A DISCRETE-TIME MODEL OF TCP WITH ACTIVE QUEUE MANAGEMENT

by

Hui Zhang
B.Sc., Institution of Information and Communication,
Dong Hua University, P. R. China, 2001

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

In the School
of Engineering Science

SIMON FRASER UNIVERSITY

August 2004

APPROVAL

Name:	**Hui Zhang**
Degree:	**Master of Applied Science**
Title of Thesis:	**A Discrete-Time Model of TCP with Active Queue Management**

Examining Committee:

Chair: **Dr. Glenn H. Chapman**
Professor, School of Engineering Science

____

Dr. Ljiljana Trajković
Senior Supervisor
Professor, School of Engineering Science

____

Dr. Uwe Gläesser
Supervisor
Associate Professor, School of Computing Science

____

Dr. Qianping Gu
Examiner
Professor, School of Computing Science

Date Defended/Approved: ______________________________

ABSTRACT

The interaction between Transmission Control Protocol (TCP) and Active Queue Management (AQM) can be captured using dynamical models. The communication network is viewed as a discrete-time feedback control system, where TCP adjusts its window size based on packet loss detection during the previous Round Trip Time (RTT) interval.

In this research, a discrete-time dynamical model is proposed for the interaction between TCP and Random Early Detection (RED), one of the most widely used AQM schemes in today's Internet. The model, constructed using an iterative map, captures the detailed dynamical behaviour of TCP/RED, including the slow start and fast retransmit phases, as well as the timeout event in TCP. Model performance for various RED parameters is evaluated using the ns-2 network simulator. The results are then compared with other existing models. Based on evaluations and comparisons of our results, we can conclude that the proposed model captures the dynamical behaviour of the TCP/RED system. In addition, the model provides reasonable predictions for system parameters such as RTT, packet sending rate, and packet drop rate. An extension of the proposed model is also presented, which is a dynamical model of TCP with Adaptive RED (ARED), a modified AQM algorithm based on RED.

ACKNOWLEDGEMENTS

I would like to thank my senior supervisor, Professor Lijljana Trajković, for her support, guidance, and knowledge. I greatly appreciate that she gave me this precious chance to continue my studies and become a member of the Communication Networks Lab. In addition, I thank her for encouragement, great help and sincere care.

At the same time, I would like to acknowledge the supports of all of the members in our lab, especially Judy Liu, Kenny Shao, Nikola Cackov, Vladimir Vukadinović, and Wan Zeng Gang.

My special thanks to Dr. Qianping Gu and Dr. Uwe Glaesser for their valuable comments and for serving on my supervisory committee.

TABLE OF CONTENTS

Approval ii

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures vii

List of Tables ix

1 Chapter One: Motivation 1

2 Chapter Two: Background 3

2.1 TCP algorithm 3

2.1.1 TCP congestion control 3

2.1.2 TCP implementations 5

2.2 RED algorithm and evaluation of its parameters 7

2.2.1 Active Queue Management 7

2.2.2 RED algorithms 8

2.2.3 Evaluation of RED parameters 11

2.2.4 Recent studies on AQM 13

2.3 Simulation tool: ns-2 14

3 Chapter Three: A survey of existing models 15

3.1 Overview of modeling TCP Reno 16

3.2 Overview of modeling TCP Reno with RED 20

4 Chapter Four: A discrete approach for modeling TCP RENO with Active Queue Management 26

4.1 TCP/RED model 26

4.1.1 Case 1: No loss 28

4.1.2 Case 2: One packet loss 30

4.1.3 Case 3: At least two packet losses 30

4.2 Properties of the S-RED Model 33

4.3 Model validation 34

4.3.1 Validation with default RED parameters 35

4.3.2 Validation for various queue weights w_q 41

4.3.3 Validation for various drop probabilities p_{max} 45

4.3.4 Validation with various thresholds q_{min} and q_{max} 49

4.3.5 Validation summary 52

4.4 Comparison of S-RED model and M-model 53

4.4.1 Comparison for default RED parameters 54

4.4.2 Comparison for various queue weights w_q 54
4.4.3 Comparison for various drop probabilities p_{max} 57
4.4.4 Comparison for various thresholds q_{min} and q_{max}.......... 58
4.5 Model modification.......... 60
4.6 Model extension: TCP with Adaptive RED (S-ARED model) 64
4.6.1 S-ARED model validation.......... 67
4.6.2 Model evaluation 70

5 Chapter Five: Conclusions and future work 72

Reference List 74

LIST OF FIGURES

Figure 1. Evolution of the window size in TCP Reno, consisting of slow start, congestion avoidance, fast retransmit, and fast recovery phases. 4

Figure 2. RED drop probability as a function of the average queue size......................... 10

Figure 3. RED drop probability with $p_{max} = 0$.. 12

Figure 4. RED drop probability with $p_{max} = 1$.. 12

Figure 5. TCP window size evolution.. 16

Figure 6. Topology of the modeled and simulated network. 27

Figure 7. Evolution of the window size in the S-RED model. The fast recovery phase has been simplified.. 34

Figure 8. Topology of the simulated network.. 35

Figure 9. Evolution of the window size with default RED parameters: (a) S-RED model, and (b) zoom-in, (c) ns-2 simulation results and (d) zoom-in. Waveforms show good match between the S-RED model and the ns-2 simulation results.. 36

Figure 10. Evolution of the average queue size with default RED parameters: (a) S-RED model and (b) zoom-in, (c) ns-2 simulation result and (d) zoom-in. The average queue size obtained using the S-RED model is higher than the average queue size obtained using ns-2 simulations. 38

Figure 11. S-RED model: evolution of the window size for $w_q = 0.006$.......................... 41

Figure 12. ns-2: evolution of the window size for w_q =0.006. 42

Figure 13. Comparison of the average queue size for various w_q.................................. 43

Figure 14. S-RED model: evolution of the window size for $p_{max} = 0.5$. 46

Figure 15. ns-2: evolution of the window size for $p_{max} = 0.5$... 46

Figure 16. Comparison of the average queue size for various p_{max}................................. 47

Figure 17. S-RED model: evolution of the window size for q_{min} = 10 packets. 49

Figure 18. ns-2: evolution of the window size for q_{min} = 10 packets. 50

Figure 19. Comparison of the average queue size for various q_{min} and q_{max}.................... 50

Figure 20. M-model: evolution of the average queue size with default RED parameters. 54

Figure 21. Comparison of the average queue size for various w_q. 55

Figure 22. Comparison of the average queue size for various p_{max}.................................. 57

Figure 23. Comparison of the average queue size for various q_{min} and $q_{max.}$.................... 59

Figure 24. S-RED model with modification: (a) evolution of the window size and (b) the average queue size. 61

Figure 25. Comparison of the average queue size for various w_q. 62

Figure 26. Comparison of the average queue size for various p_{max}.................................. 63

Figure 27. Comparison of average queue size for various q_{min} and q_{max}.......................... 63

Figure 28. Evolution of the window size: (a) S-ARED and (b) ns-2 simulation results.. 68

Figure 29. Evolution of the average queue size: (a) S-ARED and (b) ns-2 simulation results.. 69

LIST OF TABLES

Table 1. Default RED parameters in ns-2. 35

Table 2. Statistic values of state variables for S-RED model and ns-2. 40

Table 3. Statistical analysis of window size for various w_{q}. 43

Table 4. Statistical analysis of the average queue size for various w_{q}. 44

Table 5. System variable for various w_{q}. 45

Table 6. Statistical analysis of the window size for various p_{max}. 47

Table 7. Statistic analysis of the average queue size for various p_{max}. 48

Table 8. System parameters for various p_{max}. 48

Table 9. Statistical analysis of the window size for various q_{max} *and* q_{min}. 51

Table 10. Statistical analysis of the average queue size for various q_{max} *and* q_{min}. 51

Table 11. Statistic analysis of the average queue size for various w_{q}. 52

Table 12. System parameters. 53

Table 13. System variables for various w_{q}. 56

Table 14. System variables for various p_{max}. 58

Table 15. System parameters with various q_{min} and q_{max}. 60

Table 16. Default ARED parameters in ns-2. 67

Table 17. Average values of the window size, average queue size, and system parameters. 70

CHAPTER ONE: MOTIVATION

Today’s Internet applications, such as the World Wide Web, file transfer, Usenet news, and remote login, are delivered via the Transmission Control Protocol (TCP). With an increasing number and variety of Internet applications, congestion control becomes a key issue. Active Queue Management (AQM) interacts with TCP congestion control mechanisms, and plays an important role in meeting today’s increasing demand for quality of service (QoS). Random Early Detection (RED), a widely employed AQM algorithm, is a gateway-based congestion control mechanism. An accurate model of TCP with RED can aid in the understanding and prediction of the dynamical behaviour of the network. In addition, the model may help in analyzing the system’s stability margins, and providing the design guidelines for selecting network parameters. These design guidelines are important for network designers whose aim is to improve network robustness. Therefore, modeling TCP with RED is an important step towards improving the network efficiency and the service provided to Internet users.

Modeling TCP performance has received increasing attention during the last few years due to the benefits it offers to the networking community. Analytical TCP models enable researchers to closely examine the existing congestion control algorithms, address their shortcomings, and propose methods for their improvement. They may also be used to compare various TCP schemes and implementations, and to determine their performance under various operating conditions. Moreover, these models help examine the interactions between TCP and the queuing algorithms implemented in routers. Hence, they aid in the improvement of existing algorithms and in designing better algorithms, such as AQM techniques. Finally, such models offer the possibility of defining TCP-friendly behaviour in terms of throughput for non-TCP flows that coexist with TCP connections in the same network.

In this thesis, our goal is to model TCP and to investigate the nonlinear phenomena in a TCP/RED system. We are particular interested in the bifurcation and chaos phenomena that are recently observed in a TCP/IP network [24]. We use an iterative map to model the system. A second-order discrete model is derived in order to capture the interactions of the TCP congestion control algorithm with the RED queuing mechanism. We use the concepts proposed in [34] and construct a nonlinear dynamical model of TCP/RED that employs two state variables: the window size and the average queue size. The change of the window size reflects the dynamics of TCP congestion control, while the average queue size captures the queue dynamics in the RED gateway. The novelty of the proposed model is in capturing the detailed dynamical behaviour of TCP/RED. The proposed model, called *S-RED model*, considers the slow start, the congestion avoidance, the fast retransmit, and parts of fast recovery phases. It also takes into account timeout events. In addition, the interaction of TCP with Adaptive RED (ARED), a revised RED algorithm, is discussed. We also derive a discrete-time model for TCP with ARED, named *S-ARED model*, based on *S-RED model*.

