10
15
20
25
30
35
40


(b) Moderate background traffic (6Mbps)
Normalized Throughput (%)
Flow number
D2FQ
WFQ
DRR
CSFQ
180%
160%
140%
120%
100%
80%
60%
40%
20%
0
5
10
15
20
25
30
35
40

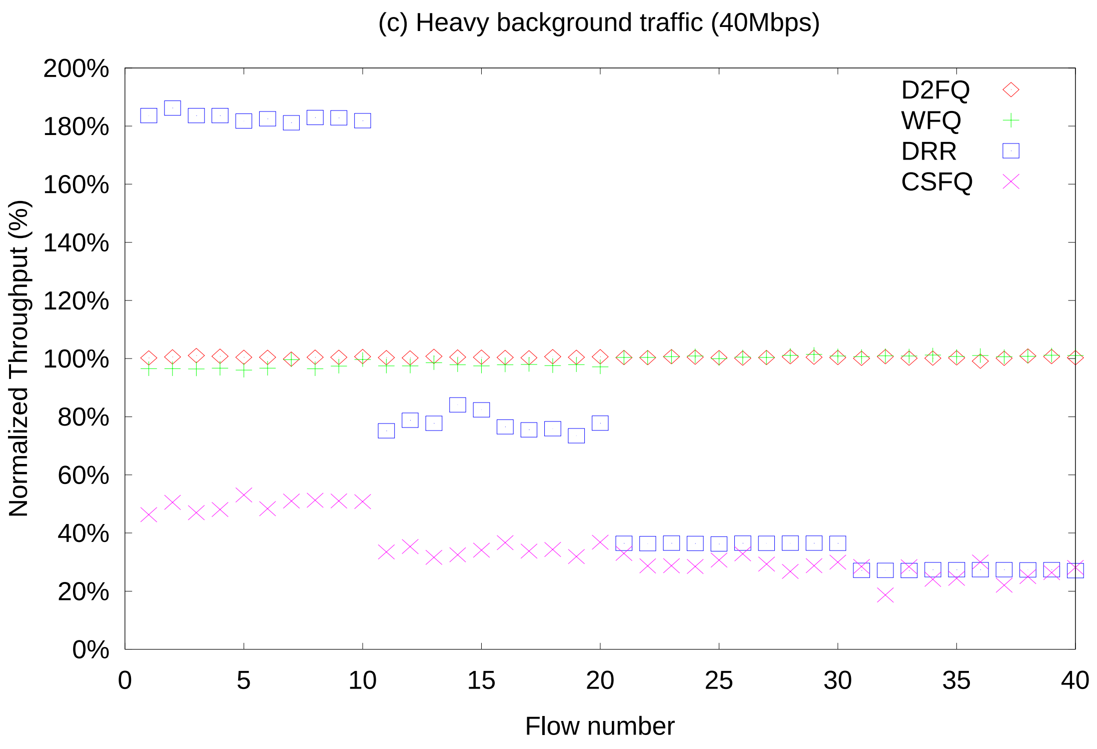
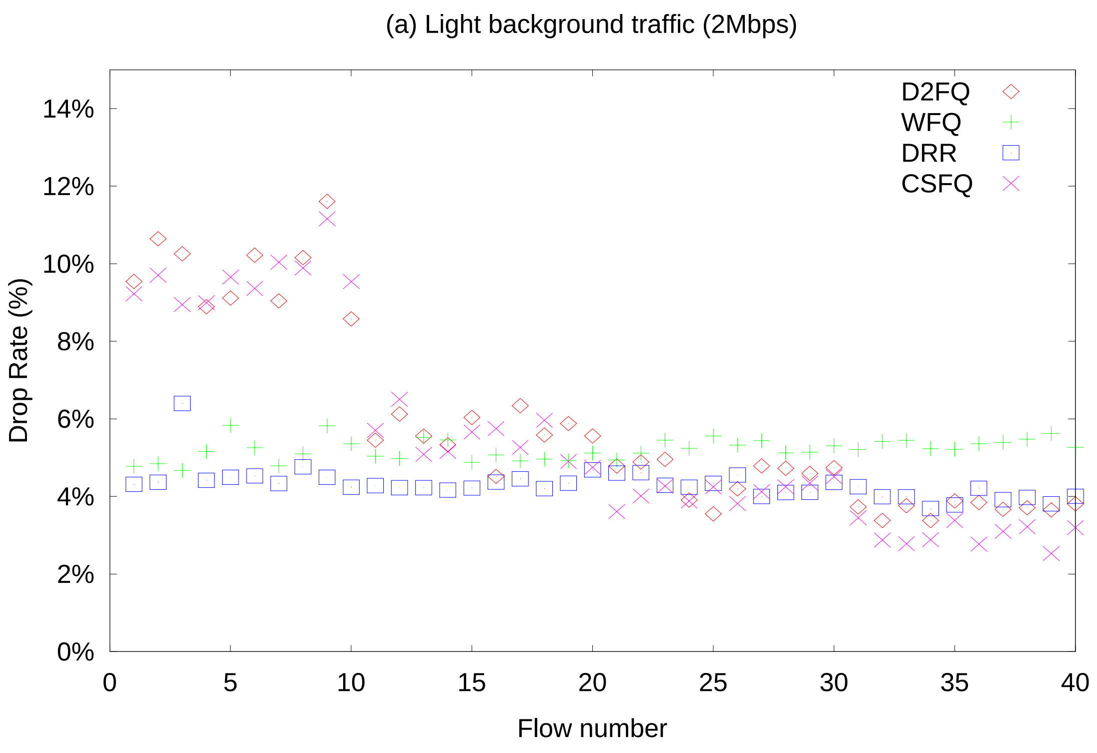
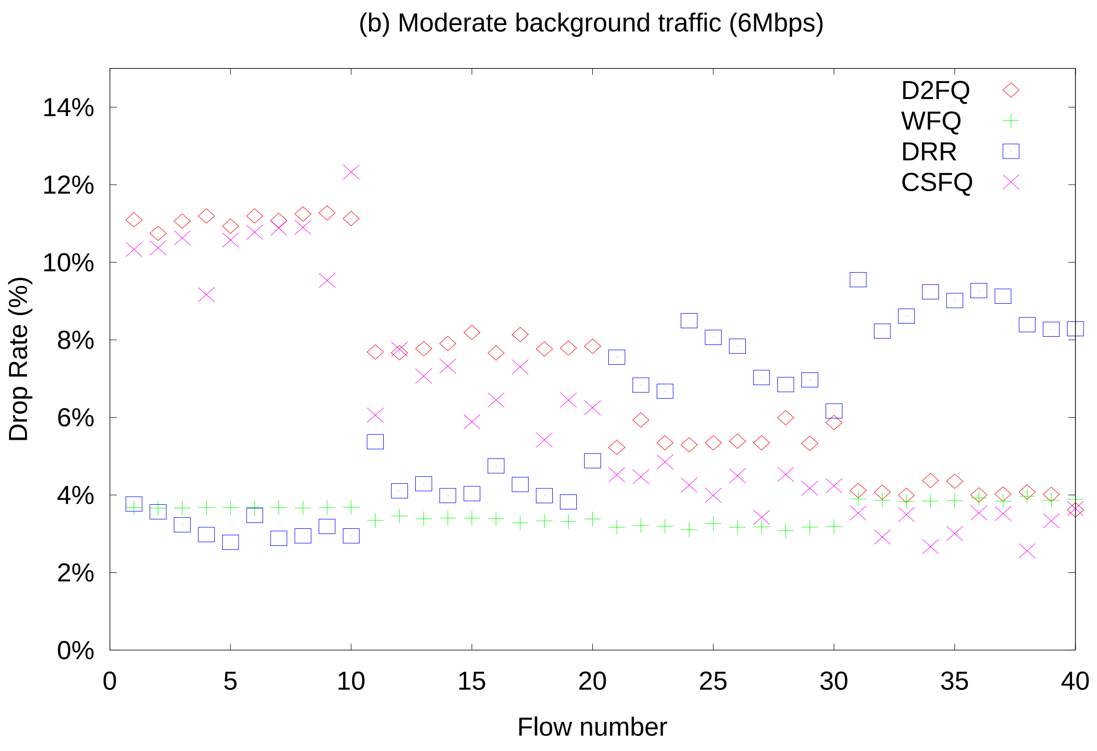


Figure 6-6 Normalized Thoughput of TCP flow with varid thoughput weight. (a) Light background traffic, (b) Moderate background traffic, (c) Heavy background traffic


(a) Light background traffic (2Mbps)
D2FQ ◇
WFQ +
DRR □
CSFQ ×
Drop Rate (%)
0%
2%
4%
6%
8%
10%
12%
14%
0
5
10
15
20
25
30
35
40
Flow number


(b) Moderate background traffic (6Mbps)
D2FQ ◇
WFQ +
DRR □
CSFQ ×
Drop Rate (%)
0%
2%
4%
6%
8%
10%
12%
14%
0
5
10
15
20
25
30
35
40
Flow number

