RQM: A new rate-based active queue management algorithm

Jeff Edmonds

Suprakash Datta

Patrick Dymond

Kashif Ali

Technical Report CSE-2006-09

September 1, 2006

Department of Computer Science and Engineering

4700 Keele Street North York, Ontario M3J 1P3 Canada

RQM: A new rate-based active queue management algorithm

Jeff Edmonds, Suprakash Datta, Patrick Dymond, Kashif Ali
Computer Science and Engineering Department,
York University, Toronto, Canada

Abstract

In this paper, we investigate a new active queue management algorithm RQM based on rates of traffic passing through a router instead of the buffer occupancy inside a router. Our algorithm uses a novel method of estimating the number of active sessions without explicitly maintaining or using per-session states at the routers. We prove that our algorithm results in fair allocations of bandwidth to competing sessions, under some simplifying assumptions. Using simulations, we show that our algorithm is superior to the well-known RED and REM algorithms in many scenarios. Our algorithm performs particularly well when the number of active sessions is very high.

1 Introduction

Active queue management (AQM) has been an important research area in networks since the early days of the Internet. Active queue management improves performance in several ways. First, it improves throughput by preventing buffer overflow and the subsequent retransmissions and window size reduction. Second it removes synchronization among flows by spreading out losses over time. Desynchronization of sessions has the additional advantage of reducing the fluctuations in the bandwidths allocated to sessions – many applications, including real-time multimedia applications, require the bandwidth allocated to them to have low variation over time. Finally, active queue management can prevent packet losses in two ways. First, it is a proactive approach to congestion control, unlike TCP congestion control, which is reactive. In other words, AQM attempts to prevent congestion from occurring, while TCP reacts when congestion occurs. Second, it can mark packets (e.g., by setting a specific bit in the header) when there is an indication of possible mild congestion instead of actually dropping them. The receiver can read these marks and ask the sender to slow down its transmission rate. Thus the sender (which runs TCP) can react in the same way that it would when it learns about a packet drop. In the remainder of the paper, we use the terms "marking" and "dropping" synonymously. We also use the terms "flows" and "sessions" synonymously.

1.1 Related Work

Existing AQM algorithms fall into two broad classes. One class consists of algorithms which infer congestion by monitoring the buffer occupancy levels at routers. These are called buffer-based or queue-based algorithms. The second class controls the transmission rate of senders by monitoring the current traffic rate at a router. Algorithms in this class are called rate-based algorithms. From a mathematical perspective, these two techniques are rather different. Buffer sizes depend on the integral of rates, since the buffer size $b(t)$ at time t is given by $b(t) = \int_0^t \max\{0, \lambda(t) - \mu(t)\}dt$, where $\lambda(t), \mu(t)$ are the arrival and service rates of packets, respectively. The primary advantage of rate-based control is that it is more direct control on sending rates of different sessions. The main advantage of queue-based AQM is that it is typically simpler and often requires little or no information to be stored at routers.

1.1.1 Queue-based AQM

The best known queue-based AQM algorithm is RED (Random early detection) [8]. RED assumes that routers do not keep per-flow state and so it adjusts bandwidth depending on the buffer buildup. Specifically, it marks each packet entering a queue with a drop probability that is a piecewise linear function of the number of existing packets in the buffer when the packet enters the buffer. Low et al [13] enumerate problems associated with RED (including setting its parameters), and demonstrate that RED behaves in an unstable manner in certain situations.

Many modifications of RED have been proposed in the literature. Feng et al [3] proposed Adaptive RED, which adjusts the packet dropping probability based on the past history of the average queue size. Floyd et al [7] improve this algorithm by modifying the adaptation algorithm.

Stabilized RED (SRED) [14] maintains a flow cache (called a *zombie list*) to keep track of recent flows. The current packet is compared with a random element in the cache. If there is a hit, the packets are assumed to be part of the same flow and the relative frequency of the cache entry is increased. Otherwise, there is a cache miss, the new entry replaces the entry it was compared to with some probability. The algorithm uses the hit frequency to estimate the number of active flows, and computes the drop probability proportional to the square of the number of active flows.

LRU-RED [15] maintains a LRU-cache at routers to record information about high-bandwidth flows. It uses this information to identify high-bandwidth flows and penalize them by modifying the drop probability function used by RED for these flows.

Exponential-RED (E-RED) [12] sets the packet marking probability as an exponential function of the length of a virtual queue whose length is slightly smaller than the actual buffer.

BLUE[5, 6] maintains a marking probability based on rate information using the following intuition (note that we have omitted several details of the algorithm here). When a packet loss event happens, the marking probability p_m is increased by a constant δ_1 and when a link is idle, the marking probability is reduced by a constant δ_2. The parameters δ_1, δ_2 are set so that p_m takes minutes to grow from 0 to 1 rather than seconds.

1.1.2 Rate based AQM:

Several rate-based algorithms have been proposed in the literature. Adaptive Virtual Queue (AVQ) [10] maintains a virtual queue that is used to perform AQM. The virtual queue has a capacity that is less than the capacity of the link. The virtual queue is updated when packets arrive at the real queue and packets are marked in the real queue when the virtual queue overflows. The virtual capacity is adapted to ensure that each link achieves a desired utilization.

Several schemes that require routers to store and maintain per-flow states have been proposed [2, 11].

Some recent papers attempt to perform AQM using both rate and queue based indicators. In Random Exponential Marking (REM) [16], the *price* $p_l(t)$ of a queue is defined as a function of the current traffic rate through it as well as the buffer occupancy. REM uses this price to generate the marking probability of a packet.

1.2 Our Contributions

The main contribution of this paper is a rate-based AQM algorithm RQM. Our algorithm differs from existing algorithms in that it uses a novel way of estimating of the number of sessions in progress. It then uses this estimate to compute the probability with which a packet entering a router is dropped or marked.

We analyze RQM and prove that in the absence of sessions entering or leaving, the algorithm quickly converges to fair allocations of bandwidth to competing sessions. We also simulate RQM in *ns* [1] and perform extensive experiments to evaluate its performance. Our simulations show that RQM performs better than RED and REM in many situations. It consistently outperforms RED and REM when the number of competing sessions is large.