CHAPTER TWO: BACKGROUND

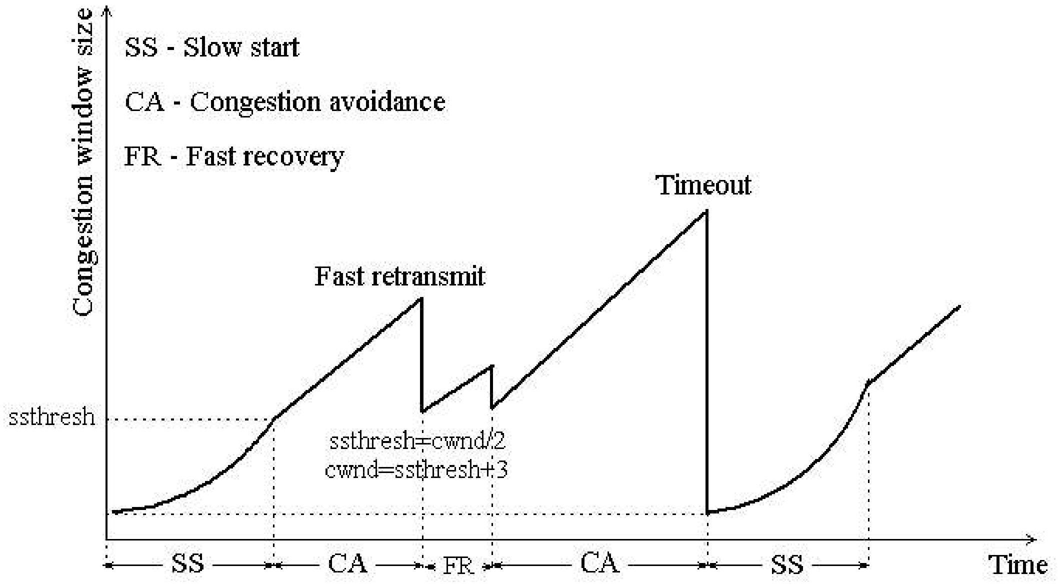
2.1 TCP algorithm

TCP, or Transmission Control Protocol, is designed to provide a connection-oriented, reliable, and byte-stream service [42]. Before communication can begin between two hosts, a connection must be established by employing a three-way handshake algorithm. This algorithm also allows the sender and receiver to initialize and synchronize important parameters, sequence numbers, advertised window size, and maximum segment size. The basic concept of the TCP congestion control algorithm is window based flow control. The window size determines the amount of data to be sent by the source. Its value will increase when all of the acknowledgements are properly received; otherwise, the window size may decrease. There are numerous versions of TCP implementations. In this research, the primary focus is on TCP Reno [1], [20], [21], as it is one of the most widely used versions in today's network. Other versions of TCP implementations are briefly overviewed in Section 2.1.2

2.1.1 TCP congestion control

To adjust the window size, the TCP Reno congestion control mechanism employs four algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery, as shown in Figure 1. They were introduced by Jacobson [20], [21] and are described in RFCs 1122 [4] and 2581 [1].

Figure 1. Evolution of the window size in TCP Reno, consisting of slow start, congestion avoidance, fast retransmit, and fast recovery phases.



In order to avoid congesting the network with large data bursts, an established TCP connection first employs the slow start algorithm to detect the available bandwidth in the network. Typically, a TCP sender initializes its congestion window (*cwnd*) to one or two segments, depending on the TCP implementation. Upon receipt of each acknowledgement (ACK) for new data, TCP increments *cwnd* by one segment size. When *cwnd* exceeds a threshold (*ssthresh*), the sender switches from slow start to the congestion avoidance phase. During congestion avoidance, *cwnd* is incremented by one segment size per Round Trip Time (RTT). Each time a packet is sent by the sender, a timer is set. Packet loss is detected by this timeout mechanism if the timer expires before receipt of the packet has been acknowledged. When multiple packets are dropped from the same window or the TCP sender does not receive enough packets to trigger fast retransmit, the TCP sender may need to wait for a timeout to occur before the lost packets get retransmitted. If a packet loss is detected by the timeout mechanism, the TCP sender adjusts its *ssthresh* and switches back to slow start. The duration of the timeout (TO) is usually updated with the RTT. If another timeout occurrs before the lost packet is successfully retransmitted, the period of timeout (TO) doubles to twice the value of the previous duration (2TO). This duration time continues to increase exponentially for each unsuccessful retransmission until the timeout period reaches 64 times the first timeout period (64TO). The timeout period then remains constant at 64TO.

The fast retransmit algorithm is used for recovery from losses detected by triple duplicate ACKs; i.e. four consecutive ACKs acknowledging the same packet. When an out-of-order segment is received by a TCP receiver, a duplicate ACK is immediately sent out to inform the sender of the sequence number for the packet that the receiver is expecting. The receipt of triple duplicate ACKs is used as an indication of packet loss. The TCP sender reacts to the packet loss by reducing *cwnd* by half and by re-transmitting the lost packet without waiting for the retransmission timer to expire.

The fast recovery algorithm is used to control data transmission after fast retransmission of the lost packet. During this phase, the TCP sender first sets its *ssthresh* to half of the current *cwnd*. Then it increases its *cwnd* for each duplicate ACK received. The fast recovery algorithm recognizes each duplicate ACK as an indication that one packet has left the channel and has reached the destination. Since the number of outstanding packets has decreased by one, the TCP sender is allowed to increment its *cwnd*. When a non-duplicate ACK is received, TCP sender sets *cwnd* equal to *ssthresh* and switches from fast recovery to the congestion avoidance phase.

2.1.2 TCP implementations

Older TCP implementations, released in the early 1980s, employed a simple window-based congestion control mechanism as specified in RFC 793 [38].

TCP Tahoe, released in the late 1980s, employed the slow start, congestion avoidance, and fast retransmit algorithms. When packet loss is detected, the sender retransmits the lost packet, reduces the congestion window to one and enters slow start. When the sender receives an ACK for the retransmitted packet, it will continue to send packets from the sequence number indicated in the ACK even if some of them are already sent. Thus, if any packet is lost in the meantime, it will be automatically retransmitted via this go-back procedure. In other words, TCP Tahoe will not have to wait for a timeout in the case where multiple packets are lost from the same window. However, frequent losses will force the TCP sender to operate in slow start phase for most of the time.

TCP Reno, introduced in the early 1990s, added the fast recovery algorithm. This algorithm does not require slow start for every packet loss. Once the three duplicate ACKs are received, the TCP sender retransmits the lost packet, reduces its congestion window by half, and enters the fast recovery phase instead of the slow start algorithm. When the first non-duplicate ACK is received, the sender exits fast recovery mode and enters the congestion avoidance phase. TCP Reno's fast recovery algorithm is optimized for the case when one packet is lost from a window. When multiple packets are lost from the same window, TCP Reno generally must wait for timeout.

TCP Reno’s congestion control algorithm works well when dealing with congestion-induced loss. However it results in reduced throughput without providing any benefits. It is also inappropriate when faced with loss due to corruption rather than congestion, such as satellite networks. In addition, using ns-2 simulations, Fall and Floyd [11] demonstrated that TCP Reno exhibits poor performance in terms of link utilization whenever multiple packets are dropped from a single window of data. To alleviate this problem, they introduced two modifications to TCP Reno: TCP New-Reno and TCP SACK [25]. A large number of Internet Web servers still use TCP Reno and its variants [35]. For example, in the TCP SACK implementation, TCP receivers can inform senders exactly which segments have arrived, rather than replying on TCP’s cumulative acknowledgements. This allows a TCP sender to efficiently recover from multiple lost segments without reverting to using a costly retransmission timeout to determine which segments need to be retransmitted. It also hastens recovery and prevents the sender from becoming window limited, thus allowing the pipe to drain while waiting to learn about lost segments. This ability is of particular benefit in keeping the pipe full and allowing transmission to continue while recovering from losses.

Other TCP implementations, such as TCP Vegas [5] and TCP Westwood [7], use various techniques to avoid congestion. They adjust the congestion window size based on estimates of the throughput at the bottleneck. For instance, TCP Vegas controls the congestion window size by estimating the RTT and by calculating the difference between the expected flow rate and the actual flow rate. It linearly adjusts TCP's congestion window size upwards or downwards, so as to consume a constant amount of buffer space in the network. It detects packet losses earlier than Reno and uses a slower multiple decrease than Reno. TCP Vegas eliminates the need to tune the receive window to serve as an upper bound on the size of the congestion window. It can avoid network congestion without overdriving the link to find the upper bound, even when operating with large windows. TCP Vegas increases its congestion window size more slowly than Reno by measuring the achieved throughput gain after each increase to detect the available capacity without incurring loss.

2.2 RED algorithm and evaluation of its parameters

2.2.1 Active Queue Management

A traditional DropTail queue management mechanism drops the packets that arrive when the buffer is full. However, this method has two drawbacks. First, it may allow a few connections to monopolize the queue space so that other flows are starved. Second, DropTail allows queues to be full for a long period of time. During that period, incoming packets are dropped in bursts. This causes a severe reduction in throughput of the TCP flows. One solution, recommended in RFC 2309 [2], is to employ active queue management (AQM) algorithms. The purpose of AQM is to react to incipient congestion before the buffer overflows. AQM allows responsive flows, such as

TCP flows, to react timely and reduce their sending rates in order to prevent congestion and severe packet losses.

2.2.2 RED algorithms

The most popular AQM algorithm is Random Early Detection (RED), proposed by Floyd and Jacobson [16]. The RED mechanism contains two key algorithms. One is used to calculate the exponentially weighted moving average of the queue size, so as to determine the burstiness that is allowed in the gateway queue and to detect possible congestion. The second algorithm is for computing the drop or marking probability, which determines how frequently the gateway drops or marks arrival packets. This algorithm can avoid global synchronization by dropping or marking packets at fairly evenly-spaced intervals. Furthermore, sufficiently dropping or marking packets, this algorithm can maintain a reasonable bound of the average delay, if the average queue length is under control. The RED algorithm is given in Algorithm 1, where $\overline{q}$ is the average queue size and q is the instantaneous queue size.

Let w_q be the weight factor and q_{k+1} be the new instantaneous queue size. At every packet arrival, the RED gateway updates the average queue size as

$$\overline{q}_{k+1} = (1 - w_q) \cdot \overline{q}_k + w_q \cdot q_{k+1} . \tag{1}$$

During the period when the RED gateway queue is empty, the system will estimate the number of packets m that might have been transmitted by the router. Hence, the average queue size is updated as

$$\overline{q}_{k+1} = (1 - w_q)^m \cdot \overline{q}_k , \tag{2}$$

$$m = idle\_time / transmission\_time , \tag{3}$$

where *idle_time* is the period that the queue is empty and *transmission_time* is the typical time that a packet takes to be transmitted.

Algorithm 1: The pseudo-code of RED algorithm.

