Stability Criteria of RED with TCP Traffic

Technical Report

Thomas Ziegler [†‡] Serge Fdida [†] Christof Brandauer[‡]

{Thomas.Ziegler, Christof.Brandauer}@salzburgresearch.at Serge.Fdida@lip6.fr

[†] Université Pierre et Marie Curie, Laboratoire Paris 6,
Rue Capitaine Scott, 8, 75015 Paris, France

[‡]Salzburg Research; University of Applied Sciences and Technologies,
Jakob Haringerstr. 5, 5020 Salzburg, Austria[*]

Document Status: Draft

Date: May 2000

Abstract

This report investigates the dynamics of RED queue management in the presence of FTP and Web-like TCP reno flows for scenarios with homogeneous and heterogeneous round trip times. We systematically derive quantitative guidelines and steady state models how to set the RED-parameters as a function of the scenario parameters bottleneck-bandwidth, round-trip-time, number of TCP flows in order to avoid severe oscillation of the queue-size. We show that properly configuring RED is feasible in case of one way bulk-data TCP traffic and knowledge of the scenario parameters, although the buffer-requirements are significant. Additionally, it is investigated in detail why RED and two-way TCP bulk-data traffic necessarily have to oscillate resulting in concerns on the deployment of RED in the TCP-dominated Internet.

* This work is partially sponsored by the EU IST project Aquila.

1 Introduction

The RED (Random Early Detection) queue management algorithm [1][2] for congestion avoidance in cooperation with end-to-end transport protocols has been widely discussed in the literature and is implemented in commercially available routers. Recent simulations [3][4][5] have shown that achieving RED's control-goal - convergence of the queue-size between *minth* and *maxth* - is sensitive to parameter setting. However, quantitative analysis giving guidelines on how to set the RED parameters dependent on the network-scenario is still in a very premature state, giving a motivation for this work. Our observation in [6] is that setting RED parameters properly in order to avoid oscillation of the average queue size is absolutely non trivial. Oscillations are harmful as they cause periods of link-underutilization when the instantaneous queue-size equals zero followed by periods of frequent "forced packet-drops" [7] when the average queue-size exceeds *maxth* or the instantaneous queue approaches the total buffer-size. Forced packet drops are in contradiction to the goal of early congestion detection and decrease the performance of ECN [8], attempting to avoid packet-loss and to provide increased throughput for low-demand flows by decoupling congestion notification from packet-dropping. In case of WRED [9] or RIO [10] oscillations may cause poor discrimination among in-profile and out-of-profile packets. When the average queue size decreases below the maximum queue-size threshold for out-of-profile packets the out-packets may enter the queue. Subsequently, the average queue-size increases again and in-packets may be dropped with high probability.

However, the primary intention of our analysis is not to investigate RED's performance decrease due to queue-size oscillations but to find the necessary and sufficient criteria to avoid these oscillations by setting the RED parameters properly. We focus on modelling RED with bulk-data TCP-Reno flows in steady state, i.e. we consider the convergence of the average queue after a phase of starting TCP flows performing their initial Slowstart. During our time-period of interest the number of TCP flows stays constant. Our arguments for investigating RED with TCP in steady state are first, the need to determine the operational bounds for any interacting set of control-mechanisms under the "simpler" steady-state assumption before investigating its behavior in transient state and second, the dominance of TCP traffic in the Internet. In section 8.2 the models for RED parameter setting are verified against realistic, Web-like traffic patterns. Contrary to other papers modelling TCP with RED ([11] [12]) our analysis is not restricted to the assumption of homogeneous round-trip-times and gives quantitative, not just qualitative guidelines how to set the RED parameters.

The problem of parametrizing RED for steady state TCP flows is definitely challenging as four inter-dependent RED parameters (*minth*, *maxth*, *maxp* and *wq*) have to be set properly as a function of three scenario parameters (bottleneck bandwidth, number of TCP flows,

round-trip-time).

Regarding the kind of convergence we may hope to achieve with RED and TCP it is important to mention that we do not mean convergence to a certain queue-size value in the mathematical sense. Contrary, we define convergence very loosely as “achieving a state of bounded oscillation of the queue-size around a value between *minth* and *maxth* so that the amplitude of the oscillation of the average queue-size is significantly smaller than *maxth-minth* and the instantaneous queue-size remains greater than zero and smaller than the total buffersize”. This state should be achieved in a few round-trip-times.

Abbreviations used throughout this paper:
- *B*: buffer-size at bottleneck in packets
- *C*: capacity of bottleneck link in Mbps
- L: bottleneck-capacity in mean packets per second
- *D*: delay of Bottleneck link in ms
- *d*: total propagation delay in ms
- d_i: delay of access link i
- *N*: number of flows
- n_i: number of flows at access link i

2 Queue Management Overview

Traditional Internet routers employ FIFO packet-scheduling at output ports, aggregating traffic in a single queue. In case of congestion "drop tail packet dropping" is performed. The principle of drop-tail is to discard all packets once the queue size has reached the total buffer-size. Although simple, drop tail has several drawbacks:

- Drop tail packet dropping exhibits a bias against bursty traffic. In case of congestion in the presence of TCP flows, the queue size converges to the buffersize as TCP increases the congestion window until drops happen. Once the queue size is at the buffer-size there is no bufferspace left to absorb bursts. Additionally, drop tail is incapable of detecting the early stages of congestion as it starts dropping packets when the buffer is already filled completely.
- Drop tail packet dropping in combination with a single FIFO queue at the output port shared by all flows causes long delay for delay sensitive interactive traffic (telnet) and real time traffic due to the large steady state queue size.
- Due to traffic phase effects drop-tail packet-drop algorithms sometimes allow a few connections to monopolize buffer space while locking out other connections causing considerable unfairness. Another non desirable phase effect of drop tail packet dropping is the tendency to synchronize multiple TCP conversations to be in phase, causing periods of heavy load followed by periods of link-underutilization. This effect is called "global synchronization" [1].

2.1 RED Gateways

The Random Early Detection (RED) algorithm [1] has been proposed in order to alleviate the problems of simple drop-tail queue management. RED monitors the average queue size (*avg*) of a shared FIFO queue at the router output port. Based on comparison of the average queue size against a minimum (min_{th}) and a maximum threshold (max_{th}) a packet

dropping probability (p_a) is computed. The RED algorithm is given in figure 1:

```
For each packet arrival:
   Calculate the average queue size
   If q > 0
      avg = (1 - w_q) avg + w_q q
   else
      avg = (1- w_q)^m avg
   If min_th <= avg < max_th
      Calculate packet dropping probability
         p_b = max_p(avg - min_th)/(max_th - min_th)
         p_a = p_b /(1 - count p_b)
      With probability p_a
         Drop the arriving packet
   else if max_th <= avg
         Drop the arriving packet.
```

Figure 1 *RED algorithm*

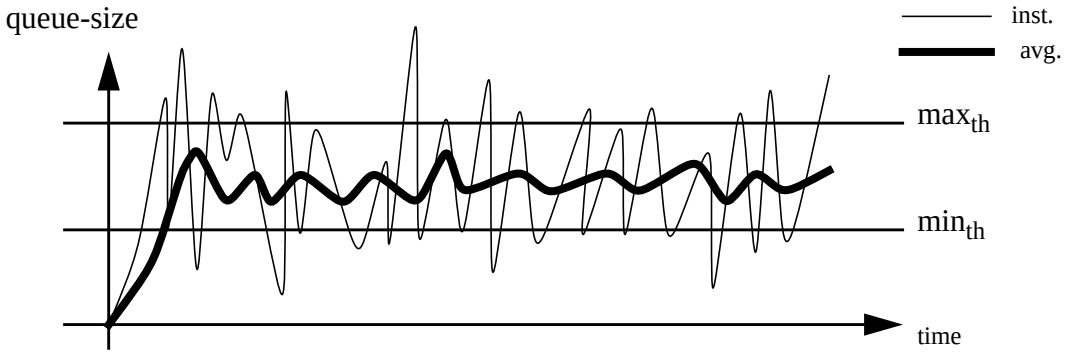


Figure 2 *Instantaneous and average queue size over time with RED*

The average queue size operates as a low-pass filter, filtering out transient congestion (and thereby accommodating traffic bursts) while tracking long-range changes in the queue size as an indicator for persistent congestion. If the queue is non empty an exponentially weighted moving average is computed, where w_q is a constant factor and q denotes the actual queue size. If the queue is empty, *avg* is computed as if m small packets had arrived with a queue size of zero. This avoids overestimation of the average queue size in case of long queue idle times.

The packet dropping probability is computed in two steps. First, p_b is derived as a linear function of the average queue size, where the constant max_p, $0 < max_p < 1$, defines an

upper bound for p_b. To avoid global synchronization, uniform distribution of packet drops in case of a constant average queue size would be desirable. This is achieved in a second step, by increasing p_a slowly as the *count* increases since the last dropped packet [2].

To account for different packet sizes, RED operating in "byte-mode" weights p_b by the ratio of the incoming packet's size to the maximum packet size.

Another operational mode of RED in collaboration with TCP data-senders and receivers is ECN (Explicit Congestion Notification) [8]. In ECN-mode RED randomly marks packets of ECN-enabled TCP flows by setting a bit in the packet header instead of dropping the packets. At reception of a marked packet, the TCP data-receiver communicates the ECN signal to the data-sender by copying the mark into the TCP-ACK. A TCP data-sender receiving an ECN-signal is expected to behave identically like a data-sender detecting that a packet has been lost (i.e. the congestion window is halved for TCP Reno).

2.2 ARED: Adaptive RED

It has been shown in [3] and [4] that the rate at which RED signals congestion indications (i.e. packet drops or ECN signals) to TCP sources has to be directly proportional to the number of TCP flows traversing the output port. RED's congestion-indication-rate can be adapted by varying the maximum-packet-drop-probability parameter (max_p). If max_p is constantly set to 1/30 as proposed in [1], RED works well in scenarios with few TCP flows. However, in case of many TCP flows RED's congestion indication-rate is too low hence the average queue-size converges to the maximum threshold (max_{th}). Once the average queue-size is at the maximum threshold RED's ability of accommodating traffic bursts and avoiding global synchronization by detecting the early stages of congestion decreases.

In order to avoid this undesirable behavior the following changes to RED have been proposed:

```
If min_th <= avg < max_th at the last packet arrival and now max_th <= avg:
     max_p = max_p / alpha
If min_th <= avg < max_th at the last packet arrival and now avg < min_th:
     max_p = max_p * beta
```

The parameters alpha and beta are set to 3 and 2, respectively. It is shown by simulation that these changes make the average queue-size converge between the minimum and maximum threshold, independently of the number of flows traversing the output port.

2.3 WRED, Weighted RED

Weighted RED [9] has been designed as a building-block for the differential-services-architecture of the Internet. WRED is executed at router-output-ports and assigns different

max_{th}, min_{th} and max_p parameters to an arriving packet marked as in-profile, respectively out-of-profile.The way WRED may provide service-differentiation among in- and out-of-profile packets is illustrated by the subsequent figure:

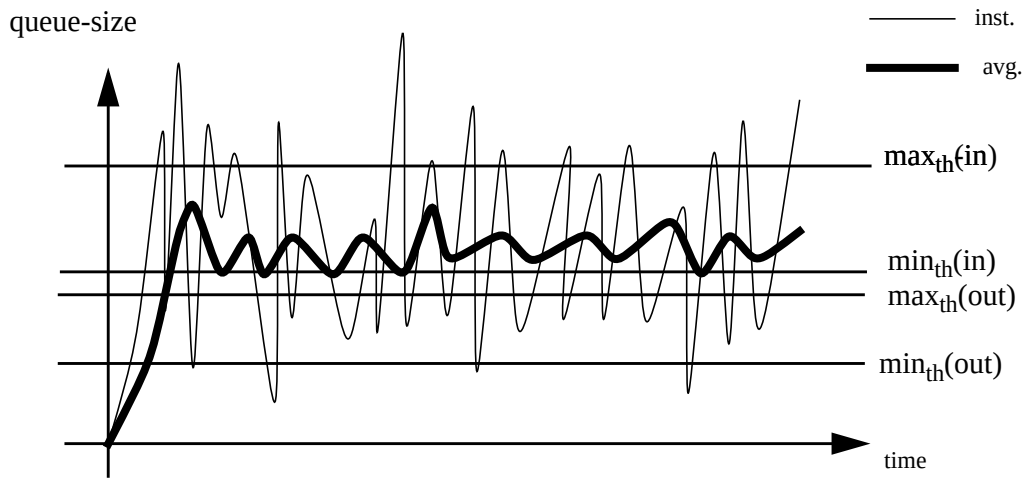


Figure 3 *Principle of WRED for service-differentiation among in and out packets.*

Out-of profile packets have lower max_{th}, min_{th} and higher max_p parameters, hence they are dropped earlier and with higher probability than in-profile packets. In steady state and in case of sufficient demand, the average queue size converges between $minth_{in}$ *and* $maxth_{in}$. Assuming $minth_{in} \geq maxth_{out}$, in-profile packets are accommodated while out-of-profile packets have a drop-probability of one if $avg > maxth_{out}$. Hence out-of-profile packets are discriminated against in-profile packets.

2.4 RIO: Red In/Out

RIO [10] slightly modifies WRED as it employs two separate average queue-sizes, one for all arriving packets and another only for in-packets. If an in-packet arrives, the drop probability is computed as a function of avg(in), max_{th}(in), min_{th}(in) and max_p(in). If an out-packet arrives the drop probability is computed as a function of avg(out), max_{th}(out), min_{th}(out) and max_p(out). It is stated in [10] that RIO should provide better service-differentiation than WRED due to the dual queue-size computation.

3 Related Research on RED Modelling and Parameter Setting

As a rule of thumb, [13] recommends to set *minth* to five packets and *maxth* to at least three times *minth*. The exact value of *minth* has to depend on the burstiness of the arrival-process, link-capacity, propagation delay and maximum buffer size. Additionally, it is remarked that the optimum setting of *minth* and *maxth* balances a trade-off between high queueing delay and poor link-utilization. [1] mentions that the difference between *minth* and *maxth* should be greater than the typical increase of the average queue-size in one round-trip-time. Without giving quantitative guidelines, the latter two statements imply that the difference between *minth* and *maxth* should be directly proportional to the bandwidth*delay product of the scenario. Based on the observation that steady state drop rates for TCP flows rarely exceed 5% in realistic scenarios, [13] recommends to set *maxp* to 0.1. Finally, it is recommended to set *wq* equal to 0.002 in order to balance a trade-off between setting *wq* too low and thereby responding too slowly to transient congestion and setting *wq* too high implying the incapability to filter out transient bursts.

In [14] measurements of TCP with RED have shown that link-utilization decreases in case of RED, small buffer-sizes and small differences between *maxth* and *minth*. This paper recommends to set the buffer-size greater or equal to the bandwidth**RTT* product of the path, which seems impossible for very high speed WANs. We will show in section 7 that, although the buffering requirements of RED are significant, setting the buffer-size equal to the bandwidth**RTT* product would be overly conservative in case of TCP flows in steady state.

As shown in [3] and [4], RED's rate to signal congestion indications to the sources (i.e. drop packets or set congestion indication bits) has to be directly proportional to the number of TCP flows traversing the output-port of a congested router. If this rate is too low the average queue-size converges to *maxth*. If the rate is too high the average queue-size oscillates around *minth*. In both cases the control goal is not achieved. An enhancement of RED called ARED (adaptive RED) is proposed (see section 2.2) to dynamically adjust the max_p parameter. As we will show in section 6, *maxp* has to be set as a function of bottleneck-bandwidth, *RTT* and number of TCP flows and not only as a function of the number of flows as assumed in [3] and [4].

Based on the investigation of RED behavior in the presence of a small number of TCP flows, [12] proposes simple performance models and guidelines for RED parametersetting. The results of this paper can be summarized as follows.

- Setting the *wq* parameter is recommended to be different for a building queue and a

draining queue. For the building queue, a model investigating RED's response to a unit step input signal shows that the average queue size equals 90% of the input signal's amplitude after 2 RTTs if *wq* is set to the inverse of the number of queue-samples per RTT. This corresponds to a *wq* set to the inverse of the pipe-size in MTU-sized packets, if the average queue size is calculated at each packet arrival. Although common practice in control theory, it may be somewhat oversimplified to model the complex behavior of a congestion-controlled traffic pattern by a unit step signal, resulting in a questionable model.

- For a draining instantaneous queue, it is recommended to set wq equal to one; i.e. the average queue size should be equal to the instantaneous queue size. This recommendation is derived from TCP behavior. If a packet loss happens, a TCP data-sender halves its congestion window. It is argued that this sudden decrease in load should be reflected in the behavior of RED's low-pass filter. However, in scenarios having a high degree of statistical multiplexing many TCP flows are likely to increase their windows while a few flows may halve their congestion window. Hence the decay of the RED queue-size is attenuated, making the recommendation of setting *wq* to one in case of a draining queue questionable for realistic scenarios.
- It is mentioned that concerning the stability and the efficiency of the system TCP with RED, it is not of importance whether RED is operated in byte or packet mode. This finding corresponds to our simulation results. Obviously, RED in byte mode is useful to avoid unfairness against flows having smaller packet sizes.
- As a rule of thumb, it is recommended to set the difference between *maxth* and *minth* in the order of 1 to 1.5 times the bandwidth RTT product of the scenario. This recommendation is not supported by a model or performance analysis.
- Based on a model for RED's response to an arriving burst, it is recommended to set *minth* in the order of 0.3 times the bandwidth RTT product of the scenario.

In [11] TCP with RED is modelled as a feedback control system using the TCP model proposed in [15]. Assuming homogeneous RTTs, a quantitative model for TCP with RED in equilibrium is derived. However, although the model is able to derive guidelines how to set the RED control function to enable convergence to an equilibrium point (constant drop probability and queue-size), convergence is considered over infinite time averages. Hence the model gives no prediction whether the system really converges to the equilibrium point or oscillates around the equilibrium point and, in case of oscillation, what is the amplitude and frequency of the oscillation. For a transient-state system, aiming at resolving these shortcomings of the model, only qualitative results can be found.

