교육학석사 학위논문

A Study on Improved Algorithm for Fair Bandwidth Allocation and Packet Processing

지도교수 : 박 승 섭

이 논문을 교육학석사 학위 논문으로 제출함.

2005년 8월

부경대학교 교육대학원

전산교육전공

김 문 경

김문경의 교육학석사 학위논문을 인준함

2005년 6월 17일

주 심	공학박사	정순호	[illegible]
위 원	이학박사	윤성대	[illegible]
위 원	공학박사	박승섭	[illegible]

Contents

Contents of Figures ---------- II

Contents of Tables ---------- III

Abstract ---------- IV

1. Introduction ---------- 1

2. Related Works ---------- 2

2.1 Active Queue Management ---------- 2

2.1.1 Random Early Detection ---------- 3

2.1.2 Flow RED ---------- 7

2.1.3 CHOKe ---------- 10

2.1.4 Disadvantage Above Algorithm ---------- 12

2.2 Core-Stateless Fair Queueing---------- 14

2.2.1 Fluid Model Algorithm ---------- 16

2.2.2 Packet Algorithm ---------- 17

2.2.3 Link Relabeling ---------- 19

3. Algorithm Deiscription ---------- 19

3.1 The Scheme in Edge Nodes ---------- 20

3.2 The Scheme in Core Nodes ---------- 22

3.2.1 Sorting Per-Flow Information ---------- 22

3.2.2 Estimating the Number of Active Flow ---------- 23

3.2.3 Packet Dropping ---------- 27

4. Simulation ---------- 28

5. Conclusion ---------- 37

Reference ---------- 39

Contents of Figures

Fig. 1-1 General algorithm for RED gateway
Fig. 1-2 Detailed algorithm for RED gateway
Fig. 1-3 Detailed algorithm for FRED
Fig. 1-4 Flow chart for FRED algorithm
Fig. 1-5 Flow chart for CHOKe algorithm
Fig. 2-1 Network architecture for CSFQ
Fig. 2-2 CSFQ's scheme in edge and core nodes
Fig. 3-1 Pseudocode for CS-FNE in edge node
Fig. 3-2 Collision resolution by chaining
Fig. 3-3 The trigger scheme
Fig. 4-1 The single congested link has the 10 Mbps capacity and the propagation delay of 1 ms.
Fig. 4-2 The average throughput over 10 sec when N=32 and all flows are UDPs.
Fig. 4-3 The normalized bandwidth of a TCP flow that competes with N-1 UDP flows
Fig. 4-4 The multiple congested link have capacity of 10 Mbps and propagation delay of 1ms.
Fig. 4-5 The normalized throughput of UDP-0 as a function of the number of congested links.
Fig. 4-6 The same plot when UDP-0 is replaced by TCP flow.

Contents of Tables

Table 1. Result of first experiment in single congested link
Table 2. Result of second experiment in single congested link
Table 3. Comparison between CSFQ and CS-FNE for processing time of first experiment in single congested link
Table 4. Comparison between CSFQ and CS-FNE for processing time of second experiment in single congested link
Table 5. Result of first experiment in multiple congested link
Table 6. Result of second experiment in multiple congested link
Table 7. Comparison between CSFQ and CS-FNE for processing time of first experiment in multiple congested link congested link
Table 8. Comparison between CSFQ and CS-FNE for processing time of second experiment in multiple congested link

공정한 대역폭 할당과 패킷처리를 위한 개선된 알고리즘에 관한 연구

김 문 경

부경대학교 교육대학원 전산교육전공

요약

네트워크 상에서 전송속도를 보장하고 공정한 대역폭할당을 위해 많은 알고리즘들이 제안되어져 왔다. 그중 가장 간단하게 구현이 가능한 FIFO 방식은 공정성을 보장할 수 없었으며 혼잡이 발생하였을 때 동작하기 때문에 혼잡을 회피할 수는 없었다. 그래서 혼잡회피를 위한 동적큐관리 기법이 제안되어졌다.

또한 라우터에서 발생하는 복잡성을 줄이기 위해 Core-Stateless 알고리즘이 제안되어졌는데 이는 네트워크를 Edge 라우터와 Core 라우터로 구분한다. Core 라우터에서 각 흐름마다의 상태관리를 하지 않게해 복잡성을 줄일수 있었다.

이 논문에서는 혼잡회피를 제공하는 동적큐관리 기법중 하나인 FNE알고리즘을 Core-Stateless 알고리즘에 적용하였으며 이를 수정 보완하였다. 이를 CS-FNE라 명명하였으며 이를 CSFQ, FRED, RED, DRR과 같은 다른 알고리즘들과 여러 환경에서 비교분석하였다. 모의실험을 통해 CS-FNE 알고리즘이 적정수준의 공정한 대역폭 할당을 달성하는지를 보였으며 다른 흐름마다 관리를 기반으로 하는 알고리즘보다 쉽게 구현할 수 있음과 패킷처리시간을 개선시킬 수 있음을 보였다.

1. Introduction

Recently, as the date communication techniques are improved, the Internet traffic also occurs more frequently. FIFO queueing with Drop Tail is the most commonly used mechanism of the existing routers. However, by FIFO, the resource is allocated unfairly. Especially, responsive flows such as TCP suffer from unresponsive flows such as UDP in a FIFO queue when a coñgestion occurs. Moreover, FIFO just reacts to the congest without any avoidances.

To solve theses defects, many improved algorithms have been proposed on the packet isolation and congestion avoidance. The AQM (Active Queue Management), for example, is proposed to solve the congestion and achieve the packet isolation. There are RED (Random Early Detection)[11], FRED(Flow RED)[2] and CHOKe(CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows)[10] as AQM scheme. In AQM scheme, a packet is dropped or admitted according to the arrival rate of the flow which the packet belongs to, the queue length that the flow occupies and the number of active flows in router. However, these AQM algorithms can't cover both existing defects and implemental complexity.

Some congestion control algorithms have adopted per-flow dropping schemes, as the result, the algorithms were implemented more complicated. In particular, fair allocation mechanisms inherently require the routers to maintain the state and perform the operations on a per-flow basis. Consequently, the mechanism which has no per-flow state is proposed to reduce the complexity. Representatively, there are CSFQ(Core-Stateless Fair Queueing)[4]. CSFQ adopt Core-Stateless network which distributes Edge nodes and Core Nodes and try to achieve more simplexity than a per-flow mechanism.

In this paper, to achieve the fairness and efficiency of packet processing, we propose the scheme which employs FNE(Flow Number Estimation)[5] algorithm in Core-Stateless network and show the improved Core-Stateless mechanism. We call the mechanism CS-FNE and evaluate the performance with alternate approaches through the simulation.

In Section 2, we mention related works of AQM and CSFQ. Then we describe the proposal simulation in Section 3. We also evaluate the different existing schemes and show the result in Section 4. Finally, the conclusion remarks are given in Section 5.

2. Related Works

2.1. Active Queue Management

Typically, Drop Tail is the most widely used queueing algorithm in current network. But it has the unfairness in the resource allocation and congestion avoidance. This is because TCP flows suffer from unresponsive flows in a FIFO queue when a congestion occurs.

To deal with that, RED, the active queue management, has been proposed. But RED requires the collaboration of end hosts congestion control mechanism to provide good service and do not work well with the growing number of UDP-based applications, which generally do not back off with the congestion indication. To resolve this problem, improved algorithms have been proposed, such as FRED and CHOKe etc.

They don't just react to the congestion, but actively try to avoid the congestion by notifying sources about the incipient congestion. The sources are expected to reduce their sending rate on receipt of such a notification.

2.1.1 Random Early Detection

The basic idea in early random drop is to start notifying connections early about the incipient congestion. Only a few connection are told to back-off which avoids underutilization and also the with connections backing off the load at the router would be avoided and there would be less probability of queue reaching its maximum limit and hence buffer overflow is avoided.