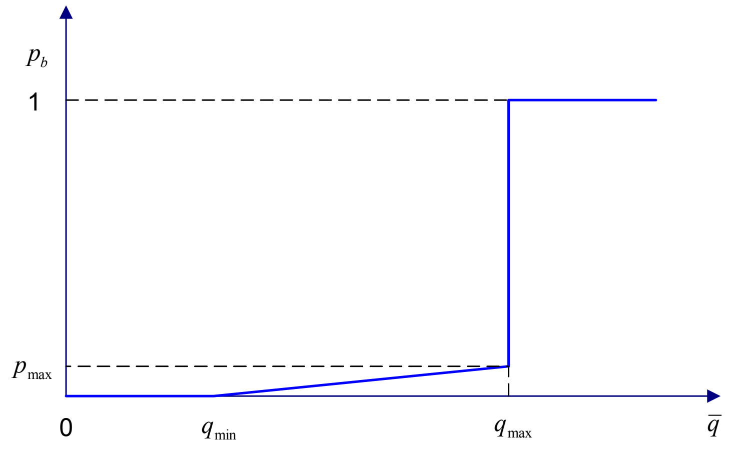
```
Initialization:
$\overline{q} = 0$; count = -1;
for each packet arrival
{
    calculate $\overline{q}$
    if (queue empty)
    {
        m = idle_time / transmission_time
        $\overline{q} = (1- w_q)^m \times \overline{q}$ ;
    }
    else
    {
        $\overline{q} = (1- w_q) \times \overline{q} + w_q \times q$;
    }
    if ($q_{min} < \overline{q} < q_{max}$)
    {
        count = count +1;
        calculate Pa:
        $Pb = p_{max} \times (\overline{q} - q_{min}) / (q_{max} - q_{min})$;
        $Pa = Pb / (1- count \times Pb)$;
        Drop arriving packet with Pa; (if arrival packet is dropped,
        count = 0)
    }
    else if (($\overline{q} > q_{max}$) or ($\overline{q}$ = buffersize))
    {
        Drop arriving packet; count = 0;
    }
}
```

The average queue size is compared to two parameters: the minimum queue threshold q_{min}, and the maximum queue threshold q_{max}. If the average queue size is smaller than q_{min}, the packet is admitted to the queue. If it exceeds q_{max}, the packet is marked or dropped. If the average queue size is between q_{min} and q_{max}, the packet is dropped with a drop probability p_b that is a function of the average queue size

$$p_{b_{k+1}} = \begin{cases} 0 & if \quad \overline{q}_{k+1} \leq q_{\min} \\ 1 & if \quad \overline{q}_{k+1} \geq q_{\max} \\ \dfrac{\overline{q}_{k+1} - q_{\min}}{q_{\max} - q_{\min}} \cdot p_{\max} & otherwise \end{cases}, \tag{4}$$

where p_{max} is the maximum packet drop probability. The relationship between the drop probability and average queue size is shown in Figure 2.

Figure 2. RED drop probability as a function of the average queue size.



The final drop probability p_a is given by Eq. (5). It is introduced in order to avoid the severe increase of the drop rate shown in Figure 2.

$$p_a = \frac{p_b}{1 - count \cdot p_b}. \tag{5}$$

Here, *count* is the cumulative number of the packets that are not marked or dropped since the last marked or dropped packet. It is increased by one if the incoming packet is not marked or dropped. Therefore, as *count* increases, the drop probability increases. However, if the incoming packet is marked or dropped, *count* is reset to 0.

2.2.3 Evaluation of RED parameters

Unlike DropTail, the performance of the RED algorithm is not determined only by the buffer size. Other parameters, such as queue weight (w_q), maximum drop rate (p_{max}), and the two queue thresholds (q_{min} and q_{max}), are also of great importance. The network designer should select these appropriately, so that the system can achieve better performance.

w_q is an exponentially weighted average factor, which determines the time constant for the averaging of the queue size. If w_q is too low, then the estimated average queue size is probably responding too slowly to transient congestion. If w_q is too high, then the estimated average queue size will track the instantaneous queue size too closely, and may result in an unstable system. In [17], quantitative guidelines are given for setting w_q in terms of the transient burst size that the queue can accommodate without dropping any packets. The optimal setting for w_q is between 0.001 and 0.005.

The packet drop probability is calculated as a linear function of the average queue size if the average queue size is between q_{min} and q_{max}. However, the maximum drop probability determines the oscillation of the drop rate. If p_{max} is set too small, then the network behaves similar to the DropTail shown in Figure 3. However, if this value is set too high (Figure 4), the packet drop probability increases, thus enforcing the system to oscillate severely and experience a decrease in throughput. It has been shown that steady-state packet drop rates of 5% to 10% would not be unusual at a router. There is, therefore, no need to set p_{max} higher than 0.1 for real network implementations [17].

Figure 3. RED drop probability with $p_{max} = 0$.

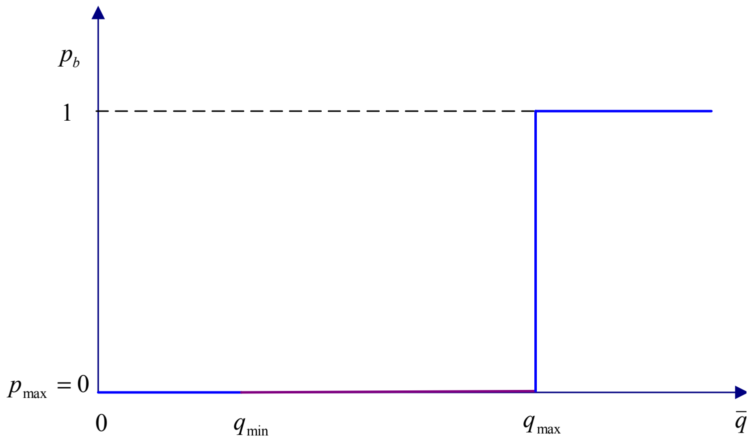
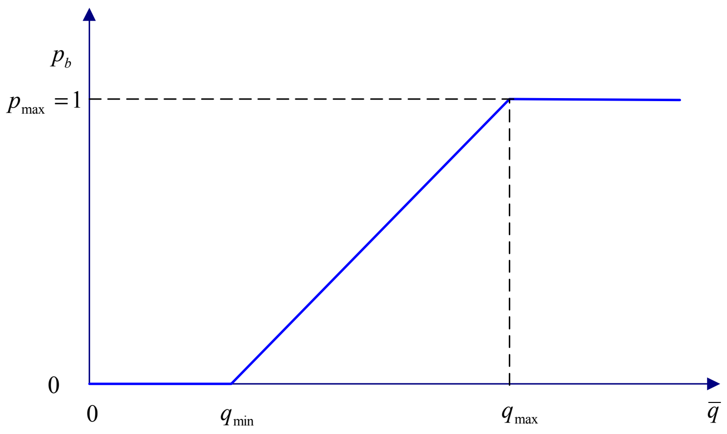


Figure 4. RED drop probability with $p_{max} = 1$.



The optimal values for q_{min} and q_{max} depend on the desired average queue size. If the typical traffic patterns are fairly bursty, then q_{min} must be correspondingly large to allow the link utilization to be maintained at an acceptably high level. This value should also be associated with the buffer size of the network. If q_{min} is set too small, it leads to low bandwidth utilization. Conversely, if q_{min} is set too high, it may result in unfair competition for bandwidth among multiple links, thereby cancelling out the benefits of the RED algorithm. The optimal value for q_{max} depends in part on the maximum average delay that can be allowed by the gateway. The RED gateway functions most effectively when the difference between q_{max} and q_{min} is larger than the typical increase in the calculated average queue size, in one roundtrip time [16]. A useful rule of thumb is to set q_{max} to at least twice the size of q_{min}. If the difference between q_{max} and q_{min} is set too small, then the computed average queue size can regularly oscillate up to q_{max}. This behaviour is similar to the oscillations of the queue size up to the maximum queue size experienced with DropTail gateways.

2.2.4 Recent studies on AQM

Although RED is one of the most prominent active queue management schemes, a number of research efforts have focused on possible shortcomings of the original RED algorithm and have proposed modifications and alternatives, such as Adaptive RED (ARED) [15], CHOKe [36], BLUE [13], Stabilized RED, Flow RED, and Balanced RED. For example, ARED, proposed in 2001, aims to improve the robustness of the original RED with only minimal changes. It attempts to achieve a stable average queue size around a pre-set target value. Its default setting is half way between q_{min} and q_{max}. It has been shown [15] through various simulation results that ARED tends to be more stable and performs better than the original RED. The key modification of ARED is updating the maximum drop rate for every interval period so that the packet drop probability changes more slowly than in the original RED algorithm. The calculation of the modified p_{max} is given in Algorithm 2.

Other algorithms are aimed at solving problems that the original RED could not address. For instance, CHOKe [36] is designed to achieve better performance on fairness, while the BLUE algorithm performs queue management directly based on packet loss and link utilization, rather than on the instantaneous or average queue lengths. In addition, the BLUE algorithm can react faster to substantial increases of traffic load than the RED algorithm.

Algorithm 2: The pseudo-code of ARED algorithm.

```
For every interval (usually the interval is set to 0.5 seconds):
{
        If (($\overline{q}$ > target ) && ($p_{max}$ <= 0.5))
        {
                $p_{max} = p_{max} + \alpha$ ;
        }
        Else if (($\overline{q}$ < target) && ($p_{max}$ >= 0.01))
        {
                $p_{max} = p_{max} * \beta$ ;
        }
}
where
 target: [$q_{min} + 0.4 * (q_{max} – q_{min})$, $q_{min} + 0.6 * (q_{max} – q_{min})$];
$\alpha$ : min(0.01, $p_{max}/4$);
$\beta$ : 0.9.
```

2.3 Simulation tool: ns-2

Ns-2 [33] is a discrete event network simulator targeted at networking research, which is currently supported by the Defence Advanced Research Project Agency (DARPA) and the National Science Foundation (NSF). It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. In this research, ns-2 was used to perform the network simulation and the simulation results were used to evaluate the proposed S-RED model.

CHAPTER THREE: A SURVEY OF EXISTING MODELS

Many papers have been published on different models for TCP. In this chapter, we describe the different models and implementations of TCP.