Nevertheless, a thorough quantitative model how to set the wq parameter is derived in [11]. It is found that setting the RED averaging interval *(I)* equal to the length of the aver-

age TCP period of window increase and decrease due to a packet drop, gives a good compromise between the opposing conditions of maintaining the moving average close to the long term average and making the moving average respond quickly to a change in traffic conditions (such as an increase in the number of flows). The length of the TCP period *(I)* is derived using results from [11]. For the sampling interval δ an upper bound equal to the RTT is derived. The queue weight can then be computed as

$$wq = 1 - a^{\delta / I},$$

where *a* is a constant parameter in the order of 0.1.

4 Simulation Settings

Simulations are performed with the NS network simulator [16]. Unless otherwise noted, *N* TCP flows start at random times between zero and 10 seconds of simulation time. Hosts at the lefthand side act as sources, hosts at the righthand side act as sinks. Host *i* starts *N/m* TCP flows to host *m+i*, where *m* denotes the number of access links connecting hosts to routers. TCP data-senders are of type Reno and New-Reno. Packet sizes are uniformly distributed with a mean of 500 bytes and a variance of 250 bytes.

Simulations are performed on two types of networks. The simulated network shown in figure 4 is used for scenarios with homogeneous RTTs. All 500Mbps links have drop-tail queue-management; buffer-sizes at access links are set sufficiently high to avoid packet loss. The link between router1 and router2 uses RED queue-management. Packets are discarded solely at the bottleneck-link from router1 to router2.

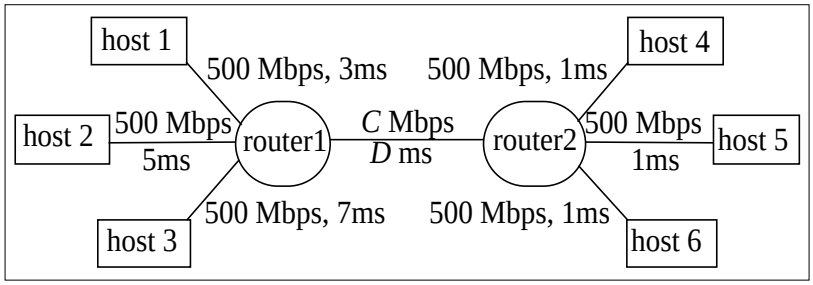


Figure 4 *Network 1*

The second network is used for scenarios with heterogeneous RTTs. Again, all access links have 500Mbps links and drop-tail queue-management; buffer-sizes at access links are set sufficiently high to avoid packet loss. The link between router1 and the sink uses RED queue-management. Packets are discarded solely at the bottleneck-link from router1 to the sink.

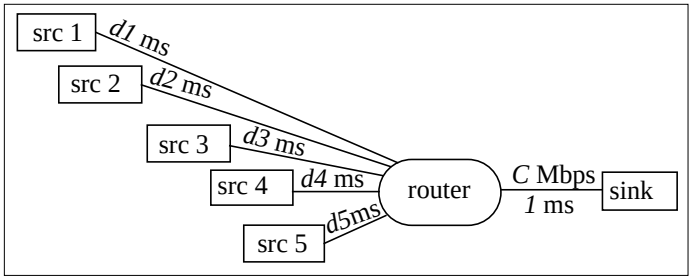


Figure 5 *Network 2*

RED is operated in packet mode as earlier simulations did not show any difference regarding the convergence behavior of the queue with RED in byte and packet mode. The “mean packet-size” parameter of RED is set to 500 bytes.

Figures showing the average and instantaneous queue-size over time should be read as follows. Part-figure a-d are ordered from left to right. In all queue-size over time figures the *avg* is plotted with a fat line, the instantaneous queue-size is plotted with a thin line. The unit of the x-axis is seconds, the unit of the y-axis is mean packet-sizes. Note that the four graphs per figure represent only a subset of the total number of simulations performed, showing the tendency of the behavior of the queue.

All simulations have been repeated several times in order to ensure convergence of results. Simulations last for 100 seconds, however only the steady state behavior is of interest, hence we plot only the last 30 seconds in queue size over time figures.

5 RED Parameter Settings for one-way bulk-data TCP Traffic

The RED parameters important for stability are the queue weight (*wq*), the maximum drop probability (*maxp*), and the difference between the queue-size thresholds (*maxth -minth*). As we will show in subsequent sections, each of these parameters has to be set as a function of the per TCP flow bandwidth RTT product; in other words knowledge of the bottleneck capacity, number of TCP flows and the delay distribution of TCP flows is required as input for our models in order to achieve the control-goal (convergence of the queue-size).

Our approach to enable systematical investigation is to use the model for setting of *wq* proposed in [11] and discussed in section 3. The sampling interval δ for the model of *wq* is set to the average packet-time on the link (mean packet size / link capacity); the constant *a* is set to 0.01. As a second step, a model is derived how to set maxp as a function of the per-TCP-flow bandwidth RTT product. This model does not assume homogeneous RTTs. Knowing how to set wq, we can set (*maxth -minth*) sufficiently high to avoid oscillations and verify the model for *maxp* by simulation. Finally, we propose an empirical model providing a lower bound for (*maxth -minth*) to avoid oscillation of the queue and verify the total model by simulation and measurements.

6 A model for maxp

We present a steady-state analysis resulting in a quantitative model on how to set *maxp* as a function of *C,N* and *RTT* [1]. As shown in [15], the rate of a TCP flow *i* (R_i) as a function of the loss-probability *p*, its round trip time (RTT_i) and retransmission time-out time (T_i) can be modelled by the following expression:

$$R_i = \frac{1}{RTT_i \sqrt{\frac{2bp}{3}} + T_i \cdot min\left(1, 3 \cdot \sqrt{\frac{3bp}{8}}\right) p(1 + 32p^2)} \qquad (1)$$

The constant *b* in the above equation denotes the number of packets received at the TCP data-receiver to generate one ACK. With a delayed-ack TCP data-receiver *b* would be equal to 2. For the moment, we do not use delayed ACKs hence *b* is set to one.

For any aggregate of *N* TCP flows in steady state the following expression is valid:

$$C = \sum_{i=1}^{N} R_i \qquad (2)$$

Substituting R_i by (1) we get a function of *p,C, N,* a distribution of round trip times and retransmission time out times. Our goal is to make the average queue-size converge at *(minth+maxth)/2,* which corresponds to a drop-probability of *maxp*/2. Substituting *p* with *maxp*/2, we can derive the optimum setting of *maxp* as a function of *C,RTT* and *N*:

$$C = \sum_{i=1}^{n} \frac{1}{RTT_i \sqrt{\frac{bmaxp}{3}} + T_i \cdot min\left(1, 3\sqrt{\frac{3bmaxp}{16}}\right) \frac{maxp}{2}(1 + 8maxp^2)} \qquad (3)$$

We are not able to provide an analytically derived closed-form expression for *maxp* as solving (3) for max*p* results in a polynomial of degree seven. However, numerical solution provided that *C,N,* and the distributions of *RTT* and *T* are given is feasible with mathemat-

[1] Note that C, N, RTT can be combined to one quantity - the per-flow bandwidth*RTT product.

ics programs [17].

ISPs are not aware of the exact distribution of RTTs of flows passing a RED queue. The best we may hope to obtain in order to provide a practical model is a histogram, grouping flows into *m* RTT classes, where each RTT class *j* has n_j flows and homogeneous round trip time. For such a model, eq. (3) can be rewritten as

$$C = \sum_{j=1}^{m} \frac{n_j}{RTT_j \sqrt{\frac{bmaxp}{3}} + T_j \min\left(1, 3\sqrt{\frac{3bmaxp}{16}}\right)\frac{maxp}{2}(1 + 8maxp^2)} \quad (4)$$

For derivation of *maxp* in all subsequent simulations we have set RTT_j and T_j as follows:

$$\begin{aligned} RTT_j &= 2d_j + (minth+maxth)/(2C) \\ T_j &= RTT_j + 2(minth+maxth)/C \end{aligned} \quad (5)$$

The term *(minth+maxth)/(2C)* matches the average queueing delay at the bottleneck, d_j denotes the total propagation delay of RTT class *j*. As done by TCP, *T* is computed as the *RTT* plus four times the variance of the *RTT*, approximated by the average queueing delay at the bottleneck.

Simulations in this chapter have the buffersize *B* set to the bandwidth RTT product of the scenario or 40 packets, whichever is higher; *maxth = 2B/3*, *minth = maxth/4*.These settings result in a sufficiently high difference between *maxth* and *minth* to avoid oscillations. *Wq* is set as explained in section 3. In our simulator RED implementation the average queue size is computed at each packet arrival. Thus δ is set to the inverse of the link capacity in packets. The constant *a* is set to 0.01, as proposed in [11].

6.1 Simulations with homogeneous RTTs

Simulations in this chapter use the network topology illustrated in figure 4. Each access link carries N/3 TCP flows. Delayed acks is deactivated.

6.1.1 Simulation1: varying bottleneck bandwidth.

Constant parameters: D = 100ms, N = 100, maxp = 1/10.

Simulation	1	2	3	4
C	0.5Mb	2Mb	10Mb	50Mb
minth	10	29	143	714
maxth	40	114	571	2857
B	60	171	857	4286
wq	0.011	0.0042	0.00063	0.000035

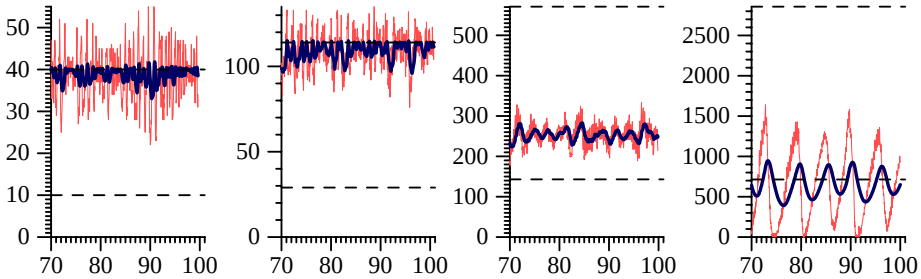


Figure 6 *a-d, inst. and average queue-size over time, maxp = 1/10*

As the increasing bandwidth causes the per-flow bandwidth*delay product to increase, the aggressiveness of the TCP flows and thereby the average queue-size decreases. In figure 6a *maxp* is too small, causing convergence of *avg* to *maxth*. In figure 6d *maxp* is too high causing *avg* to oscillate around *minth* resulting in suboptimal link utilization as the queue is often empty.

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
1/maxp	2.18	4.15	30	618

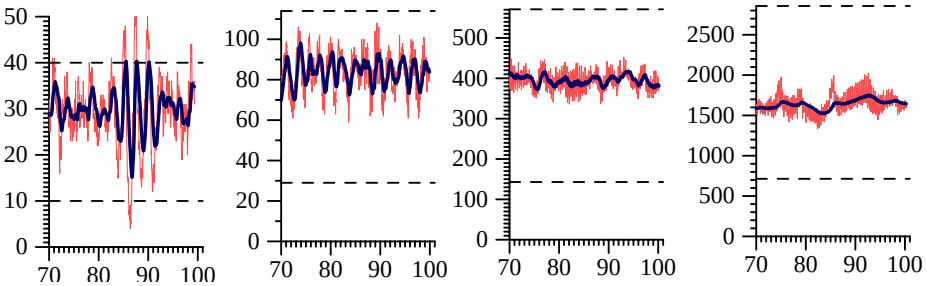


Figure 7 *a-d, inst. and average queue-size over time, maxp adapted*

Figure 7 shows that adapting maxp according to eq. 4, section 6 makes the queue size converge between minth and maxth. In part figure a, the difference between maxth and minth is somewhat too low causing oscillations, however, the mean of the average queue size is in between the two queue-size thresholds.

6.1.2 Simulation2: varying *RTT*

Constant parameters: C = 10Mbps, N = 100. The maxp parameter is set to 1/10, as recommended in [1].

Simulation	1	2	3	4
D	1ms	50ms	100ms	200ms
minth	10	71	142	285
maxth	40	286	571	1142
B	60	428	857	1714
wq	0.0134	0.0016	0.00062	0.00019

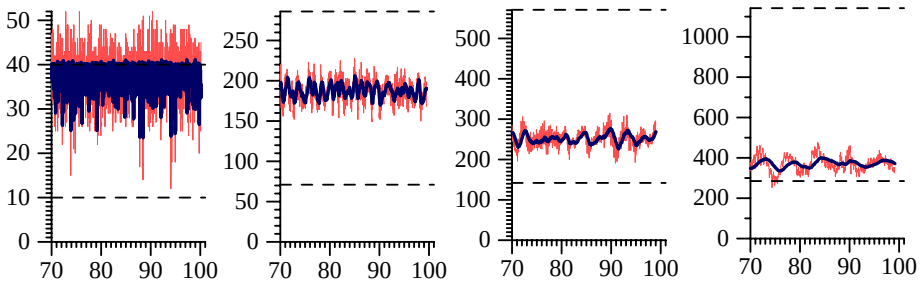


Figure 8 *1-4, inst. and average queue-size over time, maxp = 1/10*

Another parameter for the aggressiveness of TCP is the *RTT*. As the *RTT* increases the per flow bandwidth**RTT* product increases too, causing *avg* to move away from *maxth* closer

to *minth*.

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
1/maxp	2	10.6	29.7	104

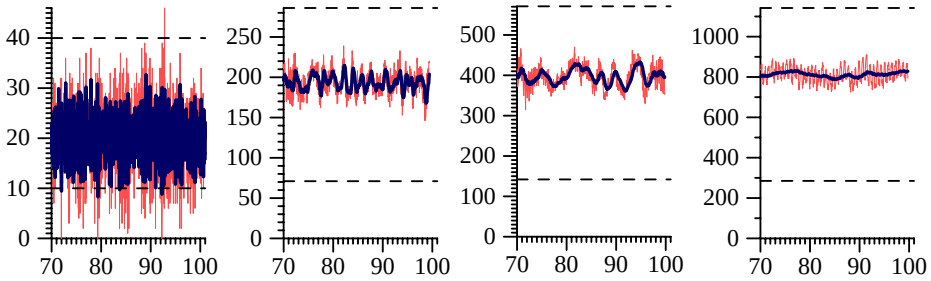


Figure 9 *a-d, inst. and average queue-size over time, maxp adapted*

The mean of the average queue size stays between minth and maxth, showing the correctness of model for maxp. Like in section 6.1.1, simply setting the difference between minth and maxth as some fraction of the bandwidth delay product, is an overly conservative choice for long, fat pipes (as indicated by the almost constant queue-size in figure d). On the other hand, for small bandwidth delay products the difference between maxth and minth is too small, causing oscillations in figure a.

6.1.3 Varying number of flows:

Constant parameters: D = 100ms, C = 10Mbps, minth = 143 packets, maxth = 571 packets, B = 857 packets, maxp = 1/10.

Simulation	1	2	3	4
wq	0.00009	0.0004	0.00063	0.00084

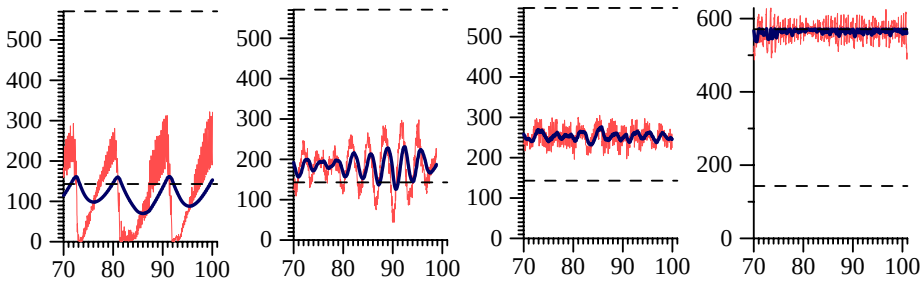


Figure 10 *1-4, inst. and average queue-size over time, maxp = 1/10*

Obviously, the per flow bandwidth**RTT* product decreases with increasing number of flows. As a consequence, the average queue-size moves from a state of oscillation around the minimum threshold to the maximum threshold in case the bandwidth is constant and the number of TCP flows is increased.

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
1/maxp	2454	104	30	4.15

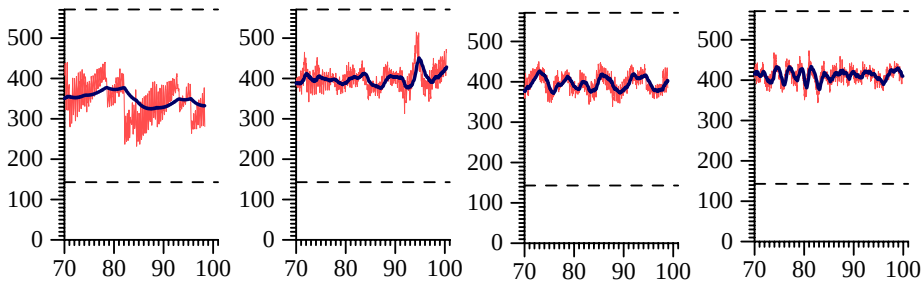


Figure 11 *a-d, inst. and average queue-size over time, maxp adapted*

Again, the average queue size converges between minth and maxth if *maxp* is set according to the model. Figure 7, figure 9 and figure 11 show that the model has a drift: for small values of *maxp* the mean queue is somewhat lower (see figure 11a) than for high values of

maxp. (see figure 11d).

6.2 Simulations with heterogeneous RTTs, various delay distributions

Simulations in this chapter use the network topology illustrated in figure 5. Access delays are distributed according to histograms which can be approximated by linear, concave or convex continuos functions. Link delays increase with increasing link number. Each access link carries N/5 TCP flows. Hence the simulation-set with the concave delay distribution has more flows with higher RTTs; vice versa for the simulation-set with the convex delay distribution. Figure 12 plots sample approximation-curves for the linear, concave and convex histogram of delay distributions.