Random Early Detection (RED) builds on the concept of Early Random Drop by introducing average queue size measure and dynamically changing the drop probability. Use of dynamic drop probability ensures the gateway reacts differently to different level of congestion anticipation i.e If queue-size is approaching thresholds the drop probability has to be higher than, say when the queue size is very less compared to the threshold.

The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold. When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability P_a, where P_a is a function of the average queue size . Each time that a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the gateway.

The RED algorithm is given in Fig. 1-1.

```
for each packet arrival
 calculate the average queue size avg
 if minth $\leq$ avg $<$ maxth
   calculate probability pa
   with probability pa :
     mark the arriving packet
 else if maxth $\leq$ avg
   mark the arriving packet
```

Fig. 1-1. General algorithm for RED gateway

Thus the RED gateway has two separate algorithms.

1. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue.

2. The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion.

As average queue size avg varies from $\min_{th}$ to $\max_{th}$, the packet-marking probability P_b varies linearly from 0 to $\max_p$:

$$P_b \leftarrow \max_p(avg - \min_{th})/(\max_{th} - \min_{th}) \qquad 1\text{-}(1)$$

The final packet-marking probability P_a increases slowly as the count increases since the last marked packet :

$$P_a \leftarrow P_b/(1 - count \cdot P_b) \qquad 1\text{-}(2)$$

The final packet-marking probability pa increases slowly as the count increases since the last marked packet. This ensures that marking of packets is fairly uniform

One option for the RED gateway is to measure the queue in bytes rather than in packets. With this option, the average queue size accurately reflects the average delay at the gateway. When this option is used, the algorithm would be modified to ensure that the probability that a packet is marked is proportional to the packet size in bytes:

$$P_b \leftarrow P_b \cdot PacketSize / MaximumPacketSize \qquad 1\text{-}(3)$$

$$P_a \leftarrow P_b / (1 - count \cdot P_b) \qquad 1\text{-}(4)$$

In this case a large FTP packet is more likely to be marked than is a small TELNET packet.

If the queue remains empty for a large period of time, the old value of avg should have a lesser share in new value of queue average size, RED ensures that with the equation 1-(5).

$$avg = (1 - w_q)^m \cdot avg \qquad 1\text{-}(5)$$

Where m denotes the time for which queue has remained idle.

The detailed algorithm for RED gateway is given in Fig. 1-2.

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

Fig. 1-2. Detailed algorithm for RED gateway

Variables :

avg : average queue size

q_time : start of the queue idle time

count : packets since last marked packet

Fixed parameter:

w_q : queue weight

min_{th} : minimum threshold for queue

max_{th} : maximum threshold for queue

max_p : maximum value for pb

Others:

pa : current packet-marking probabillity

q : current queue size

time : current time

$f(t)$: a linear function of the time t

2.1.2 Flow RED

FRED is a modified version of RED and uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use. The goal is to reduce the unfairness found in RED.

FRED uses state variables to record the current buffer share of each connection and compares it to its fair share, connections whose current share exceeds the fair share are penalized. Thus misbehaving connections are identified. FRED stores per-flow buffer count of each active connection. Here active connection refers to connection that have packets buffered in the queue. Hence the amount of accounting information stored is proportional to the buffer space.

Further FRED identifies non-responsive aggressive connections, and penalizes such connections by allowing only the such connections to buffer just their fair share of bandwidth, i.e. bursts from such connections are not be accommodated by the router and such burst packets will be dropped. On other hand if a connections has been responsive then, even if the connection has already used its fair share, the next incoming packet from that connection is not deterministically dropped but is given a random dropper where it is probabilistically dropped, depending upon the average queue length. Hence a burst of packet from such responsive connection would be accommodated in the queue. This behavior works as a sort of incentive for connections to be responsive to packet drop, and helps avoid congestion. FERD addresses the problems of very small connections like telnet that have very small data to send and hence use very less than their share of the available bandwidth. Also such connections do not have data always ready to send. FRED has provision for such connections. FRED always allows packets from such connections unless the queue as exceeded its max threshold.

FRED as it just builds on RED, continues to have its improvement over Drop-Tail and is also able to achieve isolation, protection albeit at cost of added complexity of per-active flow accounting. Like RED, FRED does not make any assumptions about queuing architecture, it will work with a FIFO gateway.

FRED uses additional variable such as max_q, min_q, ave_{cp}, $qlen_i$ and $strike_i$. FRED uses min_q and max_q as maximum and minimum number of packets a connection would be allowed to have in buffer and ave_{cp} represent the fair buffer share of a connection at the router. FRED also uses qlen as a per-flow variable which is the number of packets a connection has currently enqueued in the buffer and uses strike to identify unresponsive connection. An unresponsive connection has non-zero value for this variable. An unresponsive connection is one that tries to exceed the limit of "max_q" number of packet.

The detailed algorithm for FRED is given in Fig. 1-3.

```
for each arriving packet P
  if P is from a flow i that does not have packets in the queue
qleni = 0;
strikei = 0;
if ( qleni >= maxq ||   // the connection has exceeded its maximum limit
(avg >= maxth && qleni > 2*avecq) ||(qleni >= avecq && strikei > 1))
strikei ++;
drop packet P;
return
if (minth <= avg < maxth)
(
if (qleni >= MAX(minq, avgcq)
{
calculate drop probability pa ;
with probability pa;
drop packet P;
return;
}
else if ( avg < minth )
{
count = -1;
accept the incoming packet
}
else
{
drop packet P;
return;
}
```

Fig. 1-3. Detailed algorithm for FRED

This flow chart shows FRED scheme in general.

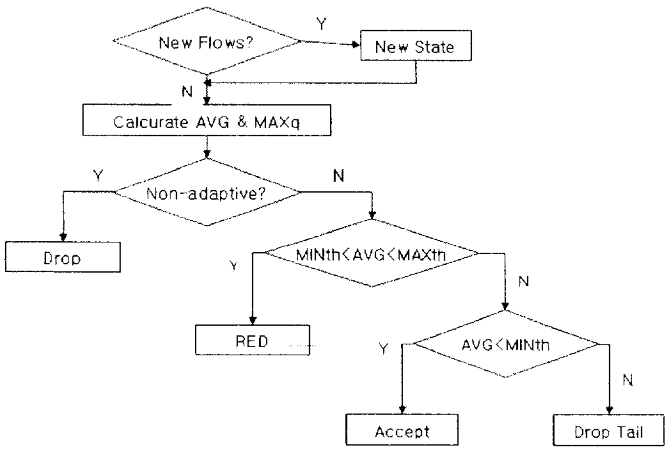

Fig. 1-4. Flow chart for FRED algorithm

2.1.3 CHOKe

The basic of idea behind CHOKe is that the contents of the FIFO buffer form a "sufficient statistic" about the incoming traffic and can be used in a simple fashion to penalize misbehaving flows.

When a packet arrives at a congested router, CHOKe draws a packet at random from the FIFO buffer and compares it with the arriving packet. If both belonging to the same flow, then they are both dropped, else the randomly chosen packet is left intact and the arriving packet is admitted into the buffer with a probability that depends on the level of congestion (this probability is computed exactly as in RED).

The reason for doing this is that the FIFO buffer us more likely to have packet belonging to a misbehaving flow and hence these packets are more likely to be chosen for comparison. Further, packets belonging to a misbehaving flow arrive more numerously and are more likely to trigger comparison. The intersection of these two high probability events is precisely the event that packets belonging to misbehaving flows are dropped. Therefore, packets of misbehaving flows are dropped more often than packets of well-behaved flows.

CHOKe calculate the average occupancy of the FIFO buffer using an exponential moving average, just as RED does. It also marks two thresholds on the buffer, a minimum threshold $\min_{th}$ and a maximum threshold $\max_{th}$.