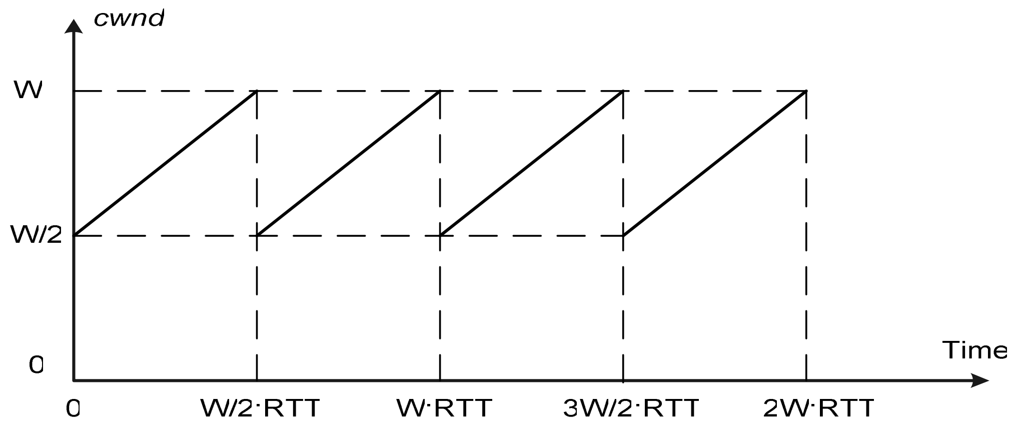
From the viewpoint of flow characteristics, analytical TCP models can be classified in three categories based on the duration of the TCP flows. This, in turn, determines the TCP congestion control algorithms to be modeled and the aspects of TCP performance that can be captured by the model [22]. The first category models short-lived flows where TCP performance is strongly affected by the connection establishment and slow start phases [30]. These models typically approximate average latency or completion time, i.e., the time it takes to transfer a certain amount of data. The second category models long-lived flows, which characterize the steady-state performance of bulk TCP transfers during the congestion avoidance phase [28], [32], [34], [41]. These models approximate aspects such as the window size, average throughput, sending rate, and goodput (goodput is used to describe the data that has been properly received). The final category includes models for flows of arbitrary duration, i.e., those that can accommodate both short and long-lived flows [6], [8]. All these models provide functions to compute the average values of throughput, sending rate, goodput, or latency of TCP flows.

From the control theoretic point of view, the developed models of TCP and TCP/RED [18], [19], [23], [24], [31], [39], [40], can be classified into two types: averaged models and iterative map models. An averaged model is described by a set of continuous differential equations. It neglects the detailed dynamics and only captures the low frequency characteristics of the system. It can be used to analyze the steady-state performance and to predict low frequency slow-scale bifurcation behaviour, such as Hopf bifurcations. Examples of such models are given in [18], [19], [23]. In contrast, an iterative map model has a discrete-time form and employs a set of difference equations. It provides relatively complete dynamical information. The discrete-time map is adequate to explore nonlinear phenomena such as period-doubling and saddle-node bifurcations, which may appear across a wide spectrum of frequencies and cause the existence of solutions in the high frequency range. Examples of iterative maps are given in [24], [39], [40].

3.1 Overview of modeling TCP Reno

Several models have been proposed in order to analyze the performance of packet networks. Prior to 1997, research was primarily focused on modeling the TCP congestion avoidance phase and ignored slow start, fast recovery, and timeout events. In [27], the authors assume random packet loss with a constant probability *p*, which means that approximately 1 / *p* packets are to be sent followed by one packet loss. Hence, the evolution of the *cwnd* results a periodic sawtooth shown in Figure 5.

Figure 5. TCP window size evolution.



The *cwnd* is halved when a packet is dropped. During this period, there are $\frac{1}{2}\cdot(\frac{W}{2}+W)\cdot\frac{W}{2}=\frac{3W^2}{8}$ packets to be delivered, where W is the maximum *cwnd*. Since, $\frac{1}{p}=\frac{3W^2}{8}$, it follows that *cwnd* is

$$W=\sqrt{\frac{8}{3p}}\,. \tag{6}$$

As *cwnd* increases by one packet size every RTT, the time required to send $\frac{3W^2}{8}$ packets is $\frac{W}{2}\cdot RTT$. Hence, the sending rate becomes

$$\begin{aligned} BW &= \frac{M\cdot\frac{3}{8}W^2}{RTT\cdot\frac{W}{2}}\,, \\ &= \frac{M\cdot C}{RTT\cdot\sqrt{p}} \end{aligned} \tag{7}$$

where constant C = $\sqrt{3/2}$.

By selecting an appropriate value for the constant C, the model is able to estimate the TCP performance over a range of loss conditions and in environments with both DropTail and RED queuing [28]. However, the prediction of the model is not very accurate, because it simplifies the TCP congestion control algorithm.

A more precise steady-state model of TCP Reno, introduced in [34], also employs the steady-state sending rate as a function of loss rate and RTT of a bulk data transfer TCP flow. The model provides a detailed explanation of the TCP algorithm. In addition to capturing the essence of TCP's fast retransmit and congestion avoidance phases, it also accounts for the effect of the timeout mechanism, which is important from a modeling perspective. The model also takes into account delayed ACKs, which are not discussed in this research. However, this model ignores the influence of queue management schemes.

In [34], authors take several steps to analyze the sending rate by separating the congestions caused by three-duplicate ACKs and timeouts. They also consider other upper bound parameters, such as advertised window size. The model describes the system in terms of duration of the RTT, called *rounds*. The model is built based on the following assumptions: RTT is a constant and it is considered to be independent of the window size; the maximum window size is also constant; only long lived connections are studied, which implies that the source buffer always has enough data to send; any packet loss is considered as link congestion; the packet loss is correlated within one round, i.e., if a packet is lost, then the remaining packets transmitted in that round are also lost. However, the packet loss is independent in different rounds. Slow start and fast recovery phases are not considered in this model. The following two situations describe the cases studied in [34].

Case one: The timeout is not considered and congestion is only detected by three duplicate ACKs.

The expectation of the total number of packets Y that are to be transmitted before a packet loss, is given by

$$\begin{aligned} E[Y] &= \frac{1-p}{p} + E[W] \\ &= \frac{1}{p} + \sqrt{\frac{8-5p}{3p}} \end{aligned}, \tag{8}$$

where p is the drop probability and W is the value of congestion window size. The time required to send Y packets is noted by A. The expectation of A is

$$E[A] = \frac{RTT}{2}(3 + \sqrt{\frac{8-5p}{3p}}). \tag{9}$$

From Eqs (8) and (9), the sending rate *BW* can be derived as

$$\begin{aligned} BW &= \frac{E[Y]}{E[A]} \\ &= \frac{\frac{1}{p} + \sqrt{\frac{8-5p}{3p}}}{\frac{RTT}{2}(3 + \sqrt{\frac{8-5p}{3p}})} . \end{aligned} \tag{10}$$

Equation (10) can be simplified to Eq. (11) for small values of *p* as

$$BW = \frac{1}{RTT}\sqrt{\frac{3}{2p}} . \tag{11}$$

Case Two: The loss indication is detected by three duplicate ACKs and timeouts:

The sending rate in Eq. (10) is modified to

$$BW = \frac{E[Y] + Q \cdot E[R]}{E[A] + Q \cdot E[Z^{TO}]}, \tag{12}$$

where Q is the probability that a packet loss is detected by a timeout, R is the total number of packets that are to be sent during the timeout interval, and Z^{TO} is the duration of the timeout. The final simplified equation (for small values of p) for the sending rate is

$$BW \approx \frac{1}{RTT\sqrt{\frac{2p}{3}} + T_0 \min(1, 3\sqrt{\frac{3p}{8}})p(1 + 32p^2)}, \tag{13}$$

where T_0 is the initial value of a timeout.

Measurements demonstrated that the model was adequate over a range of loss rates [34].

The model developed in [6] is a further extension of the model developed in [34], which aims for a more precise model of TCP latency. The model takes into account the time used to set up the connection, which is known as the three-way handshake algorithm, and the slow start phase. This information is very important for modeling short-lived connections. In addition, the model considers detailed information about data transmission including congestion avoidance, fast retransmit, fast recovery, and timeout evens. The total latency of the network is

$$E[T] = E[T_{ss}] + E[T_{loss}] + E[T_{ca}] + E[T_{delack}], \tag{14}$$

where T_{ss} is the duration of data transfer within the slow start and congestion avoidance phases, T_{loss} is the cost for timeout or fast recovery, T_{ca} is the time required to transmit the remaining packets before loss recovery, and T_{delack} is the cost of the delayed ACKs.

3.2 Overview of modeling TCP Reno with RED

A simplified first-order discrete nonlinear dynamical model was developed for simplified TCP with RED control in [24], [39], [40]. The exponentially weighted average queue size was used as the state variable. The model describes the system dynamics over large parameter variations and samples the buffer occupancy at certain time instances. This dynamical model was used to investigate the stability, bifurcation, and routes to chaos of a network for various system parameters. Based on the developed model, the authors demonstrated that nonlinear phenomena, such as bifurcation and chaos, might occur if the system parameters were not properly selected. However, this discrete model neglects the dynamics of TCP. The derived map is given by

$$q_{k+1}^{ave} = \begin{cases} (1-w)\cdot q_k^{ave} & if \quad q_k^{ave} \ge q_u^{ave} \\ (1-w)\cdot q_k^{ave} + w\cdot B & if \quad q_k^{ave} \le q_l^{ave} \\ (1-w)\cdot q_k^{ave} + w\cdot (\dfrac{N\cdot K}{\sqrt{p_k}} - \dfrac{C\cdot d}{M}) & otherwise \end{cases}, \tag{15}$$

where q_{k+1}^{ave} : average queue size in round *k+1*

q_k^{ave} : average queue size in round *k*

q_u^{ave} : upper bound of the average queue size

q_l^{ave} : lower bound of the average queue size

w *:* queue weight in RED algorithm

B : buffer size

N : number of TCP connections

K : constant [1, $\sqrt{8/3}$]

p_k : drop probability in round k

C : link capacity

d : round-trip propagation delay

M : packet size.

In [31], a second-order nonlinear dynamical model was developed to examine the interactions of a set of TCP flows with RED. The model employs the fluid-flow and stochastic differential equations. Window size and average queue length are used as state variables. From a set of coupled ordinary differential equations, the authors develop a numerical algorithm to obtain the transient average behaviour of queue length, round trip time, and throughput of TCP flows. This model is described by nonlinear differential equations that employ the average values of key network variables, given by

$$\begin{aligned} \dot{W}(t) &= \frac{1}{R(t)} - \frac{W(t)\cdot W(t-R(t))}{2R(t-R(t))}\cdot p(t-R(t)) \\ \dot{q}(t) &= \frac{N(t)}{R(t)}\cdot W(t) - C \end{aligned}, \qquad (16)$$

where $W(t)$: expected TCP window size

$q(t)$: expected queue length

$R(t)$: round trip time

$N(t)$: load factor (number of TCP sessions)

$p(t)$: probability of packet mark/drop

C : link capacity.