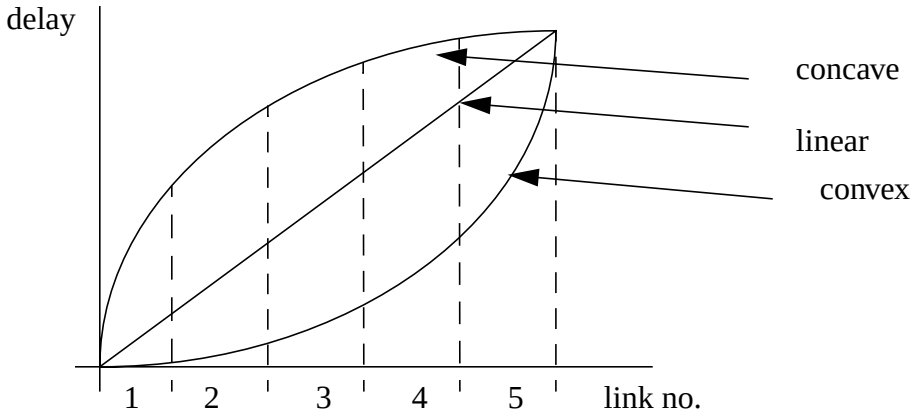


Figure 12 *Characteristics of the delay distribution*

All simulations in this chapter have a mean access delay of 100ms, N = 100, C = 10Mbps, D = 1ms.

6.2.1 Delays distributed according to convex Functions

Simulation	1	2	3	4
d1, d2, d3, d4 d5	100 100 100 100 100	70 74 85 104 130	30 39 65 109 170	1 13 51 112 199
minth	143	132	118	107
maxth	571	529	472	430
B	857	794	708	645
wq	0.00063	0.00071	0.00087	0.001

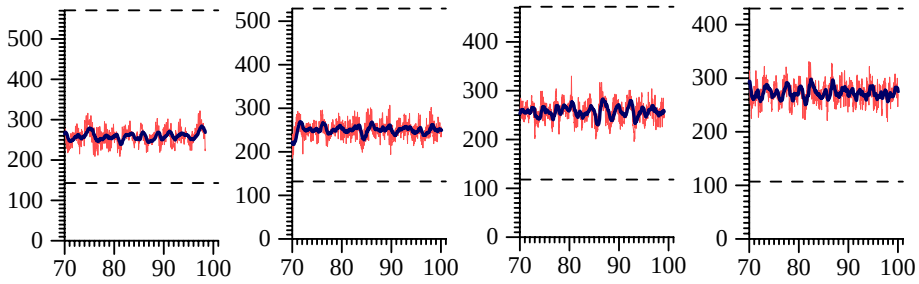


Figure 13 *a-d, inst. and average queue-size over time, maxp = 1/10*

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
maxp	30	25.5	18.6	13.3

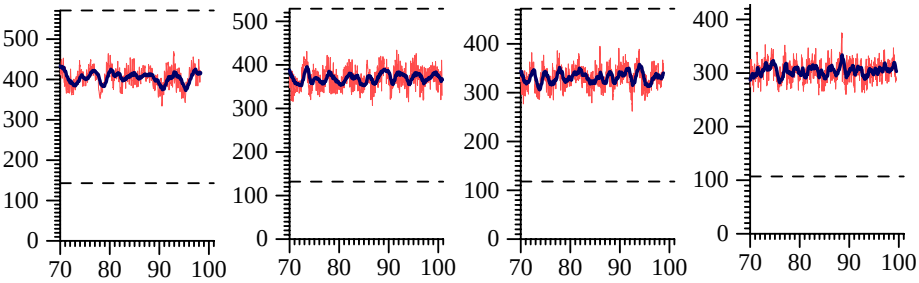


Figure 14 *a-d, inst. and average queue-size over time, maxp adapted*

6.2.2 Delays distributed according to linear Functions

Constant parameters: minth = 143 packets, maxth = 571 packets, B = 857 packets.

Simulation	1	2	3	4
d1 d2 d3 d4 d5	100 100 100 100 100	70 85 100 115 130 145	30 65 100 135 170	1 50 100 150 200
wq	0.00063	0.00063	0.00066	0.00068

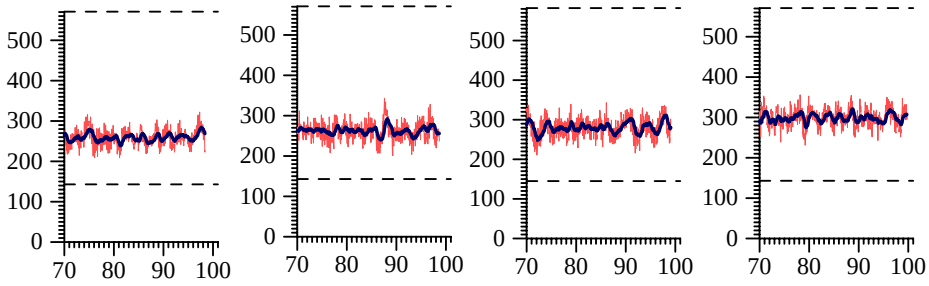


Figure 15 *a-d, inst. and average queue-size over time, maxp = 1/10*

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
1/maxp	30	29	26	22.4

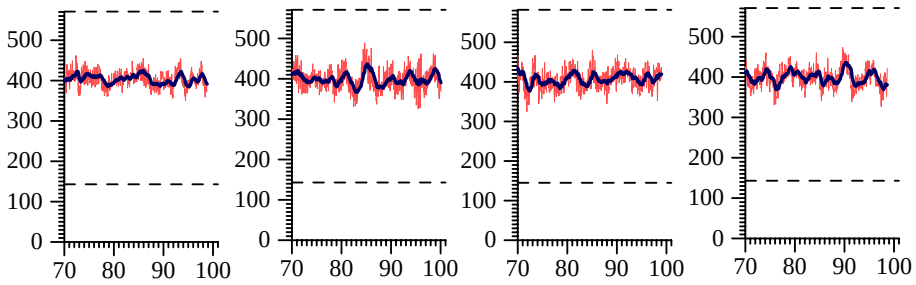


Figure 16 *a-d, inst. and average queue-size over time, maxp adapted*

6.2.3 Delays distributed according to concave Functions

Simulation	1	2	3	4
d1 d2 d3 d4 d5	100 100 100 100 100	70 88 104 118 130	30 88 122 147 170	1 99 141 172 199
minth	143	146	159	175
maxth	571	583	637	699
B	857	874	955	1049
wq	0.00063	0.00061	0.00055	0.0005

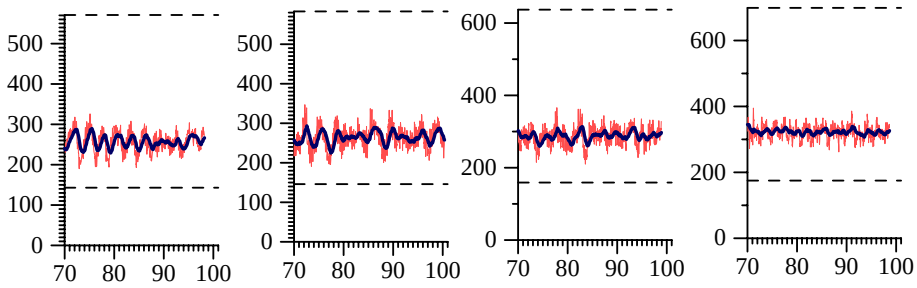


Figure 17 *a-d, inst. and average queue-size over time, maxp = 1/10*

Same simulation as above, but maxp adapted according to the model:

Simulation	1	2	3	4
1/maxp	30	30	31.5	32

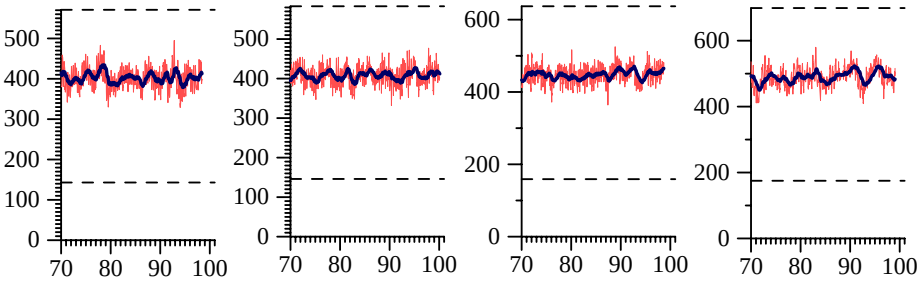


Figure 18 *a-d, inst. and average queue-size over time, maxp adapted*

In any case (convex, linear, concave), the shape of the delay distribution only marginally influences the convergence of the queue. However, as shown by the figures having *maxp* adapted according the model for heterogeneous RTTs, the queue converges at a constant

equilibrium point close to *(maxth+minth)/2*.

6.3 Simulations with heterogeneous RTTs, one Delay Distribution, varying number of Flows per Link

Constant parameters: minth = 143, maxth = 573, B = 859, C = 10Mbps, d1 = 1ms, d2 = 50ms, d3 = 100ms, d4 = 150ms, d5 = 200ms; n_i denotes the number of flows at access link i.

Simulation	1	2	3	4
wq	0.00038	0.00046	0.001	0.0012
n1 - n5	1 5 10 20 64	5 10 15 25 45	45 25 15 10 5	64 20 10 5 1
1/maxp	48	36.7	15.2	13

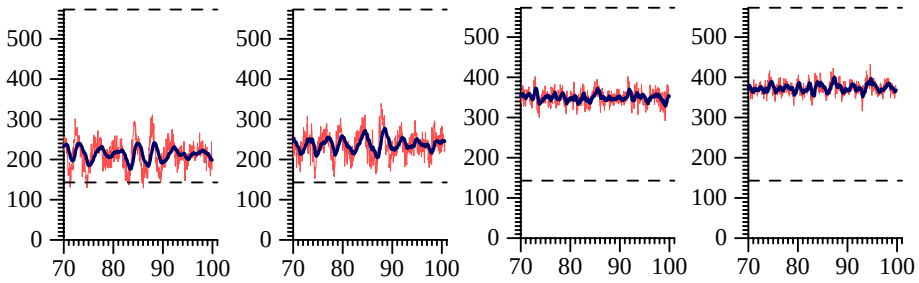


Figure 19 *a-d, inst. and average queue-size over time, maxp = 1/10*

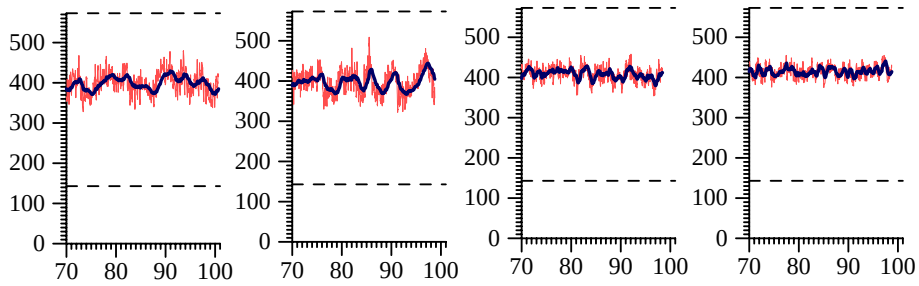


Figure 20 *a-d, inst. and average queue-size over time, maxp adapted*

Similar to section 6.2, the queue converges at a constant equilibrium point close to *(maxth+minth)/2* independently of the RTT distribution if the maxp parameter is adapted according to the model.

7 Setting maxth-minth

7.1 Constant difference between maxth and minth

This chapter investigates the behavior of the RED queue if the scenario-parameters (bottleneck bandwidth, RTT and the number of flows) are varied and the difference between maxth and minth is constant. We will see that non-adoption of the difference between maxth and minth max cause heavy oscillations in case of high bandwidth*RTT products and number of flows. Simulations in this chapter use the topology shown in figure 4.

Simulation1: varying bottleneck bandwidth

Constant parameters: D = 100ms, minth = 40, maxth = 300, B = 400. N is adapted so that the per flow bandwidth*RTT product stays constant.

Simulation	1	2	3	4
C	0.5Mb	5Mb	20Mb	50Mb
N	20	42	117	267
1/maxp	40	39	37	36
wq	0.0025	0.0011	0.00044	0.00019

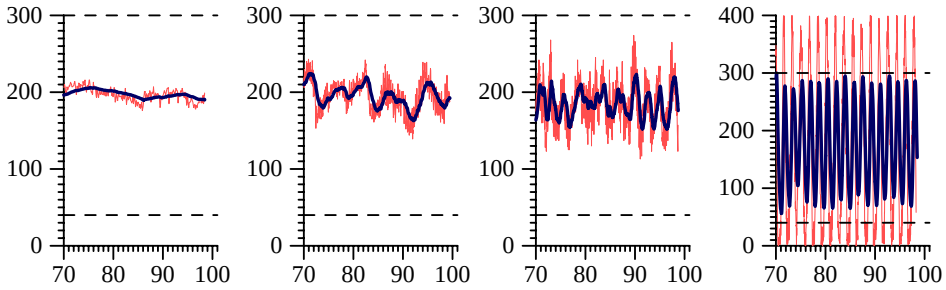


Figure 21 *a-d, inst. and average queue-size over time*

Simulation2: varying *RTT*

Constant parameters: C = 20Mbps, minth = 40, maxth = 300, B = 400. N is adapted so that the per flow bandwidth*RTT product stays constant.

Simulation	1	2	3	4
D	1ms	50ms	150ms	300ms
N	18	67	167	317
1/maxp	42	37.5	36	36
wq	0.0025	0.0015	0.0003	0.00017

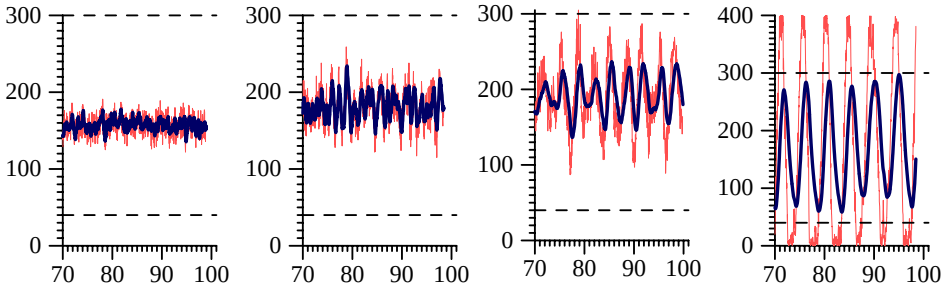


Figure 22 *a-d, inst. and average queue-size over time*

Simulation2: varying number of flows

Constant parameters: C = 20Mbps, D = 100ms, minth = 40, maxth = 300, B = 400

Simulation	1	2	3	4
N	10	100	250	300
1/maxp	4566	49	10.3	8
wq	0.00005	0.0004	0.00065	0.0007

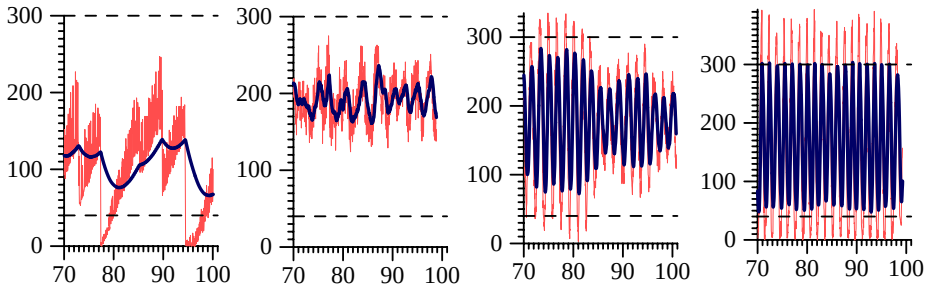


Figure 23 *a-d, inst. and average queue-size over time*

As shown in figure 21 and figure 22, the difference between *maxth* and *minth* has to be a monotonically increasing function of the bandwidth*delay product in order to avoid oscillation. If *maxth-minth* is too small compared to the bandwidth*delay product, the queue-size oscillates significantly. Figure 21 and figure 22 additionally show that the frequency of oscillation is directly proportional to the bottleneck bandwidth and inversely proportional to the RTT.

Figure 23 illustrates that in addition to the bandwidth*RTT product of the scenario, the number of flows is an important parameter to set *maxth-minth* properly. *Maxth-minth* should be sufficiently high to keep amplitude of the oscillation of the average queue-size small compared the difference between the queue-size thresholds. The difference between *maxth* and *minth* has to be inversely proportional to the average per-flow window in a scenario. The per flow window can be computed as *(bandwidth*RTT)/N*. Figure 23 additionally shows that the frequency of oscillation is inversely proportional to the number of flows.

In case of small per-flow windows (or many flows - figure 23d), the difference between *maxth* and *minth* has to be sufficiently high to avoid oscillations with high amplitudes. The reason why the amplitude of the oscillation is indirectly proportional to the per-flow window is due to TCP falling into Retransmission Timeout and Slowstart with higher probability the smaller the window. Consider the extreme case of a TCP flow with a window of only 3 packets. For such a TCP flow, Fast Retransmit/Recovery won't work and each packet loss at the RED queue will cause a Slowstart. The higher the average per-flow window, the smaller the probability TCP performs Slowstart (see figure 23c and b). However, for large per-flow windows (i.e. few TCP flows) this statement is not true anymore (see figure 23a). In this case we can't expect a regular, bounded oscillation as each packet drop causes strong fluctuation in the queue. Additionally, the probability for more than one packet drops per window increases in case of large windows, increasing the likelihood for TCP to perform Retransmission Timeout and Slowstart instead of Fast Retransmit/Recovery and keeping Self-Clocking going.

7.2 Empirical Model for maxth-minth and homogeneous RTTs.

Due to the lack of a tractable and sufficiently accurate analytical model we use an empirical approach to determine the setting of *maxth-minth* as a function of the bottleneck bandwidth, *RTT* and the number of flows. For the moment, we only consider the case of homogeneous RTTs (section 7.3 will generalize the model to the heterogeneous RTT case). The idea behind this empirical model is that we may find out the proper setting of *maxth-minth* for a specific scenario (as defined by the bandwidth*RTT product and num-

ber of flows) by trial and error, using the models already derived for maxp and wq to make the queue converge to *(maxth+minth)/2*. By performing many simulations, exploiting the maximum range of link speeds, propagation delays and number of flows the ns simulator allows on high-end Pentium PCs, we get a cloud of points in the 3-dimensional *maxth-minth/bandwidth*RTT/number of flows* space. This cloud of points is approximated by a linear least square fit, resulting in a 3D-plane representing *maxth-minth* as a function of the bandwidth*RTT product and the number of flows. The 3D plane can be extrapolated to even higher bandwidth*RTT products and numbers of flows.

