Active Queue Management and Global Fairness Objectives

XUDONG WU IOANIS NIKOLAIDIS
Computing Science Department
University of Alberta
Edmonton, Alberta T6G 2E8, Canada
{xudong, yannis}@cs.ualberta.ca

December 2002

Abstract

We present an Active Queue Management policy and we study its ability to steer an entire TCP/IP network into global max–min fairness of its long–lived flows, in a totally distributed fashion. A previously proposed technique, initially applicable to circuit–switched networks and feedback–based flow control networks is adapted to the realities of packet–switching and TCP flow/congestion control, together with a per–hop policy, called `FairShare`. The proposed solution is evaluated using several example topologies, including linear, loop, and irregular topologies as well as dynamic load fluctuations.

Keywords: TCP, Global Max–Min Fairness, Connection Lifetime, Router Policies, Active Queue Management.

1 Introduction

One aspect of determining the value of Active Queue Management (AQM) schemes is determining if they satisfy certain end–to–end performance objectives. However, the performance of AQM schemes is often studied in settings of a single bottleneck link. While this is acceptable as a baseline case, and indeed appropriate (after all most AQM schemes do not take any action, e.g., packet drops, unless a link is congested and queues build up) it is far from sufficient. For example, it is completely unknown whether applying the particular AQM on all nodes can lead the network to a specific operating point. The existence of such an operating point, as well any features that it may exhibit, are unknown. Inversely, if a certain operating point for the network is desirable by a network operator, it is not at all clear if a particular AQM scheme can achieve it. It has to be underlined that the particulars of such an operating point cannot be arbitrary. For example, an operator may have an incentive to drive the operating point of an entire network to be such that the flows of customers in its own network obtain the lion's share of available bandwidth. Clearly, the coexistence of multiple operators will be much better facilitated, if it is known that by using the same AQM scheme on all nodes, the resulting operating point would not give an unfair advantage to any particular flow. Thus, the idea of making sure that the collective operation of AQM leads to a particular globally "fair" operating point is important and forms the basis of this paper. We narrow our focus on how a particular AQM scheme that we proposed in earlier studies, `FairShare` [3], and which achieves *max–min* fairness on a single link, can be extend, using elements of Anna Charny's earlier work [9], to achieve *global max–min fairness*.

The attention to long–lived flows in particular comes from the studied impact that long–lived flows have on the Internet. Specifically, studies of Internet traffic [1, 2, 4] indicate that the lifetime distribution of TCP connections is heavy-tailed. That is, a non-negligible fraction of TCP flows exhibit long lifetimes and accounts for a substantial volume of the total carried traffic. In order to constraint the impact of long-lived on short-lived TCP flows, several DiffServ-like classification schemes [2, 3, 4, 5] have been proposed. Previously, we proposed AQM policies based on connection lifetime classification [3, 4, 5]. In our set of control policies, different control mechanisms are imposed on the two classes (short- and long-lived) and the bandwidth for aggregated short-lived TCP flows is dynamically allocated [4]. `FairShare` is applied on the long–lived flows, while the rest of the flows, being short–lived flows, are sensitive to the instantaneous bandwidth availability rather than its long–term statistics. Therefore, we take a view according to which, fairness makes sense for long–lived flows while response time[1] makes sense for short–lived flows. Indeed, most short–lived flows,

[1]We define *response* of a TCP connection, as the total time necessary for a connection to deliver reliably all its data (retransmissions included).

are predominantly HTTP requests and responses, hence quick turnaround time is of the essence. We have addressed the benefits of lifetime–based separation of flows in a previous study [5] and noted the improvement of the short–lived flow response time if their demand is being measured on–line (essentially the frequency of SYN requests) and appropriate dynamic allocation of bandwidth is performed in a Weighted Fair Queueing (WFQ) scheduler. In this paper we address the second major aspect of lifetime classification, the network-wide performance guarantees (namely fairness) on long–lived flows. We will therefore tacitly assume that each router is, at least, minimally able to schedule two classes of traffic according to WFQ, with quasi-static per-class weights. In this paper we deal with the flows in one of the two classes (the long–lived class) and how to force them to reach global max–min fairness.

We should note from the outset that the higher complexity of `FairShare` as applied to long–lived flows is amortized by the small overall number of long–lived flows present in a network at any point in time. Technically, we assume that all flows, when first starting, are short–lived. It is only after they last for a particular amount of time that they are upgraded to long-lived flows. Due to the features of the Pareto distribution, which accurately captures the lifetime distribution of TCP flows, a flow that has lasted for a certain (long) period of time is very likely to survive for a longer period of time. Few flows transit from being considered short–lived to being considered long–lived. Hence, per-flow control for such a small subset of flows becomes reasonable. We also believe that the AQM described here is a reasonable substitute for RED [6]. RED is a well-know and heavily studied policy that uses difficult to tune parameters and presents questionable stability properties [7]. However, before we claim our set of policies as a success, further research work is necessary. First, there is a need to understand the efficiency of `FairShare` policy in a large network. Specifically, we would like to determine the minimum information exchange necessary to enforce global max-min fairness using `FairShare`. Second, we would like to study the stability properties of `FairShare` policy in large networks, where possibly multiple bottlenecks are present along a path from a source to a destination. In this paper, we attack these two problems from an experimental view on simulated network configurations.