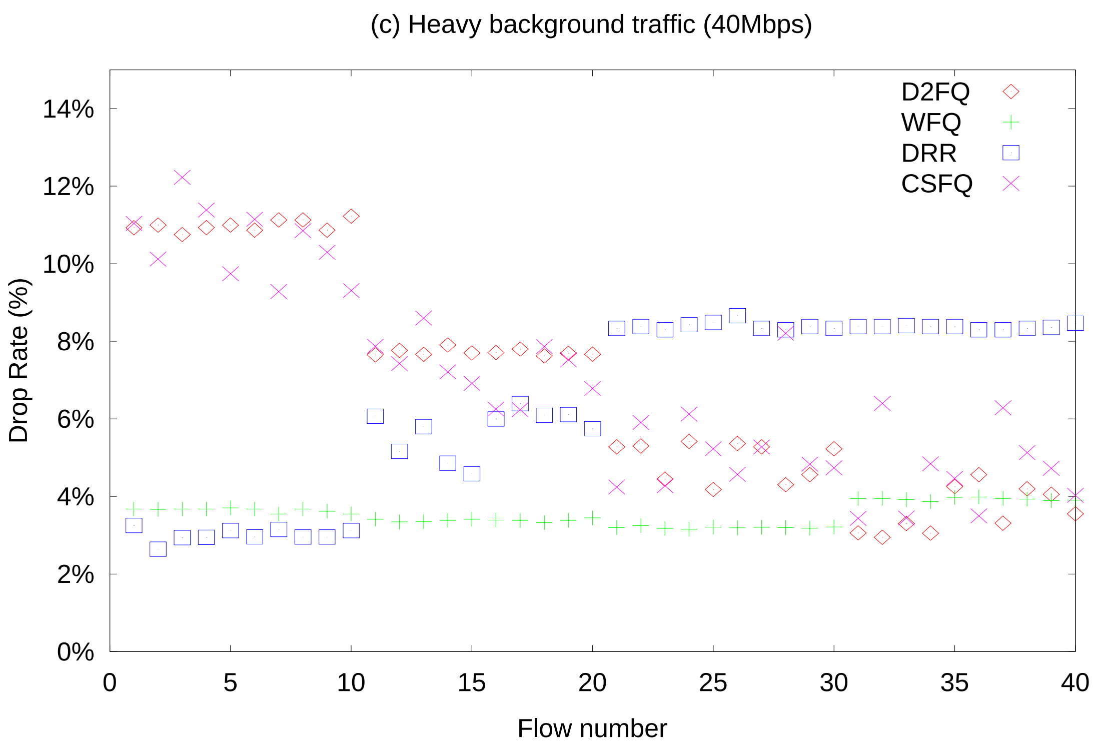
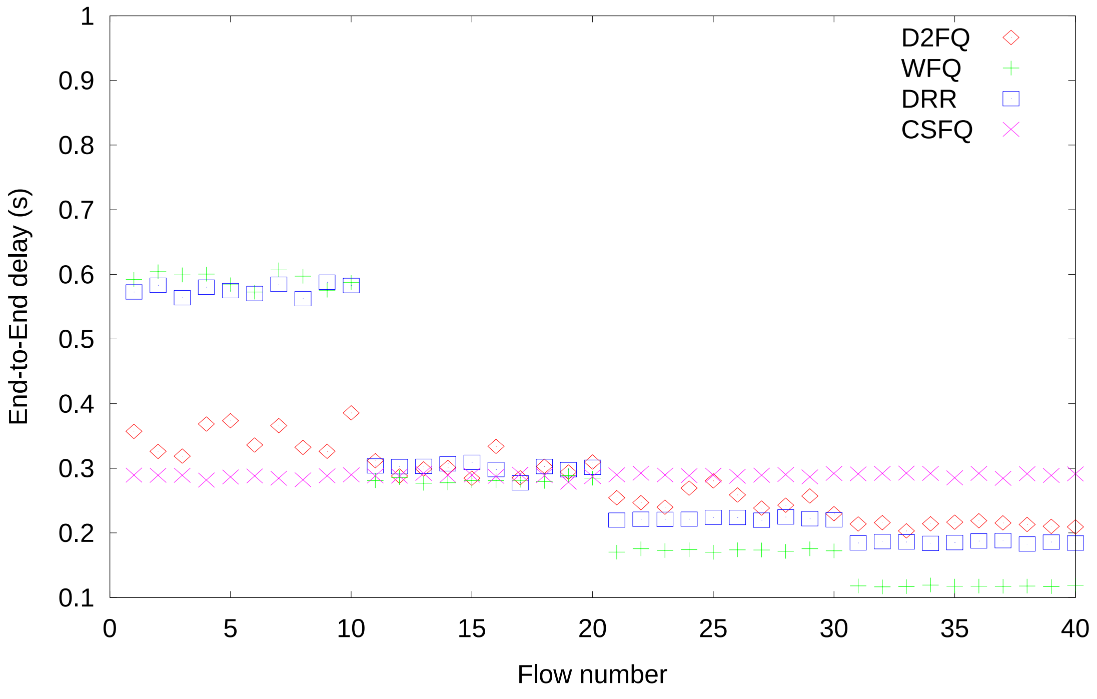
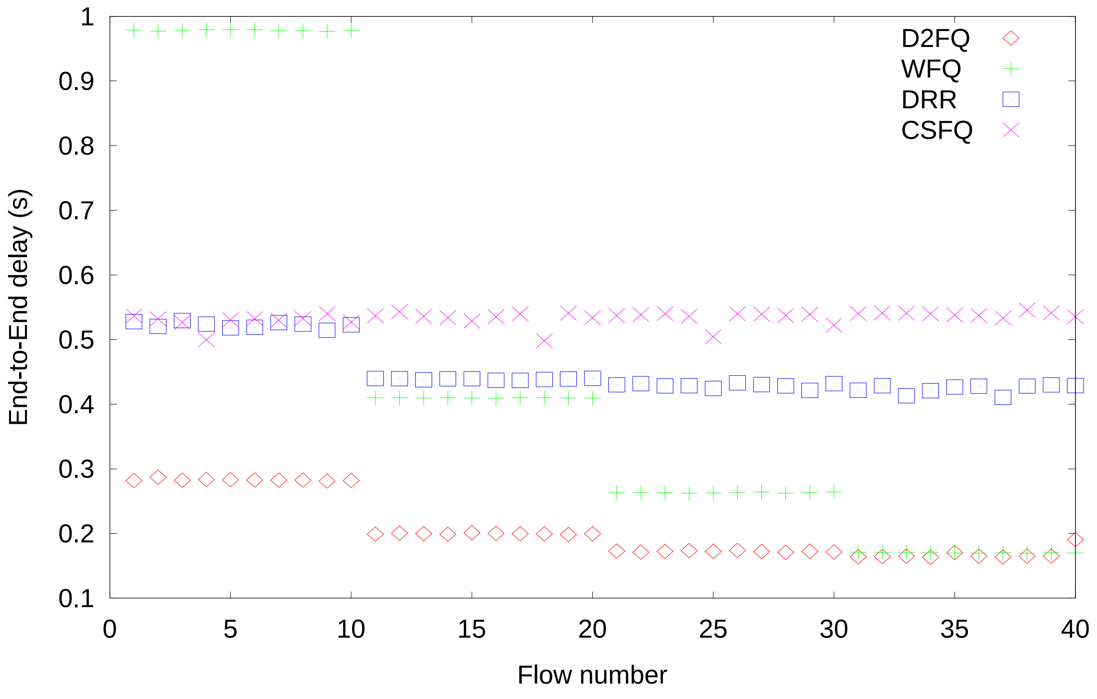


Figure 6-7 Drop rate of TCP flow with varid thoughput weight. (a) Light background traffic, (b) Moderate background traffic, (c) Heavy background traffic

(a) Light background traffic (2Mbps)



(b) Moderate background traffic (6Mbps)



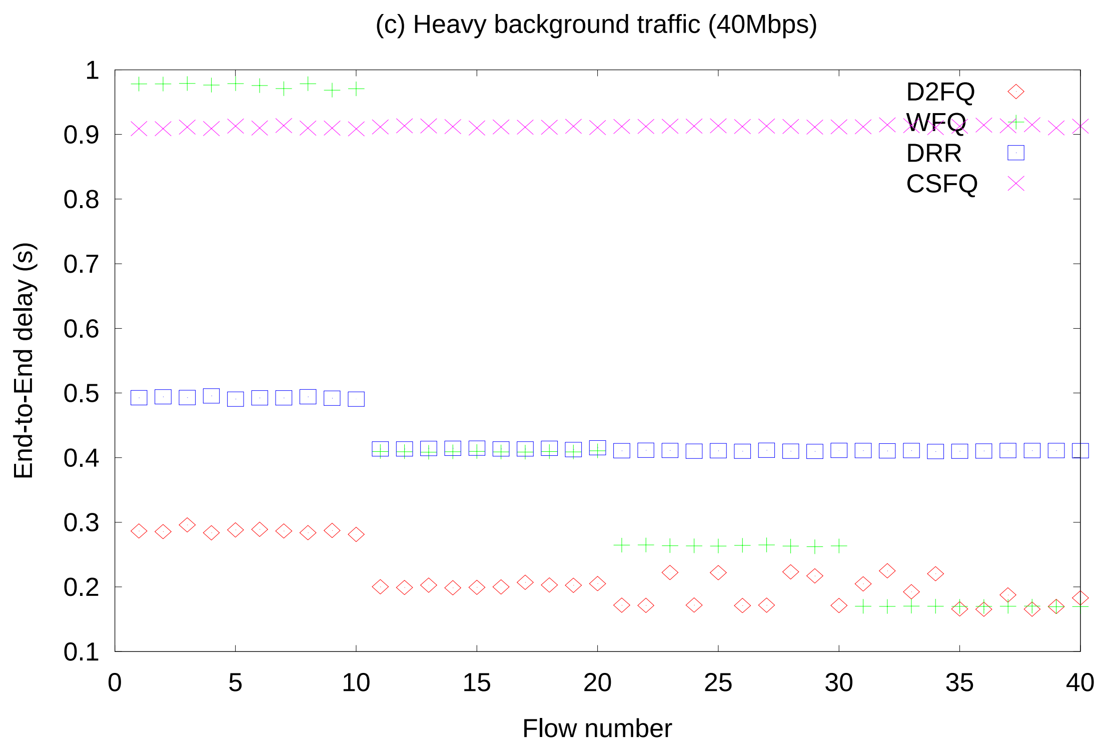


Figure 6-8 End-toEnd delay of TCP flow with varid thoughput weight

## 6.4 Competition Between UDP and TCP flow

In the real world network, UDP and TCP connection will compete for network resource simultaneously. From the view of UDP flow, TCP flow is “non-conforming traffic” that TCP source will try its best to send more packet until encounter packet dropped. A good scheduler will favor all flows including TCP and UDP in bandwidth allocation that is complied with its throughput weight. However, there is some non-conforming flows congesting a node which extend the long queueing delay and reduce the received bandwidth of conforming flows.

This section would show the competition of UDP and TCP flow under different background traffic conditions. The follow is the condition detail of this simulation.

- The simulation topology of is same to Figure 6-1. There is five congested nodes, $h_1$ to $h_5$, between source and sink.
- There is totally 40 flows transfer packets from source, $S_1$ ...$S_{40}$ to sink $h_7$
- The 40 flows divide into four groups and each group have 1o flow. i.e. 1st group to 4th group and its throughput weight of 1st group to 4th group are 1, 2, 3 and 4 respectively.
- In each group, the first five flows are new-Reno TCP source, another five flows are UDP, i.e. x1st to x5th are TCP flows and x6th to x10th are UDP flow where x is the group number.
- All UPD would send the constant arrival rate in the simulation which arrival rate is complied with its throughput weight.
- For each congested node $h_k$, 40 background CBR flow would transfer from $M_{k1\sim k40}$ *to* $D_k$. Each background CBR have same throughput weight, i.e. 1
- Total throughput weight of scheduler in each node is (1+2+3+4)*10 + 1*40 = 140 and each weight can reserve 10Mbps / 140 = 71.4kbps.
- UPD would send the constant arrival rate in the simulation. The arrival rate of UDP flow in 1st group to 4th group is 70kbps, 140kbps, 210kbps, 280kbps respectively..
- The simulation conduct separately under three offered loads of background traffic that is light (20%), normal (60%), and heavy offered load, (400%)
- In each congested node, $h_k$, offered load is controlled by the arrival rate of 40 ‘s CBR-Mk1 to CBR-Mk10.