2 Our Model and Assumptions

We assume that each router runs RQM. So, each router needs to store and update three integer variables; these will be described later. A router stores no per-session information and is not required to maintain any caches.

We also assume, for simplicity, that the network has a single bottleneck link. Such assumptions have been made and justified in the literature. Henceforth, we refer to the link capacity (respectively propagation delay) of the bottleneck as the bottleneck capacity or bandwidth (respectively bottleneck propagation delay). We also refer to the capacity of the buffer associated with the bottleneck link as the bottleneck buffer size.

For our analyses, we assume, like many other papers in the literature, that the only traffic in the network is generated by TCP, i.e., there is no UDP traffic. The significance of this assumption is that our algorithm assumes all flows to be responsive. UDP flows are unresponsive at best, and may even react selfishly in the worst case (e.g., applications could conceivably increase the sending rate) when some of its packets are dropped. We show in the simulations that RQM works well in the presence of constant-bit-rate (i.e., benign) UDP traffic. We assume that the TCP sources have standard parameters for doing additive increase ($\alpha = 1$) increase and multiplicative decrease ($\beta = \frac{1}{2}$).

3 Our Algorithm

In this section, we describe our algorithm RQM for active queue management. RQM, like RED, is a randomized algorithm. Like SRED, we estimate the number of sessions in progress, and use this to determine the dropping probability. However, unlike SRED, we do not use or maintain a cache to keep track of recent sessions. Like REM, RQM computes rate-based feedback from the buffer occupancy at different times and uses it to drop packets. Like most other AQM schemes our algorithm has several objectives:

- We want RQM to be proactive in controlling congestion and minimizing packet loss. One simple way of minimizing the number of packets being dropped is to introduce a virtual queue size that is smaller than the actual queue size.
- We want RQM to contribute towards fairness of allocations. Like RED, we would like RQM to drop more packets from sessions which have higher than their fair shares of the bandwidth, thus bringing the system to fair allocations faster.

RQM has three components: it defines a desired utilization function in terms of the number of sessions, uses this function to estimate the number of active sessions n and computes the packet drop probability as a function of n. Since RQM is a rate-based algorithm we describe it in terms of sending rates of flows (or equivalently, the bandwidth allocated to flows) rather than TCP window sizes. Let $b_{i,t}$ be the bandwidth allocated to session i at time t.

Defining a desired utilization: The desired bandwidth utilization is defined by a function $\tilde{b}(n_t)$, where n_t is the current number of active sessions. A good candidate is $\tilde{b}(n) = (1-\gamma)(1-e^{-cn})B$. Note that the parameter γ controls the size of the virtual bandwidth. The parameter c controls the tradeoff between link utilization and accuracy of inference of number of sessions: a lower value of c would result in lower utilization of the bandwidth but would allow the number of sessions to be determined more accurately.

Estimating the number of active sessions n: RQM estimates the number of active sessions n using the following intuition: when the network is at steady-state, and the current total bandwidth is b_t, the inverse function $\tilde{n}(b_t) = \tilde{b}^{-1}(b_t)$ would yield the number of active sessions. RQM uses $\tilde{n}(b_t)$ as an estimate of n. While the system will not be in steady-state at all times, we will show in Section 4 that it moves towards a steady state at all times. Thus, the estimate will almost always be fairly accurate.

Computing the drop rate: RQM uses the function $p(b_t) = \frac{\alpha}{(1-\beta)}\left(\frac{\tilde{n}(b_t)}{b_t}\right)^2$, as the probability of "dropping" packets. Note this rate depends only on the total bandwidth $b_t = \sum_i B_{i,t}$ and not on rates or numbers of individual sessions.

The intuition for the function $p(b_t)$ is as follows. Let us assume first that the roundtrip times for all sessions are equal. Then, if there are n active sessions, the the total bandwidth increases at a rate of αn, since each session increases its bandwidth at an fixed additive rate of α. RQM attempts to maintain a steady-state with fair allocation of bandwidth, i.e. that in which each of the n sessions have bandwidth $b_{i,t} = \frac{\tilde{b}(n)}{n}$ for a total of $b_t = \tilde{b}(n)$. Assuming that the network is in this steady-state, inducing a single session to reduce its rate decreases its bandwidth from $b_{i,t} = \frac{\tilde{b}(n)}{n}$ to $\beta b_{i,t} = \beta\frac{\tilde{b}(n)}{n}$. This decreases the total bandwidth by $(1-\beta)\frac{\tilde{b}(n)}{n}$. There are $\tilde{b}(n)$ packets per time unit and each gets dropped with probability $p(b_t)$, hence the frequency of rate reductions is $p(b_t) \cdot \tilde{b}(n)$. Hence, these reductions decrease the total bandwidth at a rate of $p(b_t)\tilde{b}(n)(1-\beta)\frac{\tilde{b}(n)}{n}$. RQM maintains the current total bandwidth by balancing this increase and this decrease, namely $\alpha n = p(b_t)\tilde{b}(n)(1-\beta)\frac{\tilde{b}(n)}{n}$. This would be done by setting the drop probability to $p(b_t) = \frac{\alpha}{(1-\beta)}\left(\frac{n}{\tilde{b}(n)}\right)^2$. Since the number of sessions n is unknown, RQM uses $\tilde{n}(b_t)$ as an estimate. Thus the drop rate is $p(b_t) = \frac{\alpha}{(1-\beta)}\left(\frac{\tilde{n}(b_t)}{b_t}\right)$ when the current total bandwidth is b_t.

We assumed for simplicity that all roundtrip times were equal. While this is not realistic, we will show that RQM performs well for a wide range of roundtrip times. Besides, incorporating roundtrip times in RQM is possible but cumbersome and adds unnecessary load on routers.

