On Reducing the Degree of Second-Order Scaling in Network Traffic

B. Sikdar, K. Chandrayana, K. S. Vastola and S. Kalyanaraman
Dept. of ECSE, Rensselaer Polytechnic Institute
Troy, NY 12180 USA

Abstract— **While it is well known that second order scaling in network traffic can leads to larger queueing delays, higher drop rates and extended periods of congestion, reducing the scaling exponents has remained an open problem. In this paper we evaluate some techniques to reduce the degree of scaling in TCP traffic, specifically by reducing two related causes: (1) timeouts and exponential backoffs (2) burstiness and ACK compression. We propose a simple modification to the RED algorithm, and show that it can lead to significant reductions in both multi and mono fractal properties of TCP traffic as compared to the currently implemented active and passive buffer management policies. We then evaluate TCP pacing and show that it too can reduce the multi and mono fractal scaling of traffic. We also show that though our techniques are aimed at small time-scale TCP related causes of scaling, it is also effective in reducing the degree of self-similarity in traffic even when application and user level causes are also present, as long as TCP is used as the underlying transport protocol.**

I. Introduction

The presence of scaling in the second-order properties of network traffic has been established in a variety of network environments and its causes have been traced to various factors. These scaling phenomena can be divided into three categories: (1) self-similarity or scaling as time scales go to infinity [2], [12], [25], (2) multi-fractality or scaling as time scales go to zero [5], [6] and (3) pseudo self-similarity or scaling over a limited range of timescales [10], [22]. The causes behind each of these phenomena have also been investigated [2], [25], [5], [10], [18], [22], [23], [24]. Also, it is well known that the scaling, burstiness and long-range dependence associated self-similar and multi-fractal nature of traffic can lead to a number of undesirable effects like high buffer overflow rates, large delays and persistent periods of congestion [3], [4], [19]. The severity of these conditions is dependent on various conditions like the relevant time-scales of the system [15], [9] and system utilization [3]. However, given other factors are constant, the overflow rates are proportional to the degree of self-similarity or the Hurst parameter [16].

Our focus in this paper is to study techniques which reduce the degree of second order scaling in network traffic. The time-scales over which we focus our attentions corresponds from few milli-seconds to 100s of seconds and thus corresponds to the multi-fractal and pseudo self-similar behavior. One of the main contributors to the scaling behavior in these time scales are protocol (TCP) related causes [18], [5], [10], [22], [23], [24] and these are the causes that we target.

We investigate two different techniques in this paper which reduces the degree of second-order scaling in network traffic. The underlying idea of the first technique is to reduce the incidence of timeouts and exponential backoff in TCP flows which can cause scaling on finite time-scales as shown in [10], [22]. We do this by looking at existing and also proposing a new buffer management policy to reduce correlated losses (and consequently timeouts) in TCP flows. The second technique is to eliminate the inherent burstiness and back-to-back packet transmissions in TCP flows. One of the reasons behind this behavior is the phenomenon of ACK compression in TCP flows [26]. ACK compression has also been suggested as a possible cause of the fractal nature of network traffic in [6]. We look at the impact of reducing the effects of ACK compression and burstiness of TCP sources through TCP pacing [26] on the second-order scaling of traffic.

Our results show that both these techniques are very successful in reducing the scaling exponents and burstiness in traffic. While these techniques are aimed at TCP's contribution to traffic scaling, we also show that these techniques are also effective when other factors like heavy-tailed file sizes and human causes like think times are also present, as long as TCP is used as the underlying transport protocol. Also, while we mainly concentrate on the multi-fractal and fixed time-scale properties, our simulations show that the large time scale properties are also affected by these schemes resulting in lower degrees of self-similarity or the Hurst parameter.

The rest of the paper is organized as follows. In Section II we give a brief overview of basic concepts. Section III investigates the impact of three buffer management schemes on traffic scaling: taildrop, RED and a proposed modified RED algorithm. In Section IV we concentrate on the impact of reducing the burstiness of TCP flows. Finally, Section V presents the discussions and concluding remarks.

II. Discrete Wavelet Transform and Multi-scaling

We first present a brief description of the basic concepts associated with traffic scaling and differentiate between the various types of scaling behavior studied in literature which is based partly on [3], [5]. Consider an arrival process $A(0,t)$ which counts the cumulative traffic arrival in the time interval $(0,t)$ and its associated increment process $X_\Delta(i)$ defines as

$$X_\Delta(i) = A(0, i\Delta) - A(0, (i-1)\Delta) \tag{1}$$

The basic hypothesis associated with scaling of process is that the moments of the increment process behave as

$$\sum_i X_\Delta(i)^q \sim C(q)\Delta^{-\tau(q)} \quad \text{as } \Delta \to 0 \tag{2}$$

While in theory, this behavior should hold over all time scales for the scaling hypothesis to be satisfied, in practice, the hypothesis can be said to be reasonable if the behavior is satisfied over

Supported in parts by DARPA contract F30602-00-2-0537, DoD MURI contract F49620-97-1-0382 and NSF grant ANI 9806660.

a range of timescales. The function $\tau(q)$ is called the *structure* function and for mono-fractal or self-similar processes, $\tau(q)$ is linear in q. For multi-fractal processes, $\tau(q)$ is non-linear in q and for both the processes, in general, is decreasing in q.