Figure 24 shows the results of these simulations (each point represents one simulation where the proper *maxth-minth* has been found by trial and error). In all simulations figure 24 is based on, *maxth-minth* is set so that the amplitude of the oscillation of the average and the instantaneous queue-size stays roughly constant (see figure 27). The total buffer-size (*B*) is set to 3**maxth*/2 in all simulations, which is sufficiently high to avoid frequent losses due to buffer overflow for RED with TCP flows in steady state. The *minth* parameter is set to *(maxth-minth)/3*; delayed acks is deactivated; the mean packet size equals 500 bytes; for the rest of the simulation settings we refer to section 4. Simulations in this chapter use the topology shown in figure 4.

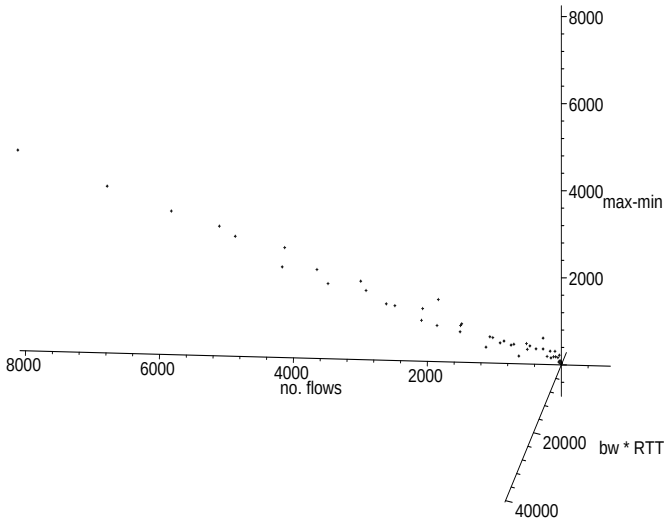


Figure 24 *scatter plot of simulation results for maxth-minth (in mean-packets) as a function of the bandwidth*RTT product (in mean-packets) and the number of flows*

Figure 25 illustrates how figure 24 has been created and ameliorates its readability. Six sets of simulations have been conducted for creation of figure 24. Each set of simulations consists of 10 simulations having identical average per-flow windows. The average per-flow window is defined as the product of bandwidth and RTT divided by the number of flows. Each set of simulations is represented by one curve in figure 25.

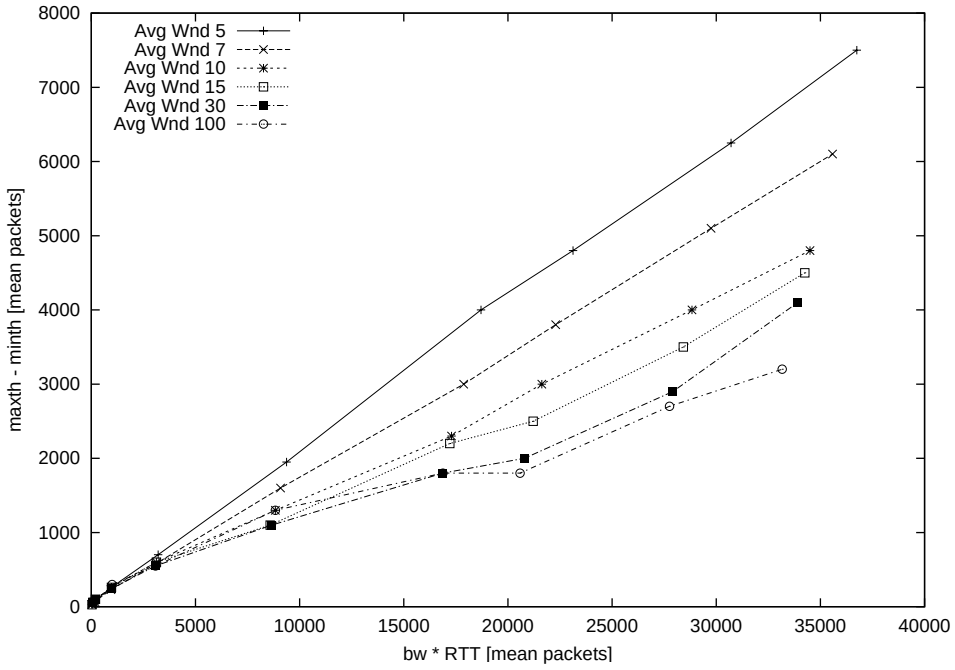


Figure 25 *maxth-minth as a function of the bandwidth*RTT product; various per-flow windows.*

Figure 26 zooms the low bandwidth*RTT product range of figure 25 in order to visualize better the setting of maxth-minth for small bandwidth*RTT products.

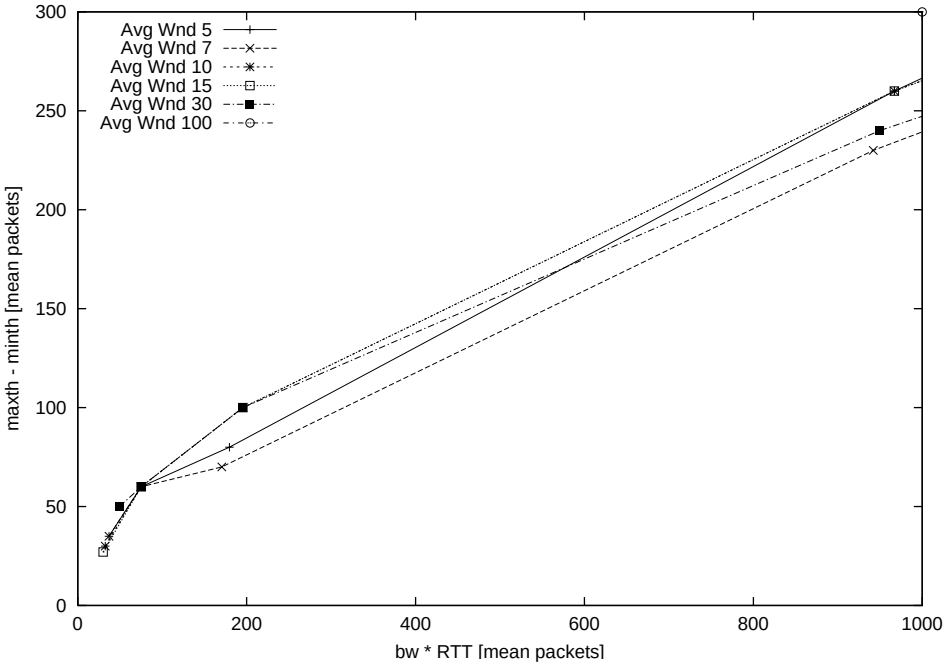


Figure 26 *maxth-minth as a function of the bandwidth*RTT product; various per-flow windows.*

Each curve of figure 25 has 10 points, thus 10 simulations with *maxth-minth* set so that the amplitude of the oscillation of the average queue stays constant. Figure 27 shows the simulations used to derive the average-window = 10 curve in figure 25. Queue-size over time figures used to derive the other curves in figure 25 are not shown in this paper.

Simulation	1	2	3	4	5	6	7	8	9	10
D	10ms	20ms	40ms	70ms	100ms	150ms	200ms	250ms	250ms	300ms
C	1Mb	2Mb	5Mb	20Mb	50Mb	100Mb	150Mb	150Mb	200Mb	200Mb
N	3	8	20	97	313	883	1729	2163	2883	3450
minth	10	20	33	87	200	433	767	1000	1333	1600
maxth	40	80	133	347	800	1733	3067	4000	5333	6400
B	60	120	200	521	1200	2600	4601	6000	8000	9600
1/maxp	47	36	37	37	37	37	36	36	36	36
wq	0.013654	0.006526	0.002556	0.000530	0.000166	0.000059	0.000030	0.000024	0.000018	0.000015

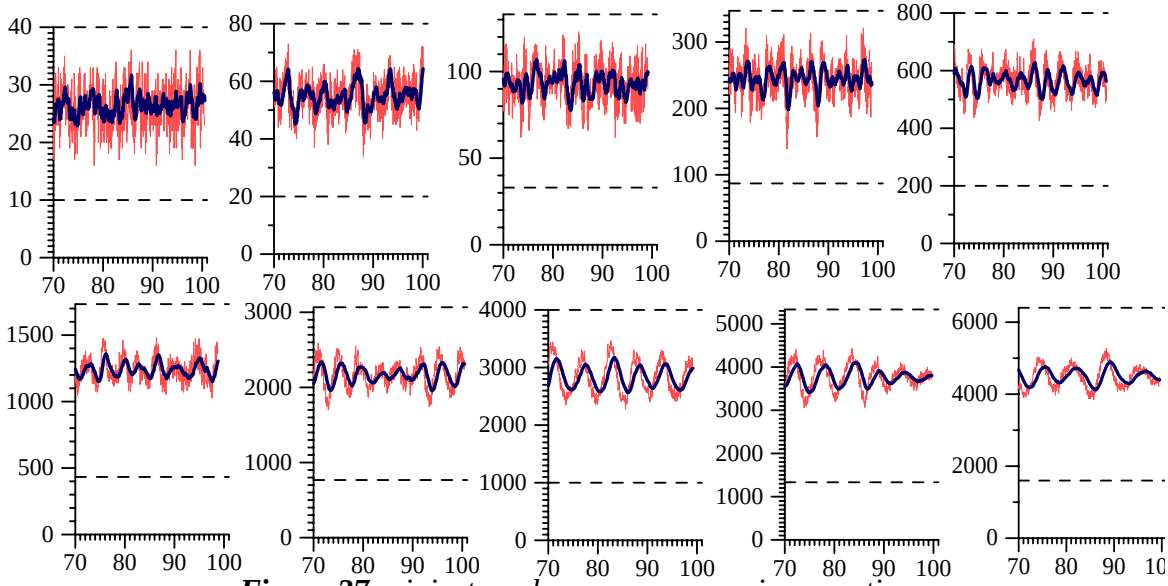


Figure 27 *a-j, inst. and average queue-size over time*

The distribution of points in figure 24 and the curves in figure 25 show that *maxth-minth* can be considered as a linear function of the bandwidth*RTT product and the number of flows. Thus the cloud of points in figure 24 is approximated by a linear least square fit. The following function for *maxth-minth* as a function of the bandwidth*RTT product and the number of flows is the result of the fit:

$$maxth - minth = 0.2158 \cdot C \cdot RTT + 0.567 \cdot N + 84.7 \qquad (1)$$

Figure 28 plots the graph for equation 1 together with the scatter plot of the simulation results in figure 24. Note that (1) gives a lower bound; simulations show that the amplitude of the oscillation decreases further if *maxth-minth* is set higher than suggested by (1).

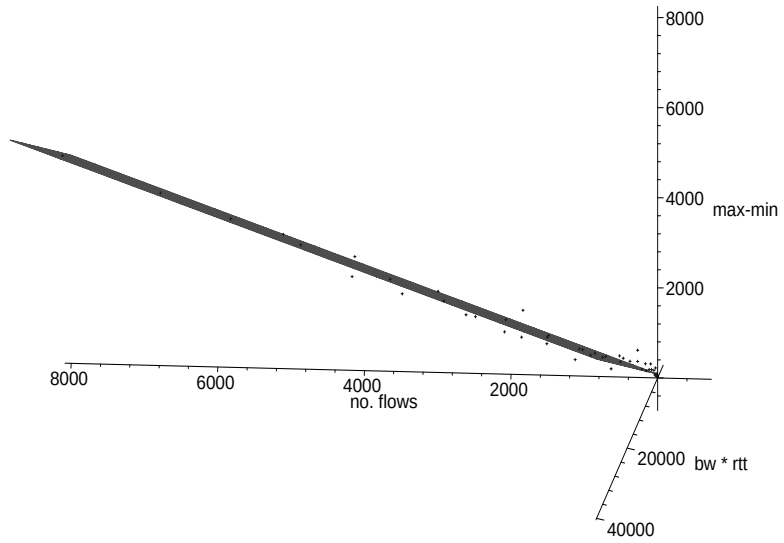


Figure 28 *Same as figure 24 but approximated with least square fit (see to eq. 1).*

Although equation 1 results in settings of *maxth-minth* significantly smaller than the bandwidth*delay product (compare to [12]), the buffer requirements of RED could still be a problem regarding its usability in high-speed WANs. For instance, considering 10000 FTP flows over a network path with a propagation delay of 50ms and OC48 links (2.5 Gbps), the difference between *maxth* and *minth* would have to be higher than 20000 500-byte packets according to the model. The bandwidth RTT product for this scenario equals 62500 500-byte packets.

7.3 Building the total Model for RED with TCP and heterogeneous RTTs

7.3.1 Incorporating heterogeneous RTTs

Equation one, section 7.2, provides a model how to set maxth-minth as a function of the bottleneck bandwidth, number of flows and the RTT, assuming that all flows have equal RTTs. This section derives a model in case the assumption of homogeneous RTTs does not hold anymore.

We propose to compute the RTT parameter for eq. 1, section 7.2 as the weighted average of the per-flow RTTs. The weights for the RTT-average (*aRTT*) are computed as a flow's rate divided by the link capacity; in other words a flow's share of the link capacity. The

reason to choose a weighted average instead of a simple mean average of RTTs is based on TCP's property of sending with a higher rate in case the RTT is shorter. The behavior of flows sending with a higher rate than others has higher influence on the queue, thus it makes sense to weight the average by a flow's transmission rate.

Similar to section 6, we assume a histogram of *m* RTT classes, where each RTT class *j* has n_j flows and homogeneous round trip time RTT_j. The aggregate rate R_j of class j can be computed according to eq. 1, section 6:

$$R_j = \frac{n_j}{RTT_j \sqrt{\frac{bmaxp}{3}} + T_j \cdot min\left(1, 3\sqrt{\frac{3bmaxp}{16}}\right) \frac{maxp}{2}(1 + 8maxp^2)} \qquad (1)$$

The average RTT, weighted by the rate of flow-aggregates can be computed as follows:

$$aRTT = \sum_{j=1}^{m} \frac{R_j}{C} \cdot RTT_j \qquad (2)$$

Note that eq. 2 simplifies to the mean average for the special case of homogeneous RTTs ($R_j = C/m$).

Finally, equation 1, section 7.2 has to be rewritten as

$$maxth - minth = 0.2158 \cdot C \cdot aRTT + 0.567 \cdot N + 84.7 \qquad (3)$$

7.3.2 Building the total Model for RED with TCP

Maxp is a function of the bandwidth, number of flows and the RTT (see section 6.1). The RTT in turn depends on the queueing delay, which may be approximated as *(maxth+minth)/(2C)* assuming that *maxp* is set correctly. Thus the setting of *maxp* depends on *maxth* and *minth*. In preceding chapters, we have assumed *maxth* and *minth* as given. Consequently, the queueing delay (and therefore the RTT) has been known, enabling the computation of *maxp* and *wq* as a function of the bottleneck bandwidth, number of flows and the RTT. For the following reason it is not possible anymore to consider the computation of *maxp* and *maxth-minth* as independent from each other: as shown in section 7.3.1, the setting of *the aRTT* quantity and thus the setting of *maxth-minth* depends on *maxp*; the setting of *maxp* depends on *maxth* and *minth*. As a consequence, equation one (1), section

7.2 and equation (4), section 6 have to be combined to a system of non-linear equations. This system of equations can be solved numerically for *maxp, minth and maxth*. Knowing how to set *maxp, minth and maxth* for a specific scenario, *wq* can be computed as done in previous sections.

The following paragraphs summarize the system of equations (equations 1,2,3) to be solved:

$$C = \sum_{j=1}^{m} \frac{n_j}{RTT_j \sqrt{\frac{bmaxp}{3}} + T_j \min\left(1, 3\sqrt{\frac{3bmaxp}{16}}\right)\frac{maxp}{2}(1 + 8maxp^2)} \quad (1)$$

$$maxth - minth = 0.2158 \cdot C \cdot aRTT + 0.567 \cdot N + 84.7 \quad (2)$$

$$minth = \frac{maxth - minth}{3} \quad (3)$$

Helper functions for eq. 1-3:

$$RTT_j = 2d_j + (minth+maxth)/(2C)$$
$$T_j = RTT_j + 2(minth+maxth)/C \quad (4)$$

$$R_j = \frac{n_j}{RTT_j \sqrt{\frac{bmaxp}{3}} + T_j \min\left(1, 3\sqrt{\frac{3bmaxp}{16}}\right)\frac{maxp}{2}(1 + 8maxp^2)} \quad (5)$$

$$aRTT = \sum_{j=1}^{m} \frac{R_j}{C} RTT_j \quad (6)$$

Finally, after maxth, minth and maxp have been derived by numerically solving the system of equations above, wq can be computed as follows (see section 5):

$$wq = 1 - a^{\delta / I}, \quad (7)$$

where the sampling interval δ is set to the average packet-time on the link (mean packet size / link capacity) and the constant *a* is set to 0.01 (see [11]). The term *I* equals the length of the TCP period, as derived in [11].

Equations 1-7 represent the total quantitative model of RED with bulk-data TCP flows, suitable for heterogeneous RTTs. This model has been implemented in the Maple mathematics program [17], requires as an input the RTT distribution, the number of flows per RTT class and the bottleneck capacity and outputs the RED parameters.