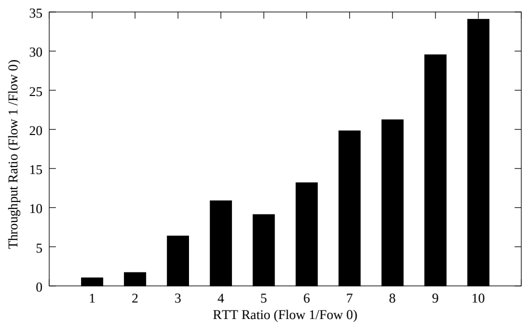
Of particular interest is the question of throughput convergence. We note that essentially no current AQM proposal exists that provably converges when deployed on realistic network topology (even in the absence of dramatic traffic demand fluctuations). In [7], RED was studied in a testbed which consisted of two CISCO routers and 16 PCs under the traffic load made of FTP traffic, HTTP traffic and UDP traffic. The research results showed that RED deployment does not improve the overall performance of network. Moreover, the study also showed parameter tuning in RED has no significant impact on the end-to-end performance. More qualitative analysis on AQM policy like RED was suggested before the universal deployment. In general, the main challenge of the analysis is the sheer complexity that is caused by the interaction of different components of the network, the incomplete information and the heterogeneity of the system. To the already complicated behavior of the end-to-end dynamics, AQM introduces a non-negligible impact from the router policies that are different than traditional First-Come-First-Served (FCFS). To cope with the complexity, we plan to extend the techniques used in [8, 9], which were used to demonstrate convergence of a policy to the optimal value regardless of initial conditions. The basic idea is to explicitly calculate the optimal value of rates for each long-lived flow and to determine if `FairShare` does indeed achieve this desired target.

What should be emphasized about the use of `FairShare` is the fact that, together with the totally distributed control for *max–min* fairness, it does not require complicated parameter tuning. Therefore, unlike schemes like RED whose ability to properly control flows lies in their accurate parameterization, the presented scheme controls all aspects of its behavior via self–contained measurement and control without user/operator intervention. To our knowledge, it is the first time that an AQM scheme is both capable of reaching a global objective as far as the TCP flow performance goes, and avoids the need for external parameterization of its operation.

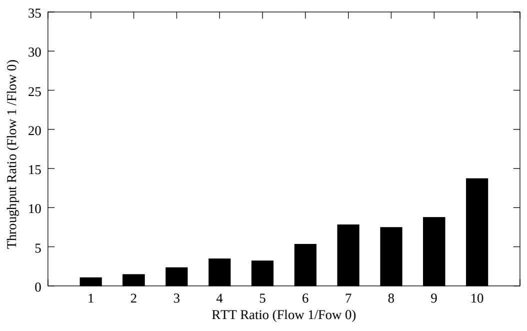
2 The FairShare Scheme

TCP unfairness rooted in the different round–trip time (RTT) values of different TCP flows has been identified a decade ago, but has yet to be solved in a satisfactory manor. Due to the different RTT of TCP flows, not all TCP flows sharing the same congested link receive the implicit congestion signals in DropTail policy. And even if they receive the signals at the same time, they react with a different speed. Thus, TCP flows with shorter RTT gain bandwidth more than their fair share. Random Early Detection (RED) was proposed as a policy to solve the problem. The essence of RED is that although not all of TCP flows will receive congestion signals at the same time, the TCP flows with more packets stored in the buffer are more likely to receive such congestion signal. However, our simulation shows that RED is far from being sufficient to ensure fairness. In Figure 1, we observe that when we compare the relative throughput of two otherwise indistinguishable TCP flows competing for the bandwidth of a congested link (the link bandwidth in the example is 100 packets per second) the ratio of the RTTs between the two flows influences the received throughput. RED improves the situation but does not correct it.

FairShare policy is proposed to solve the fairness problem of long-lived TCP flows. Essentially, FairShare is a model–based per-flow management policy that regulates individual TCP flows via scheduled losses. The steady congestion window behavior of a single isolated TCP-Reno flow is depicted in (Figure 2). The relationship between the peak value of window, $\mathtt{W}$, and the long term average value $\mathtt{W}'$ can be represented by the equation: $\mathtt{W}' = \frac{3\mathtt{W}}{4}$ or $\mathtt{W} = \frac{4\mathtt{W}'}{3}$. Thus, with the external information of RTT, once we know the expected rate for a TCP flow, we can convert the average



(a) `DropTail`



(b) `RED`

Figure 1: Comparison of the ratio of throughput achieved under, (a) `DropTail` and, (b), `RED`, of two TCP flows. The RTT of one flow is fixed at 100 msec, while the other varies from 100 msec to 1 sec.