If the average queue size is less than $\min_{th}$, every arriving packet is queued into the FIFO buffer. If the aggregated arriving rate is smaller than the output link capacity, the average queue size should not build up to $\min_{th}$ very often and packets are not dropped frequently. If the average queue size is greater than $\max_{th}$, every arriving packet is dropped. This moves the queue occupancy back to below $\max_{th}$. When the average queue size is bigger than $\min_{th}$, each arriving packet is compared with a randomly selected packet, called drop candidate packet, from the FIFO buffer. If they have the same flow ID, they are both dropped. Otherwise, the randomly chosen packet is kept in the buffer (in the same position as before) and the arriving packet is dropped with a probability that depends on the average queue size. The drop probability is computed exactly as in RED. In particular, this means that packets are dropped with probability 1 if they arrive when the average queue size exceeds $\max_{th}$. In order to bring the queue occupancy back to below $\max_{th}$ as fast as possible, CHOKe still compare and drop packets from the queue when the queue size is above the $\max_{th}$.

As algorithm is described, the flow chart shows algorithms in general.

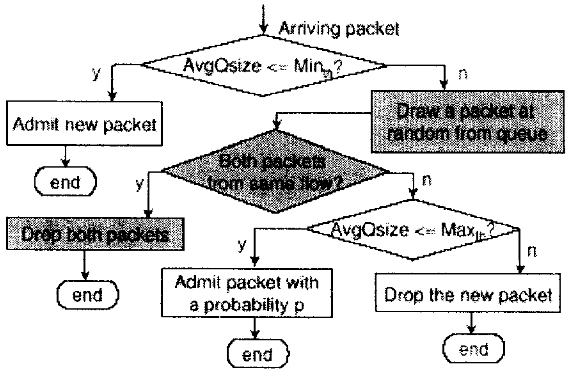

Fig. 1-5. Flow chart for CHOKe algorithm

2.1.4 Disadvantage of Above Algorithms

RED's goal is to drop packets from each flow in proportion to the amount of bandwidth when the flow uses the output link. It can be performed by dropping the each arriving packet with equal probability (provided the average queue size does not change significantly). Therefore, the connection with the largest input rate will have the biggest drop percentage among total dropped packets.

Assume that the average queue size does not change for a short period δ , so RED drops incoming packets with a fixed probability p; also assume that connection$_i$'s current input rate is λ (or $\lambda \cdot \delta$ packets per δ). The percentage of dropped packets from connectioni is:

$$\frac{\lambda_i p}{\sum \lambda_i p} = \frac{\lambda_i}{\sum \lambda_i}$$

For a FCFS service discipline, connection$_i$'s output rate is proportional to its buffer occupancy, which is determined by the percentage of accepted packets:

$$\frac{\lambda_i (1-p)}{\sum \lambda_i (1-p)} = \frac{\lambda_i}{\sum \lambda_i}$$

The above two equations imply that RED drops packets in proportion to each connection's output usage under FIFO scheduling. From each connection's point of view, however, the instantaneous packet loss rate during a short period δ is $\frac{\lambda p}{\lambda} = p$, which is independent of the bandwidth usage. If the congestion is persistent, which means the average queue length used by RED has a minimum value above $\min_{th}$, the drop probability has a nonzero minimum and therefore causes a minimum loss rate for all connections, regardless of their bandwidth usage. This unbiased proportional dropping contributes to unfair link sharing in the following ways:

1. Although RED performs better than Drop Tail and Random Drop, it still has a bias against fragile connections. The fact that all connections see the same loss rate means that even a connection using much less than its fair share will experience packet loss. This can prevent a low-bandwidth TCP from ever reaching its fair share, since each loss may cause TCP to reduce its window size by one half.

2. Accepting a packet from one connection causes higher drop probability for future packets from other connections, even if the other connections consume less bandwidth. This causes temporary undesirable non-proportional dropping even among identical flows.

3. A non-adaptive connection can force RED to drop packets at a high rate from all connections. This contributes to RED's inability to provide a fair share to adaptive connections in the presence of aggressive users even if the congestion is not severe.

Although FRED is a modified version of RED, it has extra implementation overhead since it is need to collect certain types of state information. RED with penalty box stores information about unfriendly flows while FRED needs information about active connections. Although FRED achieves more reasonable degree of fair bandwidth allocation and penalty to unfriendly flows than RED, it doesn't overcome the implemental complexity.

CHOKe is the improved algorithm on RED. CHOKe uses the match-then-drop scheme to achieve fair queueing. It chooses a packet randomly from those queue to compare with the arriving packet. If both packets belong to the same flow, they are dropped. Otherwise, the arriving packet is admitted in the queue. However, it is a fatal defect that the packet size is not distinguished. Assuming that three flows share the queue and queue is full of packets belonging to the first and second flow, their packet is more likely to be chosen for matching. Although three flows have the same arrival intensity, the packet dropping probability of the first and second flow are larger than the third flow. Then third flow is given more bandwidth than others.

Moreover, if there are increased numbers of unresponsive flows, CHOKe cause the unfairness. When unresponsive flows increase, although increasing the drop candidates enhance the ability to identify the unresponsive flows, the complexity increases with a growing number of candidates. Therefore, CHOKe works well only when there are few misbehaving flows.

2.2 Core-Stateless Fair Queueing

To achieve more reasonable degree of fair share, many algorithms are proposed. However, they are complex to implement than FIFO and Drop Tail scheme and make data processing slow. Moreover, maintaining per-flow state cause complexity so that these mechanism with maintain per-flow state are not suitable for high-speed network.

Consequently, the architecture and algorithm that allocate bandwidth fairly in an approximate fair manner and maintain no per-flow state are proposed.

For this approach, Core-Stateless network recognize an island of routers and distinguishes between the edge and core of the island. Island means the contiguous network with well-defined interior and edges. Edge node estimates per-flow rate when packet arrives the node and insert this information into packet's header. Core node adopts simple FIFO queue and keeps no per-flow state. And core node uses the information which is estimated in edge node to avoid congestion. Core-Stateless network is shown in Fig. 2-1.

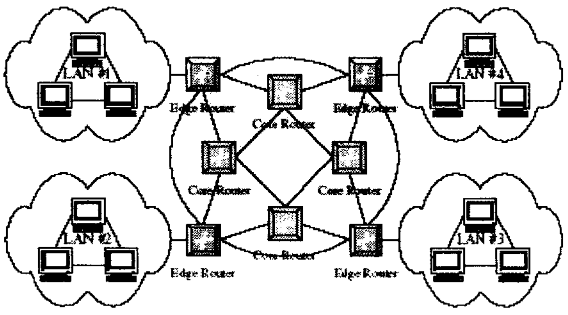

Fig. 2-1. Network architecture for CSFQ

First, to avoid maintaining per-flow state at each router, CSFQ use a distributed algorithm in which only edge routers maintain per-flow state, while core (nonedge) routers do not maintain per-flow state but instead utilize the per-flow information carried via a label in each packet's header. This label contains an estimation of the flow's rate; it is initialized by the edge router based on per-flow information, and then updated at each router along the path based only on aggregate information at that router.

Second, to avoid per-flow buffering and scheduling, as required by Fair Queueing, CSFQ use FIFO queueing with probabilistic dropping on input. The probability of dropping is a function of the rate carried in the label and of the fair share rate at that router, which is estimated based on measurements of the aggregate traffic.