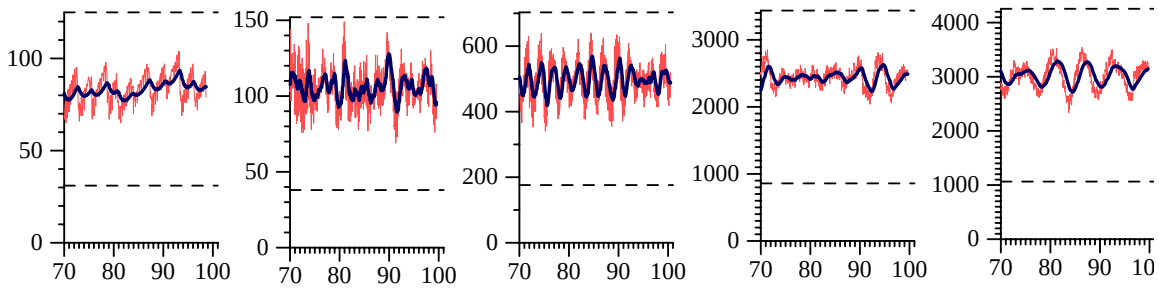
8 Evaluation of the Model

Simulations in this chapter have *maxp*,*wq* and (*maxth-minth*) set as proposed in section 7.3.2.

8.1 FTP-like Traffic

A first set of simulations repeats the scenarios used for the creation of the model for *maxth-minth*, but with values for *maxth, maxp, minth* and *wq* set according to the model in section 7.3.2.[2] These simulations show that for identical C,D,N values as used for creation of figure 24, the parameter setting recommended by the model makes the queue-size oscillate around (maxth+minth)/2. The amplitude of the oscillation is small compared to the difference between maxth and minth, thus the system behaves as desired. We show, however, only a small subset out of the set of 50 simulations performed to evaluate the model.

Simulation	1	2	3	4	5
C	1Mbps	5Mbps	50Mbps	150Mbps	150Mbps
D	.01	.04	0.1	0.2	.25
N	3	20	313	1729	2163
minth	31	38	176	859	1064
maxth	125	152	703	3436	4255
B	187	228	1054	5155	6383
1/maxp	280.22	41.6	35.25	37.4	36.96
wq	0.002540	0.0023	0.000172	2.949e-05	2.383140e-05



[2] Recall that the model for maxth-minth is based on a linear approximation of the simulation results used to derive the model. Thus these simulations are required in order to evaluate whether the approximation is sufficiently accurate.

Figure 29 *a-e, inst. and average queue-size over time*

Figure 29a shows that for small bandwidth*RTT values, the difference between maxth and minth as proposed by the model is somewhat higher that required. This is a result of marginal inaccuracy introduced by the linear approximation of the points in figure 24.

For a second set of simulations, arbitrary points have been selected in the (*C*,*RTT*,*N*) space in order to verify whether the proposed model avoids oscillations. Again, we are only able to show a small subset of the performed simulations in this document. We have selected four scenarios with small RTTs and high bottleneck capacities and vice versa, as such scenarios have not been shown before[3].

Simulation	1	2	3	4
C	10Mbps	100Mbps	2Mbps	10Mbps
D	0.001	0.05	0.2	0.3
N	8	412	32	200
minth	33	200	43	118
maxth	132	799	173	473
B	198	1198	259	710
1/maxp	74.014493	24	36.7	31
wq	0.003381	0.000185	0.0016	0.000362

[3] So far we have implicitly assumed that (besides the number of flows) the setting of maxth-minth depends solely on the *product* of bandwidth and RTT, no matter how the individual values for the bottleneck bandwidth and RTT are set in a scenario. These simulations show that this assumption holds. An additional argument why this assumption is supposed to hold is that TCP uses window flow control. Window flow control merely controls the volume of data in the net which is determined by a flow's bandwidth*RTT *product*.

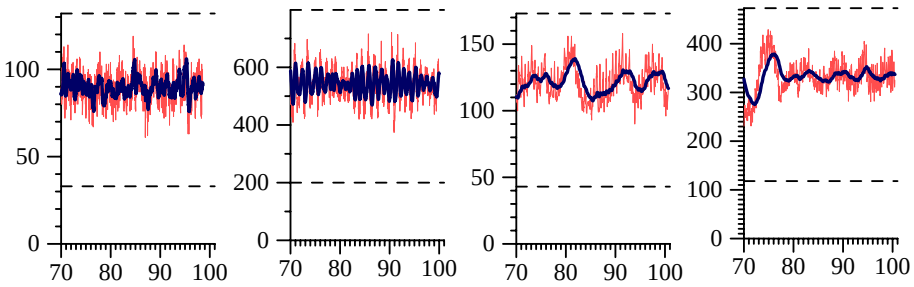


Figure 30 *a-d, inst. and average queue-size over time*

As shown in figure 30, the recommended settings for *maxp*, *wq* and *maxth-minth* avoid oscillations and makes the queue converge in between *minth* and *maxth*, also for arbitrary points in the (C,N,RTT) space.

A third set of simulations over the entire (C,N,RTT) space used to create the model investigates whether the model is valid also for TCP flows having delayed Acks switched on. Somewhat surprisingly, these simulations show that the model can be applied to TCP flows, no matter whether delayed acks is active or not. In all simulations, the queue size converges in-between *minth* and *maxth* as desired.

Finally, simulations with heterogeneous RTTs show that the model, built also for the heterogeneous RTT case, can be applied to scenarios with TCP flows having different round trip times.

We are aware of the fact that it is questionable to evaluate a model by simulation, which has partly been derived by simulation (the setting of *maxth -minth)*. Section 11 presents measurements, showing that the model is correct for a constant bandwidth*RTT product and a varying number of flows.

8.2 Web-like Traffic

We simulate *U* Web client-server connections. A client downloads a document from the server, is idle for a think-time starting at the point in time the entire document has been received by the client and downloads the next document. The think-times are exponentially distributed with a mean of μ_t seconds. According to the findings on the document size distributions of WWW traffic published in [18], the document sizes are Pareto distributed (heavy tailed) with parameter a = 1.06, b = 1000 bytes.

Subsequent simulations have homogeneous RTTs, a mean packet size of 500 bytes and

make use of the topology in figure 4. The mean think time (μ_t) equals 10 seconds and the mean document size equals 7000 bytes. The N parameter (number of TCP flows) required for the RED parameter model is estimated by the mean average of the number of active flows over the entire simulation time.

Varying parameters:

Simulation	1	2	3	4	5
C	1Mbps	1Mbps	5Mbps	5Mbps	20Mbps
D	10ms	10ms	40ms	40ms	70
N	12	119	305	1013	493
minth	32	51	85	238	156
maxth	128	204	339	951	623
B	192	306	509	1426	934
1/maxp	61	4.4	4	3	5
wq	0.0045	0.004	0.002	0.00073	0.00068
U	160	300	1200	2000	4000

Simulation1:

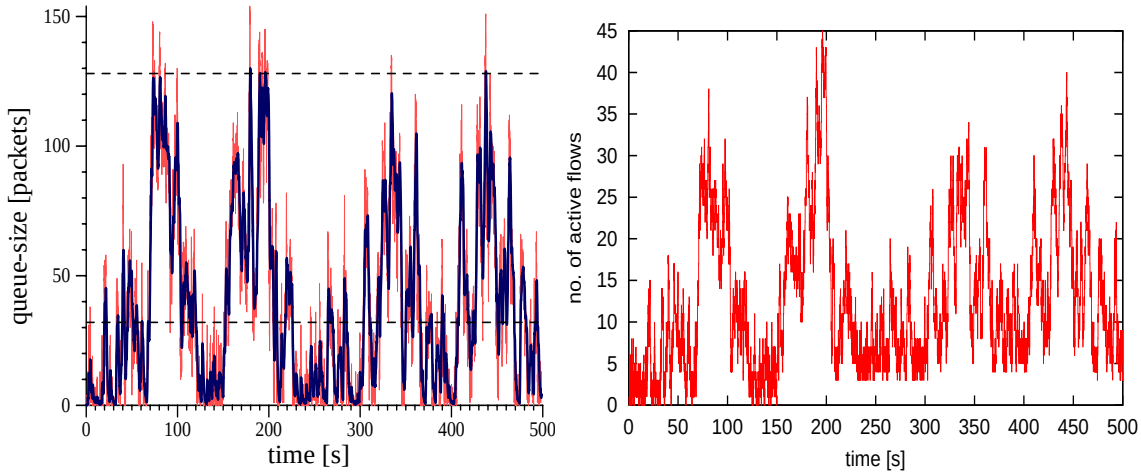


Figure 31 *a.) queue-size over time; b.) number of active flows*

Figure 31 shows a scenario with a relatively small number of client-server connections; i.e. a small number of active flows and a low degree of statistical multiplexing. For such a scenario the size of the queue mainly depends on the flow-arrival pattern and not on the properties of TCP or RED. This can be seen by comparing figures a and b. If the number

of active flows is high, the queue-size is high too and vice versa. Of course we can not expect convergence of the queue with bounded oscillation between *minth* and *maxth* similar to case of FTP-like (infinite length) flows. However, the RED model proposes reasonable parametersettings as the RED queue buffers the traffic bursts between *minth* and *maxth*.

Simulation2:

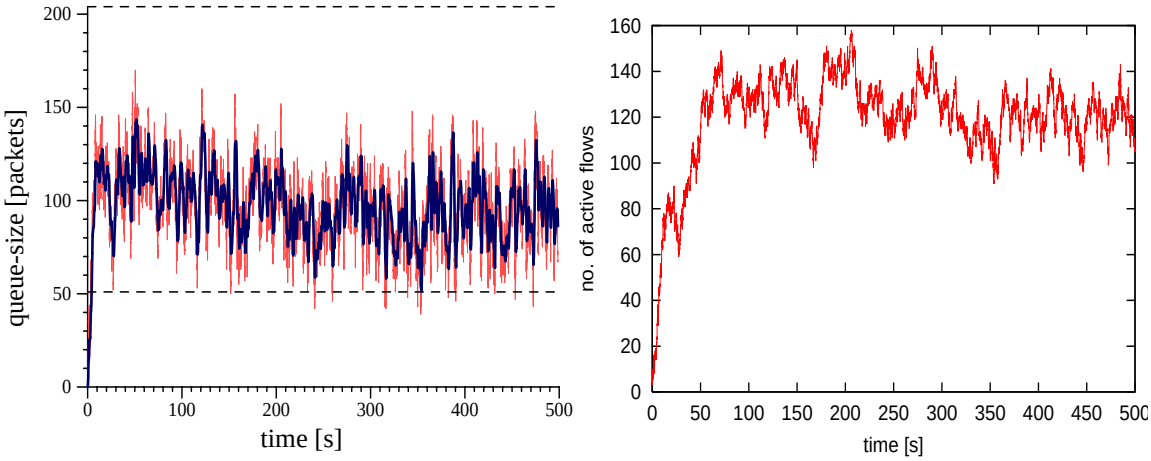


Figure 32 *a.) queue-size over time; b.) number of active flows*

Simulation 2 shows a scenario with a higher number of client-server connections and thus a higher degree of statistical multiplexing compared to simulation 1. This causes less variation in the number of active flows curve and thereby less variation in the queue-size curve. Again, the RED parameters are set properly so that the queue converges between *minth* and *maxth* with sufficiently bounded oscillation.

Simulation3:

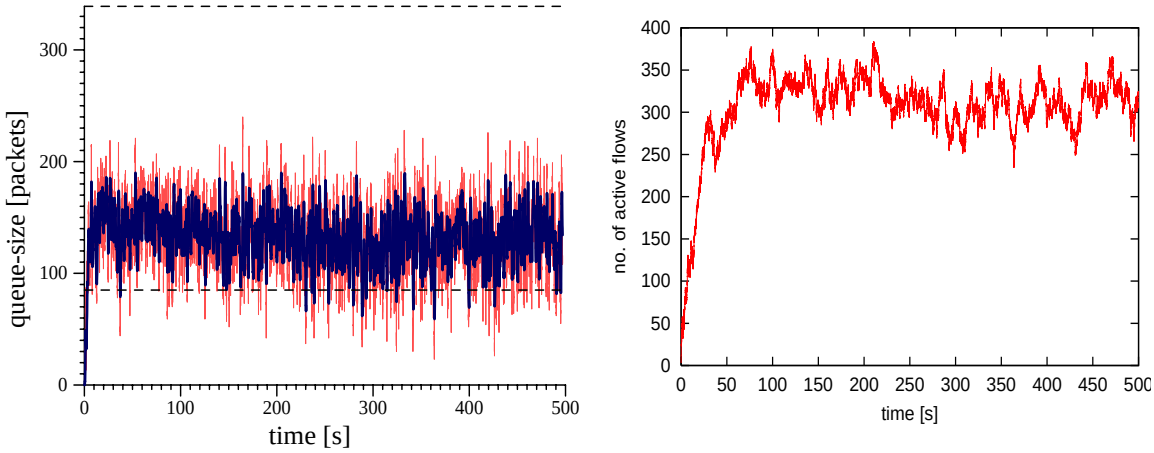


Figure 33 *a.) queue-size over time; b.) number of active flows*

Simulation4:

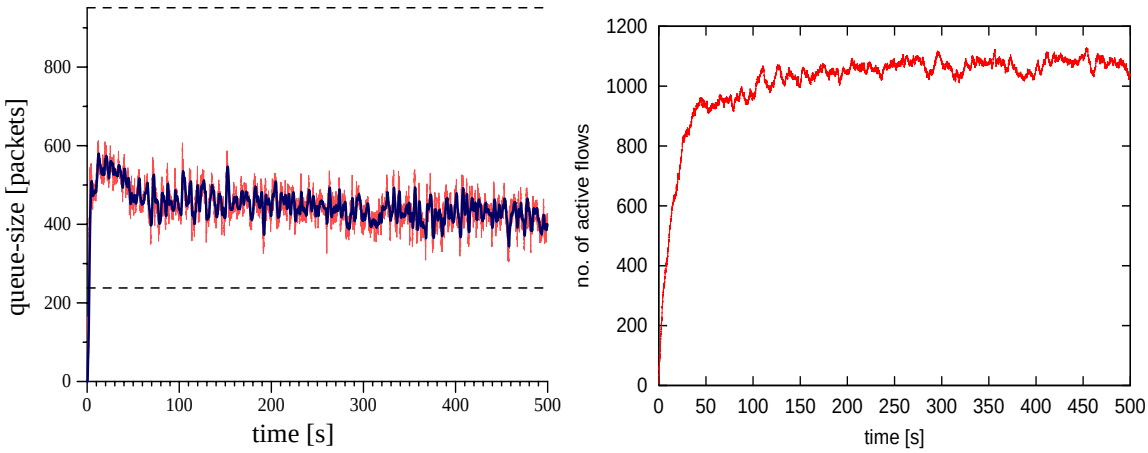


Figure 34 *a.) queue-size over time; b.) number of active flows*

Simulation5:

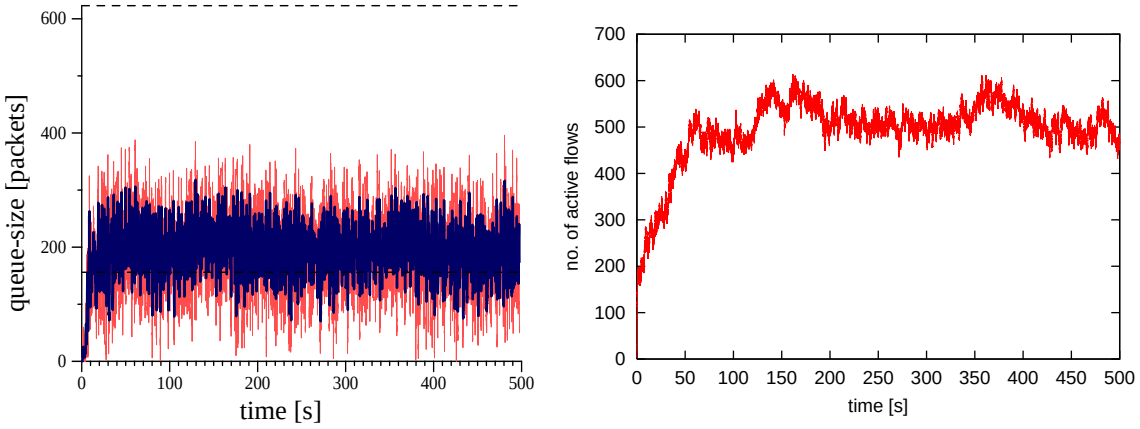


Figure 35 *a.) queue-size over time; b.) number of active flows*

Simulations 3-5 show scenarios with higher link bandwidths than scenario 1 and 2. The results are comparable. The oscillation of the queue depends on the variation in the number of active flows.

Simulations in this chapter indicate that the model for setting of the RED parameters can be applied to the case of Web-like traffic, although it has been designed assuming a constant number of flows having infinite lifetimes. However, measurements are needed to fully verify this statement.

9 RED and 2-way TCP Traffic

This chapter investigates the behavior of RED with forward and reverse TCP bulk-data traffic in steady state. Simulations use the topology of figure 4. N/2 TCP flows are started between zero and ten seconds of simulation time from hosts 1-3 to hosts 4-6, N/2 TCP flows are started between 20 and 23 seconds of simulation time in the reverse direction. For additional comments on simulation settings, not specified in this chapter, we refer to section 4.

RED parameters are generally not set exactly according to the model proposed in section 7.3.2. In case of 2way TCP traffic, packets are always mixed with acknowledgements at a router output port. This causes a shift in the packet-size distribution and - more importantly - a shift in the effect of a packet drop on the arrival rate at the router -output port. In case of 2way TCP, an ACK drop causes no substantial decrease in the arrival rate of the output port for two reasons. First, TCP is rather robust against ack-drops. An ACK drop does not cause an explicit reduction of the congestion window but merely a slight attenuation in the increase of the congestion window (during Slowstart and the congestion avoidance phase). Second, the drop of an ACK causes a decreased packet arrival-rate at the router output-port “vis-a-vis” to the output port where the ACK drop happened. E.g. if an ACK is dropped at router1, figure 4, the packet arrival rate at router2 is decreased. The decrease of the packet arrival rate in direction router2 to router1 in turn causes a decrease in the arrival rate of ACKs in direction router1 to router2. However, the length of an ACK is generally shorter than the length of a packet. Thus the decrease of the arrival rate at router1’s output port is rather small. We see, however, that the model proposed in section 7.3.2 can not be applied directly to the case of 2way TCP traffic. Thus we use the model merely as a rough indicator and perform the fine-tuning by setting the RED parameters manually in this section.

