A Note on Parameter Values of REM with Reno-like Algorithms

Sanjeewa Athuraliya

Networking Laboratory, Caltech

sanjeewa@caltech.edu

March 2002

Abstract

We explain the choice of parameter values for the REM implementation in `ns-2.1b9` network simulator, and how it can be simplified when the TCP algorithm is Reno-like, i.e., employs additive-increase-multiplicative-decrease strategy.

1 Algorithm

The design rationale of REM (Random Exponential Marking) is explained in [1]. The REM algorithm at each link (queue) l in each update period t is given by:

$$p_l(t+1) = \max\{0,\ p_l(t) + \gamma(\alpha_l(b_l(t) - b_l^*) + x_l(t) - c_l\} \quad (1)$$

$$\mathbf{prob}(t+1) = 1 - \phi^{-p_l(t+1)} \quad (2)$$

The variables are (see below for their units):

- $b_l(t)$: queue length in update period t,
- b_l^*: target queue length,
- $x_l(t)$: amount of traffic that arrives at the link in update period t,
- c_l: link capacity.

The parameter values that are recommended here assume that these variables are measured in packets. Hence, when the simulation uses packets of equal size, $x_l(t)$ and c_l are the number of packets that arrive in period t and the number of packets that can be sent in each period, respectively. Similarly, $b_l(t)$ and b_l^* are the number of packets at the queue in period t and the target number, respectively.

When packets of different sizes are used in the simulation, then these variables must be measured in bytes and then converted into the number packets of mean size. To do this, `byte` mode has to be selected in `ns-2`, i.e. set the REM parameter `bytemode` to `true`. It remains to decide the mean packet size. In the current REM implementation, the mean packet size must be selected offline. A better approach would be to calculate this online using the last N packets.

In (2), $\mathbf{prob}(t)$ is the dropping/marking probability in period t. When packets are of different sizes, $\mathbf{prob}(t)$ should be modified accordingly. In the current REM implementation, we simply scale the probability by packet size/mean packet size, as the current RED implementation in `ns-2` does. This is not ideal but it is simple to implement. It has the nice property that ACKs, which are small packets, are rarely dropped/marked in the backward path. It also means that the algorithm is biased against larger packet sizes. A more desirable approach is to think of a large packet as a collection of packets of the mean size and then calculate the probability that at least one of them will be dropped/marked. In this case, the probably that a large packet is dropped/marked is

$$\mathbf{PROB}(t+1) \quad = \quad 1 - (1 - \text{prob}(t+1))^{(\text{packet size/mean packet size})}$$

This however might be too complex to implement in a real router.

In the following, we will call the variable $p_l(t)$ 'price'.

2 Parameter values

Values for the following parameters need to be selected: update interval, γ, ϕ, α_l and b_l^*.

In general, the smaller the update interval, the better. It means that the feedback signal arriving at the hosts is more up-to-date. On the other hand a small update interval means more processing overhead. A value between 1ms and 10ms seems to work well. We do not recommend performing update on each packet arrival.

The parameter γ determines the speed of convergence of the algorithm. A larger value of γ gives a faster convergence, but it also incurs a higher risk of oscillatory queue. When choosing γ, consideration should be given to the dynamics at the router. If the congestion level at the router is changing constantly, a larger value of γ should be used for the algorithm to track the changes. A value between 0.01 and 0.001 is recommended.

The parameter ϕ determines the range of loss or marking probability, which also depends on the range of price $p_l(t)$. Ideally, ϕ should be chosen so that the end-to-end probability observed at hosts fluctuates around 0.5. However, the AIMD algorithm of TCP Reno and its variants (such as NewReno and SACK) forces the probability to be small, typically around 0.1. We recommend a value of 1.001 for ϕ for these Reno-like algorithms.

The value of α_l determines the prominence given to the queue length when determining the level of congestion. It trades off utilization and queue length in transient, with a smaller α_l producing a

higher utilization. A value of 0.1 is recommended.

A main objective of REM is to maintain low queues at routers. This is controlled by the value of b_l^*. A value of zero for b_l^* will clear the queue in equilibrium. But the oscillatory nature of Reno-like algorithms reduces the utilization in this case. To overcome this, b_l^* should be set to a small non-zero value, and we recommend $b_l^* = 20$ packets. The average queue length will have a value close to b* in equilibrium.

We **caution** that, as for RED, the parameter values, together with network parameters such as capacities, number of sources and their delays, determine stability. Hence given a set of network parameters, REM parameters can be tuned to optimize performance. However, this requires the knowledge of network parameters as well as dynamic adaptation of REM parameters, and hence may not be practical. The REM parameters recommended here are based on our simulation experiences and are not optimized for any fixed set of network parameters.

3 Simplified REM with Reno-like algorithms

When the source algorithms are Reno-like, we could further simplify the REM algorithm by removing exponentiation.

Empirical and theoretical studies suggest that the drooping/marking probability almost always is restricted to a region between 0 and 0.15. With the value of ϕ recommended above, the mapping from link price $p_l(t)$ to the probability is almost linear with a gradient of approximately 0.005 in this region of probability. Hence the update rule (1–2) can be combined into:

$$\mathbf{prob}(t+1) = \min\{0.15,\ \max\{0,\ \mathbf{prob}(t) + 0.005\gamma(\alpha_l(b_l(t) - b_l^*) + x_l(t) - c_l\}\} \tag{3}$$

It should be emphasized that this simplification is done only when Reno-like algorithms are used which constrain the probability to a small range.

A further simplification is possible if a nonzero queue target b_l^* is used. In this case, we can approximate $x_l(t) - c_l$ by $b_l(t+1) - b_l(t)$, and the update (3) becomes:

$$\mathbf{prob}(t+1) = \min\{0.15,\ \max\{0,\ \mathbf{prob}(t) + 0.005\gamma(b_l(t+1) - (\alpha_l b_l^* + (1-\alpha_l)b_l(t)))\}\}$$

This version involves only queue lengths and not rates or capacity. It is equivalent to the PI controller developed in [2].

We have retained the original version of the REM algorithm in ns-2 to provide more flexibility in choosing queue target b_l^* and experimenting with other TCP algorithms that allow a wider range of probability.

We close with a remark on non-rate-adaptive users. For Reno-like TCP algorithms, when the probability exceeds 0.15, it is usually either due to ill provisioning or non-rate-adaptive users. A

common example of the former case is that the equilibrium window size is very small, say, less than 3 packets or even fractional. In the latter case, it is important that the rate-adaptive users are not starved. We are designing a mechanism that augments the current REM algorithm to protect rate-adaptive users. In the absence of such a mechanism, it is necessary to constrain the probability to be less than some predetermined value. This will ensure that the rate-adaptive users are not starved.

References

[1] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: active queue management. *IEEE Network*, 15(3):48–53, May/June 2001. `http://netlab.caltech.edu`.

[2] Chris Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE Infocom*, April 2001. `http://www-net.cs.umass.edu/papers/papers.html`.