A third-order dynamical model that describes the interaction of TCP flows with an RED controlled queue was developed in [23]. The state variables of the model are average queue length, instantaneous queue length, and throughput. The TCP sources are idealized to operate only during the congestion avoidance phase, when the congestion window size follows the rule of linear increase and multiplicative decrease. This dynamical model is used to explore various parameter settings and observe transient and equilibrium behaviour of the system. The validity of the model is verified by comparison with simulation results. The interaction between TCP and RED is modeled as

$$\begin{aligned} &\frac{d}{dt}\bar{s}(t) = \bar{\lambda}(t-R/2)\cdot\beta(\bar{q}(t)-\bar{s}(t)) \\ &\frac{d}{ct}\bar{q}(t) = \bar{\lambda}(t-R/2)\cdot(1-\pi_K(\bar{q}(t)))\cdot(1-p(\bar{s}(t))) - \mu(1-\pi_0(\bar{q}(t))) \\ &\frac{d}{dt}\bar{\lambda}(t) = -\frac{P_L(t-R/2)}{2\cdot m}\bar{\lambda}(t)\cdot\bar{\lambda}(t-R) + (1-P_L(t-R/2))\frac{m}{R^2}\cdot\frac{\bar{\lambda}(t-R)}{\bar{\lambda}(t)} \\ &P_L(t) = p(\bar{s}(t)) + \pi_K(\bar{q}(t)) - p(\bar{s}(t))\cdot\pi_K(\bar{q}(t)) \end{aligned}, \qquad (17)$$

where $\bar{s}(t)$: expectation of the exponentially averaged queue length

$\bar{q}(t)$: expectation of the instantaneous queue length

$\bar{\lambda}(t)$: expectation of TCP throughput at the source

$P_L(t)$: loss probability in the queue at time *t*

$\pi_K(\bar{q}(t))$: steady-state probabilities for the queue to be full

$\pi_0(\bar{q}(t))$: steady-state probabilities for the queue to be empty

p : drop probability

R : round trip time

β : queue weight in RED algorithm

m : number of identical TCP sources.

In [18], [19], the authors describe a second-order linear model for TCP and RED, by linearizing the fluid-based nonlinear TCP model. Window size and average queue length are used as state variables of the system. The authors performed analysis of TCP interactions with RED from a control theoretic viewpoint. They presented design guidelines for choosing RED parameters that led to local stability of the system. In addition, they proposed two alternative controllers to improve the transient behaviour and stability of the system: a proportional (P) controller that possesses a good transient response, but suffers from steady-state errors in queue regulation, and a classical proportional-integral (PI) controller that exhibits zero steady-state regulation error and acceptable transient behaviour. One important contribution of this work is providing a good example of how to use classical control theory to solve problems in complex communication systems. The linearized model around the operating point is described as

$$\begin{aligned}\delta\dot{W}(t) &= -\frac{N}{R_0^2\cdot C}(\delta W(t)+\delta W(t-R_0)) - \frac{1}{R_0^2\cdot C}(\delta q(t)-\delta q(t-R_0)) - \frac{R_0\cdot C^2}{2N^2}\delta p(t-C) \\ \delta\dot{q}(t) &= \frac{N}{R_0}\delta W(t) - \frac{1}{R_0}\delta q(t)\end{aligned}, \quad (18)$$

where $\delta W \doteq W - W_0$

$\delta q \doteq q - q_0$

$\delta p \doteq p - p_0$

(W_0, q_0, p_0) : the set of operating points

W : expectation of the TCP window size

q : expectation of the queue length

R_0 : round trip time

C : link capacity

T_p : propagation delay

N : load factor (number of TCP sessions)

p : probability of packet mark/drop.

A multi-link multi-source model, developed in [27], was used to study the stability of a general TCP/AQM system. A local stability condition in the case of a single link with heterogeneous sources and the stability region of TCP/RED were derived. The state variables of this model are window size, instantaneous queue length, and average queue length. The authors demonstrated that TCP/RED becomes unstable when the link capacity increases. Finally, they devised a new distributed congestion control algorithm that maintains local stability for arbitrary delay, capacity, and traffic load. Preliminary simulation results were provided in order to illustrate the model's behaviour.

CHAPTER FOUR: A DISCRETE APPROACH FOR MODELING TCP RENO WITH ACTIVE QUEUE MANAGEMENT

4.1 TCP/RED model

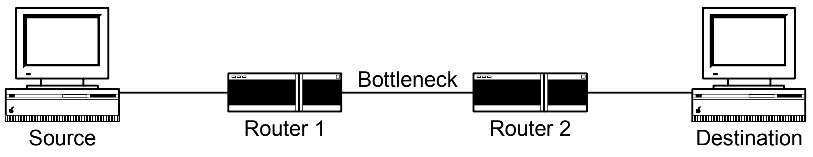
The basic idea behind RED is to sense impending congestion before it happens and to try to provide feedback to senders by either dropping or marking packets. Hence, from the control theoretic point of view, the network may be considered as a complex feedback control system. TCP adjusts its sending rate depending on whether or not it has detected a packet drop in the previous RTT interval. The drop probability of RED can be considered as the control law of the network system. Its discontinuity is the main reason for oscillations and chaos in the system. Hence, it is natural to model the network system as a discrete model. In this thesis, we model the TCP/RED system using a *stroboscopic map*, which is the most widely used type of discrete-time maps for modeling power converters [3], [9], [10], [43]. This map is obtained by periodically sampling the system state. In our study, the sampling period is one RTT. Since window size and queue size behave as step functions of RTT, one RTT is the sampling period that captures their changes [14]. Higher sampling rates would not significantly improve the accuracy of the model. Conversely, lower sampling rates would ignore the changes and affect the model accuracy.

The state variables employed in the S-RED model are window size and average queue size. These state variables capture the detailed behaviour of TCP/RED. Variations of the window size reflect the dynamics of TCP congestion control. The window size increases exponentially and linearly during the slow start and the congestion avoidance phases, respectively. It multiplicatively decreases when loss occurs. The average queue size captures the queue dynamics in RED as it is updated upon every packet arrival. In our study, we did not consider instantaneous queue size as an independent state variable.

The S-RED model is suited for investigating the nonlinear behavior of TCP/RED and analyzing the system's transitions into chaos. Hence, in this thesis, we consider a simple network topology as shown in Figure 6. It consists of one TCP source, two routers, and a destination. RED is employed in the first router. A TCP Reno connection is established between the source and the destination. Data packets are sent from the source to the destination, while traffic in the opposite direction consists of ACK packets only.

We made several assumptions in order to construct an approximate model. We assume that ACK packets are never lost. The connection is long-lived and the source always has sufficient data to send. Round trip propagation delay *d* between the source and destination is constant. Data packet size *M* is constant. The link that connects the two routers is the only bottleneck in the network. We also assume that the timeout is only caused by packet loss and that the duration of the timeout period is 5 RTTs [14]. The state variables of the system are sampled at the end of every RTT period. We assume that the queue size is constant during each sampling period. The model includes three cases, depending on the number of packets lost in the previous RTT period: no loss, single loss, and multiple packet losses.

Figure 6. Topology of the modeled and simulated network.



4.1.1 Case 1: No loss

Let W_k, q_k, and $\overline{q}_k$ be the window size, instantaneous queue size, and average queue size, respectively, at the end of the sampling period k. If no packet is dropped during the last RTT period, TCP Reno increases its window size. The window size is increased exponentially in the slow start phase and linearly in the congestion avoidance phase:

$$W_{k+1} = \begin{cases} \min(2W_k, ssthresh) & if \quad W_k < ssthresh \\ \min(W_k + 1, rwnd) & if \quad W_k \geq ssthresh \end{cases}, \tag{19}$$

where *rwnd* is the receiver's advertised window size; i.e., the largest window size that the receiver can accept in one round. Usually, *rwnd* is greater than the window size. In this case, *rwnd* does not affect the variations of the window size. In the event that the window size increases linearly and reaches the value *rwnd*, the window size is kept at *rwnd* until a loss occurs in the network.

The round trip time in *k+1* round is calculated by

$$RTT_{k+1} = d + \frac{q_k \cdot M}{C}, \tag{20}$$

where RTT_{k+1} : round trip time in round $k + 1$

d : round trip propagation delay

C : link capacity

M : packet size.

In order to calculate the average queue size given by Eq. (1), we need to know the queue size at the sampling period *k+1*. The instantaneous queue size depends on the queue size in the previous period, the current window size, and the number of packets that have left the queue during the previous sampling period. Therefore, the instantaneous queue size is given by

$$\begin{aligned} q_{k+1} &= q_k + W_{k+1} - \frac{C \cdot RTT_{k+1}}{M} \\ &= q_k + W_{k+1} - \frac{C}{M} \cdot (d + \frac{q_k \cdot M}{C}) . \\ &= W_{k+1} - \frac{C \cdot d}{M} \end{aligned} \tag{21}$$

where q_{k+1} : instantaneous queue size in round $k + 1$

q_k : instantaneous queue size in round k

W_{k+1} : current TCP window size in round $k + 1$

Substituting q_{k+1} in Eq. (1) yields the average queue size:

$$\overline{q}_{k+1} = (1 - w_q) \cdot \overline{q}_k + w_q \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0) . \tag{22}$$

Because RED updates the average queue size at every packet arrival, $\overline{q}$ is updated W_{k+1} times during the current sampling period. For the first packet arrival, the average queue size is updated according to

$$\overline{q} = (1 - w_q) \cdot \overline{q} + w_q \cdot \max(W - \frac{C \cdot d}{M}, 0) . \tag{23}$$

For the second packet arrival, the average queue size is given by

$$\overline{q} = (1 - w_q) \cdot [\ (1 - w_q) \cdot \overline{q} + w_q \cdot \max(W - \frac{C \cdot d}{M}, 0)\ ] + w_q \cdot \max(W - \frac{C \cdot d}{M}, 0) \cdot \tag{24}$$

Since we assume that queue size is constant during each period, and that there are W_{k+1} packet arrivals during this time interval, $\overline{q}$ is updated W_{k+1} times:

$$\overline{q}_{k+1} = (1 - w_q)^{W_{k+1}} \cdot \overline{q}_k + (1 - (1 - w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0) \cdot \tag{25}$$

In summary, if $p_k \cdot W_k < 0.5$, which implies that no packet loss occurred in the previous sampling period, the state variables of the model are

$$\begin{aligned} W_{k+1} &= \begin{cases} \min(2W_k, ssthresh) & if \quad W_k < ssthresh \\ \min(W_k + 1, rwnd) & if \quad W_k \geq ssthresh \end{cases} \\ \overline{q}_{k+1} &= (1-w_q)^{W_{k+1}} \cdot \overline{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0) \end{aligned} . \qquad (26)$$

4.1.2 Case 2: One packet loss

If $0.5 \leq p_k \cdot W_k < 1.5$, which implies that one packet loss occurred in the previous RTT period, the congestion control mechanism of TCP Reno halves the window size in the current sampling period:

$$W_{k+1} = \frac{1}{2} W_k . \qquad (27)$$

The average queue size is updated in the same manner as in Case 1:

$$\begin{aligned} W_{k+1} &= \frac{1}{2} W_k \\ \overline{q}_{k+1} &= (1-w_q)^{W_{k+1}} \cdot \overline{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0) \end{aligned} . \qquad (28)$$

4.1.3 Case 3: At least two packet losses

In this case $p_k \cdot W_k \geq 1.5$, which implies that at least two packets are lost in the previous RTT period. When multiple packets are lost from the same window, TCP Reno may not be able to send a sufficient number of new packets in order to receive three duplicate ACKs for each lost packet. The TCP source will probably have to wait for a timeout to occur before retransmitting the lost packet [11]. During the timeout period, the source does not send packets into the network. In our model, window size is equivalent to the number of packets that are sent by the source during one RTT period. Hence, we assume that the window size is zero during the timeout period.