The discrete wavelet transform represents a one dimensional signal $X(t)$ in terms of shifted and dilated versions of a band-pass wavelet function $\psi(t)$ and shifted versions of a low pass scaling function $\phi(t)$. For the choices of $\psi(t)$ and $\phi(t)$ which allow us to form an orthonormal basis, the signal can be represented as

$$\begin{aligned} X(t) &= \sum_k \langle X(t)\phi_{0,k}(t)\rangle\, \phi_{0,k}(t) + \\ &\quad \sum_{j=0}^{\infty}\sum_k \langle X(t)\psi_{j,k}(t)\rangle\, \psi_{j,k}(t) \end{aligned} \tag{3}$$

where $\phi_{j,k}(t) = 2^{-j/2}\phi(2^{-j}t - k)$ and $\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k)$ are the shifted and dilated version of the scaling and wavelet function respectively. The quantity $d_{j,k} = \langle X\psi_{j,k}\rangle$ is referred to as the *wavelet coefficient* at scale j and time $2^j k$. Also, $\mid d_{j,k} \mid^2$ measures the energy in the signal at time $2^j k$ and about the frequency $2^{-j}\lambda_0$ where the reference frequency λ_0 depends on the wavelet ψ. If a process has scaling in some second order statistic, then it will very often have scaling for all moments. The *partition functions* defined as

$$S_q(j) = \frac{1}{n_j}\sum_k \mid d_{j,k} \mid^q \sim C(q) j^{\alpha(q)} \tag{4}$$

and specifically the nature of the function $\alpha(q)$ can then be used to determine the nature of the scaling phenomena. For self-similar processes $\alpha(q)$ is linear and given by $\alpha(q) = Hq + q/2$. On the other hand, if $\alpha(q)$ non-linear, we say that the process shows multi-scaling. It is common in multi-fractal theory to define the exponents slightly differently: $\zeta(q) = \alpha(q) - q/2$ which results in $\zeta(q) = Hq$ for self-similar processes. In this paper, we plot $\zeta(q)$ to study the scaling behavior of network traffic.

III. Buffer Management Policies and Multi-Scaling

In this section we look at the effect of different of buffer management policies on the scaling properties of traffic passing through it. We consider both passive and active queue management algorithms as represented by taildrop and RED queues respectively. We also propose a change to the current RED algorithm which results in lower degrees of second order scaling.

A. Taildrop Queues

Taildrop queues are currently the most widely implemented queueing mechanisms in routers in the Internet [17]. The first-in-first-out (FIFO) policy of taildrop queues, coupled with the bursty nature of TCP traffic implies that the packet drops from a taildrop queue become correlated and multiple packets can get dropped from the same window. To model the effect of taildrop queues on the probability of timeouts in TCP flows, we use the correlated loss model used in [17] and [20]. In this model, a packet in a window is lost independently of losses in other rounds. However, losses within a window are correlated and all packets following the first packet to be lost in window are also assumed to be lost. As noted in [17] and [20], this model is quite realistic for taildrop queues with TCP traffic given the bursty nature of TCP sources with back to back packet transmission. With correlated losses, the probability that an arbitrary packet loss in a TCP flow with $cwnd = w$ and a loss rate of p leads to a timeout is given by [17], [20]

$$Q(w) = \begin{cases} 1 & \text{for } 1 \le w \le 3 \\ 1 - \frac{p(1-p)^{2w-1}}{1-(1-p)^w} & \text{for } 4 \le w \le 8 \\ 1 - \frac{p(2-p)(1-p)^{2w-2}}{1-(1-p)^w} & \text{for } 9 \le w \le W_{max} \end{cases} \tag{5}$$

where W_{max} is the maximum allowable window size.

B. RED Queues

RED is an active queue management algorithm which randomly drops packets before a queue becomes full, so that end nodes can respond to congestion before buffers overflow and was proposed in [8]. RED probabilistically drops packets even before the queue is full based on a weighted average of the queue length k and two threshold values: min_{th} and max_{th}. For each packet arrival at the queue, the drop probability for the packet $d(k)$ is given by

$$d(k) = \begin{cases} 0 & \text{for } k < min_{th} \\ \frac{k - minth}{max_{th} - min_{th}} max_p & \text{for } min_{th} < k < max_{th} \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

where max_p is a control variable denoting the maximum drop probability. The reader is referred to [8] for the detailed RED algorithm.

