Light-Weight Control of Non-Responsive Traffic with Low Buffer Requirements

Venkatesh Ramaswamy[1], Leticia Cuéllar[1], Stephan Eidenbenz[1], Nicolas Hengartner[1], Christoph Ambühl[2], and Birgitta Weber [‡,3]

[1] CCS-3, Los Alamos National Laboratory, USA
{vramaswa,leticia,eidenben,nickh}@lanl.gov
[2] University of Liverpool, Liverpool, Great Britain christoph@csc.liv.ac.uk
[3] Unilever R&D, Port Sunlight, UK birgitta.weber@unilever.com

Abstract. We propose ESREQM (Efficient Sending Rate Estimation Queue Management), a novel active queue management scheme that achieves Salmost perfect max-min fairness among flows with minimum (constant) per-flow state and a constant number of CPU operations to handle an incoming packet using a single queue with very low buffer requirements. ESREQM estimates sending rates of flows through a history discounting process that allows it to guarantee max-min fairness by automatically adapting parameters. It can also be used to punish non-responsive flows. The per-flow state is limited to a single value per flow, which allows the flow memory to be in SRAM, thereby making packet processing scalable with link speeds. ESREQM results in good link utilization with low buffer size requirements because it provably desynchronizes TCP flows as a by-product. We show our results through a mixture of analysis and simulation. Our scheme does not make assumptions on what transport protocols are used.

1 Introduction

Congestion in a network occurs when the sum of demand on any resource in the network is greater than its available capacity. Most common resources in a network are link bandwidth, network processors and buffer space. Congestion, if left uncontrolled, will lead to a situation known as congestion collapse, a deadly situation in which the throughput of the network approaches zero and the packet delays approach infinity. The key measures of performance of a congestion control scheme are (i) fairness in resource allocation, (ii) efficiency in resource utilization (in particular link utilization and buffer requirements[1]), and (iii) computational efficiency[2]. We propose an Active Queue Management

[1] Small buffers requirements are desirable as traditional rules of thumb result in huge memory requirements for high-speed routers.
[2] We can define computational efficiency in terms of CPU cycles used and memory required by the scheme
[‡] This work was done while the author was working for Swiss Federal Institute of Technology, Zurich

(AQM)-scheme called Efficient Sending Rate Estimation Queue Management (ESREQM) that optimizes all three measures while previous work addresses at most two of the three measures. In addition, ESREQM has the desirable features of allowing punishment of unresponsive flows and of straight-forward maximum buffer size provisioning.

Congestion control schemes can be implemented at the source or at the forwarding router. When congestion control is implemented at the source, the source is expected to curtail its sending rate when it detects congestion in the network. In the TCP congestion control algorithm, a source will reduce its sending rate when it detects congestion in the form of packet loss, ECN or source quench [1]. Congestion control at the source is effective only if all the sources in the network implement a common congestion control algorithm. Unfortunately, for a heterogeneous network such as the Internet, the TCP assumption that all the sources will implement a common congestion control algorithm is naive: network nodes are owned and operated by a multitude of commercial (and governmental) entities that might choose to optimize the throughput of their own traffic in a selfish manner without regard for network-wide optimization. While TCP congestion control has worked well in practice for a while, recent advances in streaming applications have led to increased use of UDP as the transport protocol. The unresponsive nature of UDP makes it TCP-unfriendly: a UDP flow can starve a TCP flow by taking all available bandwidth [1]. Because the malicious UDP flow is sending at a very high rate, the TCP flow gets very low throughput. Real-life occurrences of TCP starvation have been reported with increasing intensity [1]. Source-based congestion control cannot guarantee fair resource allocation. Assuming it is used in combination with reasonably configured FIFO drop-tail routers, source-based congestion control scores high on the other two metrics: resource utilization and computational efficiency.

Congestion control at the router can be classified into two main groups - scheduling algorithms and queue management algorithm [2]. Scheduling algorithms usually require a separate queue for each flow at the output port of the router. Each queue is then serviced in some predetermined order, usually in a round-robin way. Well-known scheduling algorithms are Weighted Fair Queueing (WFQ) and its variants such as Worst-case Fair Weighted Fair Queueing (WF^{2}Q), Self-clocked Fair Queueing (SCFQ) and Stochastic Fair Queueing (SFQ) [1]. Scheduling algorithms can typically guarantee quality of service in terms of throughput and delay [1]. As the number of flows increases, maintaining per-flow queue becomes computationally prohibitive. In terms of congestion control measures, scheduling algorithms provide fairness and efficient resource utilization, but they are not computationally efficient.