3.1 Translation to an algorithm

Although RQM is a rate-based algorithm, we cannot require that the routers monitor the rates of sessions continuously, since that would place excessive computational load on them. Instead, we use the buffer occupancies at the epochs at which packets enter the router queue and approximate the average bandwidth as the difference of the buffer occupancies divided by the time between the epochs. In order to remove short-term fluctuations in the rates, we adopt a exponential-moving average smoothing algorithm similar to that used in RED. Thus if the $(i-1)^{th}$ packet arrived at time t_{i-1} and the i^{th} packet arrived at time t_i, and q_{i-1}, q_i were the queue sizes before the arrival of the $(i-1)^{th}$ and i^{th} packet respectively, then the average bandwidth at time t_i is estimated using the expression $\frac{q_i - q_{i-1}}{t_i - t_{i-1}}$. There is one issue that needs to be handled: the function $p(b_t)$ may evaluate to more than 1. In that case, we drop $p(b_t)$ packets on average. The algorithm is described in Figure 1.

RQM()
1 Compute current bandwidth $b_t = \frac{q_i - q_{i-1}}{t_i - t_{i-1}}$
2 Compute smoothed average of bandwidth:
3 $\hat{b}_t = a\hat{b}_{t-1} + (1-a)b_t$
4 Compute drop rate using $p(b_t)$
5 Drop or mark $p(b_t)$ packets on average

Figure 1: Algorithm RQM

3.2 Advantages of RQM

In addition to all the advantages and disadvantages of rate-based algorithms, RQM has the following advantages.

1. It requires no per-flow state.
2. It allows the designer to control the tradeoff between link utilization and the accuracy of inference of number of sessions.
3. RQM provably goes to equilibrium.

3.3 Expected performance in different scenarios:

RQM outperforms RED: RQM can be argued to be better in the following scenario: buffers very large; rates fluctuate for very short periods, just enough to drive the buffers high.

RED outperforms RQM: RQM can be argued to be worse than RED in the following scenario: buffers very small; rates fluctuate by minuscule amounts. RQM would not drop packets, but buffer overflow would.

4 Analysis of the performance of RQM

We present a fluid-based analysis of the performance of RQM. The fluid assumptions allow us to regard a session to be infinitely divisible. To simplify the analysis, we assume that during the period of time during which the algorithm is converging to steady-state, the set $\mathcal{J}$ of active sessions remains fixed. The bandwidths b_{i,t_0} of these sessions, at the beginning of this period, however, can be arbitrary. The fluid assumptions also allow us to assume that the frequency of each job reducing its sending rate is effectively equal to the drop rate. Although our analysis allows $\tilde{b}(n)$ to be any monotonically increasing function,our proofs use the specific $\tilde{b}(n)$ defined before.

Theorem 1 *Independent of the initial bandwidths b_{i,t_0} of the n sessions, these bandwidths converge in time $\mathcal{O}(\frac{B}{\alpha n}(\ln(n)+q))$ to be within a factor of $(1+e^{-q})$ of the desired values $b_{i,t} = \frac{\tilde{b}(n)}{n}$ and the estimate $\tilde{n}(b_t)$ converges within a factor of $(1+e^{-q})$ of n.*

Proof of Theorem 1: There are two dynamics that cause this convergence to happen. We will separate them by considering them in separate stages.

The first dynamic is that as with TCP given in [4, 9], the session's bandwidths converge to being equal. Lemma 3 proves that after $\mathcal{O}(\frac{B}{\alpha n}(\ln(n)+q))$ time, each session's bandwidth $b_{i,t}$ is at most a factor of $(1+e^{-q})$ away from the fair allocation $\frac{b_t}{n}$. Note that at this point we do not know the value of b_t. This completes the first stage.

The second dynamic is that when the individual bandwidths are unbalanced, the total bandwidth b_t decreases slowly. However, Lemma 4 proves that when the bandwidths are as close to being equal as they are after the first stage, the approximate number of sessions $\tilde{n}(b_t)$ and the total bandwidth b_t converge to being within a factor of $(1+e^{-q})$ of their desired levels n and $\tilde{b}(n)$ within time $\mathcal{O}(\frac{qB}{\alpha n})$. ■

The first step is to determine how the individual bandwidths $b_{i,t}$ change. We will use $\tilde{n}$ as a short form for $\tilde{n}(b_t)$.

Lemma 1 $\frac{\partial b_{i,t}}{\partial t} = -\alpha(\frac{\tilde{n}}{b_t})^2[b_{i,t}^2 - (\frac{b_t}{\tilde{n}})^2]$.

This shows that session i's bandwidth $b_{i,t}$ moves continuously towards $\frac{b_t}{\tilde{n}}$, which corresponds to RQM's estimate of fair allocations; i.e. $b_{i,t}$ increases when it is smaller than $\frac{b_t}{\tilde{n}}$ and decreases when it is larger.

Proof of Lemma 1: Consider session i. It increases its bandwidth at an additive rate of α. It sends $b_{i,t}$ packets per time unit and each gets dropped with probability $p(b_t)$, hence the frequency of adjustments is $p(b_t)\cdot b_{i,t}$. Each such adjustment decreases its bandwidth by $(1-\beta)b_{i,t}$. In conclusion, the expected rate of change of session i's bandwidth $b_{i,t}$ is as follows.

$$\begin{aligned}\frac{\partial b_{i,t}}{\partial t} &= \alpha - p(b_t) b_{i,t} (1-\beta) b_{i,t} \\ &= \alpha - \left[\frac{\alpha}{(1-\beta)}\left(\frac{\tilde{n}}{b_t}\right)^2\right] b_{i,t}(1-\beta) b_{i,t} \\ &= -\alpha(\frac{\tilde{n}}{b_t})^2[b_{i,t}^2 - (\frac{b_t}{\tilde{n}})^2].\end{aligned}$$

■

Since sessions that have more bandwidth are likely to have packets dropped more often (i.e. with probability $\frac{b_{i,t}}{b_t}$), and they release more bandwidth when they reduce their rate, the individual bandwidths change, not linearly, but quadratically. This quadratic rate of change is similar to the differential equation $\frac{\partial y}{\partial t} = -c\cdot(y^2 - Y^2)$, for some constant Y. Despite the fact that here the "Y" keeps changing, Lemma 2 does provide the intuition to how quickly the bandwidths $b_{i,t}$ converge to $\frac{b_t}{\tilde{n}}$.