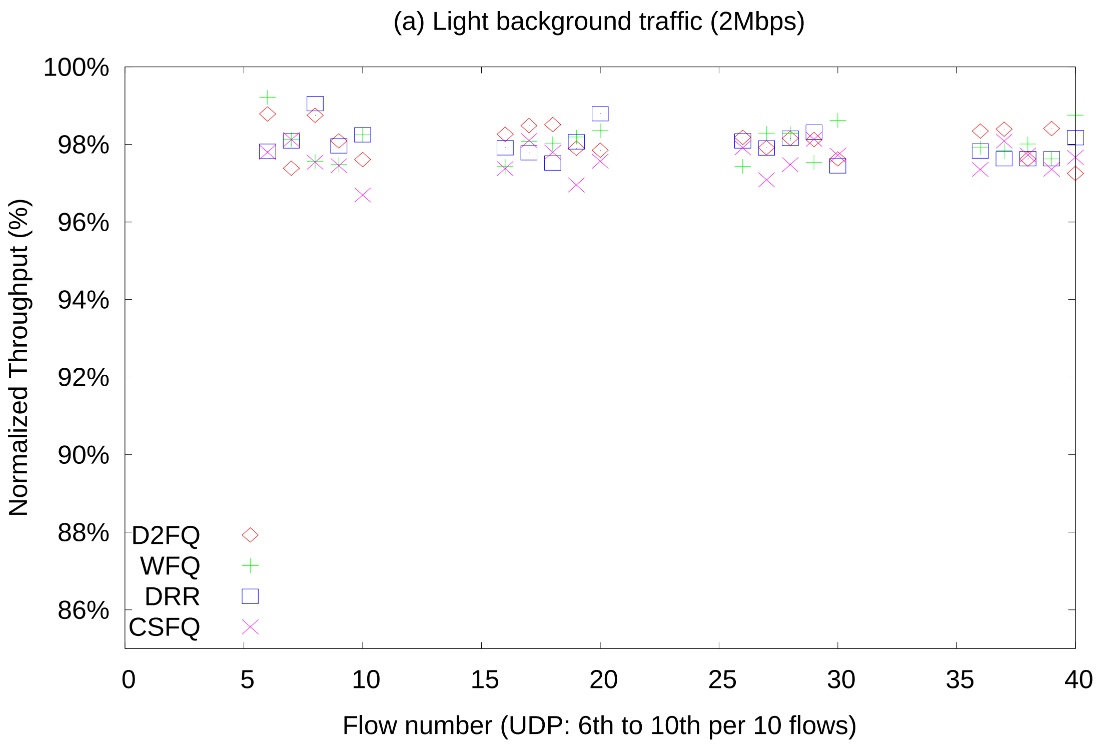
- The background traffic of offered load is 2Mbps, 6Mbps and 40Mbps respectively

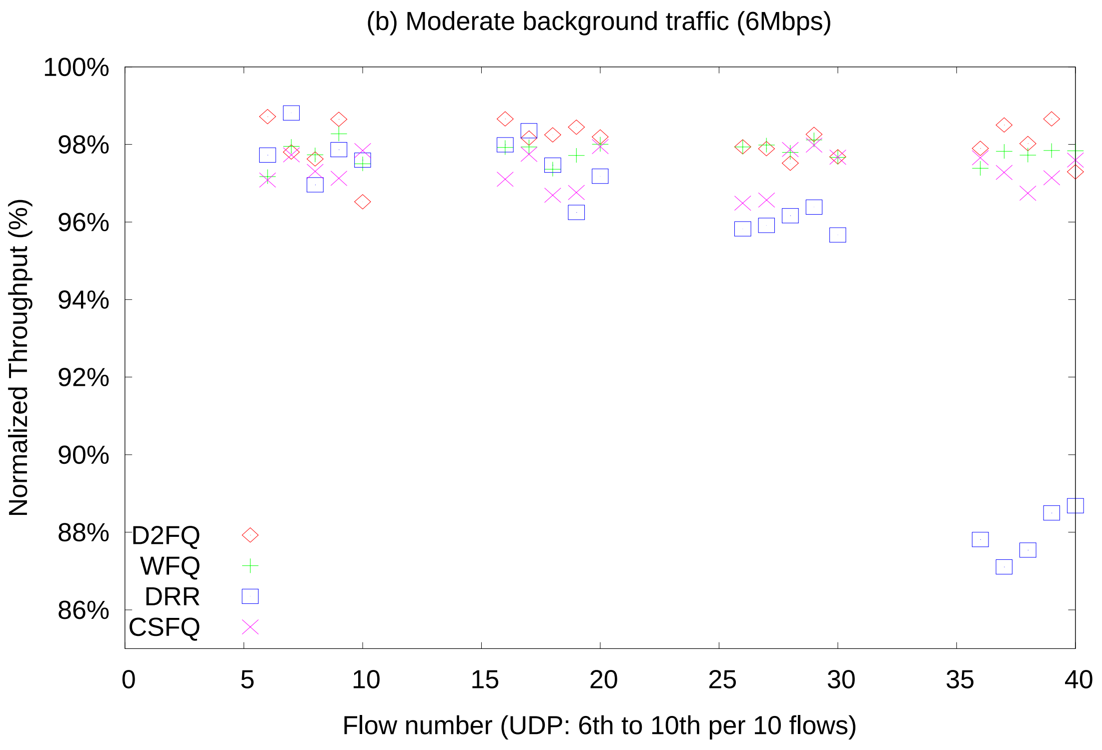
Figure 6-9 shows the simulation result of normalized throughput of TCP and UDP flows. Same as the previous simulation, the result show the normalized throughput in $D^2$FQ and WFQ scheduler is fair, especially under the moderate and heavy background traffic condition, its normalized throughput of all flows is close to 100%. In the light background traffic condition, the TCP flow shared the reminder bandwidth and thus the normalized throughput of TCP flow is higher than UDP flow. For the DRR scheduler and CSFQ scheduler, both schedulers cannot fairly share the bandwidth accord to the flow's throughput weight and DRR even cannot protect the UDP. The figure show 36th to 40th UDP flow in DRR received less bandwidth under moderate and heavy background traffic condition.

In this simulation, the UDP flows are conforming flow. Their arrival rate is complied with their throughput weight. Thus, there should be no packet drop for UDP flow. Figure 6-10 shows the drop rate of UDP flow in WFQ and $D^2$FQ is very close to 0%. That implies both WFQ and $D^2$FQ could protect the conforming flow under congested node. The drop rate of TCP flow between WFQ and $D^2$FQ is very different. WFQ have about 4% drop rate for all TCP flows regardless of the flow's throughput weight. However, the drop rate of TCP flow in $D^2$FQ is incurred by throughput weight that is also same as the result of section 6.3.

Generally, real-time network application will transfer data by UDP datagram. Therefore, the end-to-end delay of UDP flow will decide the quality of connection. Figure 6-11 shows the end-to-end delay of all flows. In $D^2$FQ, the end-to-end delay of UDP flow is much lower than end-to-end of TCP flow. Moreover, the end-to-end delays of the all flows are independent to its throughput weights. In WFQ, the end-to-end delay of UDP and TCP are correlative with flow's throughput weight. The flow with small throughput will get long delay. The $D^2$FQ can keep all UDP flows at the same end-to-end delay under light and moderate background traffic. Although the end-to-end delay of UDP under heavy background traffic is correlative with its throughput weight, the range of delay level in $D^2$FQ is much less than the range of delay level in WFQ.

Figure 6-12 shows the cumulative packet of end-to-end delay of TCP flow in $D^2$FQ. The Figure shows the cumulative packet of one TCP flow in each group. According to the figure, we can see the result data form a ripple-like line. The number of ripple in the line can tell us how many nodes are congested when the flow is transferring packet from source to destination. In the Figure 6-12, the ripple of delay is shown at 0.25sec at 30%, 0.4 sec at 70% and 0.55 sec at 100%. That means there is about 30% packet will receive 0.25second end-to-end delay, about 40% packet (70%-30%) receives 0.4 second end-to-end delay and about 30% packet (100-70%) received 0.55 second. Three clear-cut delay level implied that an unfortunate TCP packet will encounter three‘s long-queue node. Figure 6-13 shows the cumulative packet of end-to-end delay of TCP flow in WFQ. The result shows each flow produced five ripple that implied an unfortunate TCP packet will encounter five‘s long-queue node. Moreover, the first ripple of 1st flow is at 0.1 sec but the first ripple of 31st flow is at 0.4 sec. The unfairness in delay treatment is caused by the buffer management in WFQ that large throughput weight flow can get fast service but small throughput weight flow gets slow service. This simulation show $D^2$FQ can isolate UDP and TCP in term of throughput and end-to-end delay. Moreover, $D^2$FQ can decouple the two QoS performance criteria, throughput and delay.


(a) Light background traffic (2Mbps)
Normalized Throughput (%)
100%
98%
96%
94%
92%
90%
88%
86%
D2FQ
WFQ
DRR
CSFQ
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)


(b) Moderate background traffic (6Mbps)
Normalized Throughput (%)
100%
98%
96%
94%
92%
90%
88%
86%
D2FQ
WFQ
DRR
CSFQ
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)