For a RED queue the packet drop pattern is closely modeled by an independent loss model as noted in [21] and the references therein. In the independent loss model, losses in a window are assumed to be independent of each other and losses in other rounds. From [11] and [21], with the independent loss model the probability that an arbitrary packet drop leads to a timeout in a TCP flow with a window of w packets experiencing a loss rate of p is given by

$$Q(w) = \begin{cases} 1 & \text{for } 1 \le w \le 3 \\ 1 - \frac{wp(1-p)^w}{1-(1-p)^w} & \text{for } 4 \le w \le 8 \\ 1 - \frac{wp(1-p)^w}{1-(1-p)^w} & \text{for } 9 \le w \le W_{max} \\ - \frac{w(w-1)p^2(1-p)^{w+n-2}}{1-(1-p)^w} & \end{cases} \tag{7}$$

In Figure 1 we plot the probability that a loss in a TCP flow with a congestion window of 5 leads to a timeout in the case of correlated and independent losses. We see that with correlated losses, the probability of timeouts are much higher for the same loss rates. This leads to the intuition that the scaling and burstiness of traffic passing through RED queues will be much lesser than that through taildrop queues. We verify this intuition through simulations later in this section.

C. Modified RED Queues

The independent loss model for the packet drop in a RED queue is very accurate for the cases when the average queue

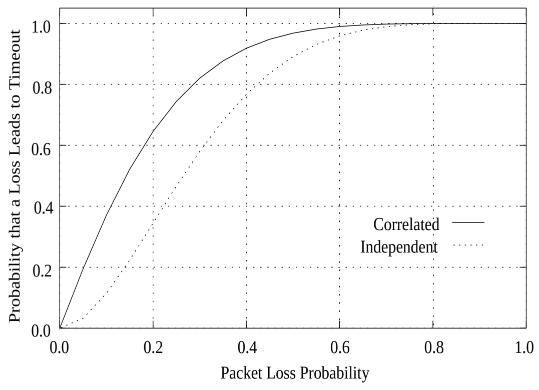


Fig. 1. Comparison of timeout probabilities for TCP flows with a window of 5 for correlated and independent loss models.

Algorithm 1 Modified Dropping algorithm of RED

```
last_drop_flag ⇐ 0
for Each Packet Arrival do
  if last_drop_flag = 1 then
    last_drop_flag = 0;
    goto enqueue;
  else if $min_{th} < avg < max_{th}$ then
    with probability $d(k)$, drop the packet
    if packet is dropped then
      last_drop_flag = 1;
    end if
  else if $max_{th} < avg$ then
    Drop the packet;
    last_drop_flag = 1;
  else
    goto enqueue;
  end if
end for
```

length stays between max_{th} and min_{th}. However, if the offered load is so high that the average queue length becomes close to max_{th}, RED fails to perform better than taildrop queues [13]. This is due to the fact that when the average queue length becomes greater than max_{th}, RED drops each packet with probability 1. This leads to multiple packet drop from the same window, resulting in timeouts.

To deal with this situation, we propose a small change to RED's dropping policy and the new algorithm for packet dropping is shown in Algorithm 1. The idea is *not* to drop any two consecutive packets which arrive at the queue, unless of course if the queue is full. Since TCP generally sends back to back packets, ensuring that no two consecutive packets are dropped will reduce the probability that multiple packets from the same window are dropped, thereby reducing the occurrence of timeouts.

To calculate the probability that any arbitrary packet arriving when the average queue size is k is dropped, $d'(k)$, we first refer to Figure 2. The three states, denoted by $\{i, j\}$ with $i, j \in 0, 1$ and $i = j \neq 1$, represent the possible conditions the queue can be in depending on whether the current or the previous packet was dropped or not. The global balance equations for the transition probabilities from the three states can be written as

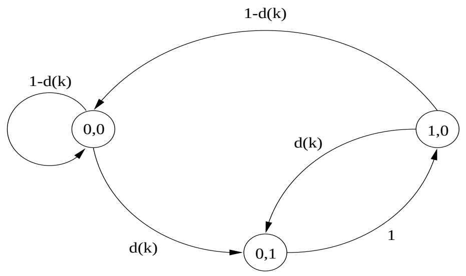

$$\begin{tikzcd}
(0,0) \arrow[loop left, "1-d(k)"] \arrow[dr, bend right, "d(k)"'] & & (1,0) \arrow[ll, bend right, "1-d(k)"'] \arrow[dl, bend right, "d(k)"'] \\
& (0,1) \arrow[ur, bend right, "1"'] &
\end{tikzcd}$$

Fig. 2. Packet drop probabilities with the modified RED algorithm

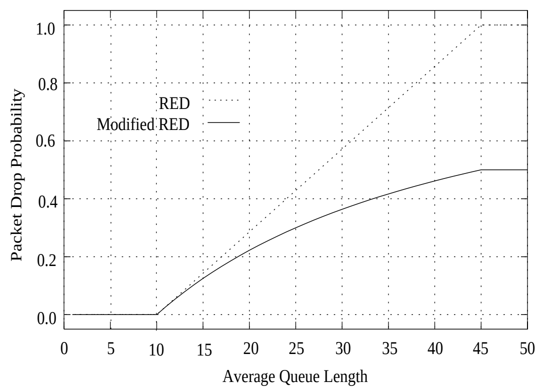


Fig. 3. Comparison of packet drop probabilities in the RED and modified RED buffer management policies.