Simulation1:

C	D	N	minth	maxth	*B*	wq	$maxp^{-1}$
0.5	50	200	10	78	104	0.008	2.4

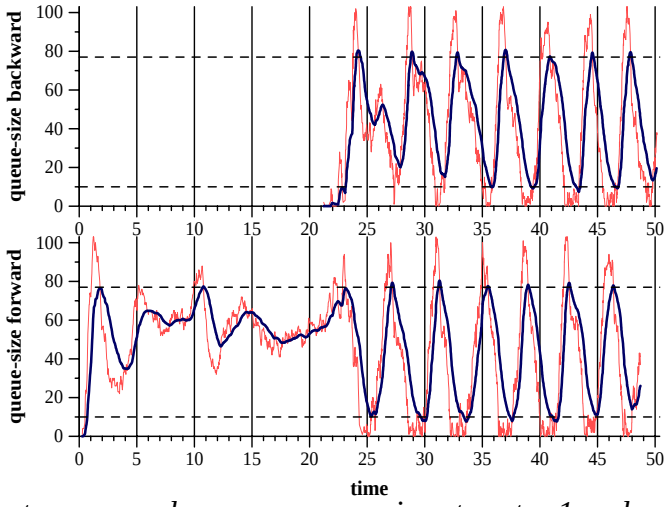


Figure 36 *Instantaneous and average queue-size at router 1 and router 2 over time*

Figure 36 shows the appearance of heavy oscillations with RED and TCP as soon as the flows in backward direction start to send. The oscillations at the backward-queue are phase-shifted by 180 degree compared to the oscillations at the forward queue.

In order to better visualize and to explain in detail the cause of oscillations with RED and two-way TCP traffic, figure 37 shows one period of oscillation in a simulation with artificially high values for *minth*, *maxth* and *B*.

Simulation2:

C	D	N	minth	maxth	*B*	wq	$maxp^{-1}$
0.5	50	100	100	505	668	0.008	5

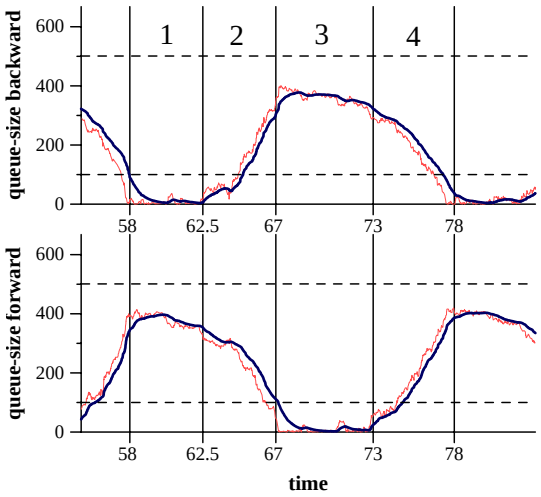


Figure 37 *Instantaneous and average queue-size at router 1 and router 2 over time*

The oscillations can be explained as interactions between queue-management algorithms and Little's law applied to window-based flow-control (the rate of the sender equals the window-size divided by RTT). One period of oscillation can be divided in four phases:

Phase1: at 58 seconds of simulation time the instantaneous queue-size at the backward queue equals zero; the average queue-size is smaller than *minth* hence the drop-probability equals zero. Instantaneous and average queue at the forward queue are close to *maxth* hence TCP flows in forward direction experience a high drop probability. As a consequence the queue-size at the forward-queue decreases between 58 and 62.5 seconds of simulation time. As the decrease of the forward queue-size causes a decrease of the round-trip-time the decrease of the sending rate of TCPs in forward direction is attenuated. Consequently, the decrease of the queue-size in forward direction is attenuated.

Phase2: at approximately 62 seconds of simulation time the backward TCP-flows have increased their rate sufficiently high to cause the backward queue to move away from zero. The drop-probability at the backward queue is still zero while the drop-probability at the forward queue is still high, hence the tendency for the backward queue to increase and the tendency of the forward queue to decrease continues. However, as shown in figure 37, the decrease of the forward queue is significantly steeper than during phase 1 as the increase of the backward queue and the decrease of the forward queue tend to cancel regarding variation of the round-trip-time. In other words, the round-trip-times stays constant during this phase.

Phase 3: at 67 seconds the backward queue has a maximum; the size of the forward queue

equals zero. Phase 3 can be explained like phase one with backward queue and forward queue exchanged.

Phase 4: at 73 seconds of simulation time the queue-size at the forward queue starts to increase causing an accelerated decay of the backward queue compared to phase 3. Phase 4 is analogous to phase 2 with backward queue and forward queue exchanged.

The initial decrease of the forward-queue due to the reverse TCP traffic can be explained similarly: when the backward queue-size is increased due to the starting TCP-flows, the RTT for the forward queue increases causing a decrease of the sending rate and thereby the size of the forward queue (see figure 36 at 23 seconds of simulation time).

Additional simulations show that higher bottleneck capacity increases the frequency of oscillation, but does not significantly decrease the amplitude. We have conducted experiments with different bottleneck-delays, different delays for hosts1-3 to router 1, different number of flows in forward and reverse direction, different start-times for the backward traffic, RED in byte and packet mode and different settings for *maxth* and the buffersize. In all these simulations the oscillations persisted. Smaller values for *minth, maxth* and the buffersize decrease the total amplitude and frequency of the oscillation but, however, can not avoid phases of buffer-overflow and empty queues.

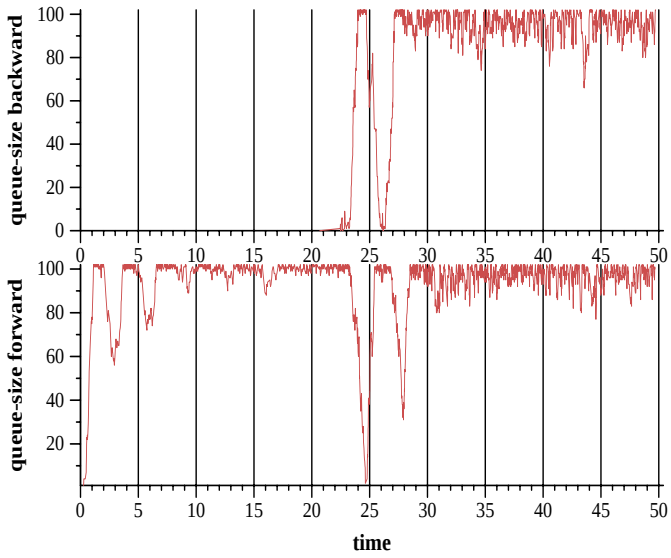


Figure 38 *Instantaneous and average queue-size at router 1 and router 2 over time*

The simulation shown in figure 36 is repeated with two drop-tail gateways instead of two

RED Gateways in figure 38. If the per TCP-flow share of the bottleneck-capacity is small, global-synchronization due to drop-tail is negligible and the queue-size stays close to the buffer-size. As a consequence, the oscillations disappear. Similarly, the oscillations disappear if TCP-Vegas [19] is used instead of TCP reno.

The problem behind these oscillations is that neither RED nor TCP-Reno does “per-se” converge to a certain buffer-utilization. If one of the two routers (like drop-tail in certain cases - see figure 38) or the end-to-end congestion control algorithm (like TCP vegas with its proactive approach to adapt the window) enforces convergence to a certain queue-size the oscillations can be avoided.

Subsequent simulations investigate the behavior of RED and 2way TCP traffic in for scenarios where the forward and backward TCP flows are coupled less strongly. In other words, instead of the simple topology in figure 4, we use a topology with several congested gateways as shown in figure 39. All routers have RED implemented in this scenario.

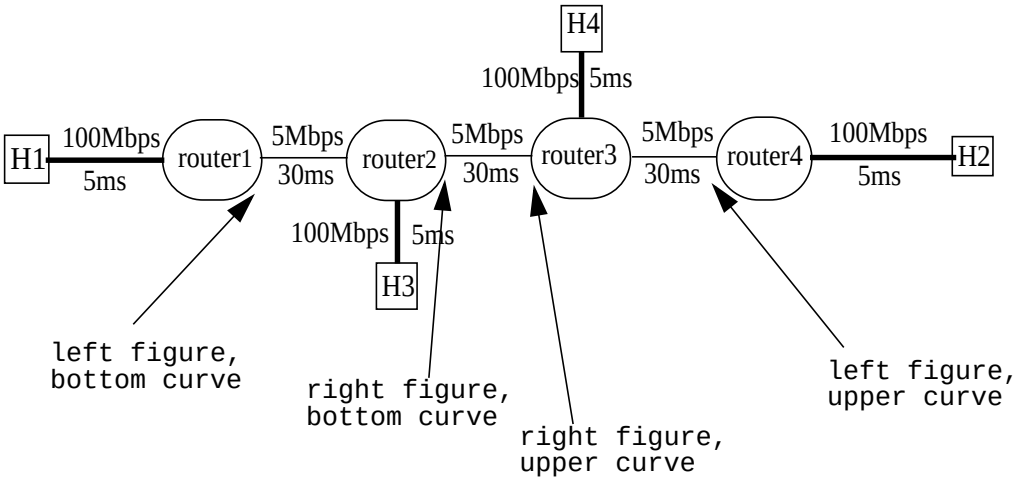


Figure 39 *Topology with several congested gateways*

15 TCP flows are started from H1 to H2 and another 15 TCP flows from H2 to H1. Additionally a varying number of cross-traffic TCP flows is started from H3 to H4. The higher the number of cross-traffic TCP flows, the weaker the coupling between the forward and backward TCP flows between H1 and H2. Thus we can expect a decrease in the amplitude of the oscillation in case of an increase in the number of cross-traffic flows. There exist 4 output ports potentially subject to congestion. First the output port from router1 to router2,

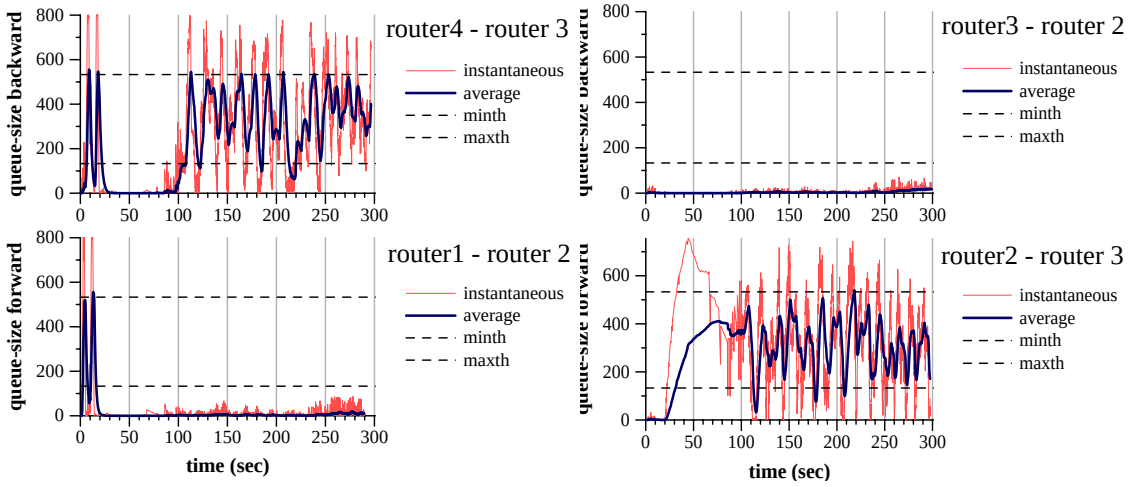
displayed in the bottom curve of the left part-figure of subsequent figures. Second, the output port from router4 to router3, displayed in the upper curve of the left part-figure of subsequent figures. Third, the output port from router2 to router3, displayed in the bottom curve of the right part-figure of subsequent figures. Finally, the output port from router3 to router2, displayed in the upper curve of the right part-figure of subsequent figures.

Default RED parameters for subsequent simulations: minth = 133, maxth = 533, B = 800, mean packet-size = 520, 1/maxp = 79, wq = 0.00045.

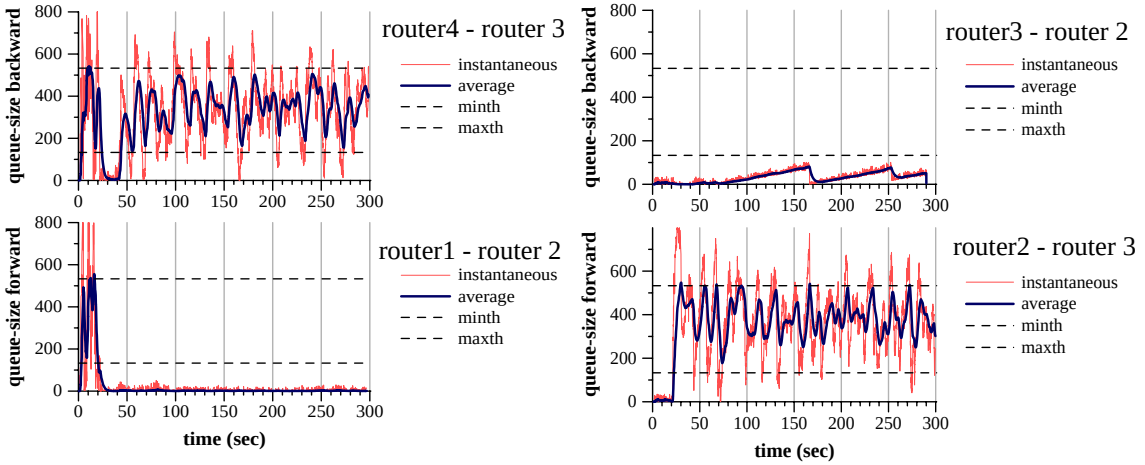
Differences from default parameter settings:

Simulation	1	2	3	4	5
no. cross-TCPs	1	3	5	20	100
1/maxp r4-r3	67	75	85	105	125
1/maxp r2-r3	10	10	10	7	2.5
1/maxp r3-r2	110	100	90	50	20

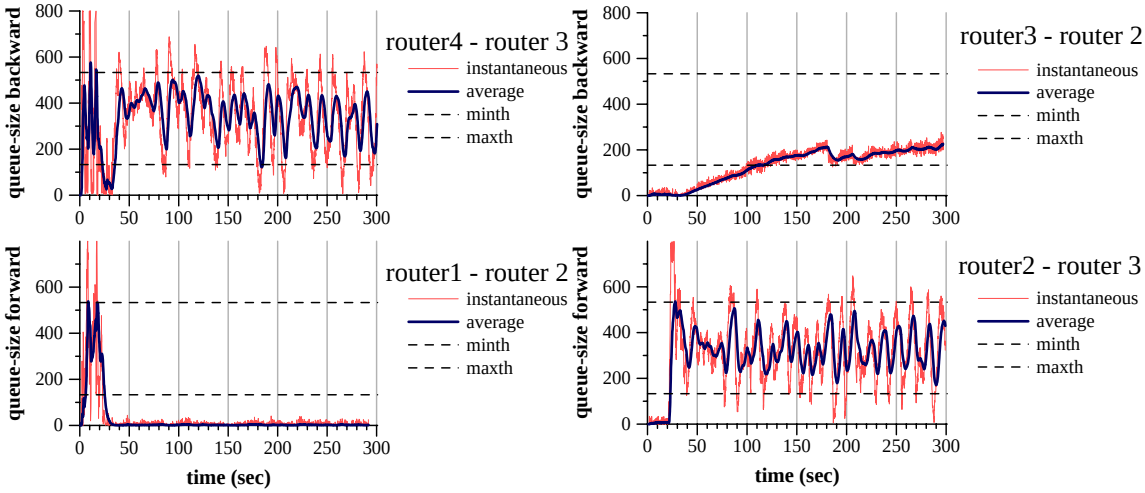
Simulation 1



Simulation 2



Simulation 3



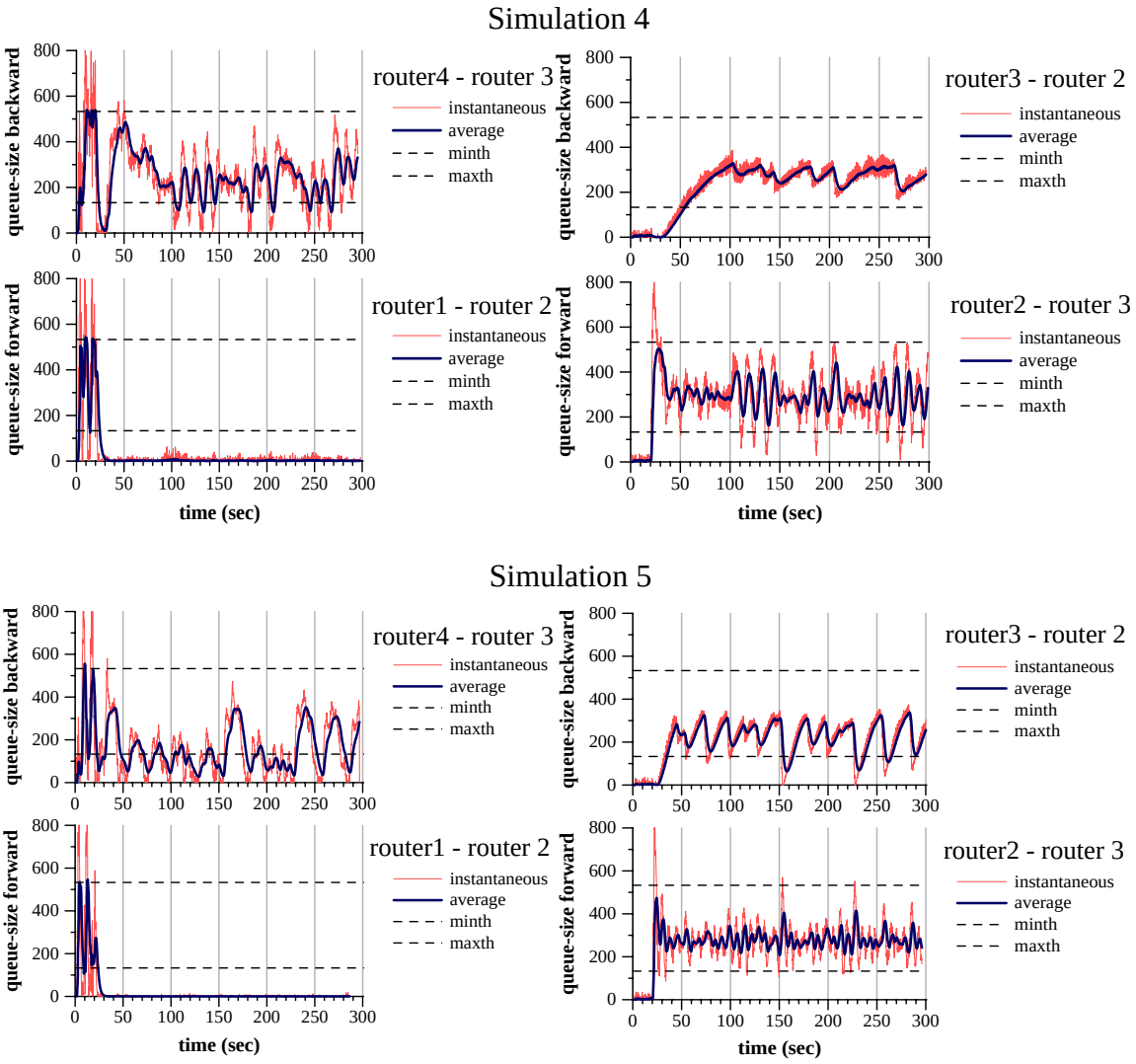


Figure 40 *left part: queue size over time at output-port router 1 to router 2 (bottom); router 4 to router 3 (top). Right part: queue size over time at output-port router 2 to router 3 (bottom); router 3 to router 2 (top)*

As soon as the cross-TCP flows start creating traffic on the link from router2 to router3, congestion is moved from the queue at router1 to the queue at router2. In the backward direction, traffic experiences a backlog at router 4. However, the higher the number of cross-traffic TCP flows, the higher the queue-size at router3 due to an increased bandwidth consumption by acknowledgements of crosstraffic TCP flows. An increased queue size at router3 causes the queue at router4 to decrease.