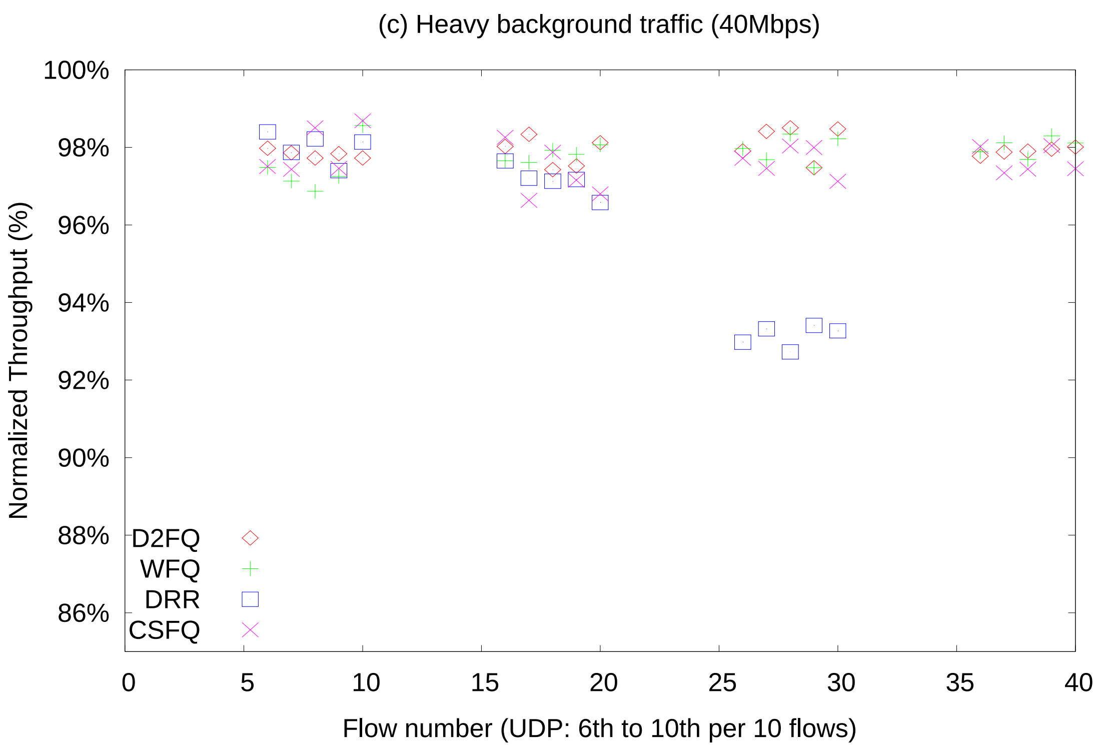
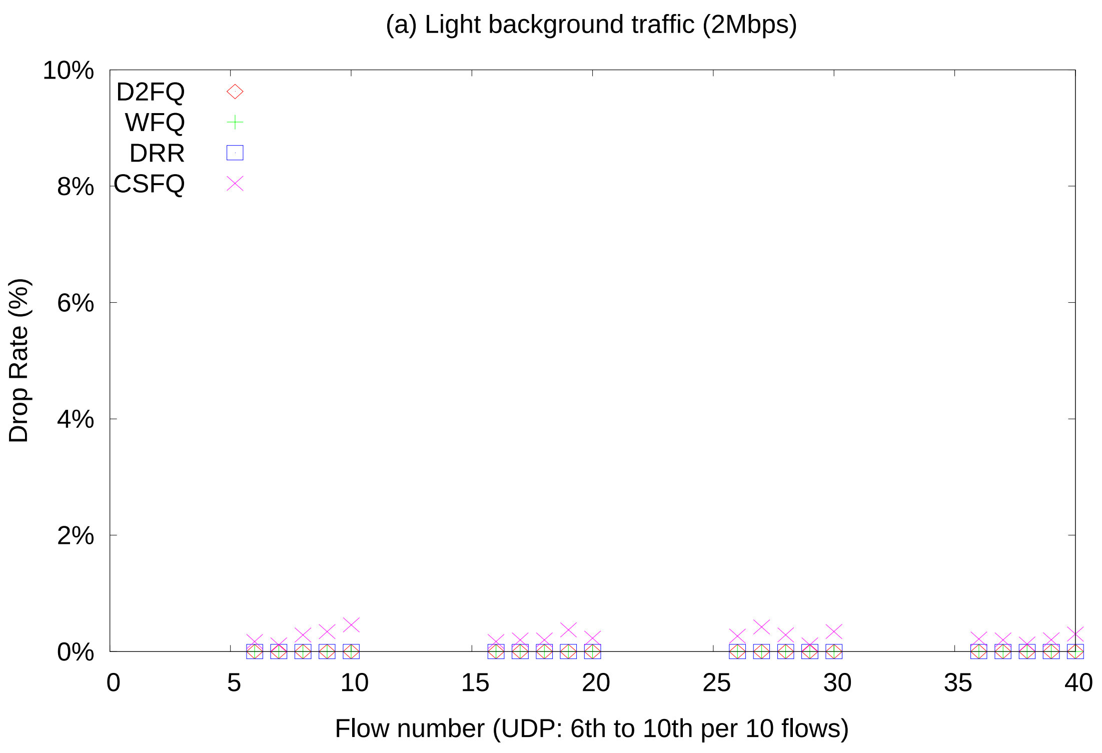
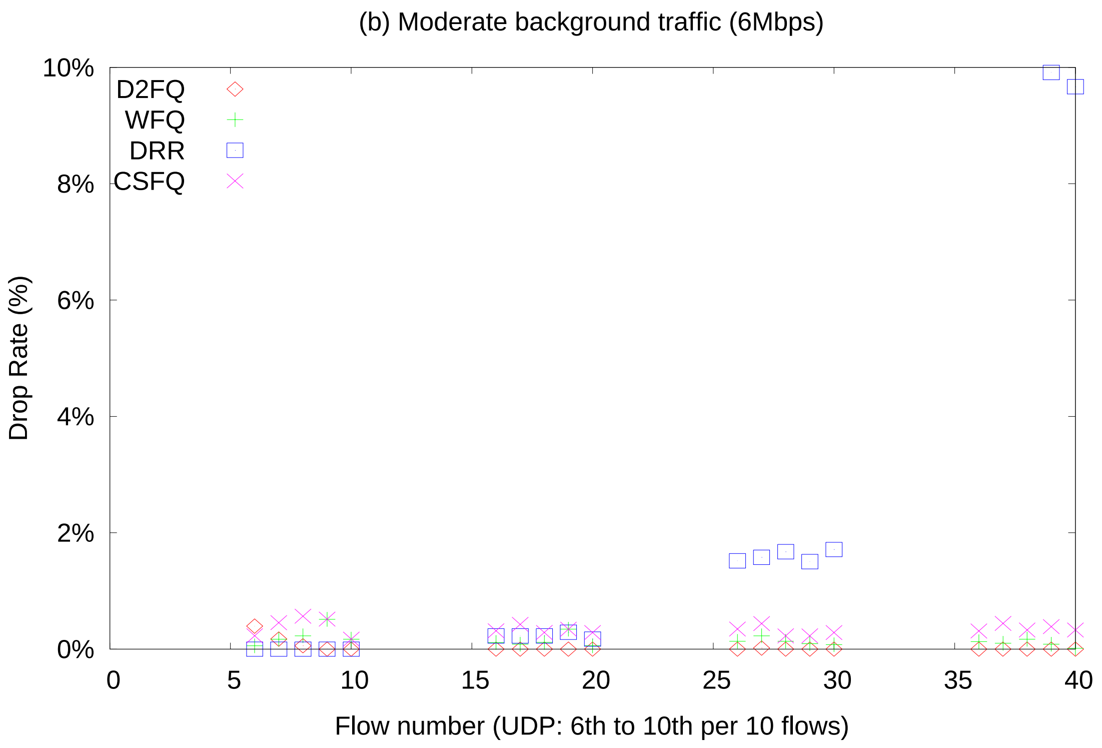


Figure 6-9 Normalized Thoughput of UDP flows. (a) Light background traffic, (b) Moderate background traffic, (c) Heavy background traffic.


(a) Light background traffic (2Mbps)
Drop Rate (%)
10%
8%
6%
4%
2%
0%
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)
D2FQ ◇
WFQ +
DRR □
CSFQ ×


(b) Moderate background traffic (6Mbps)
Drop Rate (%)
10%
8%
6%
4%
2%
0%
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)
D2FQ ◇
WFQ +
DRR □
CSFQ ×

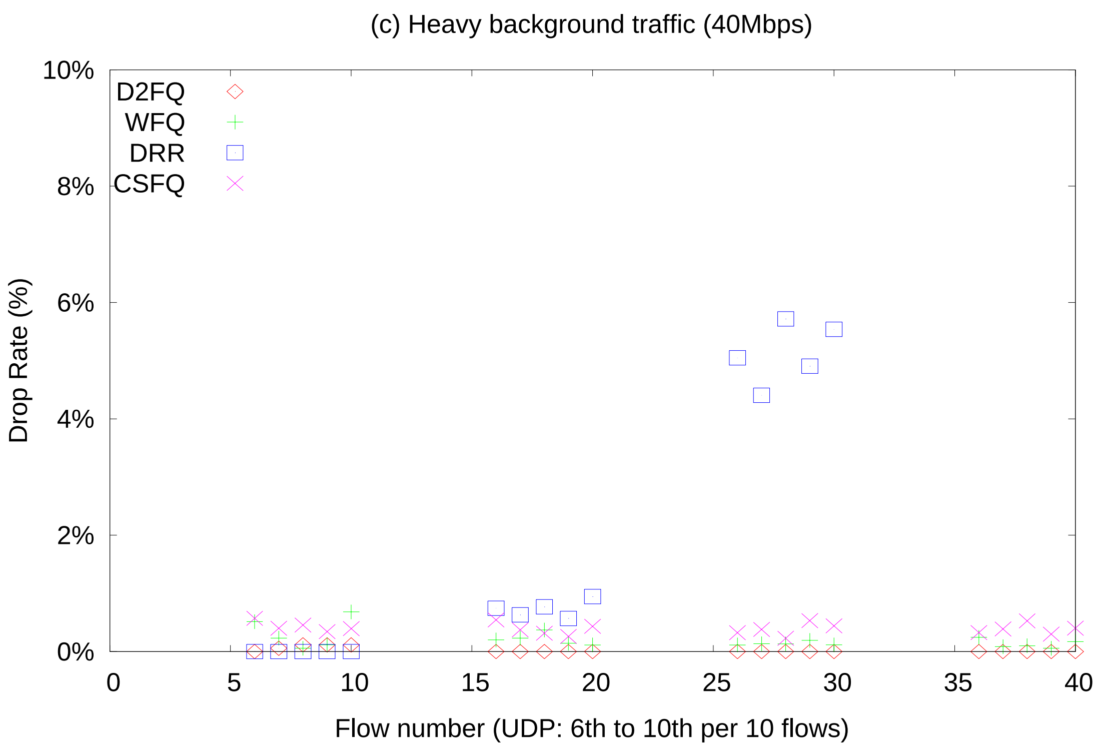
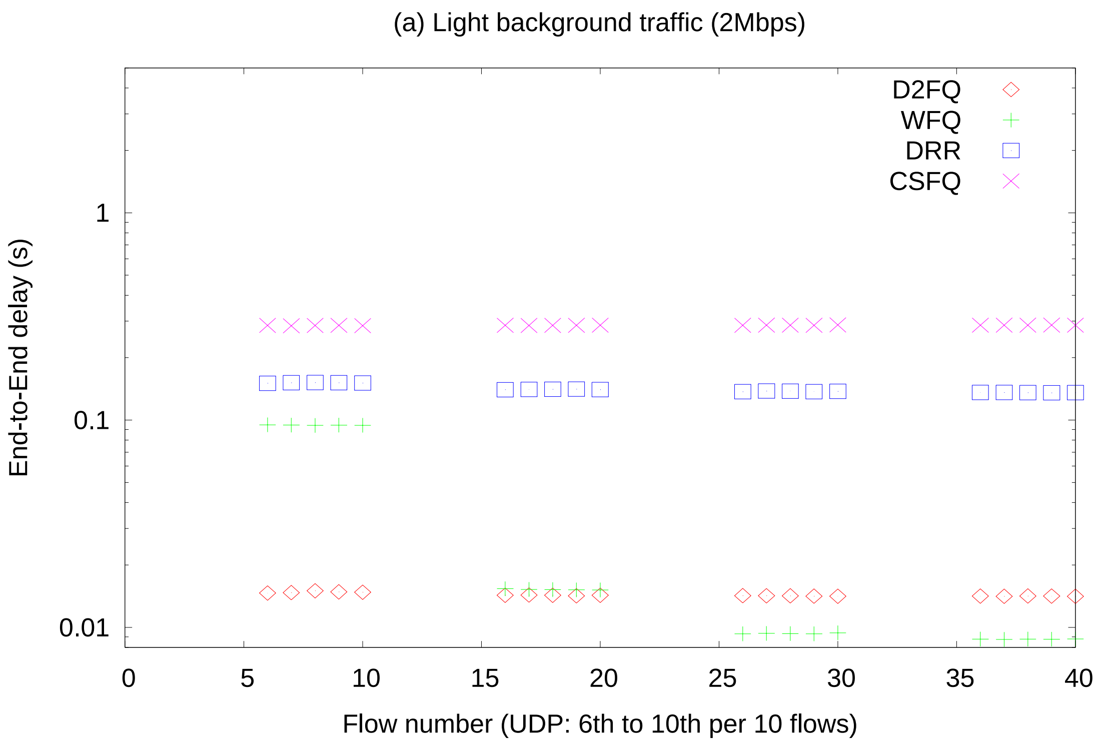
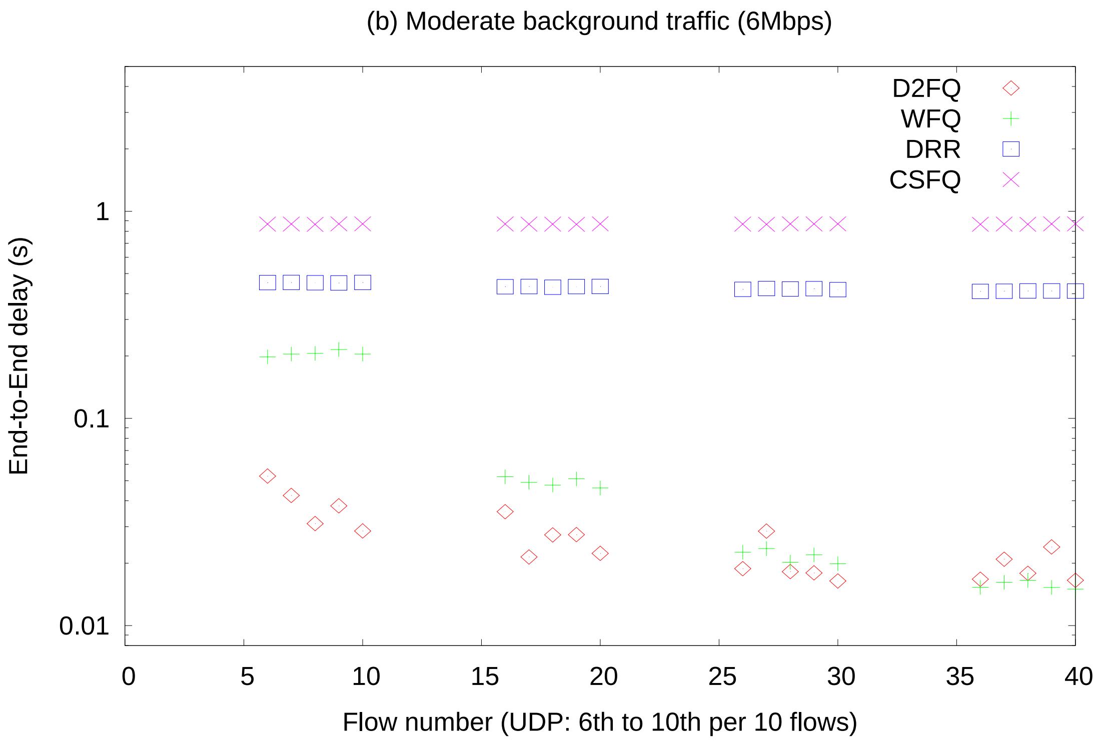