$$\begin{aligned} P_{0,0}d(k) &= P_{1,0}(1-d(k)) \\ P_{0,1} &= P_{0,0}d(k) + P_{1,0}d(k) \\ P_{1,0} &= P_{0,1} \end{aligned} \qquad (8)$$

where $P_{i,j}$ denotes the steady state probability of being in state i, j and $d(k)$ is defined in Equation (7). These steady-state are then given by

$$P_{0,0} = \frac{1-d(k)}{1+d(k)} \quad P_{0,1} = \frac{d(k)}{1+d(k)} \quad P_{1,0} = \frac{d(k)}{1+d(k)}$$

The probability that any arbitrary arriving packet is dropped when the average queue length is k, $0 \leq k < qlen$, is thus

$$d'(k) = P_{0,0}d(k) + P_{1,0}d(k) = \frac{d(k)}{1+d(k)} \qquad (9)$$

We note that if max_p is small as is suggested in literature on RED parameter configuring, then this modification does not significantly affect the drop rates while the average queue length is less than max_{th}. However, when the average queue length exceed max_{th} but is less than $qlen$, the packet drop probability becomes 0.5 as compared to 1 in RED (Figure 3). This is a sufficiently high drop rate to force TCP sources to reduce their transmission rates but without inducing a lot of timeouts. Also, since the packet drop probabilities for the modified RED queue are smaller that those for a RED queue with the same parameters for a given value of the average queue length, the corresponding probability of timeouts in the modified RED queue would also be smaller.

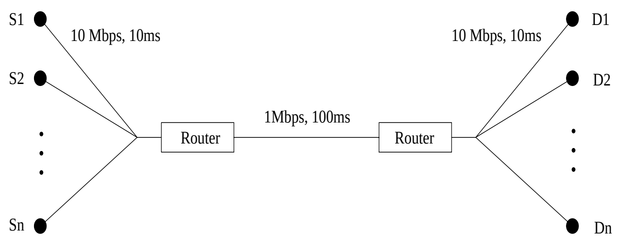


Fig. 4. Topology of the network for the validation tests.

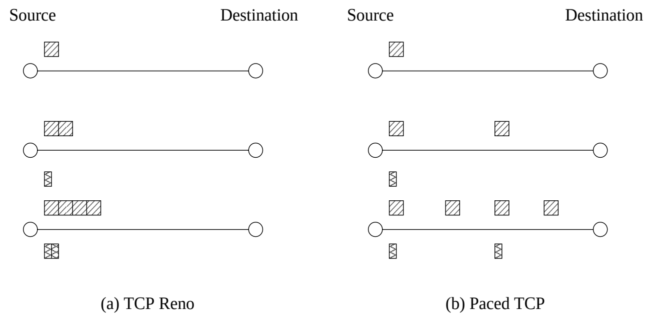


Fig. 7. The packet sending pattern of Reno and paced TCP during slow-start

D. Results

We now compare the scaling exponents of the traffic with the three buffer management policies. These results were generated by simulations using the simulator *ns*. The topology used for the simulations is shown in Figure 4. While the topology is very simplistic, it was chosen since it allows us to isolate the queueing policy's contribution to the scaling components. The simulations for each queueing policy is again broken in two parts: the presence or absence of web traffic. The web traffic is introduced as background traffic for more realistic wide area networking scenarios and to test the effectiveness of the buffer management policies against non-TCP related causes of scaling like file sizes and human factors.

In the simulations with absence of web traffic, all the sources correspond to very long TCP Reno sources which are active for the entire duration of the simulation. In the simulations with web traffic, a subset of the sources and destinations act as web traffic clients and servers. The web traffic was generated using the specifications defined in [6]. For the simulations, the buffer size of the taildrop as well as the RED and modified RED queues was kept at 100 packets. For the RED and modified RED queues, the other parameters were $min_{th} = 30$, $max_{th} = 90$, $max_p = 0.1$ and $w_q = 0.002$. All the simulations were conducted for a "simulated" time of 3600.0 seconds. For the scaling plots, we collected and analyzed the arrival statistics corresponding to the aggregate traffic arriving at the bottleneck link. For the throughput results, we present the statistics corresponding to each of the long flows in the simulation.

In Figure 5 we compare the scaling exponents $\zeta(q)$ for simulations without web traffic corresponding to 30 and 50 TCP flows. The results for other flow sizes (40 and 60) are similar. We note that, as expected, taildrop queues show a much higher variation and a non-linear pattern in the scaling exponents suggesting multi-fractal behavior. In contrast, the linear nature of $\zeta(q)$ for the RED and modified RED queues suggest less burstiness and a behavior consistent with self-similarity. This can be attributed to the reduction of timeouts and exponential backoffs in RED and modified RED queues.

The results for simulations with web traffic are shown in Figure 6. In this case, we note that the scaling exponents for all the three queues behave linearly in q suggesting self-similar behavior. The self-similar behavior in this case is due to the heavy tails introduced by the web traffic through the file size distribution and response times. For this case, we also plot the Hurst parameters for each case in Table I. We note that for lower loads the Hurst parameters corresponding to the taildrop queue are higher than both the RED and the modified RED algorithms. As the load on the network increases, the Hurst parameter of the traffic in the RED queue becomes more than the other two scheduling disciplines. This is due to the higher proportion of consecutive losses in RED queues under high loads [13]. However, we note that for all cases, the modified RED algorithm performs better than the others or equals the best performance.