Queue management algorithms such as Random Early Drop, CHOKe, or Stochastic Fair Blue (SFB) decide upon packet arrival whether to drop the incoming packet and usually maintain a single FIFO queue that is shared by all flows [2]. These schemes are usually computationally efficient as they do not need to maintain multiple queues and most can be configured to achieve good resource utilization for bandwidth and buffer size, but they do not guarantee fairness among flows [2].

The key idea of our scheme ESREQM is that dropping decisions are primarily based on the sending rates of traffic flows and only secondarily depend on aggregate queue length, which represents a departure from the traditional AQM paradigm that drops almost exclusively based on aggregate queue length. The implementation overhead of ESREQM is much lower compared to scheduling schemes such as WFQ because it only needs a single queue in contrast to per-flow queues in WFQ. ESREQM requires to store only the bare minimum state to guarantee fairness, which allows the flow memory to be Static Random Access Memories (SRAM) instead of Dynamic Random Access Memories (DRAM), thereby making packet processing scalable with link speeds [3]. These features make ESREQM very attractive compared to other similar schemes.

We describe the conceptual approach to estimating sending rates in Section 2. In Section 3 we develop ESREQM. After discussing control parameter settings in Section 4, we present simulation results that validates our fairness claim. Section 6 describes ESREMQ's buffer requirements: it turns out that buffer can be provisioned according to a simple function of link bandwidth and number of flows. ESREMQ has very small buffer requirements which finds an explanation in the de-synchronization of TCP flows that ESREMQ achieves as a side effect. We conclude in Section 7.

2 Conceptual Design

Most queue management schemes at the router either do not allocate bandwidth fairly to the flows or do not scale. While current approaches base packet dropping decisions on the aggregated queue size, we believe that the key to fair bandwidth allocation is to base dropping decision on the characteristics of each individual traffic flow and also on the aggregate instantaneous queue length. In this work we propose a novel queue management scheme called Sending Rate Estimation based Queue Management (SREQM) that drops a packet of flow j that arrives at time t with a probability $P_j(t)$ that depends on an estimate of the relative sending rate of flow j at time t and the aggregate queue size at time t. That is,

$$P_j(t) = f(H_j(t), Q^+(t)), \tag{1}$$

where $H_j(t)$ is an estimate of the sending rate of flow j at time t, $Q^+(t)$ is the instantaneous queue length and $f(H_j(t), Q^+(t))$ is a function of the estimated sending rate of flow i and the instantaneous queue length. This approach does not require per-flow queues, but only requires sending rate estimate book keeping. In order to enforce max-min fairness with bounded errors, a queueing scheme must maintain per-flow state [4]. We maintain minimum per-flow state information in the form of the relative sending rate of each flow.

Estimating the relative sending rates of flows can be very complex in terms of computation and memory. In [5] we systematically develop a scalable estimator easily amenable to high speed implementations. Here we define the estimator without providing the details of its development.

Without loss of generality, assume that time is divided into discrete time slots with a single packet from one of the flows in each time slot. A flow has a unique identifier given by the four tuple : (source address, source port, destination address, destination port). The relative sending rate of each flow, f_i, can be estimated by an exponentially weighted filter, $H_i(\cdot)$, given by

$$H_i(t) = \begin{cases} \left(1 - \frac{1}{T}\right) H_j(t-1) + \frac{1}{T} & \text{if the packet at time slot } t \\ & \text{is from flow } j \\ \left(1 - \frac{1}{T}\right) H_j(t-1) & \text{otherwise.} \end{cases}$$

where T is a parameter corresponding to the number of packets in history considered for estimation [5]. $H_i(\cdot)$ is an exponential smoother, where the most recent observation is weighted by $\left(1 - \frac{1}{T}\right)$, the second most recent observation is weighted by $\left(1 - \frac{1}{T}\right)^2$ and so on. This makes $H_i(\cdot)$ adapt easily to changes in the sending rate of flows. Moreover, we only need to keep track of a single number for each flow, whereas most other estimators require us to keep track of some form of history for each flow. Since $H_i(\cdot)$ estimates the relative sending rate of flow i, and the sum of the relative sending rates of all the flows should be one. We can show that