As expected, the higher the number of crosstraffic TCP flows, the weaker the coupling between forward and backward TCP flows. Consequently, the amplitude of oscillation decreases in case the number of cross-traffic TCP flows increases. In general wide-area Internet scenarios, we may assume that forward and backward TCP flows are rather loosely coupled, thus oscillations of RED with 2way TCP flows may not be a big problem. However, in special environments like Intranets or LAN-to-LAN interconnection via Satellites, forward and reverse TCP flows may be tightly coupled.

No matter whether the oscillation persists or not for loosely coupled forward and reverse TCP flows, it is an eligible requirement for interacting control mechanisms to show desirable behavior in simple scenarios. This requirement is not fulfilled in case of 2way TCP and RED. Additionally, these interactions between RED and 2way TCP flows make the system hard to predict.

10 What if RED parameters are not set correctly?

We perform simulations using the topology of figure 4 to demonstrate the sample queue behavior in case RED parameters are not set properly. Routers have WRED implemented. At each access link to router1 and router2 a token bucket marker is installed, marking arriving packets as in-profile or out-of-profile. Performance metrics are link utilization and the drop-probability of in-profile and out-of-profile packets. The duration of the simulations equals 100 seconds; measurements of performance metrics are performed between 50 and 100 seconds of simulation time (except for simulation 6, measuring between 150 and 300 seconds of simulation time). The network is provisioned, so that the sum of the token rates (the average bandwidth for in-profile packets) equals 0.9 times the link capacity. Bucket sizes are set to the bandwidth*RTT, respectively bandwidth*delay product of flows. The minimum queue threshold for in packets is set equal to the maximum queue threshold for out packets. Thus service differentiation between in-profile and out-of-profile packets should be maximally.

Parameter settings for simulations:

Simulation	1	2	3	4	5	6
C	20	20	10	10	10	2
D	100ms	100ms	100ms	100ms	100ms	100ms
N	10	10	500	500	500	20
$minth_{in}$	67	10	67	10	10	30
$maxth_{in}$	268	40	268	40	40	530
$maxp_{in}$	20	20	1	1	12	200
$minth_{out}$	268	40	268	40	50	543
$maxth_{out}$	400	80	400	80	80	1000
$maxp_{out}$	1	1	200	200	12	200
B	600	120	600	120	120	1800
wq	0.0009	0.0006	0.0009	0.0009	0.0014	0.00063
token rate	6Mbps	6Mbps	3Mbps	3Mbps	3Mbps	0.6Mbps
bucket size	80KB	80KB	80KB	80KB	80KB	20KB

Simulation results:

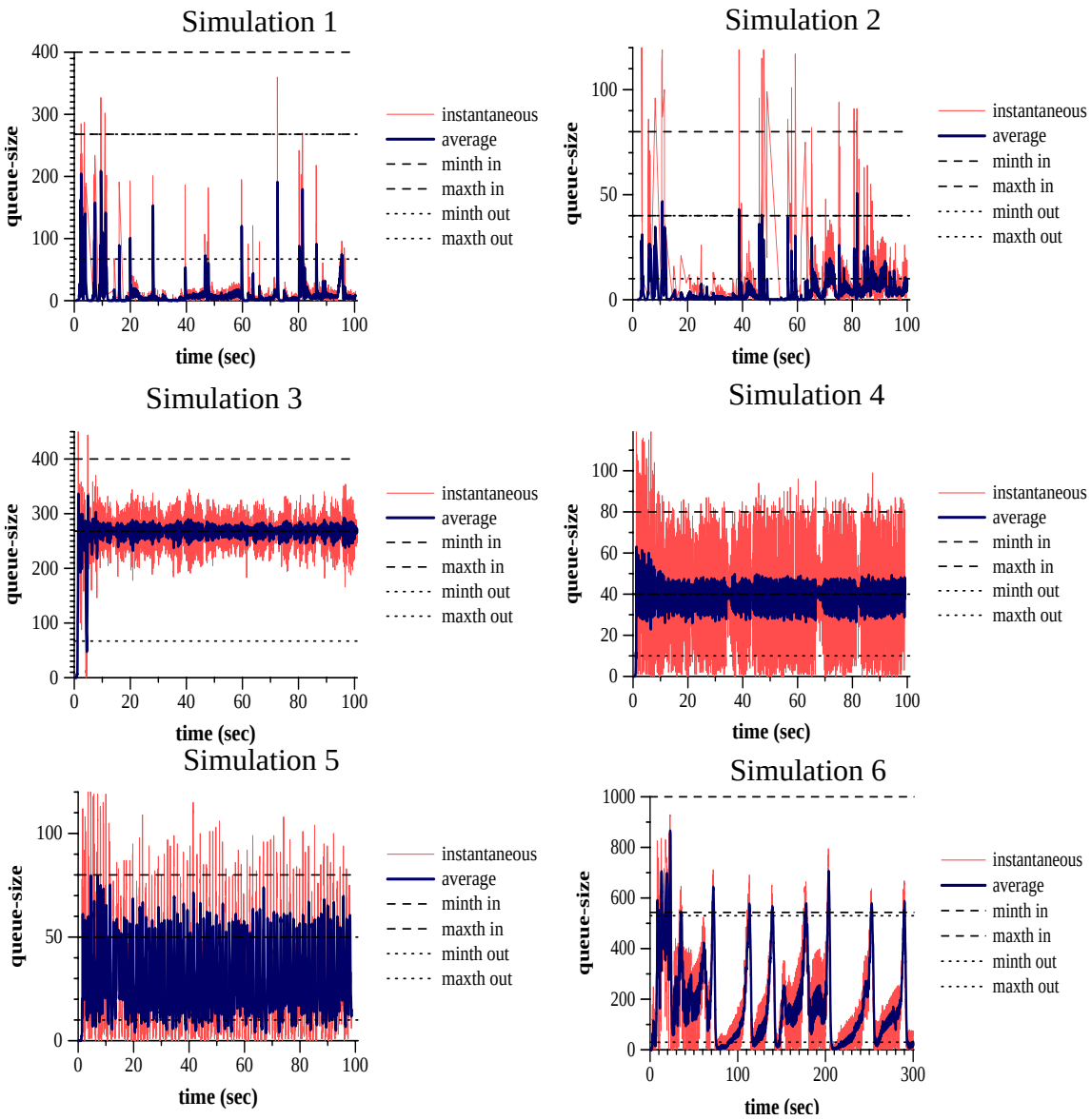


Figure 41 *a-f: queue size [packets] over time [seconds]*

Simulation	1	2	3	4	5	6
link utilization	0.76	0.63	1	0.97	0.97	0.85
$pdrop_{in}$	0	0.0008	0.05	0.08	0.008	0.000037
$pdrop_{out}$	0.02	0.11	0.43	0.41	0.33	0.05

Simulations 1 and 2 show scenarios with $maxp_{out}$ set too high considering the number of flows. This results into poor link utilization, especially in case of small minth values (simulation 2). Other simulations have shown that link utilization is even worse in case the mismatch between $maxp_{out}$ and the per-flow bandwidth*RTT product is stronger.

Simulations 3 and 4 show scenarios with $maxp_{out}$ set too small and $maxp_{in}$ set too high for the given number of flows. Setting RED parameters like done in simulation 3 and 4 causes significant drop probability for in-profile packets (5-8%) although the network is underprovisioned (90% load by in-profile traffic). Note that the drop probability for in-packets would be worse in case of $maxth_{out} > minth_{in}$.

Simulation5 shows a scenario with maxp parameters set properly, but queue-thresholds set too small for the given bandwidth*RTT product. This causes oscillation of the queue, link-underutilization and a considerable drop probability for in-profile packets.

While simulations 1-5 have one-way TCP traffic, simulation 6 employs 2way TCP traffic. The queue-size graph shows that the oscillations of RED with 2way TCP investigated in section 9, persist also for diff-serv networks. Link utilization is sub-optimal in this scenario.

11 Measurements with RED and TCP traffic

This section aims at proofing that the simulation results obtained in previous sections can be applied to the real network situation. In a first set of measurements, RED with 2way TCP traffic is investigated. The second set of measurements investigates whether the parameterization proposed by our model for RED with one-way TCP is correct for real networks. Due to restricted possibilities in varying parameters in case of lab-measurements on a real network, we can not expect an evaluation with parameters (bandwidth, delay, number of flows) varied over a range, as wide as it is possible with simulations.

In order to enable comfortable measurements with minimum errors a Generic Measurement Environment (GME) has been developed (see appendix 2). This software is written in TCL, uses standard UNIX commands for process-generation on remote-hosts and standard tools like ttcp and tcpdump to generate traffic and retrieve results. In section 11.2, we describe the configuration of this software to perform one-way delay measurements with a GPS clock. The one-way delay of packets directly corresponds to the queueing-delay at the congested router output-port, i.e. the instantaneous RED queue-size.

11.1 Network Configuration

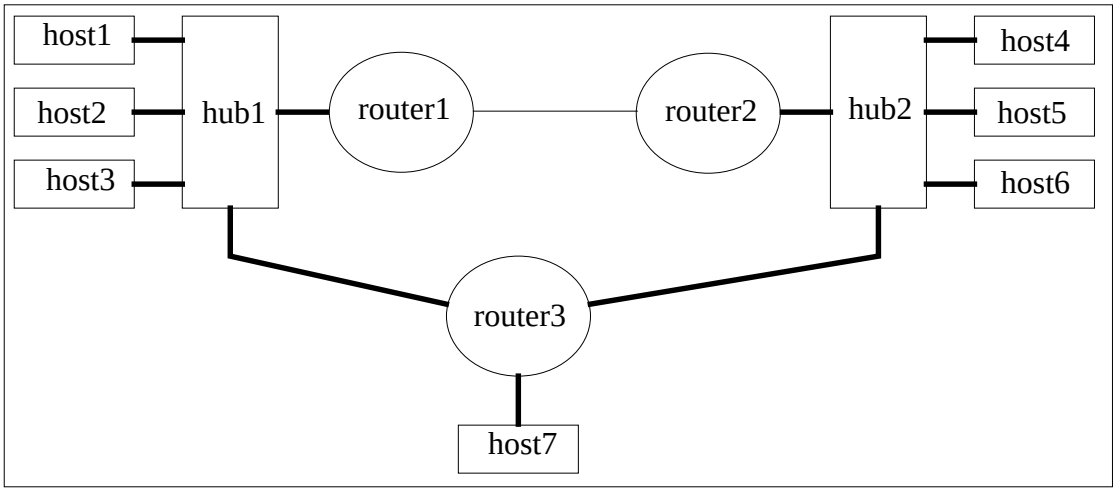


Figure 42 *Measured network*

Hardware:
- hosts1-6: PC Pentium3, 350MHz, 128MB RAM
- host7: PC Pentium MMX, 200MHz
- hub1,2: Bay Networks 10/100 Mbps Ethernet Hub
- router1: Cisco 3640, IOS 12.0 T
- router2: Cisco 2611, IOS 12.0 T

- router3: Cisco 2500, IOS 12.0 T

Links:
- hosts 1-6 to Hubs:100Mbps Ethernet
- host7 to router3: 10Mbps Ethernet
- router1 to hub1: 10Mbps Ethernet
- router2 to hub2: 10Mbps Ethernet
- router3 to hubs: 10Mbps Ethernet
- router1 to router2: Serial link running HDLC, bandwidth 500kbps

The propagation delay of all links can be assumed equal to zero (all equipment is located in one lab). In earlier measurements the bottleneck capacity has been set to 2Mbps, causing a significant amount of collisions (showing TCP's burstiness) and thereby falsifying the measurement results. Consequently the bottleneck capacity has been set to 0.5Mbps.

11.2 GME Usage for One-way Delay Measurement

In this section we show a sample application of GME; one-way delay measurements in the testbed specified in section 11.1.
The ttcp flows are created (as explained in Appendix2)from host 1 to host 4, host 2 to host 5, host 4 to host 2 and host 5 to host 1. Host 3 and host 6 are time-synchronized by a GPS clock. Additionally host3 and host6 are running tcpdump. Two one-way delay measurements are performed:
- ttcp flow A from host 1 to host 4 is tcpdumped at host 3 and host 4
- ttcp flow B from host 4 to host 2 is tcpdumped at host 4 and host 3

Tcpdump stores the time a packet is received and the TCP sequence number in a result-file. By relating the sequence numbers of ttcp flow A in the result-files at host3 and host4 and subtracting the arrival time of a TCP segment at host 3 from the arrival time of the TCP segment at host 4 the one-way delay can be measured. This equally valid for ttcp flow B. In order to avoid falsification of the delay measurements, lost packet have to be filtered out from the dump files.

Knowing the one-way delay of packets, we can easily calculate the instantaneous queue-size at the RED gateway in order to create a queue-size over time plot:

queue-size = ((delay - prop. delay) * bottleneck bandwidth) / MTU size

This configuration minimizes measurement errors as there is no traffic created during the execution of the measurement but the flows to be measured. Additionally, tcp-dumping is completely separated from the task of traffic-generation as this happens on different hosts. Hence the GME does not at all influence the measured system.

11.3 Measurements with two-way TCP traffic

Common parameters for all measurements:

Bottleneck bandwidth 500kbps; switching delay in routers 2ms, propagation delay 0ms.

All TCP flows start at the beginning of the measurement. The MTU size (configured at the bottleneck link) equals 512 bytes.

Measurement1:

Traffic: 3 tcp flows host1 -> host4, host2 -> host5, host5 -> host1, host4 -> host2 (12 in total)

Red parameters: minth = 7, maxth = 40, buffersize = 60, wq = 0.015, maxp = 1/10

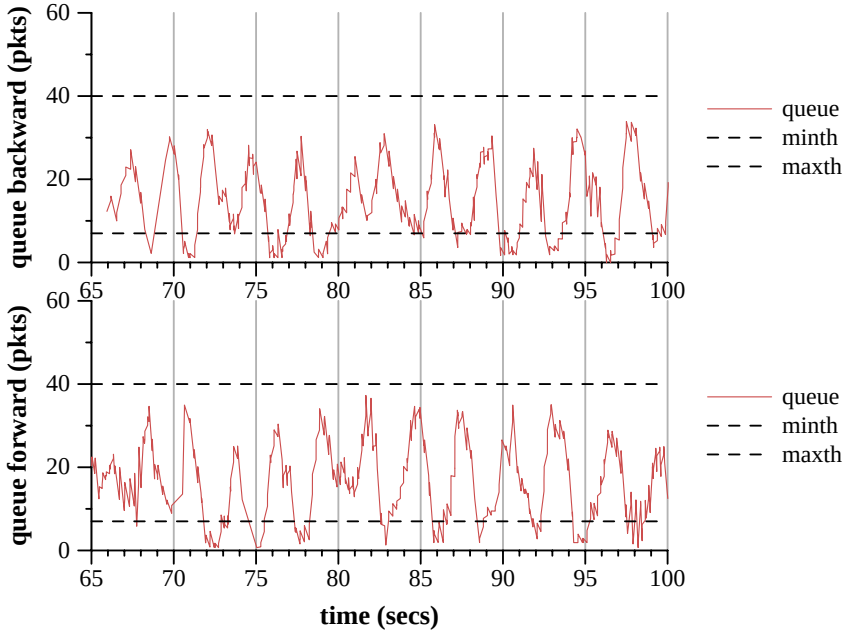


Figure 43 *queue size in forward and backward direction*

Figure 43 shows that the queue-size oscillates heavily between zero and maxth. The oscillations at the two RED-gateways are phase-shifted by exactly 180 degree. As shown by figure 43, the measurement confirms the simulation results.