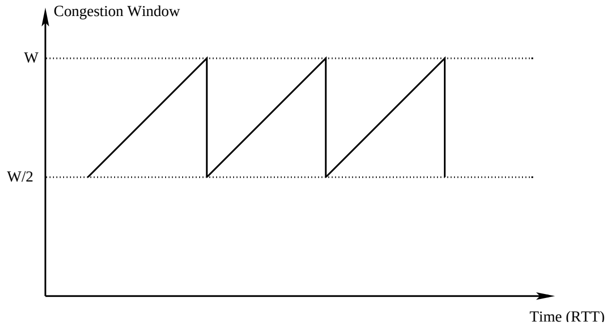


Figure 2: Example of the steady state congestion window behavior of TCP–Reno.

window value to the peak value and impose a packet loss whenever the observed congestion window exceeds the peak value. Isolation of TCP flows is guaranteed by assigning a single sub-queue for each competing TCP flow.

The algorithm of `FairShare` is described in three component schemes: the one that identify long-lived flows (Figure 3), the scheme that detects the demand (Figure 4) and the one that imposes the target rate (Figure 5). As we can see in **init_long_flow** (Figure 3), a TCP flow starting to send data through the router is initially assumed to be short lived and competes with the rest of

init_long_flow(p):

```
1. rtt_i ← rttlookup(p.src, p.dst);
2. count_i ← p.size;
3. demand_i ← $C_l^{\text{L-TCP}}$;
4. share ← maxmin(demand,$C_l^{\text{L-TCP}}$);
5. flag_i ← FALSE;
6. dropevent_i ← 0;
7. tick();
```

Figure 3: The **init_long_flow**() function.

the short–lived flows in the bandwidth pool for short-lived TCP flows, $C_l^{\text{S-TCP}}$. After a period of time, if the flow is still active, it is upgraded to long lived and entered into competition with the rest of the long lived flows, in the bandwidth pool for long-lived TCP flows, $C_l^{\text{L-TCP}}$.

In our study, TCP flows are not necessarily greedy. However, because both the existence of a remote bottleneck, as well as transitions of the flow from greedy to less active (or completely inactive) need to be tracked. In **tick** (Figure 4), the average rate of a particular TCP flow is monitored using an exponentially damped moving average estimator (line 10). That is, we produce two metrics: a measured rate over the last RTT (line 3), and an average rate over the recent past (line 10). The rate over the last RTT is the basis of the window size estimator, and hence, of the instance of dropping a packet as per the loss control algorithm. The average rate, instead, is used to infer the current level of demands. All greedy flows get exactly fair allocation of link capacity. Due to observed low demand caused by either the bottleneck of remote nodes or less active of a particular flow, unused bandwidth of a flow's allocation will be allocated to the rest of the competing flows on the basis of `max-min` fairness.

Finally, function **upon_packet_arrival** (Figure 5) is the substance of operations taking place when a packet of flow `i` arrives. The only decision taken is whether the packet ought to be enqueued or dropped. Dropping the packet is a decision based on (a) whether the **tick** function

tick():

```
1.  while (1)
2.        $m \leftarrow \texttt{count}_i$ / $\texttt{rtt}_i$;
3.        $\texttt{count}_i \leftarrow 0$;
4.        if ( $\texttt{share}_i < \texttt{demand}_i$ and
            $m >$ avg2peak($\texttt{share}_i$) ) then
5.          $\texttt{flag}_i \leftarrow$ TRUE;
6.        else
7.          $\texttt{flag}_i \leftarrow$ FALSE;
8.        endif
9.        $\texttt{demand}_i \leftarrow m \cdot \beta + \texttt{demand}_i \cdot (1 - \beta)$
10.       if ( $\sum_i \texttt{demand}_i > C_l^{\text{L-TCP}}$ ) then
11.         $\texttt{share} \leftarrow$ maxmin($\texttt{demand}, C_l^{\text{L-TCP}}$) ;
12.       endif
13.       sleep($\texttt{rtt}_i$) ;
14. endwhile
```

Figure 4: The **tick**() function.

has determined it is time to do so, and (b) if enough time has elapsed since the last drop, because, otherwise, repeated losses within an RTT time would likely force the TCP flow to timeout and its throughput deteriorate substantially.

In summary, `FairShare` is a scheme which employs measurement of the packet arrival rate of TCP flows in order to determine the flow *demands*. If the demand is higher than the fair share alloted to the flow as per the utilized fairness criterion (hereby assumed max–min fairness) it inflicts losses on the flow in order to force the long-term throughput of the TCP flow to the particular alloted value. The relation of losses inflicted and long-term throughput of the corresponding flow is established through the use of a model which analytically captures the cause–and–effect relation between loss rate and window size, and hence loss rate and throughput of a single TCP flow.

upon_packet_arrival(p):

```
1.  $now \leftarrow$ time();
2.  if ( $\texttt{flag}_i$ and $\texttt{dropevent}_i + \texttt{rtt}_i < now$ ) then
3.      $\texttt{dropevent}_i \leftarrow now$;
4.      drop($p$);
5.  else
6.      $\texttt{count}_i \leftarrow \texttt{count}_i + p.size$;
7.      enqueue($p$);
8.  endif
```

Figure 5: The **upon_packet_arrival**() function.