$$\sum_j H_j(t) = 1. \tag{2}$$

Our estimation procedure readily extends to the continuous case. Suppose that each flow is a realization from a Poisson process with parameter λ_j. For n flows arriving at the router, we can show that

$$\mathbf{E}\left[H_j(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i} \left(1 - e^{-\left(\frac{\sum_{i=1}^n \lambda_i}{T} t\right)}\right), \tag{3}$$

and at steady state,

$$\mathbf{E}\left[H_j(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}, \tag{4}$$

which is the relative sending rate of flow j.

In the following section, we describe an algorithms to guarantee fairness, which follows directly from conceptual design.

3 Efficient Sending Rate Estimate-Based Queue Management Scheme : An Algorithm for Efficiency and Fairness

Most queue management schemes do not allocate bandwidth fairly or achieve fair bandwidth allocation only at high complexity. While most queueing schemes drop packets based on the average queue size, SREQM estimates the relative sending rate of flows using the estimator $H_i(\cdot)$ and uses this estimate along with the instantaneous aggregate queue length to drop packets from each flow. The pseudo-code for the whole procedure is given in Algorithm 1.

Algorithm 1 ESREQM :: onPacketArrival(packet P)

```
1: $x \Leftarrow$ flow id of packet $P$
2: Update the sending rate of each flow based on the estimator, $H_i(\cdot)$
3: if ($H_x \leq K$) then
4:     add packet $P$ to the queue;
5: else
6:     drop packet $P$
7: end if
8: if ($count > 0$) then
9:     $count^{--}$
10: else
11:     if (queue size $< q_{min}$) then
12:         $K^{++}$
13:         $count \Leftarrow F$
14:     end if
15:     if (queue size $> q_{max}$) then
16:         $K^{--}$
17:         $count \Leftarrow F$
18:     end if
19: end if
```

The algorithm works as follows: When a packet arrives, extract the flow-id of the packet. Update the estimate $H_j(\cdot)$ for all the flows. If the value of $H_j(\cdot)$ of the flow from which the packet arrives is greater than K, drop the packet, else add the packet to the queue. The parameter K is called the fair-share parameter and it represents the maximum share of the bandwidth a flow can get. The flows need not be restricted when there is no congestion. We change the value of K dynamically to reflect the changes in the characteristics of the incoming traffic as well as the level of congestion. This change is governed by the current queue size. If the queue size is larger than some maximum threshold q_{max}, which is an indication of congestion, the value of K is decreased by one. This results in restricting the sending rate of flows. Likewise, when the current queue size is below some minimum threshold q_{min}, which is an indication of low link utilization, the value of K is increased by one, allowing flows to come in at a faster rate. To ensure a smooth variation of K, the update procedure of K is done once in F packet arrivals. The parameter F is called the congestion parameter and is a representative of how fast the system responds to congestion.

4 Setting the Parameters of ESREQM

ESREQM has three main parameters: the history parameter - T, the fair share parameter - K and the congestion parameter - F. Setting the parameters to the right values is crucial for the proper functioning of the algorithm. In this section, we give some engineering guidelines to set the parameters of the algorithm.

The effective estimation of the relative sending rate depends on appropriately selecting the value of the history parameter, T. To get a feel for how the selection of T matters, consider two extreme cases, $T = 1$ and $T = \infty$. For $T = 1$, the estimation will not be correct as $H_i(\cdot)$ can only have two values - one or zero. For $T = \infty$, $H_i(t) = H_i(t-1)$ and assuming $H_i(0) = 0$, $H_i(\cdot)$ will always be zero making the estimate incorrect. The case for $T = \infty$ is counter intuitive because higher the value of T, larger the history and we would expect the estimate to be better. It can be shown that as T goes up the variance decreases, but the bias increases, thereby making the estimate unacceptable. A value of 400 for the parameter T yields good performance [1].

The value of the parameter K varies dynamically based on the level of congestion. The initial value of K does not have a major impact on the performance of the algorithm. Our simulations suggested that the K will finally converge to $\frac{T}{n}$, where n is the number of flows. The initial value of K used in our simulations was 50.