Figure 6-10 Drop rate of UDP flows with varied throughput weight. (a) Light background traffic, (b) Moderate background traffic, (c) Heavy background traffic.


(a) Light background traffic (2Mbps)
D2FQ ◇
WFQ +
DRR □
CSFQ ×
End-to-End delay (s)
1
0.1
0.01
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)


(b) Moderate background traffic (6Mbps)
D2FQ ◇
WFQ +
DRR □
CSFQ ×
End-to-End delay (s)
1
0.1
0.01
0
5
10
15
20
25
30
35
40
Flow number (UDP: 6th to 10th per 10 flows)

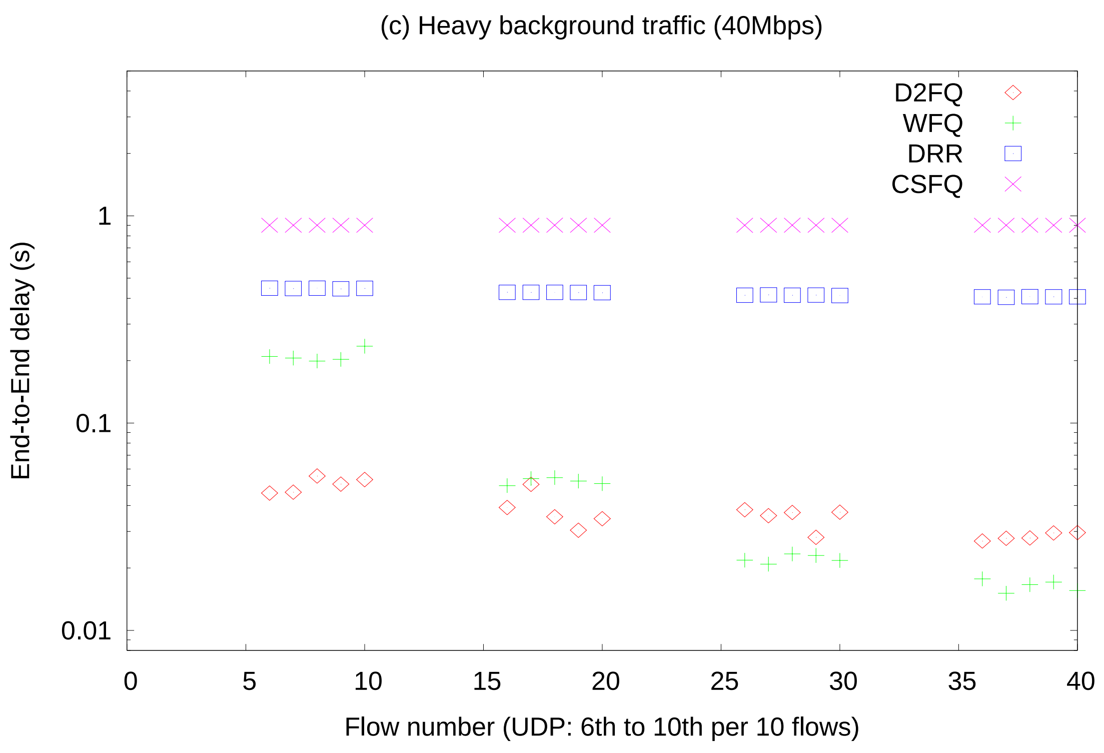
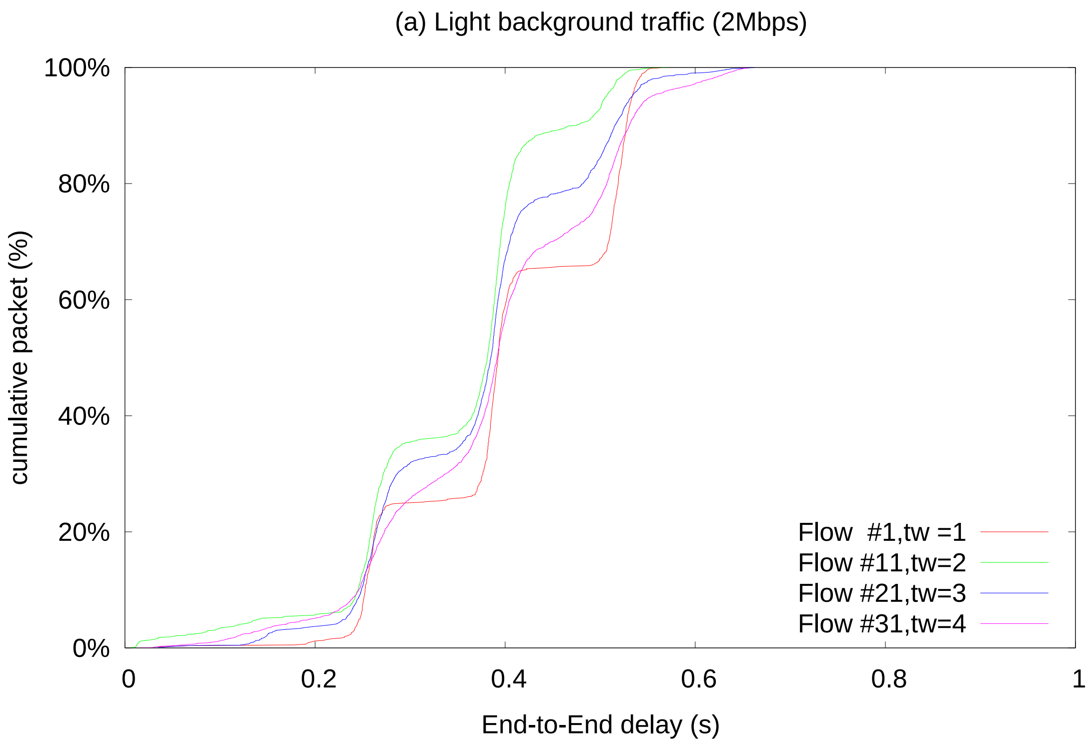
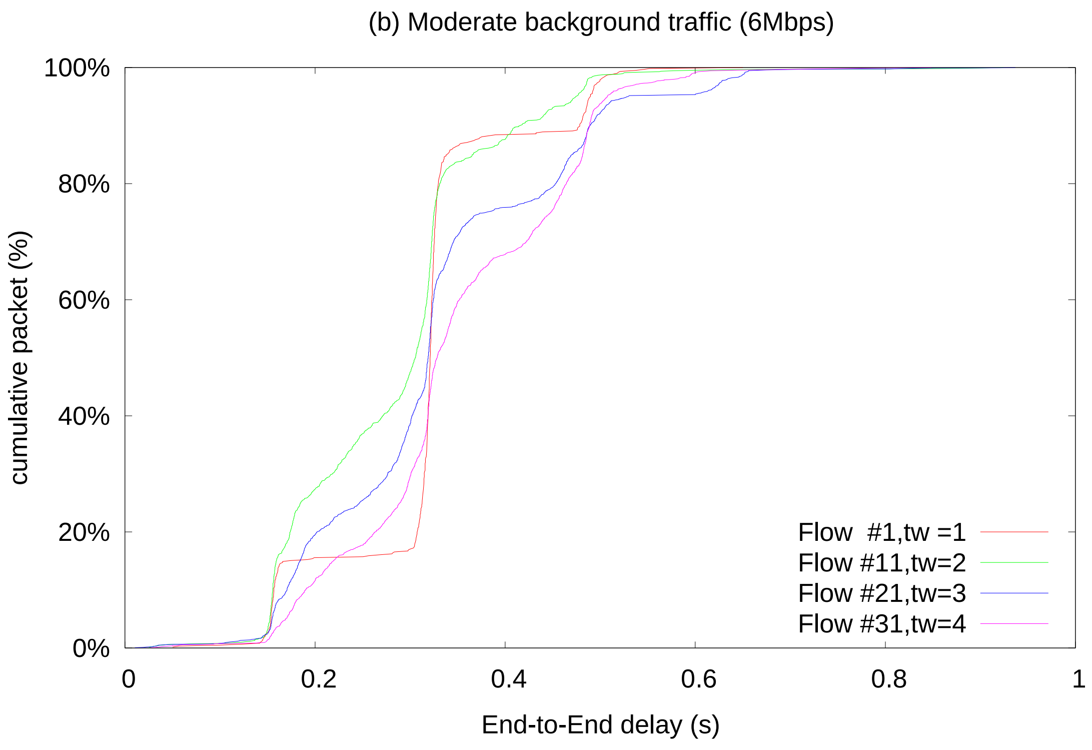


Figure 6-11 End-to-End delay of UDP flow with varid thoughput weight. (a)Light background traffic, (b)Moderate background traffic, (c)Heavy backgroundtraffic.