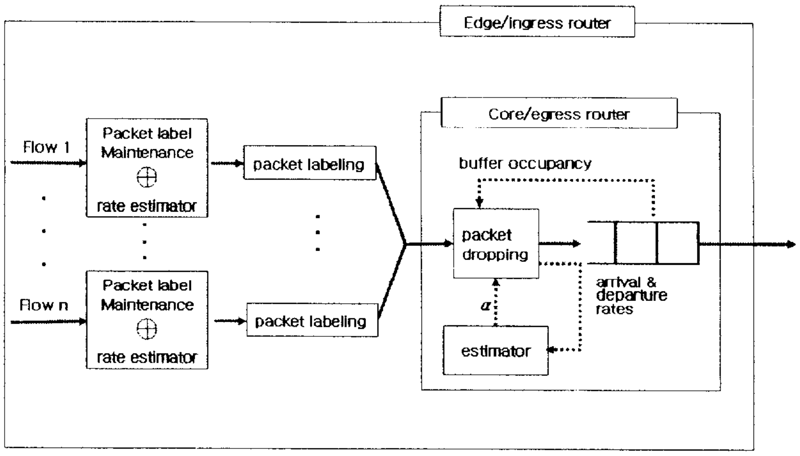

Fig. 2-2. CSFQ's scheme in edge and core node

2.2.1 Fluid Model Algorithm

The arrival rate is known exactly in fluid model algorithm because rate is bps. However, in real networks, traffic consists of packets. Thus, the algorithm is extended to situation in real router where transmission is packetized. First, we describe the fluid model algorithm.

In fluid model algorithm, C means the output link speed and $r_i(t)$ is the each flow's arrival rate. The flows are modeled as a continuous stream of bits and all flows that are bottlenecked by the router have the same output rate which is called fair share rate, α(t). In general, if max-min bandwidth allocations are achieved, each flow receives service at a rate given by $\min(r_i(t), \alpha(t))$. Let $A(t)$ denote the total arrival rate: $A(t)=\sum_{i=1}^{n} r_i(t)$. If , then the fair share is the unique solution to

$$C=\sum_{i=n}^{n}\min(r_i(t), \alpha(t)) \qquad 2\text{-}(1)$$

If $A(t)\le C$, then no bits are dropped and we will, by convention, set $\alpha(t)=\max_i r_i(t)$. If $r_i(t)\le\alpha(t)$, i.e., flow sends no more than the link's fair share rate, all of its traffic will be forwarded. If $r_i(t) > \alpha(t)$, then a fraction $\frac{r_i(t) - \alpha(t)}{r_i(t)}$ of its bits will be dropped, so it will have an output rate of exactly $\alpha(t)$. This suggests a very simple probabilistic forwarding algorithm that achieves fair allocation of bandwidth: each incoming bit of flow is dropped with the probability:

$$\max(0, 1 - \frac{\alpha(t)}{r_i(t)}) \qquad 2\text{-}(2)$$

When these dropping probabilities are used, the arrival rate of flow at the next hop is given by $\min[r_i(t), \alpha(t)]$.

2.2.2 Packet Algorithm

Packet algorithm still employ a drop-on-input scheme, except that now it drop packets rather than bits. Because the rate estimation (described below) incorporates the packet size, the dropping probability is independent of the packet size and depends only, as above, on the rate $r_i(t)$ and fair share rate $\alpha(t)$

1. Computation of Flow Arrival Rate

The rates $r_i(t)$ are estimated at the edge routers and then inserted into the packet labels. At each edge router, CSFQ use exponential averaging to estimate the rate of a flow. Let t_i^k and l_i^k be the arrival time and length of the k^{th} packet of flow i. The estimated rate of flow i, r_i, is updated every time a new packet is received :

$$r_i^{new} = (1 - e^{T_i^k/k})\frac{l_i^k}{T_i^k} + e^{-T_i^k/k} r_i^{old} \qquad 2\text{-}(3)$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a contents.

2. Link Fair Rate Estimation

The rate with which the algorithm accepts packets is a function of the current estimate of the fair share rate, which is denoted as $\widetilde{\alpha}(t)$ Letting $F(\widetilde{\alpha}(t))$denote this acceptance rate

$$F(\widehat{\alpha}(t)) = \sum_{i=1}^{n} \min(r_i(t), \widehat{\alpha}(t)) \qquad 2\text{-}(4)$$

Note that $F(\cdot)$ is a continuous nondecreasing concave and piecewise-linear function of $\widehat{\alpha}$. If the link is congested $(A(t) > C)$, CSFQ choose $\widehat{\alpha}(t)$ to be the unique solution to $F(x) = C$. If the link is not congested $(A(t) < C)$, CSFQ take $\widehat{\alpha}(t)$ to be the largest rate among the flows that traverse the link, i.e., $\alpha(t) = \max_{1 \le i \le n}(r_i(t))$. Note that if we knew the arrival rates $r_i(t)$, we could then compute $\widehat{\alpha}(t)$ directly. To avoid having to keep such per-flow state, we seek instead to implicitly compute $\widehat{\alpha}(t)$ by using only aggregate measurements of F and A.

We use the following heuristic algorithm with three aggregate state variables: $\widehat{\alpha}$ the estimate for the fair share rate; $\widehat{A}$ the estimated aggregate arrival rate; $\widehat{F}$ the estimated rate of the accepted traffic. The last two variables are updated upon the arrival of each packet. For $\widehat{A}$, we use exponential averaging with a parameter e^{-T/K_α} where is the inter arrival time between the current and the previous packet

$$\widehat{A}_{new} = (1 - e^{-T/K_\alpha}) \frac{l}{T} + e^{-T/K_\alpha} \widehat{A}_{old} \qquad 2\text{-}(5)$$

where $\widehat{A}_{old}$ is the value of $\widehat{A}$ before the updating. We use an analogous formula to update F.

The updating rule for depends on whether the link is congested or not. To filter out the estimation inaccuracies due to exponential smoothing, we use a window of size K_c. A link is assumed to be congested if $\widehat{A} \ge C$ at all times during an interval of length K_c. Conversely, a link is assumed to be uncongested if $\widehat{A} \le C$ at all times during an interval of length K_c. The value is updated only at the end of an interval in which the link is either

congested or uncongested according to these definitions. If the link is congested, then is updated based on the equation $F(\widehat{\alpha}) = C$. We approximate $F(\cdot)$ by a linear function that intersects the origin and has slope $\widehat{F} / \widehat{\alpha}_{old}$. This yields

$$\alpha_{\mp w} = \alpha_{old} \frac{C}{\hat{F}} \qquad 2\text{-}(6)$$

If the link is not congested, $\hat{\alpha}_{\mp w}$ is set to the largest rate of any active flow (i.e., the largest label seen) during the last K_c time units. The value of $\hat{\alpha}_{\mp w}$ is then used to compute dropping probabilities.

3. Link Relabeling

Estimation algorithm allows router to label packets with their flow's rate as they enter the island. Packet dropping algorithm described allows router to limit flows to their fair share of the bandwidth. After a flow experiences significant losses at a congested link inside the island, however, the packet labels are no longer an accurate estimate of its rate. CSFQ cannot rerun our estimation algorithm, because it involves per-flow state. Fortunately, the outgoing rate is merely the minimum between the incoming rate and the fair rate α. Therefore, router rewrite the packet label L as

$$L_{\mp w} = \min(L_{old}, \alpha) \qquad 2\text{-}(7)$$

By doing so, the outgoing flow rates will be properly represented by the packet labels.

3. Algorithm Description

In routing algorithms, packet isolation and reducing complexity have to be achieved simultaneously as reasonable as possible. We described proposed algorithms in above section but they can't achieve both isolation and simplexity. Although CSFQ allocate fair bandwidth approximately, CSFQ uses the exponential weight to compute flow arrival rate so that complexity isn't improved. Therefore, we consider the improvement of these algorithms and apply a novel AQM scheme which is called 'Flow Number Estimation' to Core-Stateless network.