The congestion parameter, F, determines how fast we change K. This also determines the level of penalty that we impose on flows that send at a rate higher than their fair share. The higher the value of F, the larger the level of penalty we impose. Various simulation results [1] show that a value of approximately 200 for parameter F will result in a fair bandwidth allocation.

5 Experimental Results

In order to demonstrate the effectiveness of our scheme in achieving fairness, we perform several simulation studies using *network simulator (ns-2)*. There are many definition of fairness, but here we use one of the most common definitions, which is the max-min fairness [1].

We conduct studies on a dumbbell topology as shown in Figure 1. For the dumbbell topology there are n sources and n destinations numbered S_1 to S_n and D_1 to D_n respectively. Two routers R_1 and R_2 connect the sources to their destinations. Each of the links is assigned a bandwidth of 10Mbps, which means that the bottleneck link will have n times less bandwidth than the combined bandwidth of the access links. We illustrate the the effectiveness of our scheme using an example experiment with 32 flows, out of which 31 are TCP flows and one is a UDP flow with sending rate 1Mbps. The bottleneck link bandwidth is 10Mbps, which means that the UDP is sending at a much higher rate than its fair share. Figure 2 shows the average throughput of all 32 flows over a period of 500 sec in the form of a box plot. We can see that all the flows get approximately the same bandwidth, which is close to their theoretical fair share. The UDP flow which was sending at a rate much higher than its fair share was brought down close to its fair share. More experiments with other topologies such as the parking lot topology are given in [1].

We now discuss the conclusions that can be drawn from the simulation studies. ESREQM approximates max-min fairness in all the scenarios considered. The level of fairness is much better than that of many other schemes such as RED, CHOKe, SFB or FRED. While some of the other schemes do not maintain per-flow state information, we only maintain minimum state information which is required to guarantee max-min fairness. ESREQM does not require precise parameter tuning and the max-min bandwidth allocation degrades only very gradually when the parameters are far out of range [1].

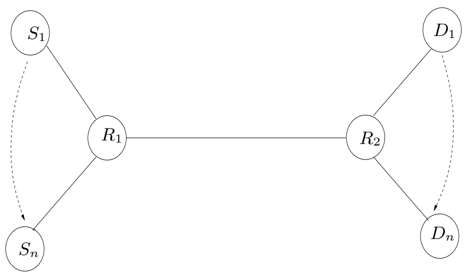

Fig. 1. Dumbbell Topology

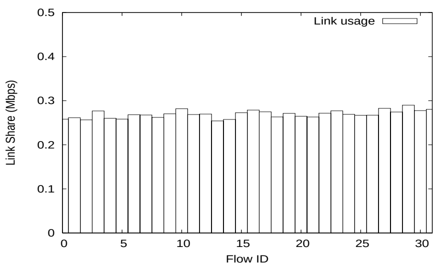

Fig. 2. Average Throughput of 32 flows

6 Buffer Requirements

Accurately sizing the buffer is one of the most challenging parts of a queueing scheme design [6]. Low buffers can cause excessive packet losses and large buffers can lead to unacceptable packet delays. The buffer requirements of the queueing scheme is heavily influenced by the packet dropping mechanism used by the scheme.

Buffer in today's routers are provisioned based on a rule of thumb by Villamiziar *et al.* [7], which says that the amount of buffer required is the end to end round trip time (RTT) times the link bandwidth (c). This rule, though developed in the early 90s is still widely accepted. The rule was developed to make a congested link busy 100% of the time assuming very few TCP flows traversing the router with a droptail queue. Based on this rule, today's routers, which operate at a speed of around 50Gbps would require 50Gbps x 250ms (typical RTT) = 12.5Gbits of memory.

Router buffers are usually built using SRAMs or DRAMs. SRAMs are fast and expensive whereas DRAMs are slow and cheap. In order to build 12.5Gb of router buffer using SRAMs, based on current standards, it will require approximately 300 SRAM chips on a board, making the board too big and too hot. On the other hand, if we use DRAM, due to slow access speed, parallel implementation will be necessary making the design too complicated [6]. Future routers will be required to operate at much higher speeds, making the buffer design a much more complicated problem. There is definitely a need for queue management schemes which has low buffer requirements.