3 RTT Estimation

A necessary ingredient of `FairShare` is the mechanism that allows the prompt estimation of a flow's RTT. We do this based on short-term sampling in the beginning stages of a flow. When a TCP connection starts, the so called three way handshake takes place. First, the client send a packet with SYN flag set. The server responds with a packets with ACK and SYN set. And then the client responds with an ACK packet. Data packets are to be transferred after handshake. To terminate a flow, the client sends a FIN packet. The server sends an ACK followed by a FIN packet. The client responds with a FIN packet. The handshake protocol of establishing a new TCP connection is described in (Figure 6).

The traditional method of measuring RTT utilizes the standard "ping" utility to collect long-term statistics. Apart from that, TCP protocol itself makes an estimation of RTT at source nodes. In order to constrain the overhead of operation and keep the semantic of TCP protocol, both methods are undesirable. However, the above methods are not applicable in our situation. We hope that edge routers performing classification have the capability of extracting the RTT information by observing arrival packet sequences through thyself on-the-fly.

We extract the information of RTT by studying the interval between first few packets of a TCP

	SOURCE		**DESTINATION**
3-Way	SYN	$\longrightarrow$	
Handshake		$\longleftarrow$	SYN/ACK
	ACK	$\longrightarrow$	
DATA	ACK	$\longleftrightarrow$	ACK
Termination	FIN	$\longrightarrow$	
		$\longleftarrow$	ACK
		$\longleftarrow$	FIN
	ACK	$\longrightarrow$	

Figure 6: TCP Handshake

flow. The first sample of RTT is the interval between SYN and ACK of three way handshake. The second sample come from the interval of last packet of three way handshake and the data packet of the first data packet. And the interval between the first data packet and the second data packet is the third sample of RTT. The technique of extracting RTT information is similar to those proposed in [11]. We use the minimum value of three RTT samples as the estimate of RTT for this TCP flow. With the measurement methodology mentioned, we analyzed RTT spectrum and distribution from the traces. Since RTT is essentially a dynamic parameter affected by queueing delay, we argue that the instantaneous RTT we measured is a low cost estimation with decent accuracy. The accuracy of similar passive RTT measuring technique is analyzed in [11].

4 A Distributed Algorithm for Global Fairness

Rigorous analysis of global fairness and distributed algorithms to achieve it were extensively studied in [9]. For a large network configuration with multiple bottleneck links and multiple flows, a *feasible* set of flow rate allocation satisfies two conditions: First, the allocated bandwidth for any flow is a non-negative number. Second, for any link in the network and the flows passing through the link,

find_global_optimal_rates:

1. Identifying the first level bottleneck link L, that is, the most congested link because of the small bandwidth or more flows passing through or both.
2. Applying local max-min fairness criteria on the flows passing through L.
3. Marking the flows passing through L as flows with limited demands calculated in the last step.
4. Adjusting the capacity of links which are passed by such flows. The new available capacity is the remaining after taking out the already allocated from the original capacity.
5. Repeating from step 1 until all flows are marked with some rates.

Figure 7: The **find_global_optimal_rates** procedure.

the sum of allocated bandwidth for these flows is smaller than the capacity of L_i. A global max–min fair allocation is a feasible allocation that maximizes the smallest element(s)/allocation(s) subject to the capacity constraints. Naturally certain exogenous constraints are also present, namely the ones on which path is used between each source–destination pair. However, the calculation of the paths is left outside the scope of this paper, and it is normally expected to be the task of a, separate, routing algorithm.

In [9], a procedure of finding global optimal rates for all flows was presented (Figure 7). The algorithm is better understood in its centralized form, where global information and synchronous operation of components in the network is needed. We will extend it to the distributed case in the next paragraph. All flows are assumed to be greedy, that is their demands are not bounded and in principle they can utilize as much bandwidth one may care to assign to them[2] The procedure illustrated in [9] terminates after a limited number of iterations. It is summarized in Figure 7.

Distributed algorithms achieving global rates were discussed in [9]. Basically, such algorithms emulate the procedure (Figure 7) in a distributed, asynchronous way. In distributed algorithms,

[2]Technically, no flow can exceed the rate of the access link rate from which the traffic is injected into the network. Thus, real systems include always flows with bounded demands.

the global information is acquired via feedback mechanism. It is observed that in order to calculate the fair allocation, a link needs to know only about the rates of the flows traversing it. In [9] the estimated allocation for the flows is carried back to the sources nodes somehow, such as piggybacking in data packets explicitly. Furthermore, each link asynchronously calculates and maintains the estimated rate for each flow an *advertised* rate according to the local max–min fairness criteria. With the feedback mechanism mentioned above, *advertised* rates for the same flow at multiple links along the path of the flow are summarized at the sources node. The source node injects data at the *stamped* rate, which is the *minimum* of all the advertised rates it received on the path. The link that corresponds to the minimum advertised rate is the *bottleneck* for the particular flow. In addition, the link maintains a *recorded* rate for each flow, which is the measured rate within the recent past. For all flows passing through a particular link, the recorded rate should be close to the advertised rate. This is the concept called M-consistency, which is crucial for the stability for the distributed algorithm. Anytime after M-consistency is violated, the advertised rate for all flows should be re-calculated. [9] proved such distributed algorithm converges to global optimum within an upper bound of time.