Lemma 2 *If $\frac{\partial y}{\partial t} = -c\cdot(y^2-Y^2)$, then independent of its initial value, y converges to within a factor $1+e^{-q}$ of Y within time $\mathcal{O}(\frac{q}{cY})$.*

We will now be more formal. We will use the function $\mathcal{M}_t = \frac{n(\sum_i b_{i,t}^2)}{(\sum_i b_{i,t})^2}$ to measure how far the individual bandwidths are from being equal. This is the reciprocal of that used in [4] for the same purpose. It has a number of useful properties. $\mathcal{M}_t$ is always in the range $[1, n]$. It is n when the total bandwidth is on one job and is 1 when the bandwidths are equal. Finally, when $\mathcal{M}_t$ is at most $1+\frac{e^{-2q}}{n-1}$, each job's bandwidth $b_{i,t}$ is at most a factor of $(1+e^{-q})$ away from the balanced level $\frac{b_t}{n}$.

Lemma 3 *Independent of the initial bandwidths b_{i,t_0} of the n jobs, after $\mathcal{O}(\frac{B}{\alpha n}(\ln(n)+q))$ time, each job's bandwidth $b_{i,t}$ is at most a factor of $(1+e^{-q})$ away from the balanced level $\frac{b_t}{n}$. More over, the balance measure $\mathcal{M}_t$, defined below, is at most $1+e^{-q}$.*

Proof of Lemma 3: The main task of the proof is that independent of the current state of the system, $\mathcal{M}_t$ decreases at a rate of $-\frac{\partial \mathcal{M}_t}{\partial t} \geq \frac{2\alpha n}{b_t}(\mathcal{M}_t - 1)$. This change causes the value of $\mathcal{M}_t - 1$ to decrease by a factor of e in at most time $\frac{b_t}{2\alpha n} \leq \frac{B}{2\alpha n}$. (We are assuming that the total bandwidth b_t never exceeds the bottleneck's capacity B.) Because initially $\mathcal{M}_t$ is at most n, $\mathcal{M}_t - 1$ becomes at most $\frac{e^{-2q}}{n-1}$ in at most $\mathcal{O}(\log(n)+q)$ such half lives. The result follows.

The change in $\mathcal{M}_t$ is computed as follows.

$$\begin{aligned}
\mathcal{M}_t &= \frac{n\left(\sum_i b_{i,t}^2\right)}{\left(\sum_i b_{i,t}\right)^2} = \frac{n\left(\sum_i b_{i,t}^2\right)}{(b_t)^2} \\
\frac{\partial \mathcal{M}_t}{\partial t} &= \frac{n}{b_t^4}\left[\left(\sum_i 2b_{i,t}\frac{\partial b_{i,t}}{\partial t}\right)(b_t^2)\right. \\
&\quad \left. - \left(\sum_i b_{i,t}^2\right)\left(2b_t\frac{\partial b_t}{\partial t}\right)\right] \\
&= \frac{2n}{b_t^4}\left[\sum_i b_{i,t}\left(\alpha\left[1-\left(\frac{b_{i,t}\tilde{n}}{b_t}\right)^2\right]\right)b_t^2\right. \\
&\quad \left. - \left(\sum_i b_{i,t}^2\right)b_t\left(\sum_i \alpha\left[1-\left(\frac{b_{i,t}\tilde{n}}{b_t}\right)^2\right]\right)\right] \\
&= \frac{2\alpha n}{b_t^4}\left[b_t^3 - \tilde{n}^2\sum_i b_{i,t}^3 - \left(\sum_i b_{i,t}^2\right)b_t n\right. \\
&\quad \left. + \left(\sum_i b_{i,t}^2\right)\frac{\tilde{n}^2}{b_t}\left(\sum_i b_{i,t}^2\right)\right]
\end{aligned}$$

At this point, it is useful to observe that $\sum_i b_i^3 \geq \frac{1}{\sum_i b_i}(\sum_i b_i^2)^2$ for any values $b_i \geq 0$. The intuition is similar to that for the standard fact that $\sum_i b_i^2 \geq \frac{1}{n}(\sum_i b_i)^2$. It is more significant to cube the individual large values before summing than only squaring them. It is interesting, however, that equality is achieved both when the values are either completely equal or completely unbalanced. The maximum difference occurs when there are two distinct values. The proof has not been included. Using it, our the second and the third terms in the above expression cancel.

$$\begin{aligned}
-\frac{\partial \mathcal{M}_t}{\partial t} &\geq \frac{2\alpha n}{b_t^4}\left[-b_t^3 + \left(\sum_i b_{i,t}^2\right)b_t n\right] \\
&= \frac{2\alpha n}{b_t}\left[-1 + \frac{n\left(\sum_i b_{i,t}^2\right)}{b_t^2}\right] \\
&= \frac{2\alpha n}{b_t}\left(\mathcal{M}_t - 1\right)
\end{aligned}$$

■

We will now show that while the individual bandwidths are unbalanced, the total bandwidth b_t and the approximation $\tilde{n}(b_t)$ both decreases slowly. However, when the bandwidths are close to each other, they converge quickly to the desired levels $\tilde{b}(n)$ and n.

Lemma 4 *When $\mathcal{M}_t \leq 1 + e^{-q}$ (i.e., the bandwidths of all sessions are nearly equal), the approximate number of jobs $\tilde{n}(b_t)$ and the total bandwidth b_t converge to being within a factor of $(1+e^{-q})$ of their desired levels n and $\tilde{b}(n)$ within time $\mathcal{O}(\frac{qB}{\alpha n})$.*

Proof of Lemma 4: By Lemma 1, the rate of change of b_t and $\tilde{n}(b_t)$ are

$$\begin{aligned}
\frac{\partial b_t}{\partial t} &= \sum_i \frac{\partial b_{i,t}}{\partial t} \\
&= \sum_i -\alpha\left(\frac{\tilde{n}}{b_t}\right)^2\left[b_{i,t}^2 - \left(\frac{b_t}{\tilde{n}}\right)^2\right] \\
&= -\alpha\left[\frac{n\left(\sum_i b_{i,t}^2\right)}{b_t^2}\frac{\tilde{n}^2}{n} - n\right] \\
&= -\alpha\left[\mathcal{M}_t\frac{\tilde{n}^2}{n} - n\right] \\
\frac{\partial \tilde{n}}{\partial t} &= \frac{\partial \tilde{n}}{\partial b_t}\frac{\partial b_t}{\partial t} \\
&= -\frac{\partial \tilde{n}}{\partial b_t}\alpha\left[\mathcal{M}_t\frac{\tilde{n}^2}{n} - n\right] \\
&= -\frac{\partial \tilde{n}}{\partial b_t}\frac{\alpha\mathcal{M}_t}{n}\left[\tilde{n}^2 - \left(\frac{n}{\sqrt{\mathcal{M}_t}}\right)^2\right].
\end{aligned}$$