Appenzeller *et al.* [6] showed that with a large number of flows and a droptail buffer, the buffer requirement can actually come down. They show that the buffer requirement can be as low as $\frac{c(RTT)}{\sqrt{n}}$, where n of the number of flows, RTT is the end to end round trip time and c is the link bandwidth. Gorinsky *et al.* [8] showed by simulation that the above result does not hold when the number of flows is in the order of 100 flows, which would be the case in slower access links serving fewer connections.

In order to get some understanding of the buffer requirements under our scheme, we perform simulation studies on a congested router which implements ESREQM. Two sets of experiments, one with just TCP traffic and other with both TCP and UDP traffic (UDP traffic less than 10% of the total traffic) is performed.

In the first set of experiments, only TCP traffic is considered, and we vary the number of TCP flows, while scaling the maximum queue capacity proportionally. Let $Q^+(t)$ be the queue size at time t and n be the number of flows. Figures 3, 4 and 5 plot the evolution of queue size divided by the number of flows, $\frac{Q^+(t)}{n}$, when the number of flows is 16, 32 and 64 respectively. Surprisingly, the variations in $\frac{Q^+(t)}{n}$ decrease as we increase the number of flows. When the number of flows approaches infinity, $\frac{Q^+(t)}{n}$ approaches a constant. Our estimation of the required buffer size is based on this observation.

Define $Q_i(t)$ to be the number of packets from flow i on the queue at time t, *i.e.,* $\sum_{\forall i} Q_i(t) = Q^+(t)$. Assume the following conditions are valid.

- $(Q_1(t), \cdots, Q_k(t))$ are exchangeable [3] [1]
- $\mathrm{Cov}(Q_1(t), Q_2(t)) = cn^{-\gamma}$

where c is a constant and $\gamma \geq 1$. With the above assumptions, the central limit theorem [1] can be applied and we can write

$$\mathbf{P}\left\{\frac{Q^+(t) - \mathbf{E}[Q^+(t)]}{\sqrt{n}\sigma} \leq x\right\} \approx \Phi(x), \tag{5}$$

[3] Any size k subset of the n random variables has the same joint distribution as $(Q_1(t), \cdots, Q_n(t))$

where $\Phi(\cdot)$ denotes a normal distribution and σ is the standard deviation of $Q_i(t)$. To empirically verify if the central limit theorem holds, we want to ascertain if the distribution of $Q_i(t)$ is Gaussian. A common technique to test if a data set comes from a Gaussian distribution is to plot the observed quantiles versus the theoretical quantiles in a quantile-quantile (Q-Q) plot. The Q-Q plot of $Q_i(t)$ for $n = 128$ is given in Figure 8. The plot is approximately linear indicating that $Q_i(t)$ is approximately Gaussian. Define $\mathbf{B}_\alpha$ to be the buffer size such that the probability that the queue size, $Q^+(t)$, will be less than or equal to $\mathbf{B}_\alpha$ is α. For the specific case of $\alpha = 0.999999$, we can compute $\mathbf{B}_{0.999999}$ as

$$\mathbf{B}_{\mathbf{0.999999}} = n\hat{\mu} + \sqrt{n}\hat{\sigma}\sqrt{12 \log 10}, \tag{6}$$

where $\hat{\mu}$ is the sample mean and $\hat{\sigma}$ is the sample standard deviation of $Q_i(t)$. Table 1 summarizes the various statistics of $Q_i(t)$ for different number of flows. Using Table 1, we can compute $\mathbf{B}_{0.999999}$ for ESREQM with 32 flows as 135KB. This means that for ESREQM the probability of exceeding a buffer size of 135KB with 32 flows is less than 10^{-6}. Note that with the same settings, droptail queue will require 500KB.

To investigate the effect of link bandwidth, we kept both the number of flows and the parameters of the algorithm constant, while varying the link capacity. We observed an increase in the average sending rate of flows, with $Q_i(t)$ remaining the same. In other words, the change in the link bandwidth did not affect the average queue size and our estimation of $Q^+(t)$ remains valid. Note that in all our experiments, the link utilization was 100%.