We use the same concepts as in [9] but we translate them to the abilities of the `FairShare` policy. First, we note that the TCP congestion avoidance algorithm is a feedback mechanism. Any loss along the path will be sent back to the source node and cause the reduction of sending rate. With such mechanism, the sending rate of a TCP is determined by the link on which the bandwidth it receives is the smallest. This is the point of the bottleneck, accumulation of the queue, and eventual loss. When `FairShare` policy is deployed in the network, it imposes a link-wide ”local” max-min fairness. The max-min rate for each flow (“advertised” rate) is calculated and imposed by the scheduled losses on per-flow basis. When a TCP flow goes through multiple `FairShare` links, its sending rate is determined by the smallest “advertised” rate along its path.

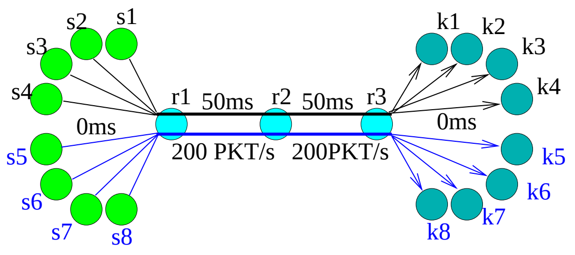
We note that by identifying a flow as ”bottlenecked elsewhere” `FairShare` avoids victimizing a flow multiple times across its path, it lets the `FairShare` instance on the bottleneck link (from the perspective of a flow) to be the only instance that will inflict losses on the flow. Apart from the scheduled loss mechanism, the `FairShare` policy also measures the delivering history of each past flow (capturing the “recorded” rate aspect). When the difference between the measured rate (“recorded” rate) and the expected max-min rate (“advertised” rate) exceeds a certain level (or M-consistency is violated), the `FairShare` policy will re-calculates the rate allocation according to max–min fairness.

5 Simulation Study

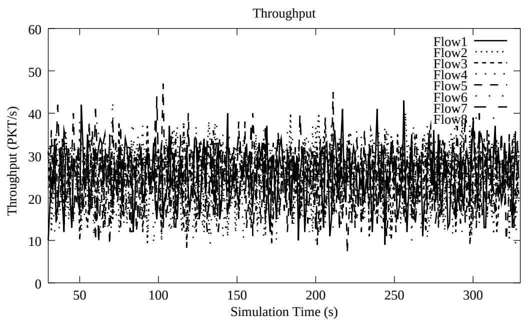
The performance of `FairShare` policy in small network environments is presented in [3]. The experiments in this section demonstrate that `FairShare` allows flows to adapt quickly to the changes of network load caused by the arrival of new (and termination of old) flows and by the variation in the demands of flows. Different network topologies are considered. The simulation experiments are conducted using ns–2 [10].

5.0.1 Experiment 1

In the first set of experiments, we investigate the impact of the bottleneck location on the performance of `FairShare`, assuming linear topology with multiple hops. The topology is depicted in Figure 8(a), which consists of eight source nodes, eight sink nodes and three router nodes. Eight TCP Reno flows are initiated from eight pairs of source and sink nodes. The bandwidth of link between router nodes are 200 packets/sec with delay of 50ms. In the first scenario, all eight flows are congested at the link r1-r2, while they are congested at the link r2-r3 in the second scenario (Figure 9(a)).



(a) Topology



(b) Goodput

Figure 8: Linear topology. Scenario 1.

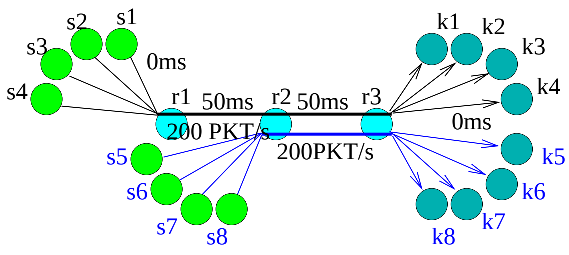
In both scenarios, the max-min rates of eight flows is expected to be 25. The throughput in both scenarios is plotted in Figures 8(b) and 9(b). The average goodputs are summarized in Table 1. We note that in the first scenario the bottleneck is the r1-r2 link, while in the second it is the r2-r3 link. The performance of our proposed FairShare policy does not appear to be affected by the location of the congestion spot along the path from source to destination.