(a) Light background traffic (2Mbps)
cumulative packet (%)
End-to-End delay (s)
0%
20%
40%
60%
80%
100%
0
0.2
0.4
0.6
0.8
1
Flow #1,tw =1
Flow #11,tw=2
Flow #21,tw=3
Flow #31,tw=4


(b) Moderate background traffic (6Mbps)
cumulative packet (%)
End-to-End delay (s)
0%
20%
40%
60%
80%
100%
0
0.2
0.4
0.6
0.8
1
Flow #1,tw =1
Flow #11,tw=2
Flow #21,tw=3
Flow #31,tw=4

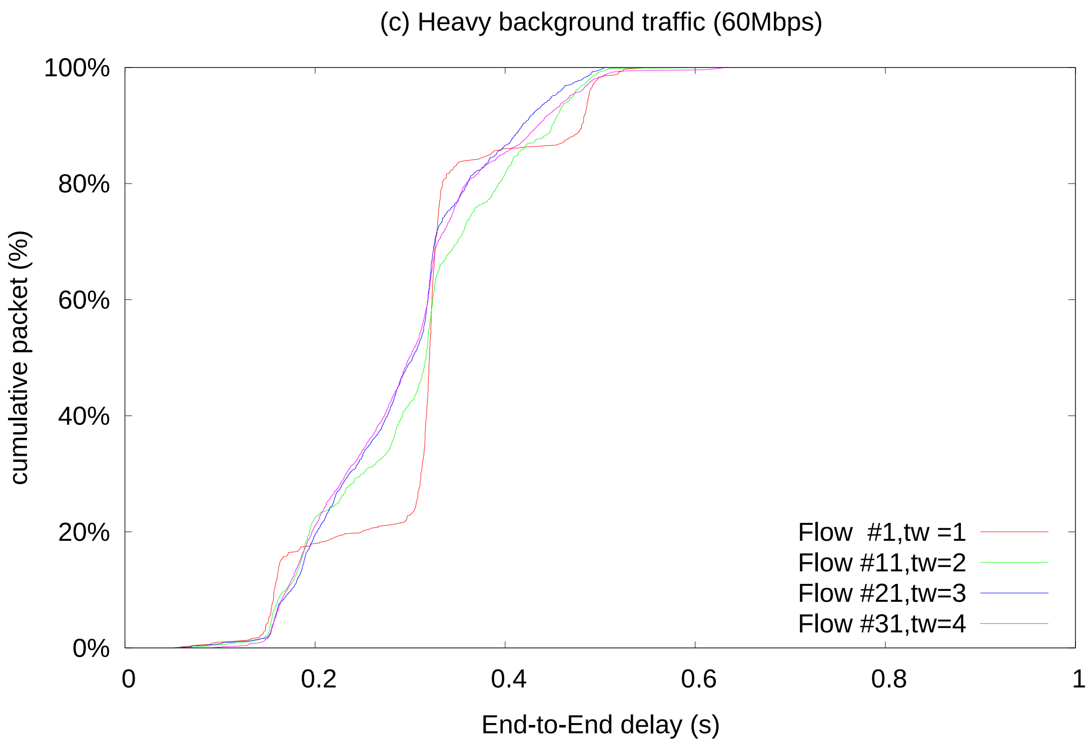
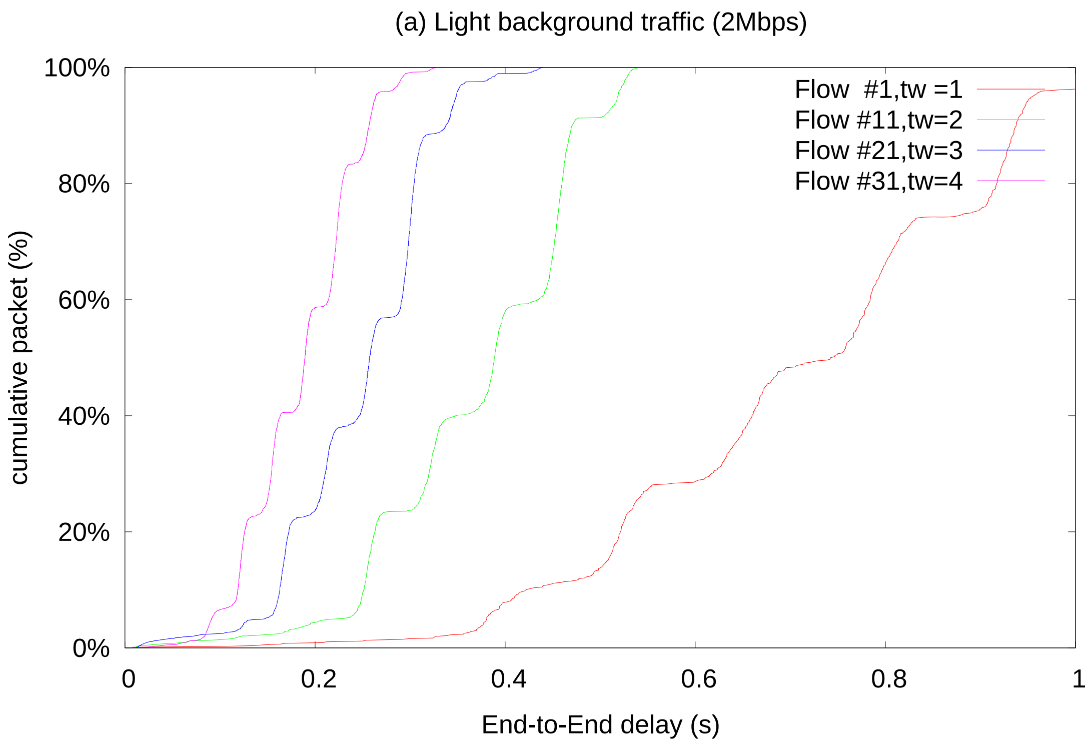
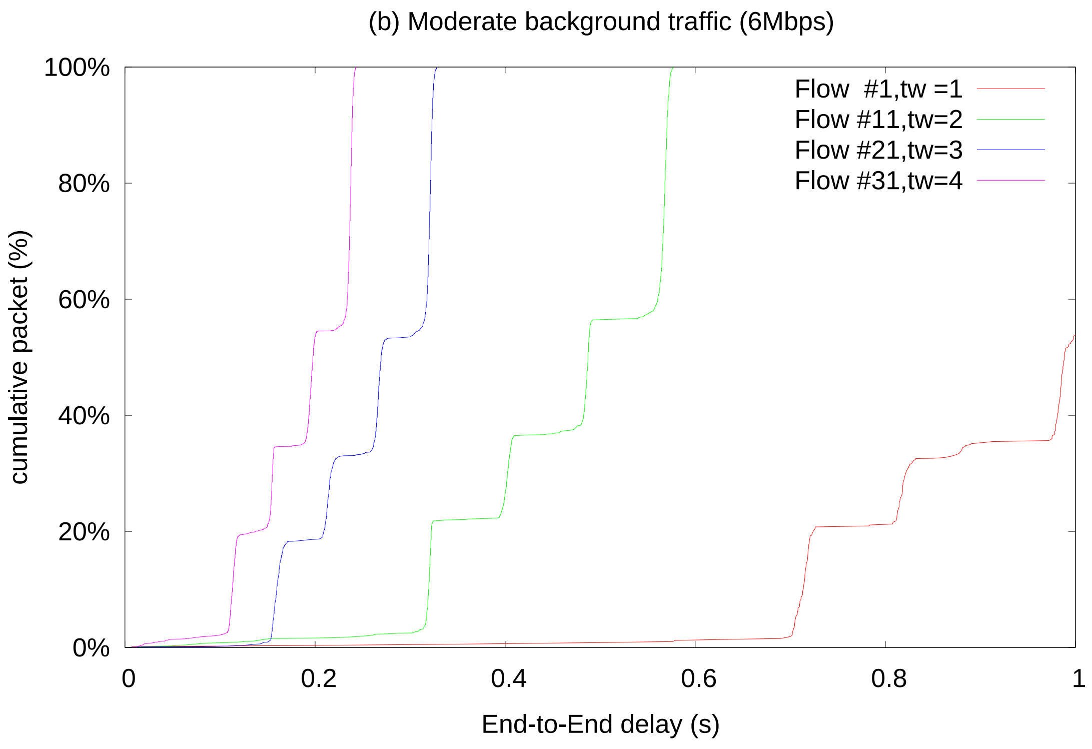


Figure 6-12 Cumulative distribution of End-to-End delay of TCP packet in D$^2$FQ scheduler. (a) Light background traffic, (b)Moderate background traffic, (c)Heavy background traffic.