In Table I we also compare the throughputs of the long TCP flows in the simulation scenarios. The improvement in the throughput with RED queues over taildrop queues is around 1-3% for the cases without web traffic. In the presence of web traffic this improvement increases to between 5-11%. In the absence of web traffic, the throughput of the RED and modified RED are almost identical. However in the presence of web traffic, the throughput of the modified RED generally increases and the increase is around 5%.

IV. Reducing Source-Level Burstiness of TCP Flows

TCP traffic is inherently bursty in nature and TCP sources tend to send back to back packets. One of the key reasons behind this behavior is ACK compression as described in [26]. The self-clocking mechanism of TCP depends on the arrival of ACKs at the same spacing with which they were generated by the receiver. However, in the presence of two-way traffic, queueing on the reverse path can alter this spacing and the ACKs arrive closer together than they were sent. The first immediate consequence of this is that the sender becomes bursty and sends more back to back packets. Additionally, with the ACKs being received quicker than they were sent, the sender might be misled into sending more data than the network can accept [14]. This in turn contributes to the losses and timeouts experienced by the TCP flow. It was also conjectured in [6] that the phenomenon of ACK compression might be responsible for the fractal behavior of network traffic. Thus, the prospect of reducing scaling in network traffic by undoing the effects ACK compression on TCP dynamics is very promising and worthy of further exploration.

One of the most widely reported mechanisms for smoothing out TCP traffic is through evenly spacing or "pacing" a window of packets over the round-trip time and was first proposed in [26]. Since then, pacing has been proposed for cases where the ACK clocking is lost to avoid slow starts at the beginning of connections, after losses or at the resumption of idle connection. For more details on these and other environments where pacing is used, we refer the reader to [1] and the references therein. Pacing is accomplished at the sender (receiver) if instead of trans-

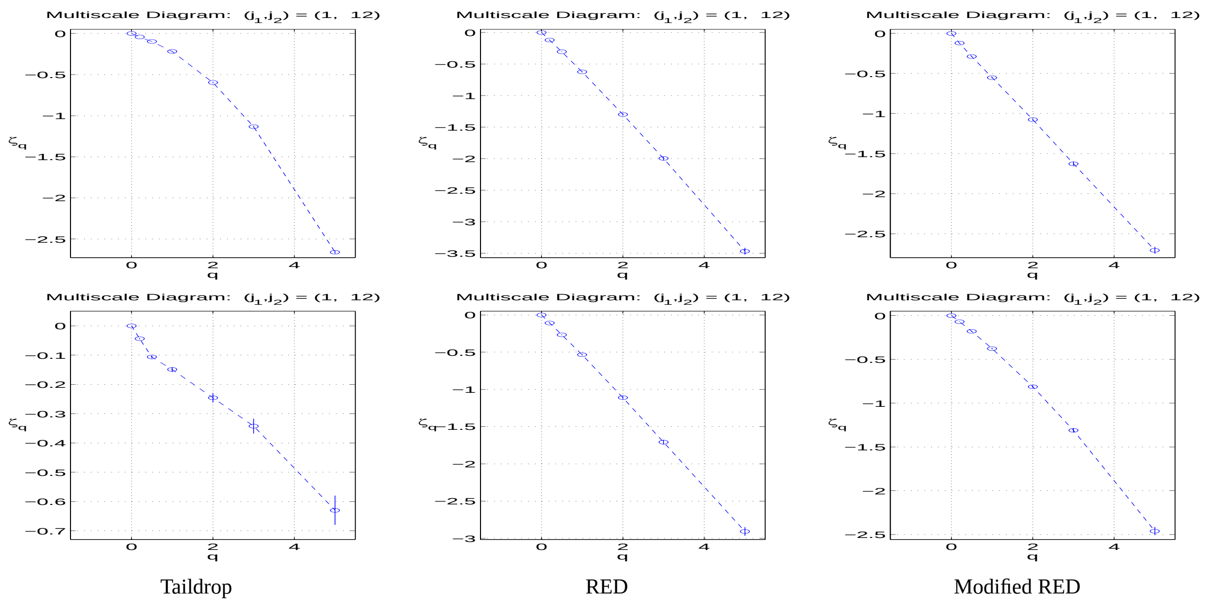


Fig. 5. Scaling behavior without web traffic: 30 flows (top) and 50 flows (bottom).