This has the form $\frac{\partial y}{\partial t} = -c\cdot(y^2 - Y^2)$ where $y = \tilde{n}$, $Y = \frac{n}{\sqrt{\mathcal{M}_t}}$, and $c = \frac{\partial \tilde{n}}{\partial b_t}\frac{\alpha\mathcal{M}_t}{n}$. (We will treat Y as being constant because $\mathcal{M}_t$ remains between 1 and $1+e^{-q}$.) Hence, according to Lemma 2, $\tilde{n}$ converges

to $(\frac{n}{\sqrt{\mathcal{M}_t}})^2$. When the individual bandwidths $b_{i,t}$ are unbalanced, the measure $\mathcal{M}_t$ is large, and hence both b_t and $\tilde{n}(b_t)$ decrease slowly than. However, when $\mathcal{M}_t \leq 1 + e^{-q}$ then $\tilde{n}$ converges to within a factor $1+\frac{1}{2}e^{-q}$ of $\frac{n}{\sqrt{\mathcal{M}_t}} = (1+\frac{1}{2}e^{-q})n$, i.e. within a factor $1+e^{-q}$ of n, within time $\mathcal{O}(\frac{q}{cY}) = \mathcal{O}(\frac{q}{\frac{\partial \tilde{n}}{\partial b_t}\alpha}) = \mathcal{O}(\frac{q}{\alpha}\frac{\partial \tilde{n}}{\partial b_t})$.

Since RQM approximates the number of sessions using $\tilde{n}(b_t) = \tilde{b}^{-1}(b_t)$ when the bandwidth is b_t, so as n increases from zero to some value, $\tilde{b}(n)$ can't increase by more than the capacity B of the buffer and in fact squeezes closer and closer to B. Hence, it is reasonable to assume that $\frac{\partial \tilde{b}(\tilde{n})}{\partial \tilde{n}}$ is relatively small and is at most $\frac{B}{n}$. This gives the stated converging time of $\mathcal{O}(\frac{qB}{\alpha n})$.

Finally, the difference, $|b_t - \tilde{b}(n)|$, between the actual and the desired total bandwidth will be at most $e^{-q}\tilde{b}(n)$, because the difference, $|\tilde{n}(b_t) - n|$, between the bottleneck's approximation and the actual of the number of active jobs is at most $(1/\frac{\partial \tilde{b}(n)}{\partial n}) \cdot e^{-q}\tilde{b}(n) \geq (\frac{n}{\tilde{b}(n)}) \cdot e^{-q}\tilde{b}(n) = e^{-q}n$. ■

5 Empirical Studies

In this section, we confirm our intuitive expectation about situations in which RED performs better than RQM and those in which RQM outperforms RED.

Next, we investigate the performance of RQM extensively. We use three performance metrics used commonly in the literature: the goodput, expected delay of TCP packets and the number of TCP packets dropped. The goodput is defined as usual, i.e., as the number of packets that reach their destinations. The delay is the total time that elapses between the first transmission of a packet and its reaching its destination. The number of packets dropped counts all drops of TCP packets, and multiple drops of a packet are counted separately.

5.1 Simulation of RQM

We evaluate the performance of our protocols using the well-known ns-2 simulator [1]. We explored the performance of our algorithms for a network with a single bottleneck, as shown in Figure 2. The bottleneck capacity (in Mb/s) B and the propagation delay (in milliseconds) d are varied in our experiments. Also varied is the buffer size q at the bottleneck link.

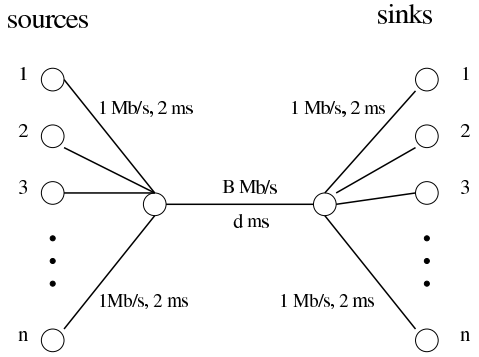

Figure 2: The network used in our experiments

It is worth noting that instead of transferring files of a given length, ns supports TCP sessions that start and terminate at specified times. Thus the delay in transferring a file cannot be measured in this simulator; instead, we measure the number of (unique) packets that reach their destinations and the average delays of these packets. We also measure the number of packets lost under each protocol.

In order to introduce fluctuations in the buffers, we used 2 constant bit rate (CBR) UDP sessions throughout our simulations. Each UDP session generated data at 1 Mb/s.

5.2 Parameter settings

We investigated the effect of parameters c, γ and a in RQM. We omit detailed results due to lack of space. We chose the best values of each in the following simulations. These were $c = 1.0, \gamma = 0.01, a = 0.5$.

5.3 Scenario where RQM outperforms RED

As pointed out in Section 3.2, we expect RQM to outperform buffer-based algorithms like RED when the bottleneck has a very large buffer, and rates fluctuate for very short periods, just enough to drive the

buffers high. This intuition was verified in the following experiment. We used a buffer of size 10000, buffer capacity $B = 10$ Mb/s, delay $d = 40$ ms, and used 10 TCP sessions and 5 UDP sessions. Each UDP session injects packets at 1Mb/s. The network was simulated for 200 timesteps. The results are given in Table 1.

	Goodput	delay	num drops
RQM	216160	0.1353	131
RED	213619	0.1082	1455

Table 1: RQM outperforms RED

5.4 Scenario where RED outperforms RQM

In Section 3.2, we argued that RED would outperform RQM when the bottleneck buffer is very small, and rates fluctuate by small amounts. This intuition was verified in the following experiment. We used a buffer of size 10, buffer capacity $B = 10$ Mb/s, delay $d = 40$ ms, and used 10 TCP sessions and 5 UDP sessions. Each UDP session injects packets at 1Mb/s. The network was simulated for 200 timesteps. The results are given in Table 2.