(a) Light background traffic (2Mbps)
cumulative packet (%)
End-to-End delay (s)
0%
20%
40%
60%
80%
100%
0
0.2
0.4
0.6
0.8
1
Flow #1,tw =1
Flow #11,tw=2
Flow #21,tw=3
Flow #31,tw=4


(b) Moderate background traffic (6Mbps)
cumulative packet (%)
End-to-End delay (s)
0%
20%
40%
60%
80%
100%
0
0.2
0.4
0.6
0.8
1
Flow #1,tw =1
Flow #11,tw=2
Flow #21,tw=3
Flow #31,tw=4

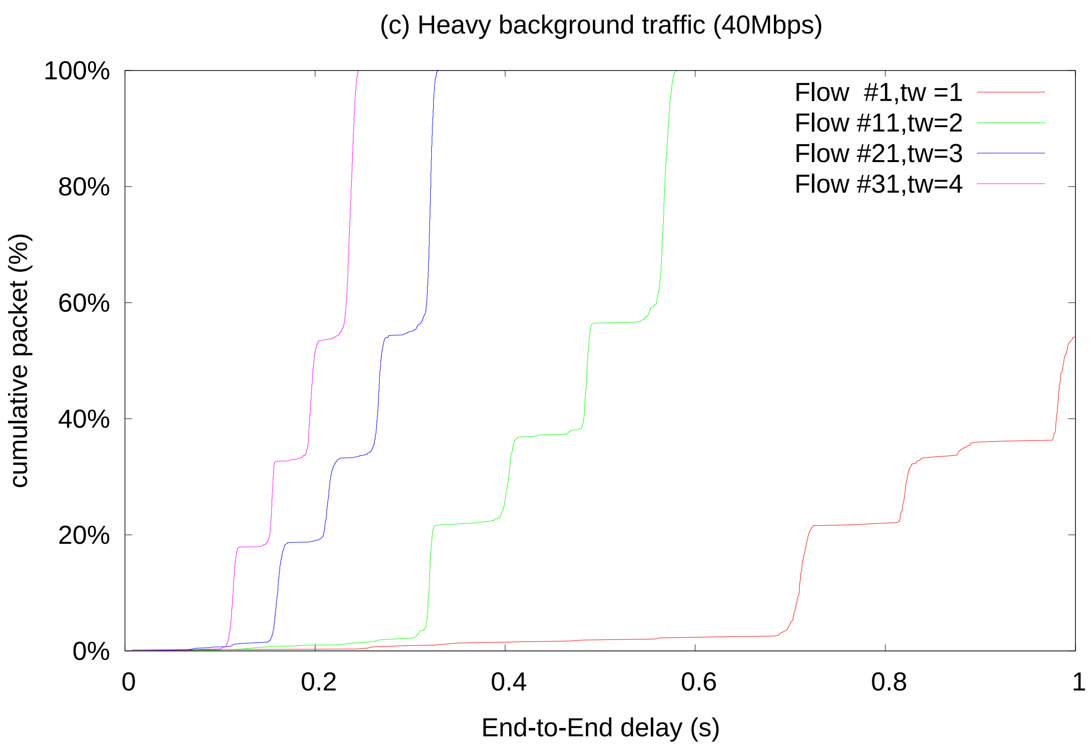


Figure 6-13 Cumulative distribution of End-to-End delay of TCP packet in WFQ scheduler. (a) Light background traffic, (b)Moderate background traffic, (c)Heavy background traffic.

# Chapter 7. Conclusion

In this thesis, one has presented a new, efficient and simple scheduling algorithm called Delay-Differentiable Fair Queuing ($D^2FQ$). $D^2FQ$ is designed to overcome the coupling effect between throughput and delay. $D^2FQ$ provides a fair bandwidth sharing and mean delay differentiation. The throughput fairness is the same as many other scheduling algorithms, but the mean delay differentiation is quantifiable so that it can achieve different delay requirements among flows.

One has presented the requirements of network applications in the real world. The traditional best-effort network cannot fulfill various requirements while Quality-of-Service (QoS) is the correct direction to improve the development of network applications. Scheduling algorithm in network node plays a critical role in providing the QoS with guarantees required by various network applications, especially real-time applications. One also presents the scheduling discipline and the constraints of the queuing delay treatment in scheduling, such as conservation law in total mean delay, and the tradeoff between the throughput and delay in the heavy loaded node.

Therefore, one proposes a scheduling algorithm, $D^2FQ$, which achieves fair bandwidth sharing by the quantum concept of DRR with throughput weight, $tw_i$. The delay differentiability uses an extra delay-slot, and each flow hops based on its delay weight, $dw_i$. $D^2FQ$ punishes the non-conforming flows not only on the fair bandwidth sharing but also by means of the longer delay. The buffer management and delay differentiability in $D^2FQ$ can protect all conforming flows. Moreover, the delay differentiability does not induce further complexity

We have proved analytically the fairness properties of $D^2FQ$. The difference in the bandwidth allocated to any two backlogged flows is bounded by ($3Q+2M$), where Q is the quantum of the round and $M$ is the maximum packet size of the outgoing link. $D^2FQ$ also belongs to the class of *LR* server. *LR* server is characterized by the latency bound to describe the worst-case behavior of scheduling algorithms and *LR* server. We have shown that the work complexity of $D^2FQ$ is independent of the number of flows, i.e. O(log $L$), where L is the size of the delay-slot. Therefore, $D^2FQ$ can be easily implemented in networks and the delay-slot of $D^2FQ$ can be implemented by hardware. Then the complexity O(log $L$) can be transformed from time domain into space domain.

We have studied the performance of $D^2FQ$ in a single-node topology and a multi-node topology. The performance study of a connectionless traffic in $D^2FQ$ scheduler conducts in a single-node topology. In the fairness study on bandwidth allocation, $D^2FQ$ scheduler can allocate bandwidths precisely according to the flow's throughput and its performance in terms of bandwidth allocation is very close to the well-known rate scheduler, WFQ. The buffer management in $D^2FQ$ also protects the conforming flows, which have no packet drops under the condition of overloading. At the same time, the working complexity of $D^2FQ$ is only O(log $L$), which is lower than the working complexity of WFQ of O(N), where N is number of flow. $D^2FQ$ can also isolate the performance criteria, i.e. throughput and delay. Compared with the delay in WFQ scheduler, the mean delay of flow in $D^2FQ$ is not incurred by throughput weight. The flow with same delay weight can receive same mean delay in $D^2FQ$.

We use various delay weights among the flows to evaluate its delay differentiability during increasing the offered load. In the simulation of varied delay weights, the results show the mean delay of flow is consistent with its delay weight. Moreover, all non-conforming flows are punished by long delay bound, and delay performance of conforming flow is influenced slightly under heavy congestion.

In the study of varied shares of non-conforming flows in the congested node, the received delay in $D^2$FQ scheduler can be kept in low level. Although there are 99% of non-conforming traffics in a congested node, the conforming flows can be received with a low mean delay. Thus, $D^2$FQ scheduler can guarantee QoS under DDoS attack. The performance study of varied packet sizes is also conducted in single-node topology. The simulation results show that $D^2$FQ scheduler maintains a relatively fair performance in terms of mean delay.  The received mean delay from flows with different packet size only varies in a narrow range.

In the performance study of connection-oriented traffic on $D^2$FQ, multi-node topology is used. We used three famous TCP types, i.e. Tahoe, New-reno, and SACK, to evaluate the outgoing throughput of a TCP connection. The simulation results show that $D^2$FQ scheduler generally provides higher throughput than WFQ scheduler in these three TCP types of multi-node topology, especially in SACK-TCP. This result also reflects that $D^2$FQ is a TCP-friendly scheduler. This propriety is very important because most of network traffics in real world are TCP-traffics.

The study of connection-oriented traffics in $D^2$FQ scheduler also evaluates the performance of bandwidth allocation among TCP flows with different throughput weights. The result shows that $D^2$FQ scheduler can achieve fair bandwidth allocation for the flows with different throughput weights whereas WFQ scheduler is  unfair for the flows having low throughput weight, with which the received bandwidth is less than the reserved bandwidth of the flows. In the study of competition between UDP and TCP flows, $D^2$FQ scheduler also guarantees a good performance in terms of bandwidth allocation and mean delay. Conforming UDP flow is protected and no packet drops occur in $D^2$FQ scheduler. Thus, the mean delay of all conforming UDP flows is much lower than the mean delay of TCP flows. In addition, TCP flows can fairly share the reminder bandwidths according to the max-min share policy [24]. Thus, the link of the $D^2$FQ- equipped network can be fully utilized.

Recently, some real-time applications such as Voice over IP (VoIP) have become more popular.  Its traffic characteristic is low- bandwidth-and-low-delay (LBLD) requirements. In order to achieve the low mean delay, the router always needs to reserve more bandwidths than its actual requirement and the over-claimed reservation cannot fully utilize the bandwidth of the output link. Using the proposed $D^2$FQ algorithm, the requirements of delay and bandwidth can be isolated with the rarity of network resources, so the bandwidth can be fully utilized. On the other hand, these LBLD traffics are always grouped into a small number of classes, usually less than five. If some non-conforming traffic can “cheat” the scheduler server to obtain low delay service, then all flows in that class cannot be guaranteed to have a low delay service. $D^2$FQ can also handle many flows or classes because its complexity cost is low and the flow is isolated from other flows.

The decoupling between throughput and delay in $D^2$FQ can guarantee a lower mean delay for conforming flows, This characteristic of $D^2$FQ can keep a low delay service when the server is attacked by the DDoS-flooded packets. The quantifiable delay control mechanism also provides a new platform for business on the charging of the network services. The service for LBLD flows could become practicable.

# Appendix A Latency Rate Servers

In this section, some definitions and notations for understanding the concept of *LR* servers[8] is presented.

**Definition A.1** A system busy period *is defined as the maximal time interval during which the server is continuously transmitting packets*.

**Definition A.2** We define a flow as active during an interval of time, if at all instants of time during this interval, it has at least one packet awaiting service or being served.

We now define the notion a busy period, an essential component of the concept of *LR* servers.

**Definition A.3** A busy period of a flow is defined as the maximal time interval during which the flow is active if it served at exactly its reserved rate.

Let $\rho_i$ be the reserved rate for flow *i*. Also let $Arrived_i(t_1, t_2)$ denote the total number of bits of flow *i* that arrive at the scheduler during the time interval $(t_1, t_2)$. Consider an interval of time $(\tau_1, \tau_2)$ which represents a busy period for flow *i*. Then for any time interval $(\tau_1, t)$ such that $t \in (\tau_1, \tau_2)$, the number of bits that arrive during this interval is greater than or equal to the number of bits that would exit the scheduler if the flow received service at its reserved rate, $\rho_i$. In other words, for all $t \in (\tau_1, \tau_2)$,

$$Arrivedi(\tau_1, t) \geq \rho(t-\tau_1)$$

A graph of *Arrivedi*($\tau_1$, *t*) against time is plotted in figure A-1. Figure A-1 illustrates two busy periods, ($t_1$, $t_2$) and ($t_3$, $t_4$) for flow *i*. It is important to understand the basic difference between a session busy period and a session active period. The definition of the busy period supposes that flow *i* is served at the constant reserved rate, and therefore, depends only on the reserved rate of the flow and the packet arrival pattern of the flow. An active period of a flow, however, reflects the actual behavior of the scheduler where the instantaneous service offered to flow *i* varies according to the number of active flows. If during a busy period of flow *i*, the instantaneous service rate offered to flow *i* is greater than the allocated rate, then the flow may cease to be active. Thus, a busy period of a flow may include multiple active periods for that flow. The start of a busy period of a flow is always caused by the arrival of a packet belonging to the flow.