The goal of fairness should be defined first: Assume that n flows pass through the router with link capacity C. For flow i, r_i denotes its arrival rate and r_i' denotes its admitted rate into the router. By using the threshold rate, the router drops the packets of flow i with the probability $\max[0,(r_i-CL)/r_i]$ (the CL is means cutting line or threshold). If the total arrival rate is smaller than or equals to the link capacity, $\sum r_i' \le C$, no packets should be dropped even through the rate of one flow is much larger than that of the others. The solution to maximize $\sum r_i'$ is must founded.

Assuming that n flows pass through the router and the sum of their arrival rates is larger than the output link capacity, packets of misbehaving flows should be dropped. Without sacrifice of generality, the arrival rates of all incoming flows are sorted to be $r_1 \le r_2 \le \ldots\ldots \le r_n$. The solution of CL exists in one of the $n+1$ region, that is, $0 < CL \le r_1$, $r_1 \le CL \le r_2$, ... , $r_{n-1} \le CL \le r_n$ or $r_n \le CL < \infty$. Therefore, the problem is to find CL to achieve the maximum in the $(n+1)$ regions of

$$\sum_{i=1}^{n} \min(r_i, CL) = \frac{n \cdot CL + \sum_{i=1}^{n} r_i - \sum_{i=1}^{n} |r_i - CL|}{2} \qquad 3\text{-}(1)$$

CS-FNE is constructed from three function blocks: (1) measuring the flow rate in edge nodes (2) estimating the active flows and (3) dropping the packet in core nodes.

3.1 The scheme in Edge Nodes

To measure intensity of flows, CS-FNE adopts Time Sliding Window. Although the rate of TCP flows fluctuates due to congestion avoidance, TSW attenuates the influence of rate fluctuation by estimation the bandwidth according to a short-term past history. The measuring algorithm is shown in pseudocode.

```
Initially:
Win_length = constant;
Avg_rate    = flow's minimum guaranteed rate;
For each arriving packet P:
Bytes_in _TSW = Avg_rate * Win_length;
New_bytes = Bytes_in_TSW + pkt_size
Avg_rate = New_bytes / (Now - T_front + Win_length);
T_front   = now;
If (Avg_rate <= flow's minimum guaranteed rate)
    not insert Label;
Else
    insert Label
```

Fig. 3-1. Pseudocode for CS-FNE in edge node

As the above description, CS-FNE measure the flow rate differently in edge node comparing CSFQ. And measured flow rate is inserted as label in packet's header and it is employed in packet dropping algorithm in core nodes.

3.2 The scheme in Core Nodes

3.2.1 Storing per-flow information

In Core-Stateless Network, core nodes don't maintain per-flow state. However, maintaining per-flow state occurs complexity so that CS-FNE adopts scheme that just stores per-flow information without per-flow state. Then, CS-FNE can reduce complexity and achieve core-stateless network.

To maintain per-flow information, CS-FNE employ a hashing function with m slots to locate the record each flow. CS-FNE use the source-destination address pair of a flow as the key and then calculate the location of the slot in the hash table through a hashing function.

The general hashing function is implemented with the division method. Using the division method, CS-FNE map a key S_D_i into one of the m slots by the remainder of S_D_i divided by m, where S_D_i is the source-destination pair of flow i.

However, maintaining per-flow information by the hash table sometimes results in unfairness due to collision, which occurs when two or more flows map to the same slot. Since the packet drop probability of these flows are calculated according to the per-flow information in th same slot, all flows mapped to the same slot share one per-flow bandwidth.

To avoid collision, there are three ways: (1) hashing flows with direct-address table, (2) rehashing and (3) chaining. CS-FNE adopts chaining to avoid collision, which hash flows to the same slot in a linked list. Each slot in the hash table contains a pointer to the head of the list that stores the per-flow information of all flows hashed to the slot. Fig. 3-2. in the below show the resolution by chaining.

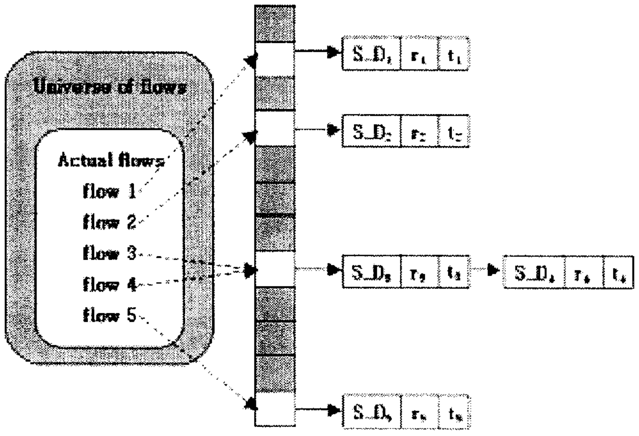

Fig. 3-2. Collision resolution by chaining

As the source-destination pair, the estimated flow rate and the late calculation time are stored in bucket of the linked list, CS-FNE can distribute the output link bandwidth fairly than the scheme which just link the packets as a list, when the packet of two or more flows are hashed to the same slot. Moreover, since very bucket of the lists stores the per-flow information, it is impossible for two flows to have the same per-flow information. Therefore, the flows passing through the router can be regulated correctly according to the accurate per-flow information.

3.2.2 Estimating the Number of Active flow

To estimate fair share and determine that packet dropping probability, CS-FNE estimate the number of active flows, N_{act}, which is estimated by comparing the incoming packet with the packet selected ar random in the queue.

Assuming that n flows pass through the router, r_i and p_i denote the flow rate estimated via the TSW and the buffer occupancy of flow i, respectively. Additionally, there are two auxiliary rates involved in calculating N_{act} : r_{hit} and r_{miss}. Intuitively, r_{hit} is the rate that the flow ID of an incoming packet matches that of one randomly selected packet in the queue, while r_{miss} denotes the rate that two compared packet are not of the same flow. The two auxiliary rate are shown as

$$r_{hit} = \sum_{i=1}^{n} r_i \cdot p_i \qquad 3\text{-}(2)$$

$$r_{miss} = \sum_{i=1}^{n} r_i \cdot (1 - p_i) \qquad 3\text{-}(3)$$

The number of active flow is defined as

$$N_{act} = \frac{r_{hit} + r_{miss}}{r_{hit}} \qquad 3\text{-}(4)$$

For example, p_i is proportional to r_i in FIFO queue. From 3-(2), 3-(3) and 3-(4) We can get

$$N_{act} = \frac{n}{C_r^2 + 1} \qquad 3\text{-}(5)$$

where C_r is the coefficient of variation in flow rate and equals $\sigma_r / E[r]$. If there exists a flow with a much larger rate, C_r equals $n-1$ and N_{act} equals 1. If all flows have the same rate, C_r equals 0 and N_{act} equals n

For more detail, formulas is derived as following.

Basically, p_i is direct proportional to r_i in the FIFO queue and equals

$$p_i = \frac{r_i}{\sum_{i=1}^{n} r_i} = \frac{r_i}{n \cdot E[r]}$$

From the formula of r_{hit},

$$r_{hit} = \sum_{i=1}^{n} r_i \cdot p_i = \frac{\sum_{i=1}^{n} r_i^2}{n \cdot E[r]} = \frac{Var[r] + E[r]^2}{E[r]}$$

The sum of r_{hit} and r_{miss} is equals to the aggregation of the arrival rate of all flows. Therefore, from formulas above, the number of active flows is estimated to be

$$N_{act} = \frac{r_{hit} + r_{miss}}{r_{hit}} = \frac{n \cdot E[\lambda]^2}{Var[\lambda] + E[\lambda]^2}$$

Let us define C_r≡coefficient of variation of arrival rate = $\sigma_r/(E[r])$ and the C_r^2≡ square of coefficient of variation = $(Var[r])/(E[r]^2)$ or $(E[r^2])/(E[r]^2)-1$. Futhermore, from the definition above and formula of N_{act}, the number of active flows can be expressed as

$$N_{act} = \frac{n}{C_r^2 + 1}$$

If there exists a flow with much larger rate than that of the others, that is, for $\forall j \neq i$, $r_j \ll r_i$, we obtain

$$E[r^2]=\frac{r_i^2}{n} \text{ and } E[r]=\frac{r_i}{n}$$

Then, the square of the coefficient of variation can be expressed as

$$\frac{E[r^2]}{E[r]^2}-1=\frac{r_i^2/n}{(r_i/n)^2}-1=n-1$$

Therefore, the estimated number of active flows equals 1. If all flows have the same rate, that is, for $\forall j,\ r_j \neq \overline{r}$,

$$E[r^2]=\overline{r^2} \text{ and } E[r]=\overline{r}$$

Then, the square of the coefficient of variation can be expressed as

$$\frac{E[r^2]}{E[r]^2}-1=\frac{\overline{r^2}}{(\overline{r})^2}-1=0$$

Therefore, from formulas, the estimated number of active flows equals n. Only the comparison result and the relevant of the arriving packet without p_i is viable. When a packet belonging to flow i arrives, the probability that the drop candidate and this packet are of the same flow is equal to the queueing occupancy of flow i.