The RED mechanism updates the average queue size for each packet arrival. However, when the queue is empty, it calculates the duration of this period named *idle time*. Then, the average queue size is updated based on estimating the number of packets that could have been sent during this time. Our model does not take into account the *idle time*. Therefore, during the timeout period, there are no packet arrivals. The average queue size is not updated and has the same value as in the previous RTT period. The TCP/RED system during the timeout period is described at

$$\begin{aligned} W_{k+1} &= 0 \\ \overline{q}_{k+1} &= \overline{q}_k \end{aligned} . \tag{29}$$

The pseudo-code describing the S-RED model is given in Algorithm 3.

Algorithm 3: The pseudo-code of the S-RED model.

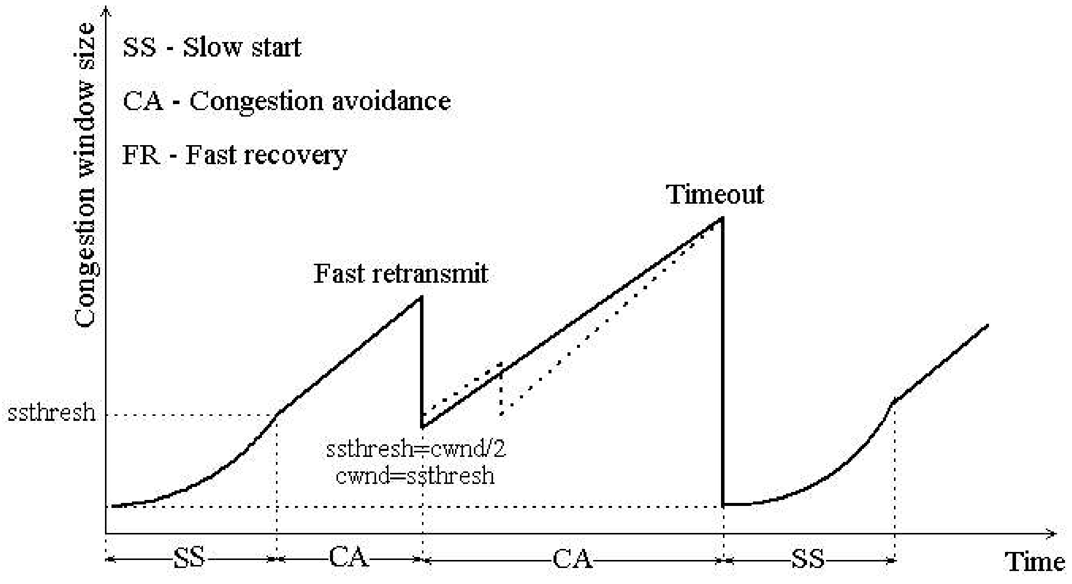
```
Initialization:
        $\bar{q}_0 \leftarrow 0;$
        $q_0 \leftarrow 0;$
        $p_0 \leftarrow 0;$
for every round
{
        calculate the product of $p_k \cdot W_k$
        if $p_k \cdot W_k < 0.5$
        {
                compare $W_k$ with ssthresh
                if $W_k$ < ssthresh
                {
                        $W_{k+1} \leftarrow \min(2W_k, ssthresh)$
                        $\bar{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \bar{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                }
                else
                {
                        $W_{k+1} \leftarrow \min(W_k + 1, rwnd)$
                        $\bar{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \bar{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                }
                calculate the drop probability using Eq. (4)
        }
        elseif $0.5 \le p_k \cdot W_k < 1.5$
        {
                $W_{k+1} \leftarrow \frac{1}{2} W_k$
                $\bar{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \bar{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                calculate the drop probability using Eq. (4)
        }
        else
        {
            $W_{k+1} \leftarrow 0$
            $\overline{q}_{k+1} \leftarrow \overline{q}_k$ .
            calculate the drop probability using Eq. (4)
        }
    }
```

4.2 Properties of the S-RED Model

The proposed second-order discrete-time model captures the interactions between the TCP Reno congestion control algorithm and the RED mechanism. It models the dynamics of the TCP/RED system. Unlike past models [23], [24], [34], [39], [40], the S-RED model includes the slow start phase. It also takes into account timeout, common in TCP [34], that the models in [23], [24], [39], [40] ignore. However, our model does not capture all the details of the fast recovery phase. Congestion window size in the S-RED model is not increased for each duplicate ACK received after retransmitting the lost packet. Instead of inflating the window size, we assume that the TCP sender switches to the congestion avoidance phase without performing slow start. Evolution of the window size in the S-RED model is shown in Figure 7.

The S-RED model captures the most important characteristics of the RED algorithm. The average queue size is updated after each packet arrival; i.e., W_{k+1} times in the sampling period $k+1$. In contrast, models presented in [24], [39], [40] update the average queue size only once every RTT period. However, in deriving the new model, we have also made simplifications to the RED algorithm. RED uses *count* to modify the drop probability according to Eq. (5), while we ignored the *count* that keeps track of the number of packet arrivals since the last packet drop. We have also ignored the *idle time* period, since it has no significant impact on the dynamics of the system.

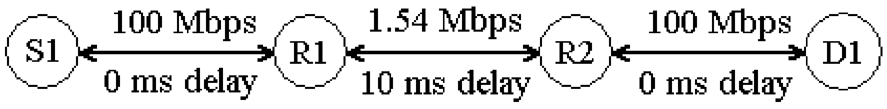
Figure 7. Evolution of the window size in the S-RED model. The fast recovery phase has been simplified.



4.3 Model validation

In order to evaluate the accuracy of the S-RED model, we compare its performance with the ns-2 simulation results. The topology of the simulated network is shown in Figure 8. It consists of one source, one sink, and two routers: R1 and R2. RED is employed in router R1. The link between R1 and R2 is the only bottleneck in the network. Its capacity is 1.54 Mbps and its propagation delay is 10 ms. The capacity of the links between the source and R1, and between R2 and the sink, is 100 Mbps. This is sufficient to guarantee no congestion in these links. Their propagation delay is 0 ms. We evaluated the window size and the average queue size in the S-RED model using MATLAB [29] and compared the results with ns-2 simulation results.

Figure 8. Topology of the simulated network.



The model validation is divided into four stages. First, we used ns-2 default RED parameters. Second, we chose various queue weights w_q, while keeping other system parameters constant. In the third scenario, we varied the maximum drop probability p_{max}. Finally, we varied the minimum and maximum queue thresholds q_{min} and q_{max} simultaneously, while keeping their ratio $q_{max} / q_{min} = 3$. In each simulation scenario, we observed system behaviour and captured average RTT, sending rate, and drop probability.

4.3.1 Validation with default RED parameters

In order to validate that the S-RED model can capture the details of the system behaviour, we evaluated the time waveforms of the two state variables: window size and average queue size. The ns-2 simulation results are obtained using default RED parameters listed in Table 1.

Table 1. Default RED parameters in ns-2.

Packet size M (bytes)	500
Maximum drop probability p_{max}	0.1
Minimum queue threshold q_{min} (packets)	5
Maximum queue threshold q_{max} (packets)	15
Queue weight w_q	0.002

The waveforms of the window size for various time scales are shown in Figure 9. The proposed S-RED model and ns-2 simulation results are quite similar, especially during the steady-state.

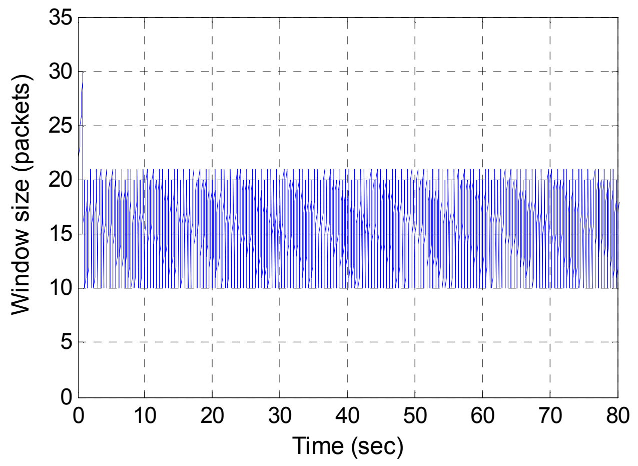
Figure 10 shows waveforms of the average queue size for various time scales. The average queue size in the S-RED model is approximately one packet larger than the average queue size resulting from the ns-2 simulation results. This difference is due to the simplifications that we introduced in the S-RED model. Our S-RED model employs packet-marking/drop probability calculated by Eq. (4), while the RED algorithm adopts a smooth drop probability p_a that increases slowly as the *count* increases:

$$p_a = \frac{p_b}{1 - count \cdot p_b}, \tag{30}$$

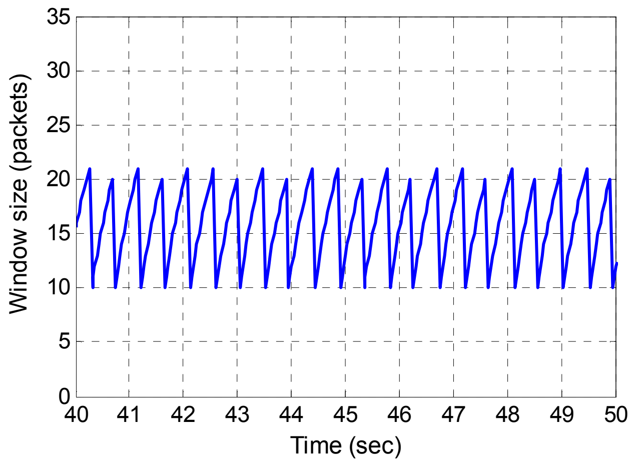
where p_b is the drop probability given by Eq. (4) and *count* measures the number of packets that have arrived since the last dropped packet. In the S-RED model, p_b is used as the final drop probability and *count* is ignored. Since $p_b < p_a$, the average queue size of the S-RED model is larger than that obtained via ns-2 simulations.

Figure 9. Evolution of the window size with default RED parameters: (a) S-RED model, and (b) zoom-in, (c) ns-2 simulation results and (d) zoom-in. Waveforms show good match between the S-RED model and the ns-2 simulation results.