We still see the convergence when there are UDP and TCP flows. The average queue size for the case when there are 64 TCP flows and one UDP flow and well as the case when there are 128 TCP flow and one UDP flow are shown in Figure 6. We can observe that the variations in the average queue size is lower in the case with 128 TCP flows.

The convergence of the average queue length was observed in RED routers under very special conditions of dropping function when there are only TCP flows [9]. In our scheme the convergence happens without any special conditions and also with TCP and UDP traffic.

The two main conclusions that we can draw from the simulation results presented in this section are as follows. ESREQM requires much lower memory than most of the other queueing schemes for same performance under similar network conditions. For a given number of flows, with ESREQM, we can accurately estimate the amount of buffer needed. This is very important for the optimal performance of any queue management scheme.

6.1 An Explanation for the Low Buffer Requirement

An intuitive explanation for the low buffer requirements of our queueing scheme is presented in this section. The congestion window process of a TCP flow typically has a form of a saw-tooth waveform [10]. Define $\tilde{W}_i(t)$ to be process describing the congestion window for flow i and $Q^+(t)$ be the aggregate queue length at time t. The queueing process [10] at the router can be described as

$$Q^+(t+1) = Q^+(t) + \sum_{i=1}^{n} \tilde{W}_i(t) - c\tau, \tag{7}$$

where c is the rate at which the router serves the packet τ is the average end to end round trip time of each flow. $\tau = \tau_p + \tau_q$, where τ_p is the average propagation delay and τ_q is the average queueing delay. The above equation can be re-written as

$$Q^+(t+1) = Q^+(t) + \sum_{i=1}^{n} \tilde{W}_i(t) - c\left(\tau_p + \frac{Q^+(t)}{c}\right) = \sum_{i=1}^{n} \tilde{W}_i(t) - c\tau_p. \tag{8}$$

It is obvious from the above equation that the maximum queue size is reached when all the flows are synchronized and the maximum queue size will depend on the maximum congestion window for each flow. When there are many synchronized TCP flows with each of these flows having a “saw-tooth” congestion window, the effective congestion window will also have a form of a saw-tooth. This is a consequence of the fact that adding many synchronous saw-tooth waveforms will lead to a single large saw-tooth. For synchronous TCP flows, this phenomenon is illustrated in [10]. Previous studies have shown that even though two TCP flows start at different times, they get quickly synchronized in real networks when the routers are employed with droptail or RED [6]. Therefore, from (8), we expect no improvement in the buffer size requirement when there are many synchronized TCP flows.

With our queueing scheme, the TCP flows get de-synchronized quickly as the dropping criteria for packets from a flow is not completely dependent on other flows (the dropping is a function of both $H_i(t)$ and $Q^+(t)$ and not just $Q^+(t)$). If we add together many de-synchronized saw-tooth waveforms, their sum is less likely to be like a saw-tooth waveform as they smooth each other out, and we get a wave form which is less varying. When we add many de-synchronized congestion window processes, the same thing happens. The resulting window process will have very less variation and the peak of the effective window process comes down, which results in much smaller buffer requirement. This is illustrated in Figure 7. We plot the sum of congestion window ($\sum_{i=1}^{n} \tilde{W}_i(t)$) of all the flows when the flows are synchronized (as in the case of droptail) and when the flows traverse through a congested router implementing ESREQM. Since the flows get desynchronized with ESREQM, the maximum of $\sum_{i=1}^{n} \tilde{W}_i(t)$ is much lower resulting in a low buffer requirement.

7 Conclusion

In this paper, the problem of allocating max-min fair bandwidth to flows in a congested router is addressed. We presented architecture and algorithms,

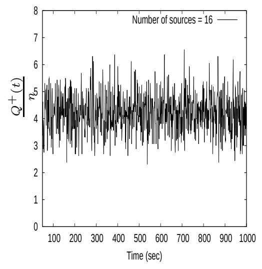

Fig. 3. Average queue length over time with 16 TCP flows.