	Goodput	delay	num drops
RQM	207894	0.1102	1399
RED	209796	0.1104	1367

Table 2: RED outperforms RQM

5.5 Performance of RQM

We investigate thoroughly the performance of RQM under different settings of the buffer size, bandwidth and propagation delay of the bottleneck link. In our experiments, we used two settings each for the bandwidth and propagation delay and three settings for the buffer size. The bandwidth values used were 10 Mb/s (high) and 1.5 Mb/s (low). The former setting would leave about 8Mb/s for the TCP sessions whereas the latter would not even have enough capacity for the UDP sessions. We used three buffer sizes, viz., 100, 1000, 10000, to correspond to low, medium and high buffer sizes. We compared RQM to RED, REM and DropTail. DropTail refers to queues which do not run any AQM algorithm and drop packets when the buffer overflows.

5.5.1 A homogeneous network

Our first set of experiments were done with a homogeneous network, where all sessions use paths with identical capacities, buffer sizes and propagation delays. The results are displayed in Figures 3 through 10. For lack of space we do not include results for the number of packets dropped in this paper.

Goodput results: We present only the results for buffer size 1000. These are shown in Figures 3 through 6. The graphs show that when the number of sessions exceeds five, RQM outperforms RED and REM. The only exception is the scenario when there is little or no congestion and the bottleneck delay is high. However, RQM does not perform as well as REM and RED when the number of sessions is low.

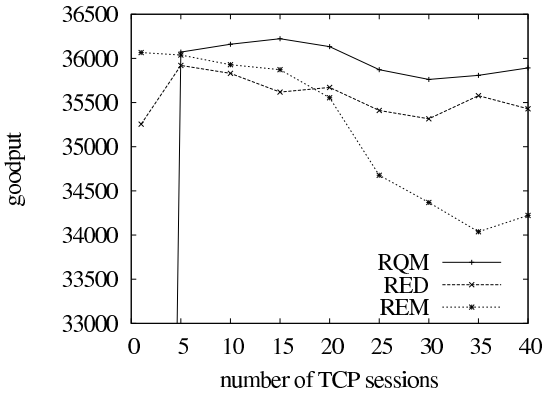

Figure 3: Low bandwidth, low delay, medium buffer.

Delay results: The delay results are shown in Figures 7 through 10. The graphs show RQM outperforms RED and REM whenever the number of sessions exceed 5 for low bandwidth and delay. For high bottleneck propagation delay, it outperforms REM and RED when the number of sessions is 25 or more.

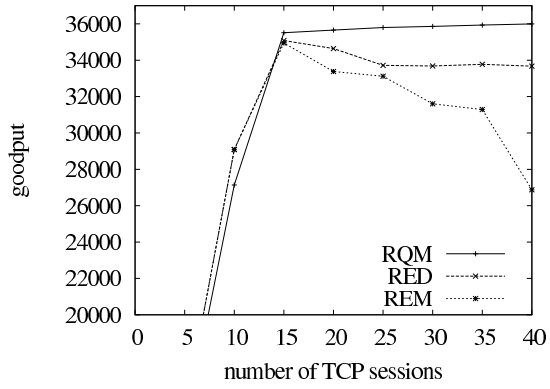

Figure 4: Low bandwidth, high delay, medium buffer.

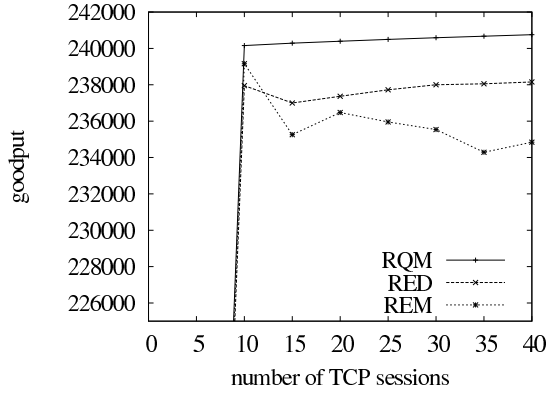

Figure 5: High bandwidth, low delay, medium buffer.

5.5.2 Heterogeneous networks

Since we designed RQM using intuition about homogeneous networks, we need to check whether it performs well on heterogeneous networks. In order to test the performance of our algorithm on heterogeneous networks, we used the same topology but changed the propagation delays of the links incident on the sources as shown in Figure 11 so that the i^{th} link has delay $2i$ ms.

Goodput results: As shown in Figures 12 through 19, RQM performs better than RED and REM when the number of sessions are high. The only exception is the case when the bottleneck has

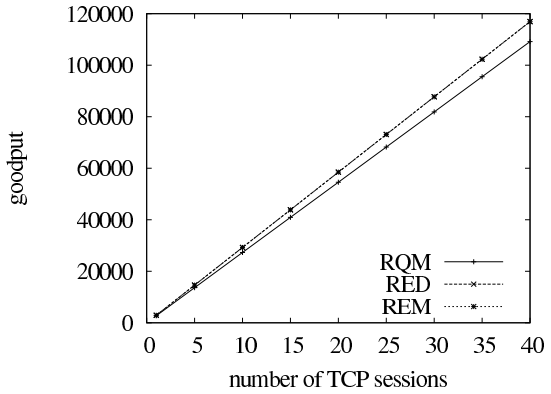

Figure 6: High bandwidth, high delay, medium buffer.

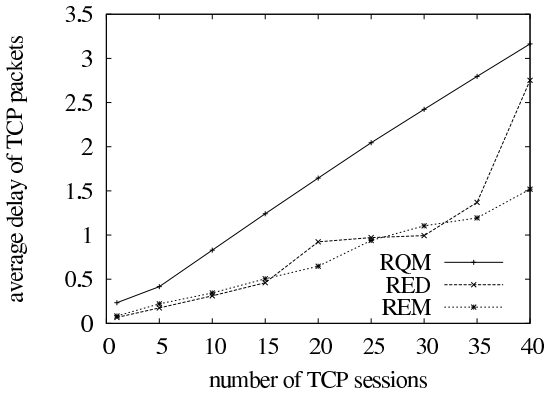

Figure 7: Low bandwidth, low delay, medium buffer.