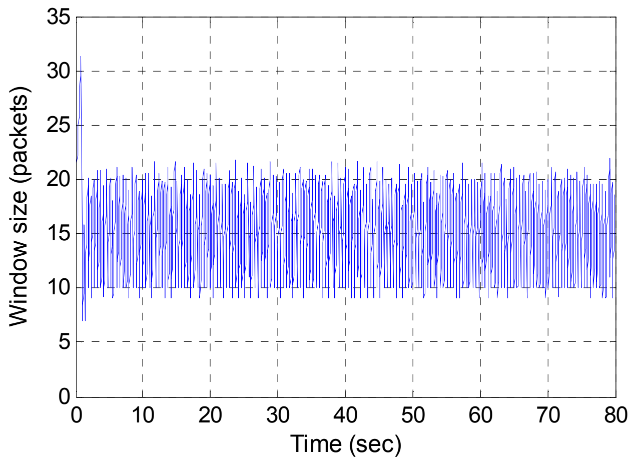
(a)



(b)



(c)



(d)

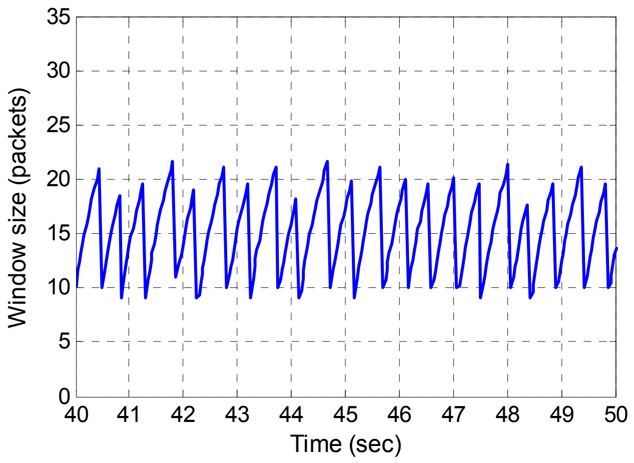
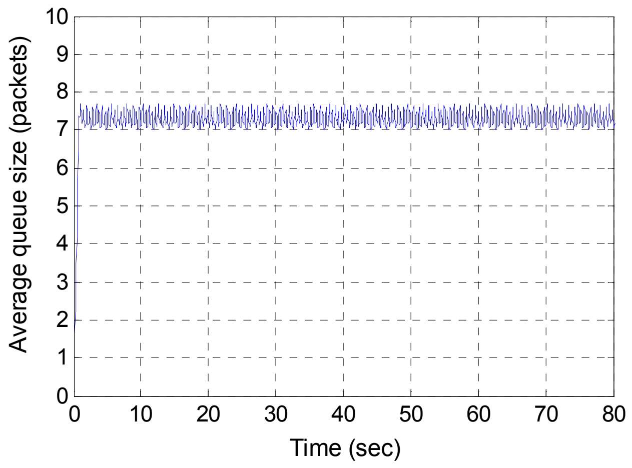
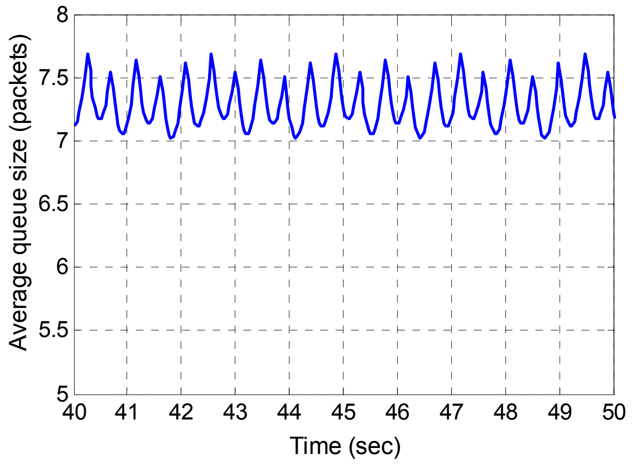


Figure 10. Evolution of the average queue size with default RED parameters: (a) S-RED model and (b) zoom-in, (c) ns-2 simulation result and (d) zoom-in. The average queue size obtained using the S-RED model is higher than the average queue size obtained using ns-2 simulations.

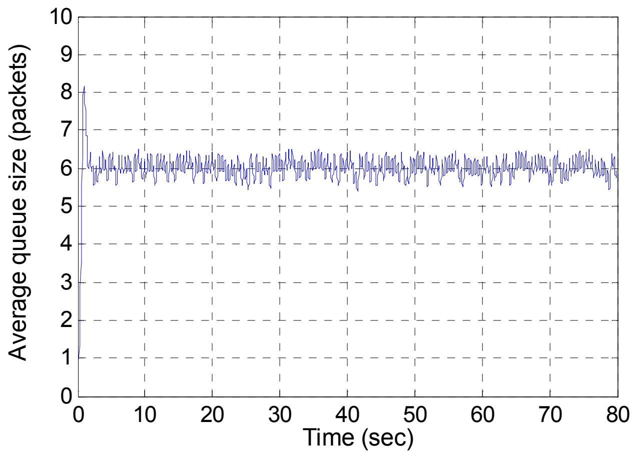
(a)



(b)


Average queue size (packets)
Time (sec)
8
7.5
7
6.5
6
5.5
5
40
41
42
43
44
45
46
47
48
49
50

(c)


Average queue size (packets)
Time (sec)
10
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80

(d)

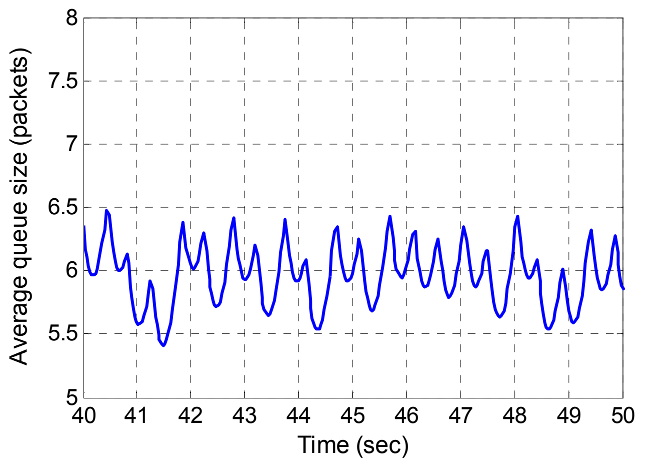


Table 2 summarizes the statistic values of the two state variables: window size and average queue size. It also compares the results with the ns-2 simulation. We calculated the average, maximum, and minimum values of the window size and average queue size over the duration of the simulation. In addition, the maximum and minimum values during the steady-state are also computed. We observed that the window size obtained from the S-RED model is similar to that obtained via the ns-2 simulation. The difference of the average queue size is approximately one packet.

Table 2. Statistic values of state variables for S-RED model and ns-2.

	Window size (packets)			Average queue size (packets)		
	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)
Average	15.35	15.01	2.25	7.24	5.95	21.63
Max	30	31.38	-1.20	7.69	8.14	-5.60
Min	2	2	0	0.12	0.05	77.45
Max (steady-state)	21	21.86	-3.93	7.69	6.51	18.16
Min (steady-state)	10	9	11.11	7.02	5.41	29.79

4.3.2 Validation for various queue weights w_q

In order to evaluate the S-RED model for various system parameters, we varied the queue weight w_q between 0.001 and 0.01. The window size and the average queue size obtained from S-RED model are compared to those from ns-2 simulation. The window size of S-RED model and ns-2 are similar. For example, Figures 11 and 12 show the waveform of the window size, for $w_q = 0.006$, for S-RED model and ns-2, respectively.

Figure 11. S-RED model: evolution of the window size for $w_q = 0.006$.

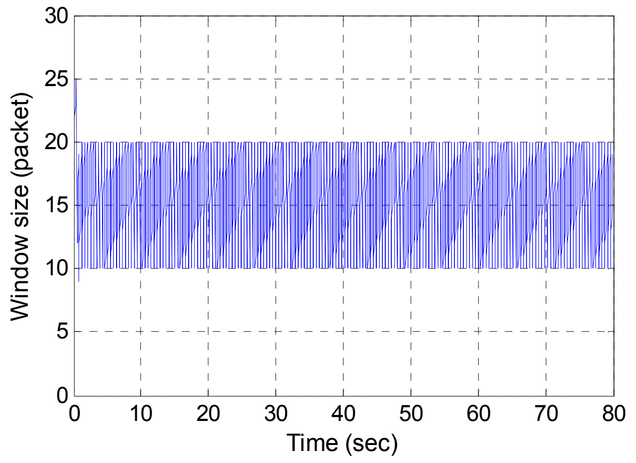
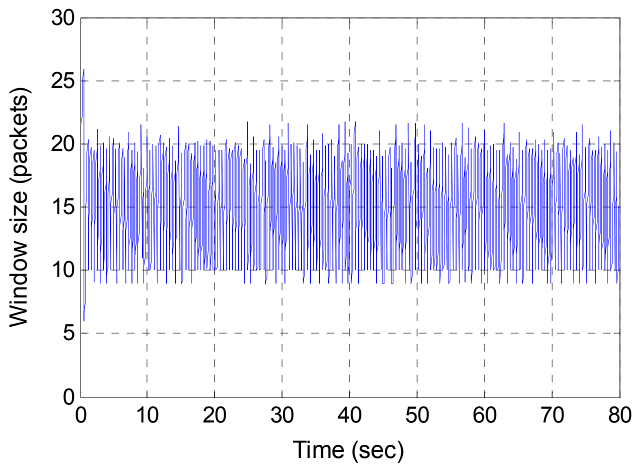


Figure 12. ns-2: evolution of the window size for w_q =0.006.



The waveforms in Figures 11 and 12 show that S-RED model and ns-2 have similar results for the window size. However, ns-2 has coarser waves. We also compared the waveforms of the window size between S-RED model and ns-2 for various w_q. The results are similar to those presented in Figures 11 and 12. Therefore, the S-RED model can capture the dynamical behaviour of the congestion window size, an important variable of the TCP algorithm.

Figure 13 shows the simulation results of the average queue size for various w_q during steady-state. When w_q increases, the average queue size slowly decreases. The difference in the average queue size between S-RED model and ns-2 simulations is approximately one packet for various $w_{q.}$ Tables 3 and 4 summarize the statistical analysis of the window size and average queue size for various w_q during steady-state.

Figure 13. Comparison of the average queue size for various w_q.