3.2.3 Packet Dropping

As previously description, admitted rate is regulated to be min(arrival rate, threshold rate). Therefore, threshold rate is estimated and then router can determine that the packet is dropped or not. The method employes a queue-length bound, the target which is shown in Fig. 3-3.

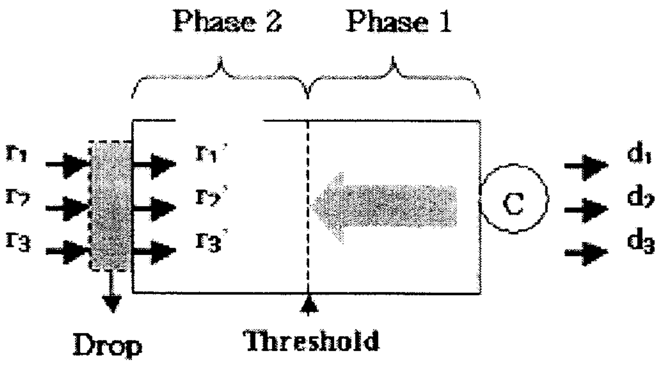

Fig. 3-3. The trigger scheme

When the queue length is below the target, no packet will be dropped. It is reasonable that packets should not be dropped if the output link capacity is sufficient. Once when the queue length exceeds the target, the resources may be insufficient and packet should be dropped. The threshold rate is determined as C/N_{act}. It is possible only for the flows whose rate exceeds the threshold rate to have their packets dropped.

4. Simulation

All simulations are performed in NS-2(Network Simulation). To achieve the propriety of simulation, we consider the single congested link and multiple congested links in the same environment as CSFQ's simulation [4]. We compare CS-FNE's performance to four additional algorithms i.e., DRR (Deficit Round Robin), CSFQ, RED and FRED. DRR is one of scheduling algorithm and CS-FNE employs active queue management, however, we can use DRR as benchmark for fair share.

We use the following parameter for the simulation. Each output link has a capacity of 10Mbps, a latency of 1ms, and a buffer of 64KB. In RED and FRED cases the first threshold is set to 16KB, while the second one is set to 32KB. The averaging constants K (Used in estimation the flow rate), K_a (used in estimation the fair rate), and K_c (used in making the decision of whether a link is congested or not) are all set to 100ms unless specified otherwise. Finally, in all topologies the first router on the path of each flow is always assumed to be an edge router; all other router are assumed without exception to be core router.

First, we consider single congested link which is shared by N flows. The topology is shown in Fig. 4-1.

Fig. 4-1 The single congested link has the 10 Mbps capacity and the propagation delay of 1 ms.

We also perform two related experiments.

In the first experiment, we have 32 UDP flows, indexed from 0, where flow i sends $i+1$ times more than its fair share of 0.3125Mbps. Thus flow 0 sends 0.3125Mbps, flow 1 sends 0.625Mbps, and so on. Fig. 4-2 shows average throughput of each flow over a 10 sec interval.

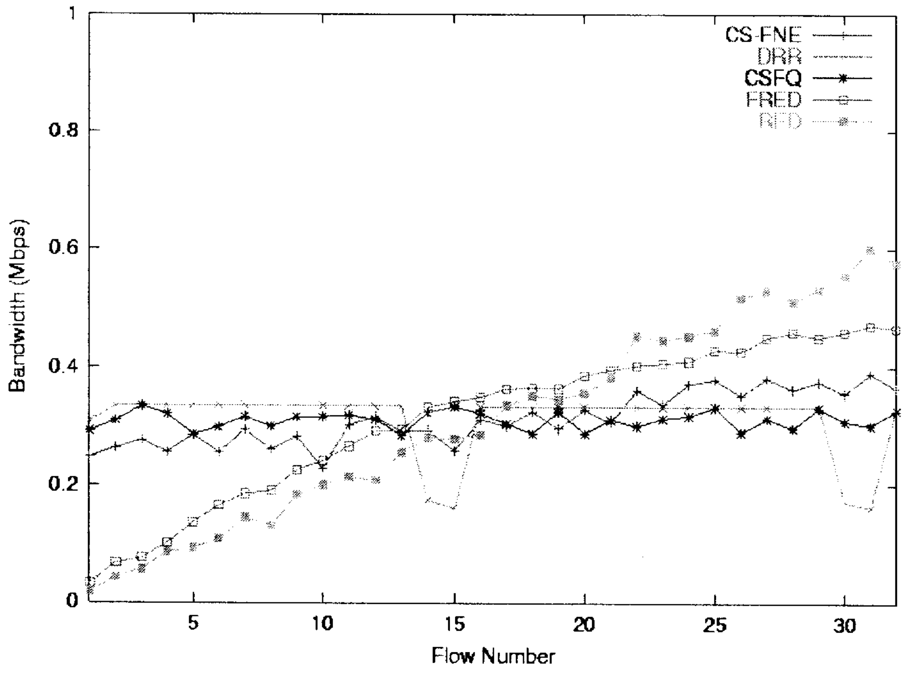

Fig. 4-2. The average throughput over 10 sec when N=32 and all flows are

As number of flow increase, we can see increasing of throughput in RED and FRED. Therefore, RED and FRED's performance is construed that these algorithms fail to ensure fairness from difference of arriving rate. On the other hand, DRR, CSFQ and CS-FNE is achieve fairness approximately. Although the line vibrate a little, flow's throughput is almost similar to fair share rate. We evaluate that CS-FNE can achieve fairness against different flow rate.

For more detail, following table shows the numerical result of performance. In standard deviation, CS-FNE, DRR and CSFQ perform similar degree of fairness. In Table 1, Although CSFQ is better performance than others in Max-Min and Standard Deviation, CS-FNE perform similar degree with DRR and better than RED and FRED.

	Average	Max - Min	Standard Deviation
CS-FNE	0.312	0.162	0.046
DRR	0.312	0.176	0.056
CSFQ	0.310	0.050	0.015
RED	0.312	0.438	0.132
FRED	0.312	0.582	0.176

Table 1. Result of first experiment in single congested link.

In second experiment, we measure how well the algorithms can protect a single TCP flow against multiple ill-behaving flows. We perform 31 simulation, each for a different value N, N = 2, 3, ... , 32. In each simulation we take one TCP flow and N-1 UDP flows; each UDP sends at twice its fair share rate of (10/N)Mbps. Figure 4-3 plots the ratio between the average throughput of TCP flow over 10 seconds and the fair share bandwidth it should receive as a function of the total number of flow N.