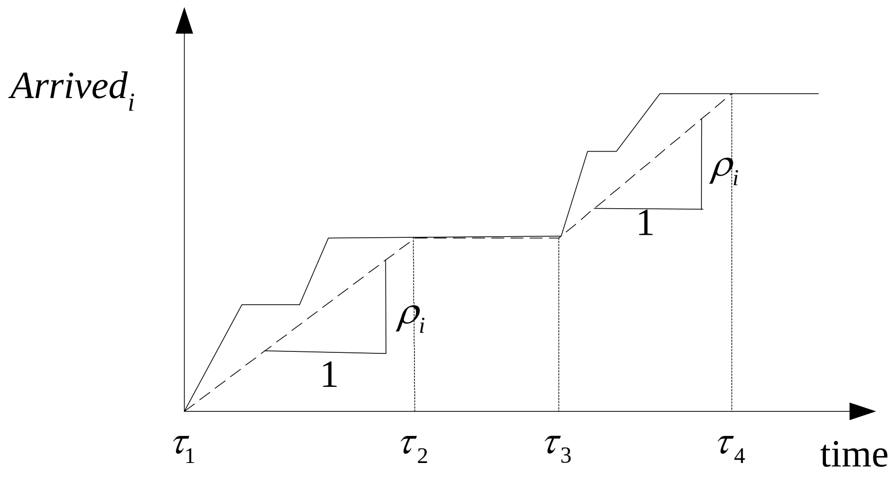


Figure A-1 Two busy periods for flow i

Note that, when the same traffic distribution is applied to two different schedulers with identical reserved rates, the ensuing active periods of the flows can be quite different. This makes it difficult to make use of active periods to analyze a broad class of schedulers. On the other hand, the busy period of a flow depends only on the arrival rate of the flow and its reserved rate. Therefore, the busy period can be used as an invariant in the analysis of different schedulers. It is because of this important property that the definition of an LR server is based on the service received by a flow during a busy period.

The following definitions lead to a formal notion of latency in the case of guaranteed rate servers. The reader is referred to [8] for a more detailed discussion.

**Definition A.4** Define $Sent_i(t_1, t_2)$ as the amount of service received by flow *i* during the interval $(t_1, t_2)$.

**Definition A.5** Let time instant $\alpha_i$ represent the start of a certain busy period for flow *i*. Let $t > \alpha_i$ be such that the flow is continuously busy during the time interval $(\alpha_i, t)$. Define $S_i(\alpha_i, t)$ as the number of bits belonging to packets in flow *i* that arrive after time $\alpha_i$ and are scheduled during the time interval $(\alpha_i, t)$.

Note that, $Sent_i(\alpha_i, t)$ is not necessarily equal to $S_i(\alpha_i, t)$. This is because, during this interval of time, the scheduler may still be serving packets that arrived during a previous busy period. $S_i(\alpha_i, t)$, therefore, is not necessarily the same as the total number of bits scheduled from flow i in this interval. We are now prepared to present the definition of latency in *LR* servers.

**Definition A.6** The latency of a flow is defined as the minimum non-negative constant £i that satisfies the following for all possible busy periods of the flow,

$$S_i(\alpha_i, t) \geq max\{0, \rho_i * ( t - \alpha_i - \Theta_i)\} \qquad \text{(A-1)}$$

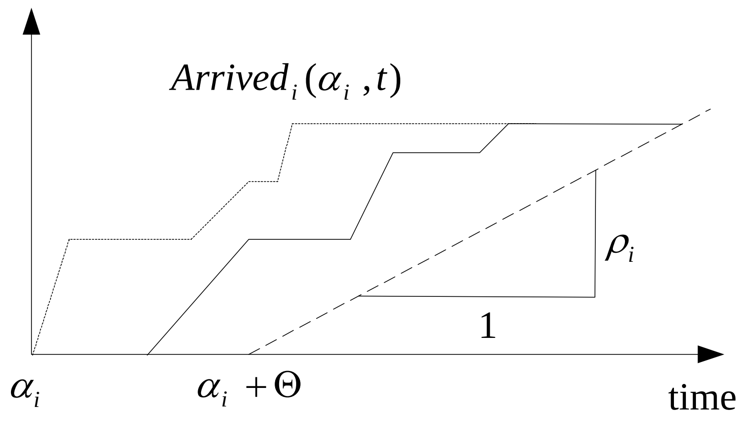


Figure A-2 An example of the behavior of an LR Server

As defined in [8], a scheduler which satisfies Equation (A-1) for some non-negative constant value of $\Theta_i$ is said to belong to the class of Latency Rate (*LR*) servers. The above definition captures the fact that the latency of a guaranteed-rate scheduler should not merely be the time it takes for the first packet of a flow to get scheduled, but should be a measure of the cumulative time that a flow has to wait until it begins receiving service at its guaranteed rate. A graph of $S_i(\alpha_i, t)$ against time is plotted in Figure A-2. The RHS of the above equation defines an envelop which bounds the minimum service received by a flow *i* during the busy period $(\alpha_i, t)$. The dashed line in figure A-2 corresponds to this envelop. For a particular scheduling algorithm several parameters such as its transmission rate on the output link, the number of the other flows sharing the link and their reserved rate may influence the latency. It has been proved in [8] that the maximum end-to-end delay experienced by a packet in a network of schedulers can be calculated from only the latencies

of the individual schedulers on the path of the connection and the traffic parameters on the connection that generated the packet. Since the end-to-end delay increases directly in proportion to the latency of the schedulers, the model highlights the significance of using low-latency schedulers for achieving low end-to-end delays.

# Reference

[1] A. Demers, S. Keshav, and S. Shenker, “Design and analysis of a fair queuing algorithm,” in Proceedings of ACM SIGCOMM, Austin, pp. 1-12, September 1989.

[2] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control— the single node case,” in Proceedings of IEEE INFOCOM, Florence, Italy, pp. 915-924, May 1992.

[3] A.K. Parekh and R.G. Galleger, “A generalized processor sharing approach to flow control in integrated services networks: the single node case,” IEEE/ACM Trans. Netw., vol. 1, no. 3, pp. 344-357, June 1993.

[4] A.K. Parekh and R.G. Galleger, “A generalized processor sharing approach to flow control in integrated services networks: the multiple node case,” IEEE/ACM Trans. Netw., vol. 2, no. 2, pp. 137-150, June 1994.

[5] C. B. Stunkel, “The SP2 high-performance switch,” IBM Systems Journal, vol. 34, no. 2, pp. 185-204, February 1995.

[6] D. Bertsekas and R. Gallager, Data Networks. Prentice Hall, 1987.

[7] D. Ferrai and D. Verma, “A scheme for real-time channel establishment in wide-area networks,” IEEE J. Select. Areas Communications, pp.389-279, April 1990.

[8] D. Stiliadis and A. Verma, “Latency-rate servers: A general model for analysis of traffic scheduling algorithms,” IEEE Transactions on Networking, vol. 6, no. 3, pp. 611-624, October 1996.

[9] F. Risso, “Decoupling Bandwidth and Delay Properties in Class Based Queueing,” in proceedings of IEEE Computers and Communications, 2001.

[10] Francini, A., “A weighted fair queueing scheduler with decoupled bandwidth and delay guarantees for the support of voice traffic” GLOBECOM '01. IEEE, vol.3 pp. 1821-1827, 2001.

[11] H. Kroner, G.Hebuterne, P.Boyer, A. Gravery, “Priority Management in ATM Switching Note,” IEEE JSAC, April 1991.

[12] Hui Zhang and Srinivasan Keshav, “Comparison of rate-based service disciplines,” In Proceedings of ACM SIGCOMM , pp. 113-121 , August 1991.

[13] J. Barlow and W. Thrower, “TFN2K – an analysis,” February 2000, http://packetstorm.securify.com/distributed/TFN2k_Analysis.htm.

[14] J. C. R. Bennett and H. Zhang, “WF$^2$Q: worst-case fair weighted fair queuing,” Proc. of IEEE INFOCOM, San Francisco, CA, pp. 120-128, March 1996.

[15] J. Liebeherr and N. Christin. “JoBS: Joint buffer management and scheduling for differentiated services”. In Proceedings of IWQoS 2001, Karlsruhe, Germany, pp. 404-418, June 2001.

[16] Jaikumar Vijayan, “Credit card firm hit by DDoS attack” “http://www.computerworld.com/securitytopics/security/story/0,10801,96099,00.html,” Computerworld, September 22, 2004.

[17] L. Kleinrock, “Queueing Systems Volume 2: Computer Applications,” John Wiley, Inc., New York, 1975.

[18] M. G. Hluchyj and M. J. Karol, “High-performance Packet Switching,” IEEE

Journal of Selected Areas in Communications Vol. 6, No. 9, pp. 1587-1597, December 1988.

[19] M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round robin,” in Proceedings of ACM SIGCOMM, Boston, MA, September 1995.

[20] M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round-robin,” IEEE Transactions on Networking, vol. 4, no. 3, pp. 375–385, June 1996.

[21] R. Cruz, “A calculus for network delay. i. network elements in isolation,” IEEE Transactions on Information Theory, vol. 37, pp. 114–131, January 1991.

[22] R.Jain, K. Ramakrishnam, “Congestion Avoidance in Computer Networks, with a connectionless Network Layer: Concepts, Goals, and Methodology,” Computer Networking Symposium, April 1988.

[23] S. J. Golestani, “A self-clocked fair queuing scheme for broadband applications,” in Proceedings of IEEE INFOCOM, Toronto, Canada, pp. 636-646, June 1994.

[24] S. Keshav, “An Engineering Approach to Computer Networks,” Addison-Wesley Publishing Company, Reading, MA, 1997.

[25] S. Keshav, “On the efficient implementation of fair queuing,” Journal of Internetworking Research and Experience, vol. 2, no. 3, pp. 3-26, September 1990.

[26] S. S. Kanhere, H. Sethu, and A. B. Parekh, “Fair and efficient packet scheduling using elastic round robin,” IEEE Transactions on Parallel and

Distributed Systems, vol. 13, no. 3, pp. 324-326, March 2002.

[27] S. Tsao and Y. Lin, “Pre-order deficit round robin: a new scheduling algorithm for packet-switched networks,” Computer Networks, vol. 35, no. 2-3, pp. 287-305, February 2001.

[28] Y, Bernet, “A Framework for Integrated Services Operation over Diffserv Networks,” RFC2998, November 2000.

[29] Y. Zhou and H. Sethu, “On the relationship between absolute and relative fairness bounds,” IEEE Communication Letters, vol. 6, no. 1, pp. 37–39, January 2002.

[30] UCB/LBNL/VINT “Network Simulator - ns(version 2)”, 1997, URL: http://www.isi.edu/nsnam/ns.

[31] S. Floyd, T. Henderson, “The NewReno Modification to TCP's Fast Recovery Algorithm,” RFC 2582, April 1999.

[32] K.Fall and S. Floyd, “Simulation-based Comparison of Tahoe, Reno, and SACK TCP,” ACM Computer Communication Review, Vol.26, No.3, pp. 5-21, July 1996.

[33] M.Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgement Options,” RFC 2018, October 1996.

[34] R.Braden, D.Clark, and S. Shenker. “Integrated service in the internet architecture”, RFC 1633, June 1994.

[35] Y. Bernet et.al. “A framework for differentiated services, Internet Draft, draft-ietf-diffserv-framework-01.txt,” November 1998.

[36] Ion Stoica, Scott Shenker, Hui Zhang. “Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks,” IEEE/ACM Transactions on Network. Vol. 11, no. 1, pp. 33-46, 2003