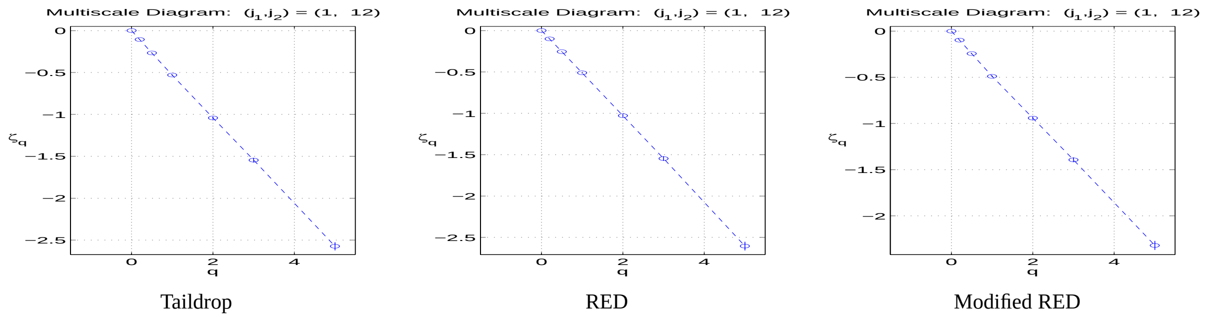


Fig. 6. Scaling behavior with web traffic: 30 TCP flows and 10 web sessions.

Queues with web traffic

Configuration	Taildrop		RED		Modified RED	
	Throughput	H	Throughput	H	Throughput	H
10 Long, 5 web	63988.44	0.58	72982.00	0.50	63055.77	0.50
15 Long, 5 web	42099.70	0.67	44785.19	0.60	47422.37	0.57
10 Long, 10 web	33005.78	0.74	37165.11	0.76	34517.33	0.75
15 Long, 10 web	22318.52	0.77	23449.48	0.84	25204.88	0.77
20 Long, 10 web	17508.89	0.80	19952.00	0.91	21275.88	0.80

TABLE I

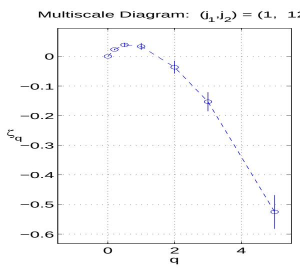
THROUGHPUT (IN BITS/SEC) AND HURST PARAMETERS FOR THE THREE BUFFER MANAGEMENT POLICIES.

mitting a packet (ACK) everytime an ACK (packet) is received, it is delayed to maintain the proper spacing between two successive packets (ACKs). The delay between two successive packets is given by

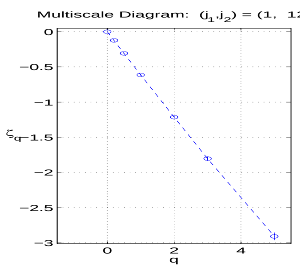
$$delay = \frac{RTT}{cwnd} \tag{10}$$

where $cwnd$ is the current value of the congestion window. The concept of TCP pacing and its difference from conventional TCP versions is illustrated in Figure 7. The figure shows the $cwnd$ increase and the packet transmission patterns of TCP Reno and TCP pacing.

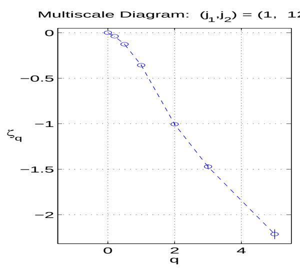
In Figures 8 and 9 we plot the behavior of the scaling exponent of paced TCP and TCP Reno for taildrop and RED queues. We again note that while TCP Reno shows multi-fractal behavior, specially with taildrop queues, while multi-fractal scaling is absent for paced TCP for both taildrop and RED queues. In both the cases, pacing the TCP packets instead of sending them in bursts leads to significant reductions in the traffic burstiness and consequently the scaling. We also note that though we present the results only for TCP Reno, comparisons with other versions of TCP yield similar results since other versions also send back to back packets whenever possible.



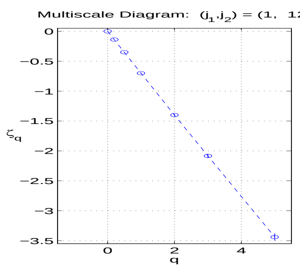
60 Flows, Reno



60 Flows, Paced

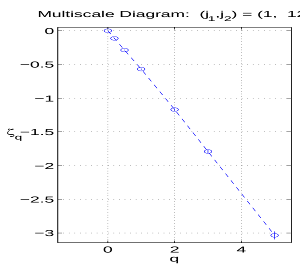


70 Flows, Reno

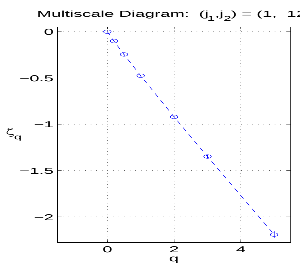


70 Flows, Paced

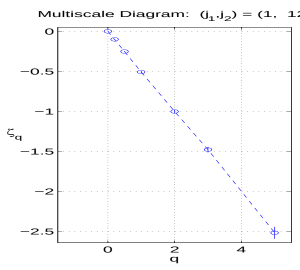
Fig. 8. Comparison of scaling behavior without web traffic for Reno and Paced TCP: Taildrop queue.