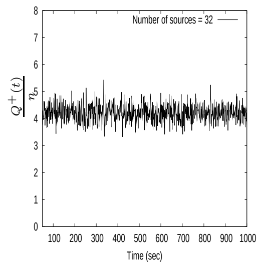

Fig. 4. Average queue length over time with 32 TCP flows

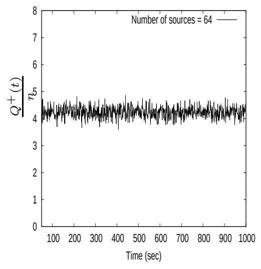

Fig. 5. Average queue length over time with 64 TCP flows

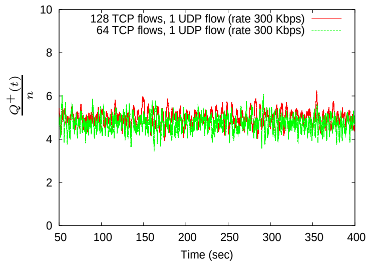

Fig. 6. Average queue length over time with 1 UDP flow and different number of TCP flows

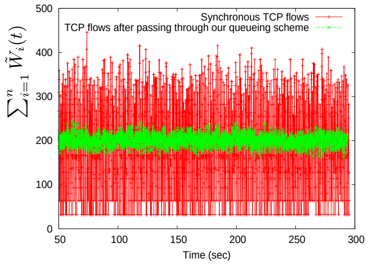

Fig. 7. Aggregate Congestion Window

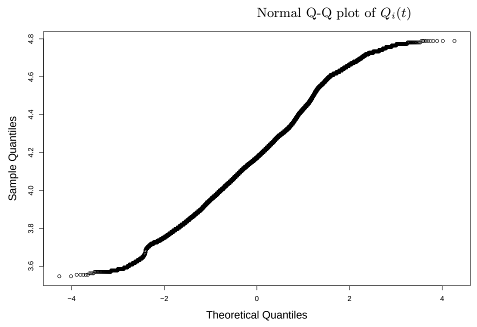

Fig. 8. QQ plot of $Q_i(t)$ when n=128

Table 1. statistics of $Q_i(t)$ for different number of flows (n)

n	Average (pkts)	Std. deviation
8	4.250	1.1077
16	4.217	0.7605
32	4.209	0.5388
64	4.222	0.4545
128	4.2452	0.3423

along with simulation results of a simple scalable scheme which can provide approximate max-min fair bandwidth allocation. The buffer requirements of the scheme is shown to be much lower than the conventional routers, which is a great advantage from the implementation perspective. We showed by a combination of analysis and simulation that our scheme performs well simultaneously in terms of three key measures (i) fairness in resource allocation (ii) efficiency in resource utilization and (ii) computational efficiency.

References

1. V. Ramaswamy, "Efficient Control of Non-Cooperative Traffic Using Sending Rate Estimate-Based Queue Management Schemes," Ph.D. dissertation, The University of Mississippi, 2006.
2. R. Pan, B. Prabhakar, and K. Psounis, "CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," in *Proc. of IEEE INFOCOM'00*, July 2000, pp. 942–951.
3. G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*, 1st ed. Morgan Kaufmann, December 2004.
4. A. Das, D. Dutta, A. Goel, A. Helmy, and J. Heidemann, "Low State Fairness: Lower Bounds and Practical Enforcement," in *Proc. of IEEE INFOCOM'05*, Month 2005, pp. 2436–2446.
5. V. Ramaswamy, L. Cuellar, S. Eidenbenz, and N. Hengartner, "Preventing Bandwidth Abuse at the Router through Sending Rate Estimate-Based Queue Managemnet Schemes," in *Proc. of IEEE ICC'07*, May 2007.
6. G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proc. of ACM SIGCOMM'04*, August 2004, pp. 281–292.
7. C. Villamizar and C. Song, "High Performance TCP in ansnet," *ACM Computer Comm. Review*, vol. 24, no. 5, pp. 45–60, 1994.
8. S. Gorinsky, A. Kantawala, and J. Turner, "Link Buffer Sizing: A New Look at the Old Problem," in *Proc. of ISCC'05*, June 2005, pp. 507–514.
9. P. Tinnakornsrisuphap and A. M. Makowski, "Limit Behavior of ECN/RED Gateways Under a Large Number of TCP Flows," in *Proc. of IEEE INFOCOM'03*, April 2003, pp. 873–883.
10. J. Sun, M. Zukerman, K. Ko, G. Chen, and S. Chan, "Effect of Large Buffers on TCP Queueing Behavior," in *Proc. of IEEE INFOCOM'04*.