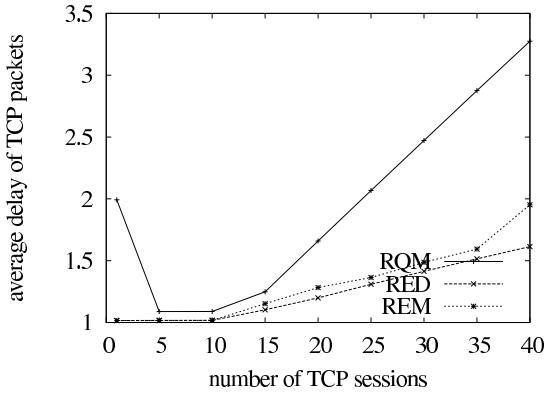

Figure 8: Low bandwidth, high delay, medium buffer.

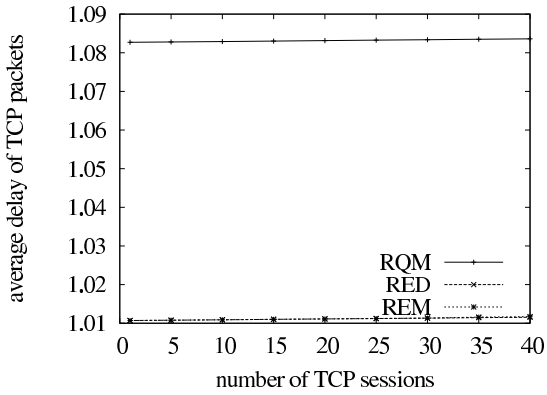

Figure 10: High bandwidth, high delay, medium buffer.

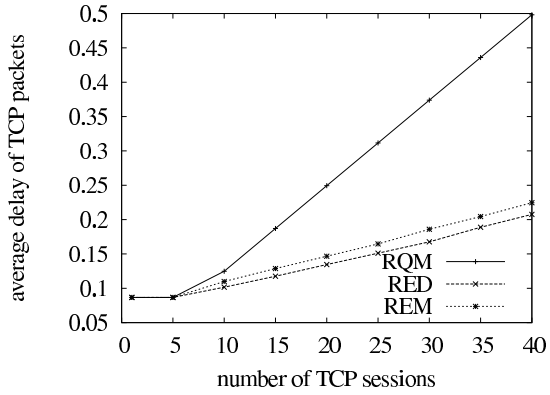

Figure 9: High bandwidth, low delay, medium buffer.

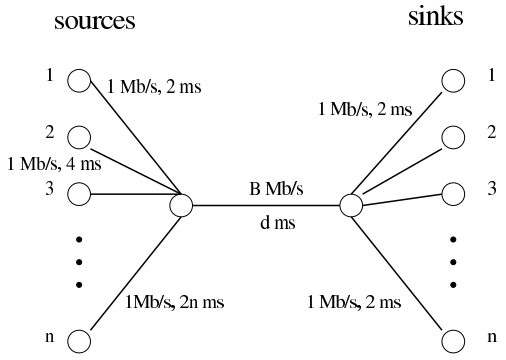

Figure 11: The network used in our experiments

high bandwidth and high propagation delay. RQM far outperforms its competitors when the bottleneck delay is low.

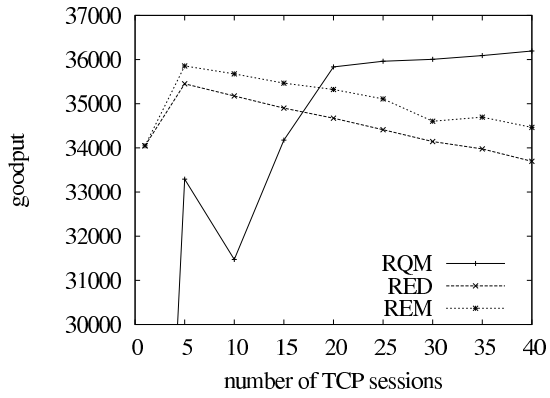

Figure 12: Low bandwidth, low delay heterogeneous network.

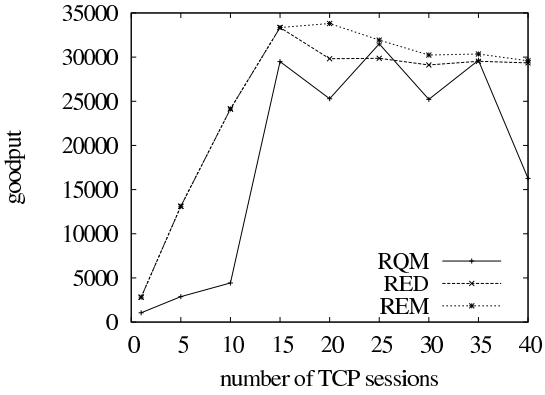

Figure 13: Low bandwidth, high delay heterogeneous network.

Delay results: Figures 16 through 19 compare the expected delay of TCP packets for the three algorithms. In this case RQM has a delay slightly higher than it competitors when the bottleneck bandwidth is high. However, RQM has a significantly higher delay when the bottleneck bandwidth is low.

6 Discussion

In this paper, we have proposed a new AQM algorithm, RQM, and evaluated its performance using both analytical and experimental studies. RQM is outperformed by RED and REM when the number of sessions is low. This can be attributed to the nature of our utilization curve. Interestingly, RQM has a higher goodput than its competitors when the number of sessions is high. We are currently investigating ways to combine RQM with existing algorithms to produce algorithms that always outperform RED and REM. We are also investigating the performance of AQM algorithms for networks with multiple bottlenecks.