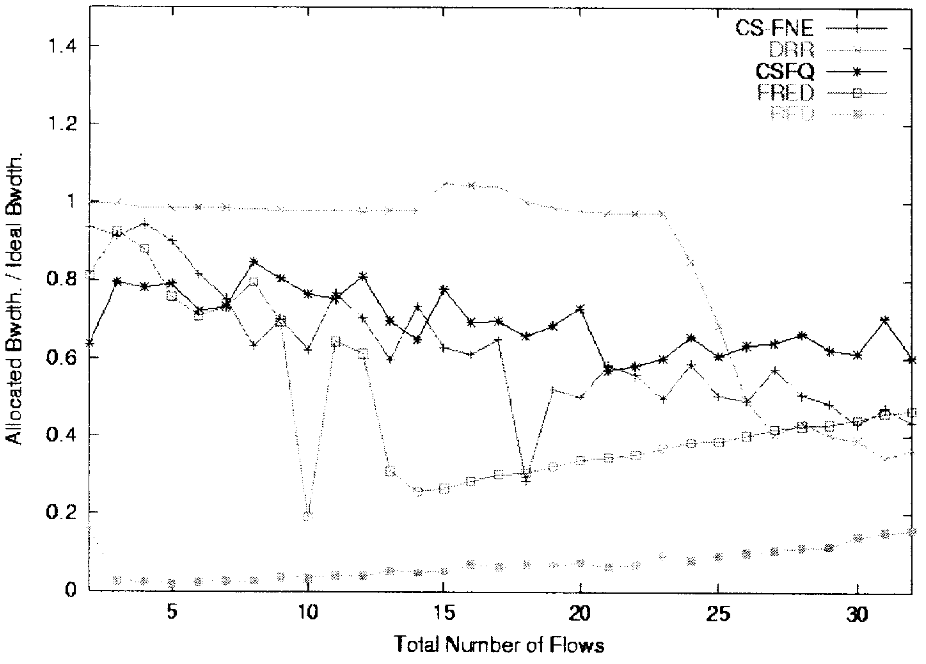

Fig. 4-3. The normalized bandwidth of a TCP flow that competes with N-1 UDP flows.

DRR perform very well when the number of flows is under 22. However, it's performance decrease afterwards. This is because the TCP flow's buffer share is less than three buffers, which significantly affects its throughput. CSFQ and CS-FNE perform than DRR when the number of flows is large. This is because CSFQ and CS-FNE cope better with TCP burstiness by allowing the TCP flow to have several packets buffered for short time intervals which is possible by employing core-stateless network. Across the entire range, CSFQ and CS-FNE provide better performance than RED and FRED.

	Average	Max - Min	Standard Deviation
CS-FNE	0.643	0.690	0.171
DRR	0.850	0.704	0.248
CSFQ	0.703	0.432	0.093
RED	0.339	0.920	0.331
FRED	0.101	0.078	0.042

Table 2. Result of second experiment in single congested link.

As shown in table, DRR's average is better than CSFQ and CS-FNE, however decreasing since the number of flows increase, make standard deviation large. Thus, we can't evaluate that DRR's performance is better than others. CSFQ and CS-FNE shows similar performance.

In above mention, we notice that CSFQ and CS-FNE perform similar degree of fairness. Although DRR's performance is better, DRR have complexity since it schedule packets so that it is impossible to implement DRR in real networks.

Thus, we compare CS-FNE to CSFQ in different angle. For each experiment, we compute the processing time. In Table 3, we assumed that each UDP flow's sender transmits packet over one second in first experiment and compute the average processing time in edge and core node. In Table 4, we also compute the average processing time in edge and core node while the TCP flow's sender transmits the packets over one second in second experiment. As we adopt FNE to Core-Stateless network, we can reduce complexity of estimating. Therefore, routers can process packets rapidly. The tables are express in ms.

	Edge Node	Core Node	Summation
CSFQ	2877	2943	5820
CS-FNE	2621	2787	5408

Table 3. Comparison between CSFQ and CS-FNE for processing time of first experiment in single congested link.

	Edge Node	Core Node	Summation
CSFQ	2889	2936	5825
CS-FNE	2597	2714	5311

Table 4. Comparison between CSFQ and CS-FNE for processing time of second experiment in single congested link.

In both first experiment and second experiment, CS-FNE process packets faster than CSFQ about 8%.

Second, we consider multiple congested link and topology is shown in Fig 4-4. All UDPs, except UDP-0, send at 2 Mbps.

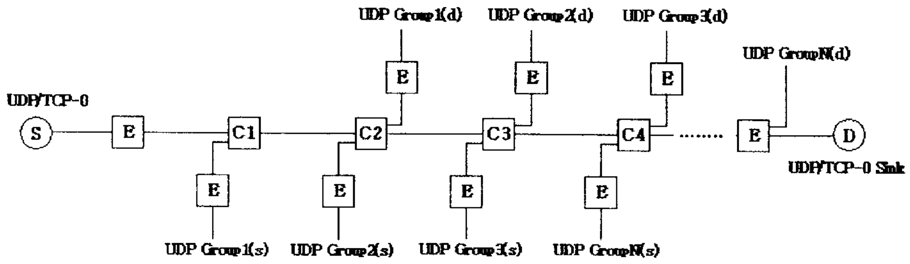

Fig. 4-4 The multiple congested link have capacity of 10 Mbps and propagation delay of 1ms

In first experiment, we have a UDP flow (denoted UDP-0) sending at its fair share rate of 0.909 Mbps.

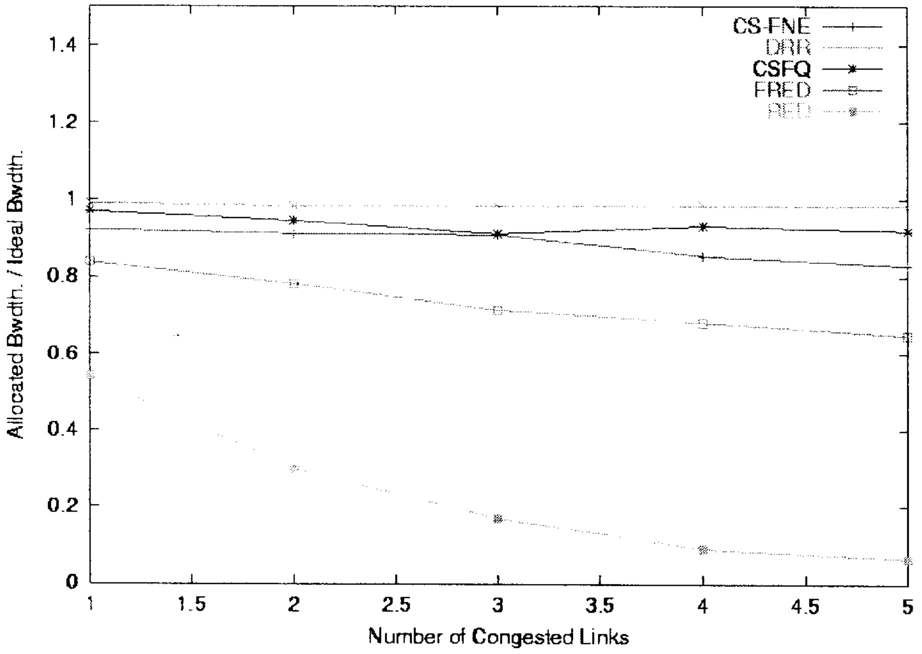

Fig. 4-5 The normalized throughput of UDP-0 as a function of the number of congested links

This figure shows the fraction UDP-0's traffic that is forwarded versus the number of congested link. CS-FNE and CSFQ preform reasonable well, although not quite as well as DRR. However, DRR is hard to implement in real network, as mentioned above. If we consider complexity, DRR's performance is not better.

	Average	Max - Min	Standard Deviation
CS-FNE	0.889	0.088	0.038
DRR	0.986	0.008	0.003
CSFQ	0.936	0.058	0.023
RED	0.732	0.192	0.078
FRED	0.233	0.476	0.195

Table 5. Result of first experiment in multiple congested link.