Scenario	F1	F2	F3	F4	F5	F6	F7	F8
1	25.01	24.99	24.93	24.93	24.99	25.03	25.08	24.94
2	25.04	25.03	24.96	24.99	24.92	25.06	25.03	24.97

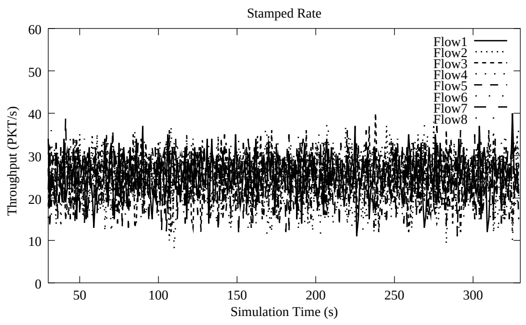
Table 1: Goodput of the eight flows in the linear topology scenarios.

5.0.2 Experiment 2

Our second set of experiments are conducted in the loop topology depicted in Figure 10(a). Four router nodes, r1, r2, r3 and r4, form a loop. The links between router nodes are with capacity of 100 packets/sec and the delay of 10ms. Four TCP Reno flows, F1, F2, F3, and F4, are initiated



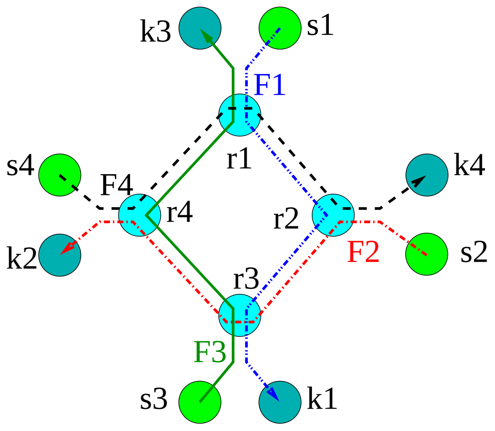
(a) `Topology`



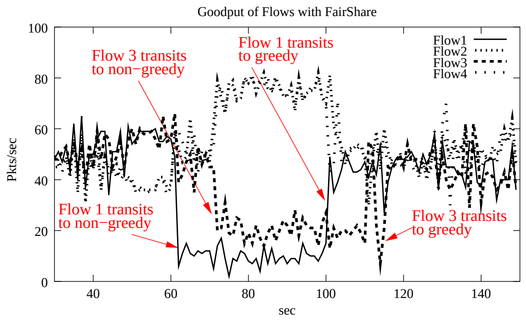
(b) `Goodput`

Figure 9: Linear topology. Scenario 2.

between the correspondent pair of source node and sink node respectively. The detailed paths for each flow are given in Table 2.



(a) `Loop Topology`



(b) `Goodput`

Figure 10: Fairness between TCP Reno flows with limited demands.

In the experiments, we investigate the performance of our proposed `FairShare` policy in dynamic environment. That is, the demands of some flows change in the process of experiments. The unit of time is presented in seconds. Specifically, flow 2 and 4 are always greedy, that is, they have

Flow ID	Path Description
F1	s1→r1→r2→r3→k1
F2	s2→r2→r3→r4→k2
F3	s3→r3→r4→r1→k3
F4	s4→r4→r1→r2→k4

Table 2: Detailed paths of the flows in the loop topology scenario.

infinite demands during the whole period of the simulation. Flow 1 is greedy from 0. At time 60, it switches to limited demands, that is 10 packets/sec. And finally, it resumes to be greedy at 100. In the similar transition process of flow 1, flow 3 transits to the status of limited demands (20 packets/sec) at time 70 and resumes to be greedy at time 110. All flows are associated with the same propagation delay (40ms).

The throughput of flows is depicted in Figure 10(b). Before time 60, due to the uniform bottlenecks, every flow gets the same bandwidth, that is, 50 packets/sec. At time 60 , the demand of flow 1 drops to 10 packets/sec. The link r4-r1 and the link r3-r4 become first level bottleneck links. Consequently, flow 3 equally share the capacity of r4-r1 and r3-r4 with flow 4 and flow 2 separately, 50 packets/sec each. The max-min rates of flow[1-4] are 10, 50, 50, and 50, respectively.

Then at time 70, the demand of flow 3 drops to 20 packets/s and the deadlock is broken. Bounded by r4-r1 and r3-r4, the throughput of flow 4 and flow 1 switch 80 packets/sec. The max-min rates of flow[1-4] are 10, 80,20, and 80, respectively. At time 100, flow 1 resumes to being greedy. The first level bottlenecks are r1-r2 and r2-r3. The max-min rates of flow[1-4] are 50, 50, 20, and 50, respectively. Finally, flow 3 resumes greedy at time 110. The evolution of the max-min rates is illustrated in Table 3. Figure 10(b) shows that with `FairShare` policy, flows determine and stabilize at the max-min rates in a dynamic environment with loop topology.

Flow ID	Phase I	Phase II	Phase III	Phase IV	Phase V
F1	50	10	10	50	50
F2	50	50	80	50	50
F3	50	50	20	20	50
F4	50	50	80	50	50

Table 3: The evolution of max-min rates in loop topology.