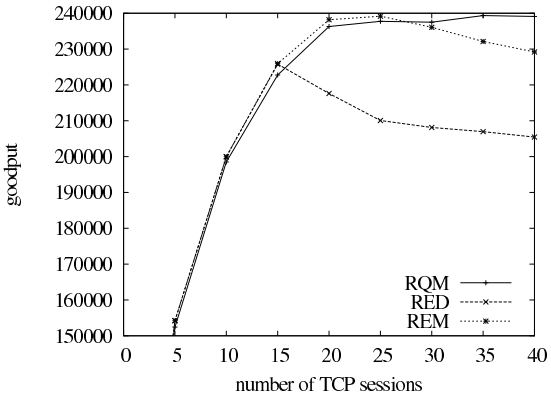

Figure 14: High bandwidth, low delay heterogeneous network.

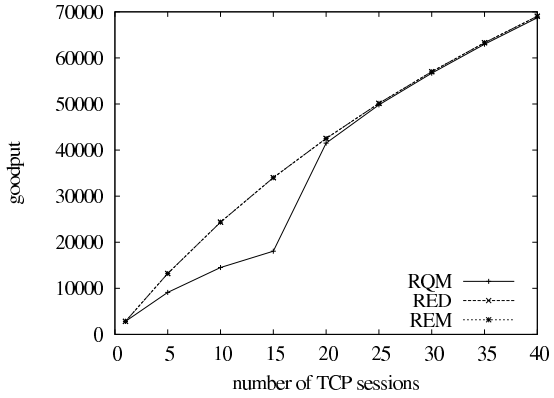

Figure 15: High bandwidth, high delay heterogeneous network.

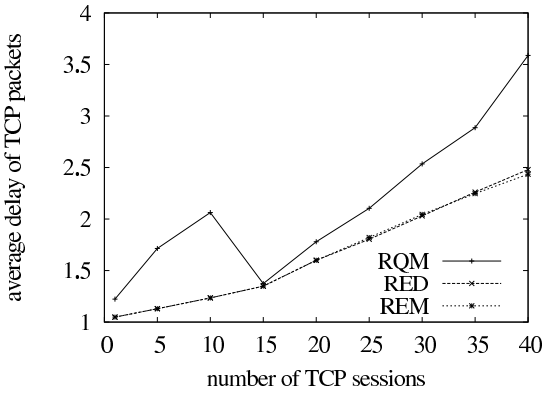

Figure 17: Low bandwidth, high delay heterogeneous network.

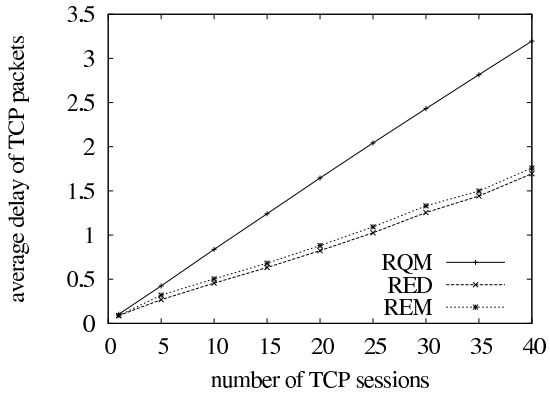

Figure 16: Low bandwidth, low delay.

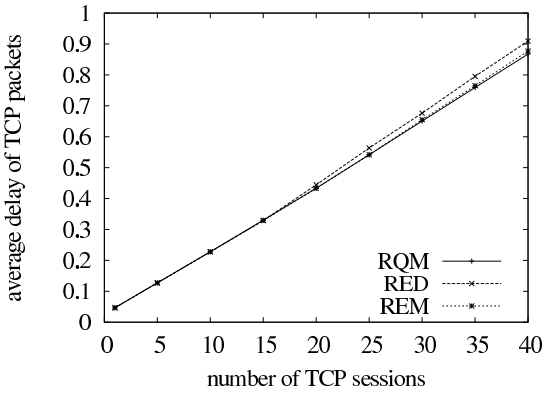

Figure 18: High bandwidth, low delay heterogeneous network.

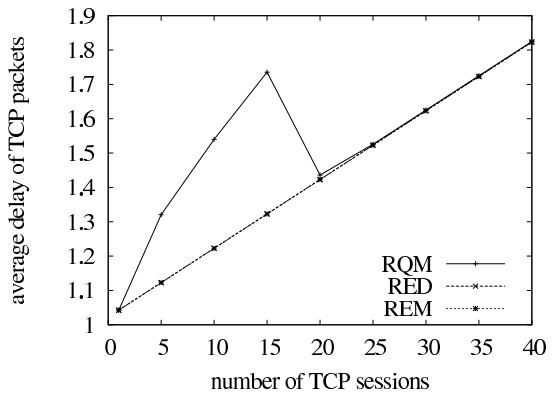

Figure 19: High bandwidth, high delay heterogeneous network.

References

[1] The network simulator: ns-2. Available at http://www.isi.edu/nsnam/ns/.

[2] F. M. Anjum and L. Tassiulas. Balanced-RED: An algorithm to achieve fairness in the internet. In *Proceedings of Infocom*, March 1999.

[3] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring RED gateway. In *IEEE INFOCOM*, 1999.

[4] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer networks and ISDN systems*, 17(1):1–14, 1989.

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: An alternative approach to active queue management. In *Proceedings of NOSSDAV 2001*, June 2001.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin. The blue queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[7] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, 2001.

[8] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[9] P. W. D. Jeff Edmonds, Suprakash Datta. TCP is competitive against a limited adversary. In *SPAA*, pages 174–183, 2003.

[10] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management. In *Proceedings of SIGCOMM 2001*, San Diego, California, USA, 8 2001.

[11] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of SIGCOMM 97*, 1997.

[12] S. Liu, T. Basar, and R. Srikant. Controlling the internet: A survey and some new results. In *IEEE Conference on Decision and Control*, 2003.

[13] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle. Dynamics of tcp/red and a scalable control. In *Proceedings of IEEE INFOCOM '02*, 2002.

[14] T. J. Ott, T. V. Lakshman, and L. H. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOM '99*, pages 1346–1355, March 1999.

[15] S. A. L. N. Reddy. LRU-RED: An active queue management scheme to contain high bandwidth flows at congested routers. In *IEEE Globecom*, pages 2311–2315, 2001.

[16] S. H. L. S. Athuraliya, V. H. Li and Q. Yin. REM: Active queue management. *IEEE Network*, May 2001.