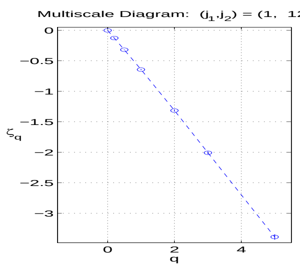
40 Flows, Reno



40 Flows, Paced

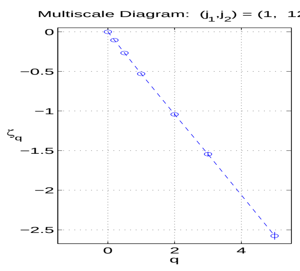


80 Flows, Reno

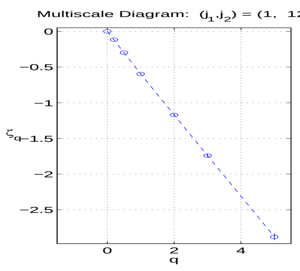


80 Flows, Paced

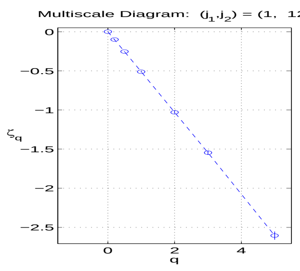
Fig. 9. Comparison of scaling behavior without web traffic for Reno and Paced TCP: RED queue.



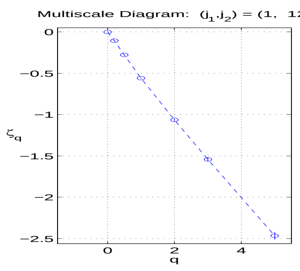
Reno, Taildrop



Paced, Taildrop



Reno, RED



Paced, RED

Fig. 10. Comparison of scaling behavior with web traffic (20 long flows and 10 web session) for Reno and Paced TCP.

Queues with web traffic

Configuration	Taildrop		RED	
	Reno	Paced	Reno	Paced
10 Long, 5 web	0.58	0.50	0.50	0.50
15 Long, 5 web	0.67	0.50	0.60	0.50
10 Long, 10 web	0.74	0.50	0.76	0.50
15 Long, 10 web	0.77	0.50	0.84	0.50
20 Long, 10 web	0.80	0.54	0.91	0.58

TABLE II
HURST PARAMETERS FOR RENO AND PACED TCP WITH WEB TRAFFIC FOR TAILDROP AND RED QUEUES.

In Figure 10 we compare the scaling exponents for paced and TCP Reno in the presence of web traffic. While we show only the case with 20 long TCP flows and 10 web sessions, the results for other cases (enumerated in Table II) are similar. We note that the behavior of $\zeta(q)$ is linear in q suggesting the presence of self-similar properties. In Table II we compare the Hurst parameters for the simulations with web traffic. We note that pacing is very successful at reducing the degree of self-similarity even with the presence of session and user level causes. Also, we note that pacing leads to larger reductions in H as compared to the modified RED algorithm. This leads us to believe (without rigorous proof) that the inherent burstiness in TCP flows is a greater contributor to traffic self-similarity than timeouts and exponential backoffs. This makes pacing more successful at reducing traffic self-similarity over a large number of scenarios as compared to the modified RED algorithm.

V. SUMMARY

In this paper we explored some methods for reducing the degree of second order scaling of TCP traffic. The methods were based on two of the causes which contribute to the self-similarity and multi-fractal nature of network and in particular TCP traffic: timeouts and the burstiness of TCP traffic. While we considered only the causes of multi and mono fractality from the TCP point of view, our solutions are also effective against other causes of self-similarity like session interarrival times and heavy tailed distributions in the file sizes, introduced by web traffic.

We first investigated the impact of various buffer management policies on the second order scaling of TCP traffic. Our results show that while taildrop queues lead to a multi-fractal behavior in traffic, RED and modified RED queues result in self-similar traffic. Also, in the presence of web traffic, the traffic has monofractal characteristics due to the heavy tails associated with web server file sizes and inter-file separation times. For these cases, RED and modified RED algorithms have lower degrees of self-similarity than taildrop queues and the modified RED queue consistently gives the lowest degree of self-similarity for all these scenarios. This is due to the reduction of timeouts and exponential backoffs in RED and modified RED queues which has been shown to be a contributor to the scaling of network traffic.

Another factor contributing to the multi-fractal scaling of network traffic is the inherent burstiness of TCP traffic which can be reduced by paced TCP. Our simulations show that pacing in TCP eliminated the multi-fractal scaling of TCP traffic under both taildrop and RED queues. Also, in the presence of web traffic, paced TCP results in significant reductions in the Hurst parameter for both RED and taildrop queues at the bottleneck when compared to other versions of TCP and is in fact more successful at it than the modified RED algorithm.

REFERENCES

[1] A. Aggarwal, S. Savage, and T. Anderson, “Understanding the performance of TCP pacing,” *Proceedings of IEEE INFOCOM,* pp. 1157-1165, Tel-Aviv, Israel, March 2000.

[2] M. Crovella and A. Bestavros, “Self-similarity in World Wide Web traffic: Evidence and possible causes,” *IEEE/ACM Trans. on Networking,* vol. 5, no. 6, pp. 835-846, Dec 1997.