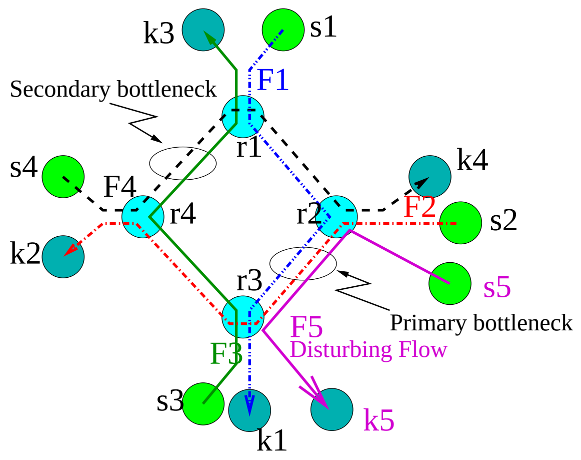
5.0.3 Experiment 3

In the third set of experiments, we investigate the impact of a disturbing flow on the performance of `FairShare`. The topology (Figure 11(a)) is similar with that of previous experiments (Figure 10(a)), except a disturbing flow 5 share link r2-r3 with flow 1 and 2. With the presence of the flow 5, link r2-r3 becomes the first level bottleneck, while link r4-r1 is the second level bottleneck.

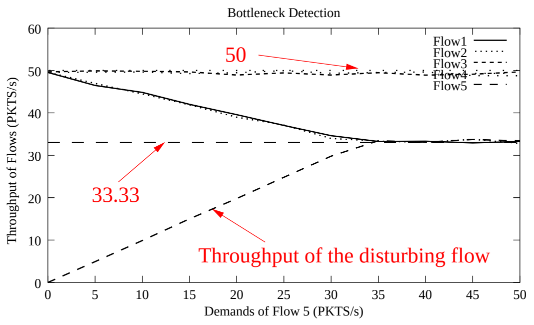
In the experiments, the demands of flow 5 increase from 0 to 50. As it is shown in Figure 11(b), when the demands of flow 5 is smaller than 33.33, the goodput of flow 5 achieves its demands, while flow 1 and 2 split the remaining capacity equally. As the demand of flow 5 is larger than 33.33, flow 1,2 and 5 share the capacity of link r2-r3 equally. In both scenario, flow 3 and 4 are bounded by the second level bottleneck, link r4-r1. In summary, with `FairShare` policy flows detect the bottlenecks of different levels and achieve the max-min rates consistently.

5.1 Experiment 4

In the last experiment we conducted simulations with the most sophisticated configurations studied in Anna Charny's work [9] . The topology and the capacities of links are shown in Figure 12(a). Like the corresponding study in [9], the object of this experiment is to investigate the response of our algorithm to dynamic changes in a sophisticated distributed environment. The applications associated with flows are of ftp style, that is, the applications always have data to deliver as long as the flows are active. The unit of time is presented in seconds and the throughput is presented



(a) `Loop topology with a disturbing flow`



(b) `Goodput`

Figure 11: Fairness between TCP Reno flows with limited demands.

in packets per second.

In this experiment, three levels of bottleneck links are presented. In the simulation, all five flows start at time 0 (Phase I). Transient portion of traces (0 - 30) are removed. In this Phase, flow 2, 3 and 4 share the first-level bottleneck link of 5 (r4-r6). Flow 1 is bottlenecked by link 1 (r1-r2). And flow 5 is bottlenecked by link 4 (s5-r4). Thus, the expected rates of flow[1-5] by global max-min fairness standard are as follows: 40, 20,20,20, and 60, respectively.

Then flow 3 terminates at time 100 (Phase II). Flow 2 and 4 share bottleneck 5 (r4-r6) equally. Like the previous phase, flow 1 is bounded by link 1 (r1-r2), and flow 5 is bounded by link 4 (s5-r4). As the result, the expected max-min rates of flow[1-5] are as follows: 30, 30,0,30, and 60, respectively.

At time 150, flow 1 and 2 also terminate (Phase III). In this phase, only two flows, 4 and 5, are active. Consequently, flow 4 is bounded by link 5 (r4-r6) and flow 5 is bounded by link 4 (s5-r4). The max-min rates for flow[1-5] are 0,0,0,60, and 60, respectively.

Finally, flow 1 resumes at time 200 (Phase IV). In this phase, the bottleneck links are link

3 (r3-r4) shared by flow 1 and 4, and link 6 (r4-r5) shared by flow 1 and 5. Consequently, the max-min rates for flow[1-5] are 50, 0, 0, 50, and 50, respectively.

In Table 4, the status of flows, the expected (analytically) max-mix rates of flows, and the measured goodputs of lows are given. Figure 12(b) shows that with our `FairShare`, all flows quickly determine and stabilize at their max-min share.

(30s - 100s) PHASE I			
Flow ID	Operate	Expected Goodput	Measured Goodput
F1	YES	40	39.40
F2	YES	20	19.80
F3	YES	20	20.20
F4	YES	20	20.11
F5	YES	60	58.98
(100s - 150s) PHASE II			
F1	YES	30	29.19
F2	YES	30	30.77
F3	NO	-	-
F4	YES	30	29.10
F5	YES	60	59.99
(150s - 200s) PHASE III			
F1	NO	-	-
F2	NO	-	-
F3	NO	-	-
F4	YES	60	60.00
F5	YES	60	60.00
(150s - 200s) PHASE IV			
F1	YES	50	49.19
F2	NO	-	-
F3	NO	-	-
F4	YES	50	50.70
F5	YES	50	50.82

Table 4: The performance of `FairShare` Policy in dynamic, complex distributed environment.