Measurement2:

Traffic: 7 tcp flows host1 -> host4, host2 -> host5, host5 -> host1, host4 -> host2 (28 in total)

Red parameters: minth = 10, maxth = 60, buffersize = 100, wq = 0.002, maxp = 1/10

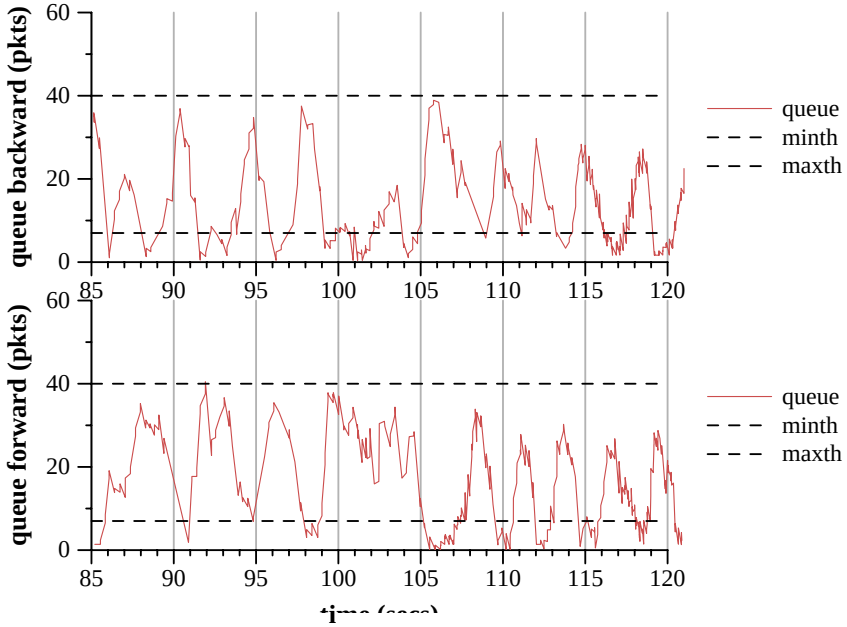


Figure 44 *queue size in forward and backward direction*

Measurements 1,2 show that frequency and amplitude of oscillations are independent of the number of TCP flows.

11.4 Measurements with one-way TCP traffic

Common parameters for all measurements:

Bottleneck bandwidth 500kbps; switching delay in routers 2ms, propagation delay 0ms.

All TCP flows start at the beginning of the measurement. The MTU size (configured at the bottleneck link) equals 512 bytes.

Measurement1:

Traffic: 3 tcp flows host1 -> host4, host2 -> host5 (6 in total)

Red parameters: minth = 10, maxth = 70, buffersize = 100, wq = 0.125, maxp = 1/40

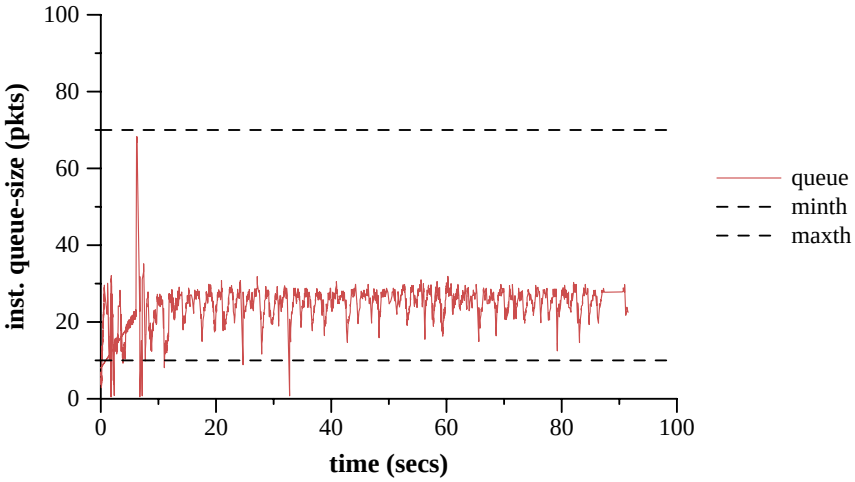


Figure 45 *queue size in forward and backward direction*

Measurement2:

Traffic: 6 tcp flows host1 -> host4, host2 -> host5 (12 in total)

Red parameters: minth = 10, maxth = 70, buffersize = 100, wq = 0.015, maxp = 1/10

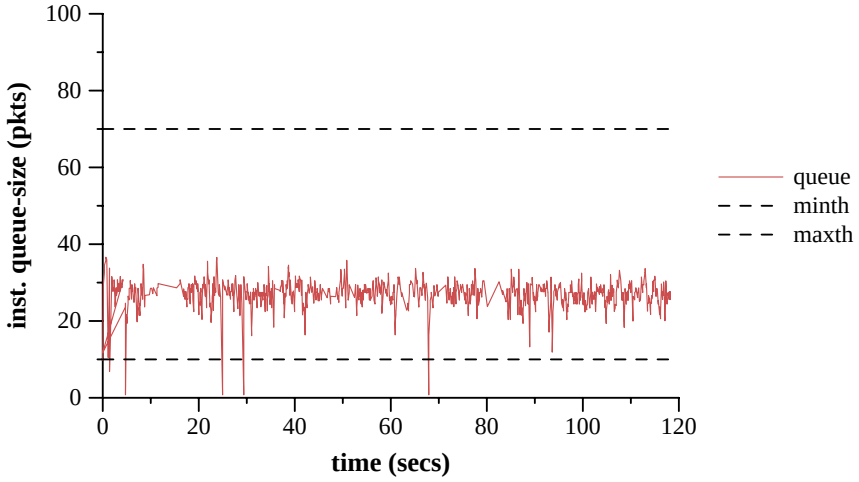


Figure 46 *queue size in forward and backward direction*

Measurement3:

Traffic: 14 tcp flows host1 -> host4, host2 -> host5 (28 in total)

Red parameters: minth 10, maxth 70, buffersize 100, wq 0.125, maxp 1/6

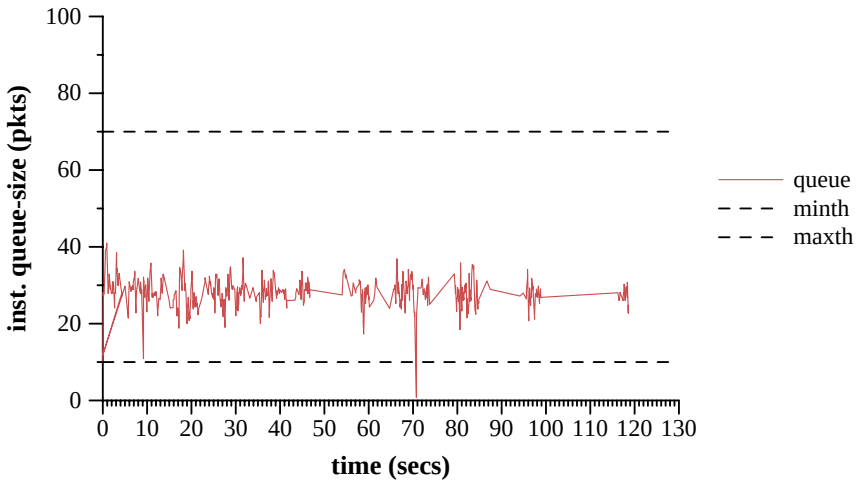


Figure 47 *queue size in forward and backward direction*

Measurement4:

Traffic: 28 tcp flows host1 -> host4, host2 -> host5 (48 in total)

Red parameters: minth = 10, maxth = 70, buffersize = 100, wq = 0.125, maxp = 1/4

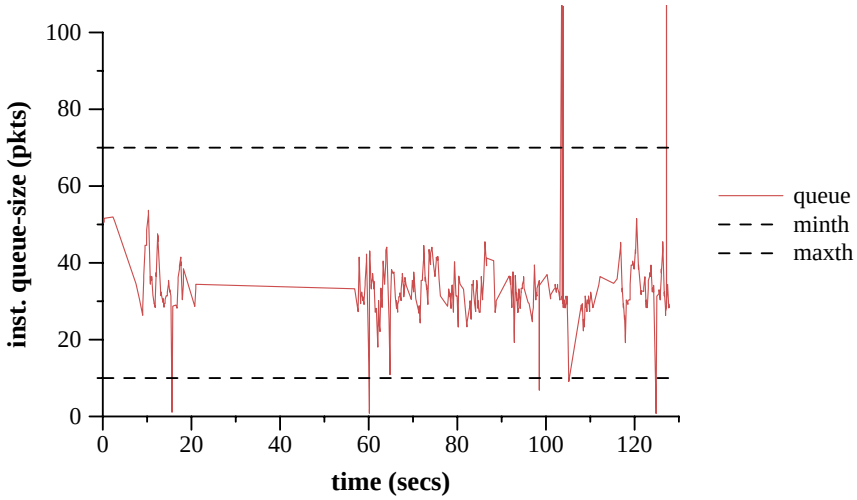


Figure 48 *queue size in forward and backward direction*

Measurements in this chapter show that the maxp, wq, minth and maxth values proposed by the model (see section 6) make the queue-size converge between minth and maxth independently of the number of flows. Hence the simulation results for RED parameterization in case of oneway TCP traffic are confirmed for the investigated scenario.

In figures 6 and 7 many packet losses and retransmissions happen. For a dropped packet one way delay can not be measured, causing time-periods of insufficient number of samples (e.g. figure 7 between 20 and 60 seconds).

12 Summary and Conclusions

This report aims at systematically investigating the stability criteria of RED and deriving quantitative guidelines for the setting of RED parameters. As the stability of the system does not solely depend on the RED parameters but also on the behavior of end-to-end congestion control and the time-scales of the arrival-process, we focus on bulk-data TCP traffic for derivation of the models.

For the assumed traffic-pattern, stability (i.e. bounded oscillation of the queue-size between minth and maxth) can be achieved assuming the scenario-parameters (bottleneck bandwidth, number of flows, round-trip-time distribution) are known. We show that the queue oscillates heavily if the difference between *minth* and maxth is set too small and that the queue converges to *maxth* or oscillates around *minth* if *maxp* is not set correctly. Subsequently, based on a model for the *wq* parameter [11], we analytically derive a model how to set *maxp* as function of the bottleneck bandwidth, number of flows, and the round-trip-time distribution. This model is suitable for heterogeneous RTTs. As a next step, lacking an analytical model to obtain quantitative values for *maxth-minth*, we perform simulations over the maximum possible parameter range (bandwidth, number of flows) the NS simulator allows and determine the proper settings for *maxth-minth* by trial and error. These simulations result in a cloud of points which is approximated by a linear least square fit, giving a quantitative formula how to set *maxth-minth* as a function of bottleneck bandwidth, number of flows and the RTT. Subsequently, we extend our model for *maxth-minth* to the heterogeneous RTT case and explain why the setting of *maxp* and *maxth-minth* exhibits interdependencies. These interdependencies demand for combining the two models into a system of non linear equations. Finally, this system of equations is solved numerically and we may obtain the total model how to set the RED parameters as a function of bottleneck bandwidth, number of flows and round-trip-time distribution. The model is evaluated by measurements and simulations with bulk-data and Web-like TCP traffic.

The model shows that the buffer-requirements of RED are significant in case of bulk-data TCP traffic making the usability of RED in networks with high bandwidth*delay products questionable.

Additionally, we examine the reasons for the existence of heavy oscillations with RED and two-way bulk-data TCP-Reno traffic. We find that these oscillations are inevitable in case of tight coupling between forward and reverse TCP traffic, caused by interactions of queue-management and TCP Reno's window-based congestion-control algorithm and can

be avoided if the transport protocol's congestion control algorithm or the queue-management algorithm converge to a certain buffer-utilization. Finally, we investigate the decrease in service differentiation and link utilization by WRED in case parameters are not set properly and in case of 2way TCP traffic.

All in all we that it is questionable whether appropriate setting of RED parameters is possible in a realistic Internet environment. Although the quantitative models proposed in this thesis make an important step ahead, it is questionable whether ISPs will be able to retrieve sufficiently accurate information about the RTT of flows and the number of flows traversing a router. Additionally, the models proposed in this paper, although definitely sophisticated to derive, can only be applied to simple topologies with one way TCP traffic. For sophisticated topologies with 2way TCP and cross-traffic, possibly consisting of flows not congestion controlled by TCP an extension of the model is required. The complexity of the model for TCP with RED indicates that future queue-management algorithms should employ a smaller parameter-set than RED does.

13 References

[1] S. Floyd, V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, August 1993

[2] B. Braden, D. Clark et al., “Recommendations on Queue Management and Congestion Avoidance in the Internet”, RFC 2309, April 1998

[3] W. Feng, D. Kandlur, D. Saha, K.G. Shin, “ A self-configuring RED gateway”, Proc. of IEEE Infocom 1998

[4] W. Feng et al., “Techniques for eliminating Packet Loss in congested TCP/IP Networks”, University of Michigan CSE-TR-349-97, November 1997

[5] T.J. Ott, T.V. Lakshman, L.H. Wong, “SRED: Stabilized RED”, Proc. of IEEE Infocom 1998

[6] T. Ziegler, U. Hofmann, S. Fdida, "RED+ Gateways for detection and discrimination of unresponsive flows", Technical Report, December1998, http://www.newmedia.at/~tziegler/papers.html

[7] S. Floyd, K. Fall, K. Tieu, “Estimating arrival rates from the RED packet drop history”, March 1998, unpublished, http://www.aciri.org/floyd/end2end-paper.html

[8] K.K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC2491, January 1999

[9] Cisco Web pages, http://www.cisco.com/warp/public/732/netflow/qos_ds.html

[10] D. Clark, “Explicit Allocation of Best Effort Packet Delivery Service”, http://www.ietf.org/html.charters/diffserv-charter.html

[11] V. Firoiu, M. Borden, “ A Study of active Queue Management for Congestion Control”, Proc. of IEEE Infocom, Tel Aviv, 2000

[12] V. Jacobson, K. Nichols, K. Poduri, “RED in a different Light”, Draft Technical Report, Cisco Systems, Sept. 1999

[13] S. Floyd, “Discussions on setting RED parameters”, http://www.aciri.org/floyd/red.html , Nov. 1997

[14] C. Villamizar, C. Shong, “ High performance TCP in ANSNET”, Computer Communication Review, V.24. N.5, October 1994, pp. 45-60

[15] J. Padhye et al.,”Modeling TCP Troughput: A simple Model and its empirical Validation”, Proceedings of ACM SIGCOMM, August 1998

[16] NS Simulator Homepage, http://www-mash.cs.berkeley.edu/ns/

[17] Maple home page, http://www.mit.edu/afs/athena.mit.edu/software/maple/www/home.html

[18] B. Mah. ‘‘An Empirical Model of HTTP Network Traffic’’, Proceedings of INFOCOM ’97, Kobe, Japan, April 1997

[19] L. S. Brakmo, L. L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet”. IEEE Journal on Selected Areas in Communications (JSAC), 13(8):1465-1480, October 1995

[20] M. May et al, “Reasons not to deploy RED”, Technical Report, Feb. 1999

14 Appendix, Generic Measurement Environment (GME)

The GME enables flexible and comfortable performance evaluation by measurement in the Internet. The main-program (a TCL script) is running at the GME-Server, controlling the entire measurement-process. This script in turn creates and distributes scripts for flow-generation and traffic monitoring, starts flows and traffic-monitors on remote hosts, and finally gathers the measurement results at the GME-server after the measurement has terminated.

For generation of flows the "ttcp" tool is used:

- first a ttcp server (receiver) has to be started at the receiving host with "ttcp -rs -p x". The option "-rs" tells ttcp to operate as data-receiver tcp and to discard all incoming data; "-p x" specifies the port-number to listen at.
- then the ttcp sender is started with "ttcp -ts -p x receiving-host". The "-ts" option tells ttcp the operate as transmitter and to source arbitrary data into the net to the host running the ttcp receiver with port-number "x". The ttcp sender is aware of several other TCP-specific options and additionally allows creation of UDP flows.

Traffic monitoring is performed with tcpdump. Tcpdump switches a host's ethernet-interface into promiscuous mode and dumps header-information of all packets broadcasted at the Ethernet Segment matching a filter condition. For instance, in order to filter all packets of one TCP flow from host A to host B with source port-number 4711, the call to tcpdump is as follows: "tcpdump src host A and dst host B and proto TCP and src port 4711"

For remote procedure calls and file-copying the standard Unix-commands rsh and rcp are used. Note that for proper operation of the rsh and rcp commands ".rhosts files" are required at remote hosts enabling access from the GME-server.

The main TCL script running at the GME-server can be outlined as follows:

1. create traffic-files, send traffic-files to hosts: there are two types of traffic-files: receiver-files: shell scripts starting ttcp receivers and sender-files starting ttcp senders. Among a sender-file on a host A and a receiver file on a host be exists a one-to-one relation determining the traffic-flows: for each ttcp-receiver process on host B listening on port C, a ttcp sender is created (with adjustable options) sending to host B, port C. In the current version of GME, all ttcp senders in one sender-file start immediately. However, in future versions the individual ttcp sender processes can be scheduled at arbitrary points time (this would be facilitated by running TCL not only at the CME-Server but also at the remote hosts running the traffic-files).

After creation of the sender- and receiver-files for host A and B the sender file is copied to host A; the receiver file is copied to host B with the Unix command "rcp".

2. create tcpdump-files, send tcpdump files to hosts. Tcpdump files are shell scripts con-

taining one line of code, starting tcpdump with a filter-function (see above).

3. Start tcpdump files: the tcpdump-files are started with the "rsh" command as background processes on remote hosts.

4. start ttcp receivers: the ttcp receiver-files are started with the "rsh" command as background processes. Note that it is important to start the ttcp receivers earlier than the ttcp senders as ttcp receivers and senders have a client-server relationship.

5. start ttcp senders: the ttcp sender-files are started with the "rsh" command as background processes. As the remote hosts are not time-synchronized it may happen that two sender-files at two distinct remote hosts are not started at the same point in time. Assume that the delay from the CME-server to a host A is longer than the delay between CME-server and a host B or, assume that some rsh-packets are lost on the way to host A and all rsh.packets to host B arrive. In both cases the sender-file at host B (and thereby the ttcp flows at host B) will be started before the sender file at host A. It is the responsibility of the engineer to estimate whether the connectivity between CME-caller and the remote hosts is sufficient to satisfy the time-accuracy required for a specific measurement. For future versions of the tool an intelligent agent would be desirable, detecting the delayed start of ttcp flows and terminating the measurement in case accuracy requirements are not met.

6. wait for the estimated duration of the measurements

7. gather ttcp results: each ttcp-result contains the measurement results for a single tcp (or udp) flow. All ttcp-result files are copied from the remote hosts to the CME-sever by rcp.

8. gather tcpdump results. TCPdump result files are copied from the remote hosts to the CME-sever by rcp.