[3] A. Erramilli, O. Narayan, A. Neidhardt and I. Sainee, “Performance impacts of multi-scaling in wide area TCP/IP traffic,” *Proceedings of IEEE INFOCOM*, pp. 352-259, Tel Aviv, Israel, March 2000.

[4] A. Erramilli, O. Narayan and W. Willinger, “Experimental queueing analysis with long-range dependent packet traffic,” *IEEE/ACM Transactions on Networking,* vol. 4, no. 2, pp. 209-223, April 1996.

[5] A. Feldmann, A. C. Gilbert and W. Willinger, “Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic,” *Computer Communications Review,* vol. 28, no. 4, pp. 42-58, 1998.

[6] A. Feldmann, A. C. Gilbert, P. Huang and W. Willinger, “Dynamics of IP traffic: A study of the role of variability and the impact of control,” *Proceeding of ACM SIGCOMM,* Boston, MA, August 1999.

[7] D. R. Figueiredo, B. Liu, V. Misra and D. Towsley, “On the autocorrelation structure of TCP traffic,” Technical Report TR 00-55, University of Massachusetts, Computer Science Department, Amherst, MA, 2000.

[8] S. Floyd and V. Jacobson, “Random early detection gateways for TCP congestion avoidance,” *IEEE/ACM Transactions on Networking* vol. 1, no. 4, pp. 397-413, August 1993.

[9] M. Grossglauser and J-C. Bolot, “On the relevance of long-range dependence in network traffic,” *IEEE/ACM Transactions on Networking,* vol. 7, no. 5, pp. 629-640, October 1999.

[10] L. Guo, M. Crovella and I. Matta, “How does TCP generate pseudo-self-similarity?,” *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems,* Cincinnati, OH, August 2001.

[11] A. Kumar, “Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link,” *IEEE/ACM Trans. on Networking,* vol. 6, no. 4, pp. 485-498, August 1998.

[12] W. E. Leland, M. S. Taqqu, W. Willinger and D. V. Wilson, “On the self-similar nature of Ethernet traffic (extended version),” *IEEE/ACM Transactions on Networking,* vol. 2, no. 1, pp. 1-15, Feb 1994.

[13] M. May, T. Bonald and J.-C. Bolot, “Analytic evaluation of RED performance,” *Proceedings of IEEE INFOCOM,* pp. 1415-1424, Tel-Aviv, Israel, March 2000.

[14] J. Mogul, “Observing TCP dynamics in real networks,” *Proceedings of ACM SIGCOMM,* pp. 305-317, Baltimore, MD, August 1992.

[15] A. L. Neidhardt and J. L. Wang, “The concept of relevant time scales and its application to queueing analysis of self-similar traffic,” *Proceedings of ACM SIGMETRICS,* pp. 222-232, Madison, WI, June 1998.

[16] I. Norros, “A storage model with self-similar input,” *Queueing Systems,* vol. 16, pp. 387-396, 1994.

[17] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, “Modeling TCP Reno performance: A simple model and its empirical validation,” *IEEE/ACM Trans. on Networking,* vol. 8, no. 2, pp. 133-145, April 2000.

[18] K. Park, G. Kim, and M. Crovella, “On the relationship between file sizes, transport protocols, and self-similar network traffic,” *Proceedings of International Conference on Network Protocols,* pp. 171-180, Columbus, OH, October 1996.

[19] V. Paxson and S. Floyd, “Wide area traffic: The failure of Poisson modeling,” *IEEE/ACM Trans. on Networking,* vol. 3, no. 3, pp. 226-244, June 1995.

[20] B. Sikdar, S. Kalyanaraman and K. S. Vastola, “An integrated model for the latency and steady state throughput of TCP connections,” *Performance Evaluation,* vol. 46, no. 2-3, pp. 139-154, September 2001.

[21] B. Sikdar, S. Kalyanaraman and K. S. Vastola, “TCP Reno with random losses: Latency, throughput and sensitivity analysis,” *Proceedings of IEEE IPCCC,* pp. 188-195, Phoenix, AZ, April 2001.

[22] B. Sikdar and K. S. Vastola, “The effect of TCP on the self-similarity of network traffic,” *Proceedings of the 35th Conference on Information Sciences and Systems,* Baltimore, MD, March 2001.

[23] A. Veres and M. Boda, “The chaotic nature of TCP congestion control,” *Proceedings of IEEE INFOCOM,* pp. 1715-1723, Tel-Aviv, Israel, 2000.

[24] A. Veres, Z. Kenesi, S. Molnár and G. Vattay, “On the propagation of long-range dependence in the Internet,” *Proceedings of ACM SIGCOMM,* pp. 243-254, Stockholm, Sweden, September 2000.

[25] W. Willinger, M. S. Taqqu, R. Sherman and D. V. Wilson, “Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level,” *IEEE/ACM Trans. on Networking,* vol. 5, no. 1, pp. 71-86, 1997.

[26] L. Zhang, S. Shenker, and D. Clark, “Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic,” *Proceedings of ACM SIGCOMM,* pp. 133-147, Zurich, Switzerland, September 1991.