6 Conclusion

We have presented evidence of what we believe is the first time that an AQM scheme demonstrates capabilities to predictably force the entire network into a fairness objective. The solution is the synthesis of a per-link mechanism, in the guise of the `FairShare` algorithm, and a totally distributed rate allocation algorithm which is an adaptation of [9]. Through simulation experiments, we have shown the resulting AQM scheme converges to the optimal rates in various network configurations and in dynamic load environments. The `FairShare` policy appears to be self-stabilizing in the presence of dynamic network changes. The result of our investigation is crucial towards the practical applicability of global control algorithms for TCP flows in realistic environments. Our future research includes the combined global rate allocation algorithms for long–/short–lived TCP flows and a closer look into the processing and hardware needs for the implementation of these algorithms.

References

[1] N. Brownlee and K.C. Claffy, "Understanding Internet Traffic Streams: Dragonflies and Tortoises," *submitted for publication (http://www.caida.org/outreach/papers/2002/Dragonflies/).*

[2] L. Guo and I. Matta, "The War Between Mice and Elephants," *Technical Report BU-CS-2001-005,* Computer Science Department, Boston University, May 2001.

[3] Xudong Wu and Ioanis Nikolaidis, "Sociable Elephants: Fairness Among Long Lived TCP Flows," *Proc. of SPECTS 2002*, pp 502-506, July 2002.

[4] Xudong Wu and Ioanis Nikolaidis, "A Dynamic Bandwidth Allocation Scheme Based on Lifetime Classification," *submitted for publication to ICC 2003,* August 2002.

[5] Xudong Wu and Ioanis Nikolaidis, "On the Advantages of Lifetime and RTT Classification Schemes for TCP Flows," *accepted for publication in IPCCC 2003,* October 2002.

[6] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993.

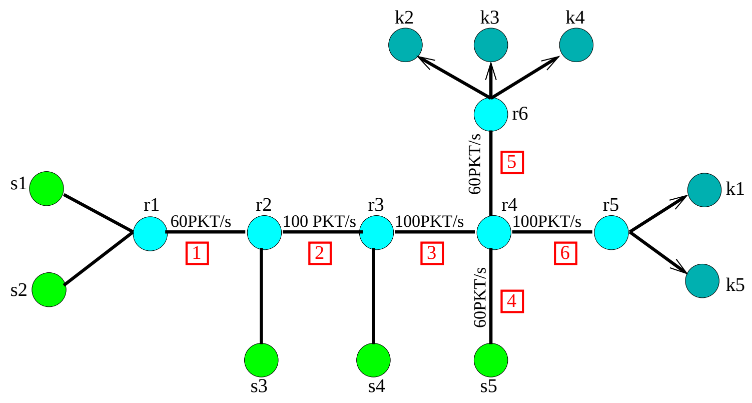
[7] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," *Proc. of the IEEE/IFIP International Workshop on Quality of Service (IWQoS'99),* June 1999

[8] K.K. Ramakrishnan, R. Jain, and D.-M. Chiu, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer. Part IV: A Selective Binary Feedback Scheme for General Topologies Methodology," DEC-TR-510.

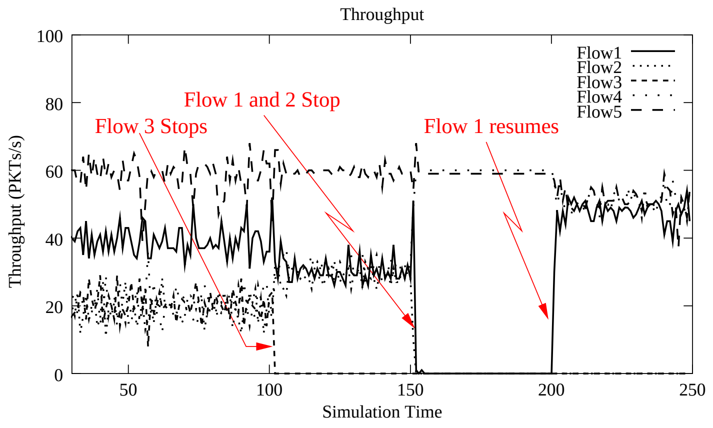
[9] Anna Charny, "An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback," *Master Thesis for EECS&EE*, MIT, 1994.

[10] UCB/LBNL/VINT Network simulator - ns (version 2), http://www-mash.cs.berkeley.edu/ns/

[11] H. Jiang and C. Dovrolis, "Passive Estimation of TCP Round-Trip Times", *ACM Computer Communication Review*, July, 2002



(a) `Complex Topology`



(b) `Goodput`

Figure 12: The most sophisticated configuration exploited by Anna Charny