As shown in table, we also notify that CS-FNE and CSFQ perform similar degree and better than RED and FRED.

In second experiment, we replace UDP-0 with a TCP flow.

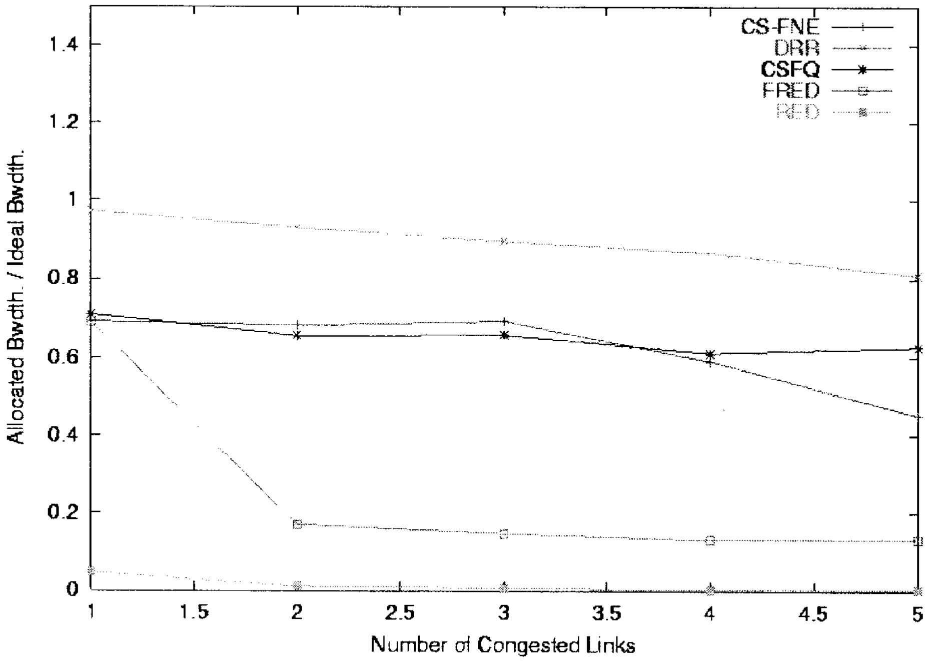

Fig. 4-6. The same plot when UDP-0 is replaced by a TCP flow.

Similarly, DRR is most ideal and CS-FNE and CSFQ perform resonable degree fairness. FRED and RED perform significantly worse. Flows with different end-to-end congestion control algorithms will achieve different throughput even if routers try to fairly allocate buffer.

	Average	Max - Min	Standard Deviation
CS-FNE	0.621	0.241	0.104
DRR	0.859	0.165	0.062
CSFQ	0.651	0.099	0.038
RED	0.143	0.689	0.306
FRED	0.013	0.048	0.023

Table 6. Result of second experiment in multiple congested link.

We also evaluate the performance of CS-FNE and CSFQ samely in above mention. CS-FNE's average is similar to CSFQ although CS-FNE's difference of ratio is larger than CSFQ.

In multiple congested link, CS-FNE's performance is similar to CSFQ so that we also consider processing time. Similarly, we compute the processing time in each experiment while sender transmits packets over one second. Table 7 shows when the flow is UDP which is denoted UDP-0 in the topology. Also, when the UDP is replaced by TCP, the processing time is shown in Table 8. The tables are express in ms.

	Edge Node	Core Node	Summation
CSFQ	2562	2832	5394
CS-FNE	2467	2448	4915

Table 7. Comparison between CSFQ and CS-FNE for processing time of first experiment in multiple congested link.

	Edge Node	Core Node	Summation
CSFQ	2501	2743	5244
CS-FNE	2387	2394	4781

Table 8. Comparison between CSFQ and CS-FNE for processing time of second experiment in multiple congested link.

The improved processing time is also shown in multiple congested link. In first experiment and second experiment, the processing time is improved about 8% similarly

5. Conclusion

In the aspect of guaranteeing throughput and reducing complexity, Drop Tail algorithm has the limitation. Therefore Active Queue Management Scheme, such as RED, have been proposed. However, this scheme still has problems of isolation, complexity and mis-behaving flow effect.

To reduce the complexity of maintaining per-flow state, Core-Stateless network is proposed. In this network, nodes are separated as the edge nodes and core nodes; Edge nodes maintain per-flow state, estimate the arrival rate and insert the label into the packet header and core nodes maintain no per-flow state and drop packets by probability. Although CSFQ is proposed to reduce complexity, it adopts exponential average so that it is necessary to reduce complexity more.

In this paper we apply FNE queueing mechanism on Core-Stateless network. In edge node, it measures the packet intensity and inserts label into packet's header. In core node, it estimates active flows as comparison between incoming packet and randomly selected packet. By using this information, it can compute threshold and decides that packet will be dropped. The steps of measuring packet intensity, estimating active flows and computing dropping probability have complexity O(1) since no packet scheduling is required.

In simulation, we evaluate it comparing with CSFQ, FRED, RED and DRR. In the single and multiple congested link, CS-FNE's performance is more effective than RED and FRED and similar to CSFQ. From another angle, we test CS-FNE's processing time and compare it to CSFQ. CS-FNE's processing time is improved about 8%. Therefore, we evaluate that CS-FNE can achieve the fairness approximately and reduce the complexity to reasonable degree than other algorithms.

Even though ideal fairness result is not proposed, CS-FNE is meaningful in the point of being implemented through the hardware more easily because CS-FNE is implemented in the traditional Drop Tail routers by simply employing a packetfiltering module in front of the single FIFO queue. Moreover it is implemented cost-efficiently since it doesn't adopt the exponential average to compute the fair rate.

From now, to get more effective solution, the improved fairness and congestion control with TCP data are necessary.

Reference

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," J. Internetw.Res. Experience, pp. 3-26, Oct.1990

[2] D. D. Clark and W. Fang, "Explicit allocation ofbest-effort packet delivery service," IEEE Trans (1998),362-373.

[3] D. Lin and R. Morris, "Dynamics of random early detection," in Proc. ACM SIGCOMM, Cannes, France, Oct. 1997, pp. 427-137

[4] Ion Stoica, Scott Shenker, and Hui Zhang,Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks," in Proceeding of SIGCOMM'98, Oct, 1997

[5] Jung-Shian Li and Ming-Shiann Leu, "Fair bandwidth share using flow number estimation," Communications, 2002. ICC 2002. IEEE International Conference on, Vol. 2, pp. 1274-1278, May 2002

[6] L Zhang, "Virtual clock: a new traffic control algorithm for packet switching networks," in Proc ACM, SIGCOMM 90, 1990, pp. 19-29.

[7] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," IEEE/ACM Trans. Networking, pp. 375-385, June 1996.

[8] NS simulator, available from http://www.isi.edu/nsnam/ns.

[9] Parekh, A. A generalized processor sharing approach to flow control

[10] R. Pan, B. Prabhakar, and K. Psounis, CHOKe: A Stateless active queue management scheme for approximating fair bandwidth allocation, Proc 19th Annu Joint Conf of the IEEE Computer and Communication Societies, INFOCOM 2000, 2000, pp. 942-951

[11] S. Floyd and V. Jacobson, "Random early detection for congestion avoidance," IEEE/ACM Trans. Networking, Vol. 1, pp. 397-413, July 1993.

[12] Z. Cao, Z Wang, E. Zegura, "Rainbow fair queueing: fair bandwidth sharing without per-flow state", Proceedings INFOCOM. March 2000, pp. 922-931

[13] Mun-Kyung Kim, Kyoung-Hyun Seo, Seung-Seob Park, "Fair Bandwidth Allocation in Core-Stateless Networks: Improved Algorithm and Its Evaluation", The 9th CDMA International Conference. October 2004, pp.526