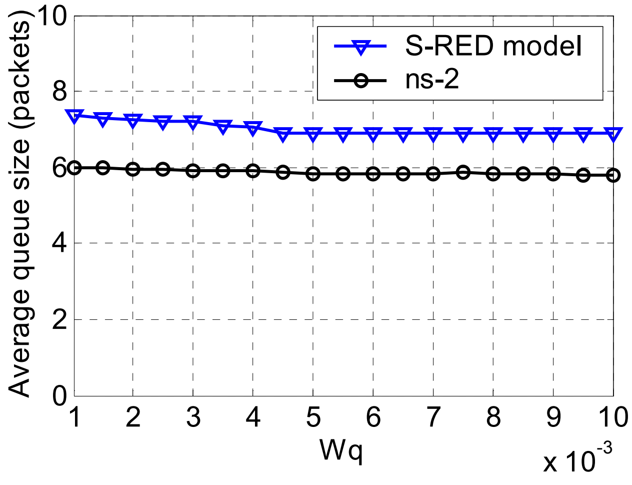


Table 3. Statistical analysis of window size for various w_q.

Parameters	Average window size (packets)			Variance: $\frac{n\sum x^2-(\sum x)^2}{n(n-1)}$	
weight (w_q)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
0.001	15.39	15.09	1.99	11.62	12.74
0.002	15.35	15.01	2.25	11.27	10.71
0.004	15.14	14.98	1.07	10.61	10.43
0.006	15.02	14.84	1.21	10.01	10.67
0.008	15.00	14.89	0.74	10.04	11.07
0.010	15.00	13.83	8.45	10.04	9.11

Table 4. Statistical analysis of the average queue size for various w_q.

Parameters	Average queue size (packets)			Variance: $\frac{n\sum x^2 - (\sum x)^2}{n(n-1)}$	
weight (w_q)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
0.001	7.38	5.98	23.41	0.007	0.021
0.002	7.24	5.95	21.63	0.027	0.046
0.004	7.06	5.91	19.45	0.096	0.122
0.006	6.90	5.82	18.55	0.165	0.293
0.008	6.89	5.84	17.97	0.287	0.540
0.010	6.89	5.79	18.99	0.435	0.618

From Tables 3 and 4, we observed that as w_q increases, the average queue size and the window size slowly decrease. However, w_q has an obvious impact on the variance of the average queue size. Large w_q implies large variance of the average queue size. This may lead to an unstable system.

For w_q between 0.001 to 0.008, the window size of the S-RED model is in agreement with that of ns-2. Although the results of S-RED model and ns-2 have some difference, they follow the same trend.

We also evaluated the S-RED model for other system variables: average RTT, sending rate, and packet loss rate. The results are summarized in Table 5. Values obtained using the S-RED model and ns-2 are quite similar. We observed that small variations in queue weight have significant influence on the average RTT.

Table 5. System variable for various w_q.

Parameters	Average RTT (msec)			Sending rate (packets/sec)			Drop rate (%)		
weight (w_q)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)
0.001	40.3	36.1	11.63	384.99	384.71	0.073	0.55	0.54	1.29
0.002	39.9	36.0	10.83	384.98	384.77	0.056	0.56	0.55	2.56
0.004	39.4	36.2	8.80	385.11	384.79	0.083	0.59	0.56	6.12
0.006	39.0	35.8	8.93	385.08	384.73	0.093	0.60	0.56	7.91
0.008	39.0	35.8	8.90	385.10	384.68	0.109	0.61	0.55	11.11
0.010	38.9	35.7	8.96	385.02	384.70	0.083	0.61	0.55	11.72

4.3.3 Validation for various drop probabilities p_{max}

In this simulation scenario, we evaluated the S-RED model for various p_{max}. Other parameters remain as listed in Table 1. When the maximum drop probability is set to a very small value, the RED algorithm has only a small influence on the system behaviour. In this case, the system behaves similarly to TCP with DropTail, which leads to bursty packet losses and longer queuing delays. However, if the value of p_{max} is close to one, a high dropping rate will cause the system to become unstable and decrease the throughput.

We show the waveforms of the window size with $p_{max} = 0.5$. Figure 14 gives the results from the S-RED model and Figure 15 gives the results from the ns-2 simulation. The results of the average queue size are shown in Figure 16. The average queue size decreases as the maximum drop rate increases. Results obtained from the S-RED model and ns-2 simulation results show the same trend.

Figure 14. S-RED model: evolution of the window size for $p_{max} = 0.5$.

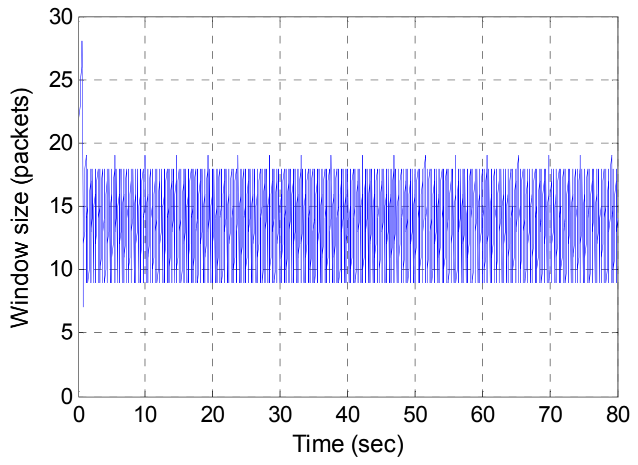


Figure 15. ns-2: evolution of the window size for $p_{max} = 0.5$.

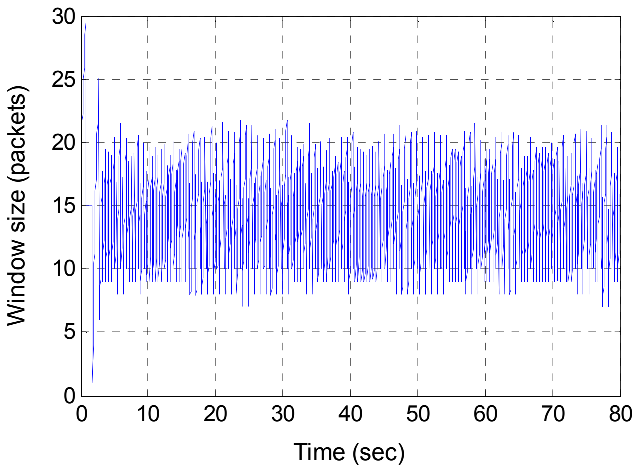
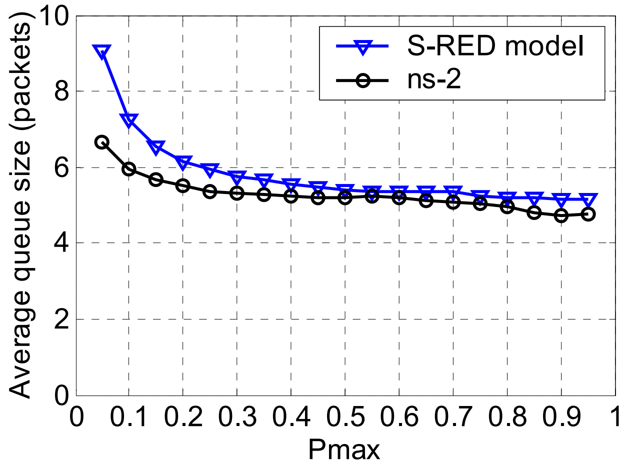


Figure 16. Comparison of the average queue size for various p_{max}.



Statistical analyses of the two state variables are listed in Tables 6 and 7. When p_{max} increases, the average value of the window size and the average queue size decrease.

Table 6. Statistical analysis of the window size for various p_{max}.

Parameters	Average window size (packets)			Variance: $\frac{n\sum x^2 - (\sum x)^2}{n(n-1)}$	
p_{max}	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
0.05	17.02	15.67	8.61	14.03	12.61
0.10	15.35	15.01	2.25	11.27	10.71
0.25	14.00	14.40	-2.77	10.01	10.12
0.50	13.54	14.17	-4.44	8.44	11.04
0.75	13.37	13.62	-1.84	8.63	16.51

Table 7. Statistic analysis of the average queue size for various p_{max}.

Parameters	Average queue size (packets)			Variance: $\frac{n\sum x^2 - (\sum x)^2}{n(n-1)}$	
p_{max}	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
0.05	8.05	6.66	20.87	0.046	0.082
0.10	7.24	5.95	21.63	0.027	0.046
0.25	5.94	5.36	10.82	0.016	0.029
0.50	5.39	5.21	3.45	0.013	0.044
0.75	5.25	5.00	5.00	0.015	0.417

Validation results for the average RTT, sending rate, and drop rate are listed in Table 8. The results show that the system variables in our S-RED model change in a similar manner to the results obtained from the ns-2 simulations. As expected, when the maximum drop probability increases, the actual drop rate increases. It leads to a decrease of the sending rate. At the same time, the average RTT decreases, which indicates a lower queuing delay.

Table 8. System parameters for various p_{max}.

	Average RTT (msec)			Sending rate (packets/sec)			Drop rate (%)		
p_{max}	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)
0.05	44.3	38.1	16.27	385.13	384.70	0.11	0.45	0.51	-11.76
0.10	39.9	36.0	10.83	384.98	384.77	0.06	0.56	0.55	2.56
0.25	36.5	34.5	5.80	384.93	384.73	0.05	0.65	0.59	11.28
0.50	35.3	34.0	3.80	384.98	379.37	1.48	0.73	0.61	19.09
0.75	34.8	35.1	-0.85	384.63	357.55	7.60	0.74	0.65	14.37

4.3.4 Validation with various thresholds q_{min} and q_{max}

In this validation scenario, the model is evaluated for various queue thresholds. Values of q_{min} and q_{max} are changed simultaneously, while maintaining the ratio $q_{max}/q_{min} = 3$, as recommended in [2]. Here, we show the waveform of the window size of the S-RED model (Figure 17) and ns-2 (Figure 18), with $q_{min} = 10$ packets and $q_{max} = 30$ packets. The simulation results for the average queue size are shown in Figure 19.

Figure 17. S-RED model: evolution of the window size for $q_{min} = 10$ packets.

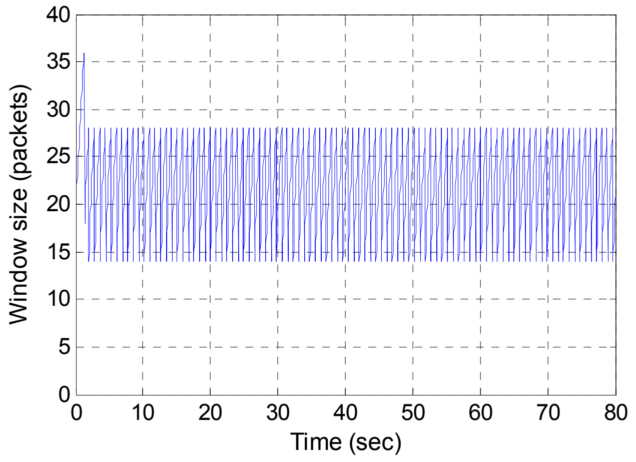


Figure 18. ns-2: evolution of the window size for q_{min} = 10 packets.

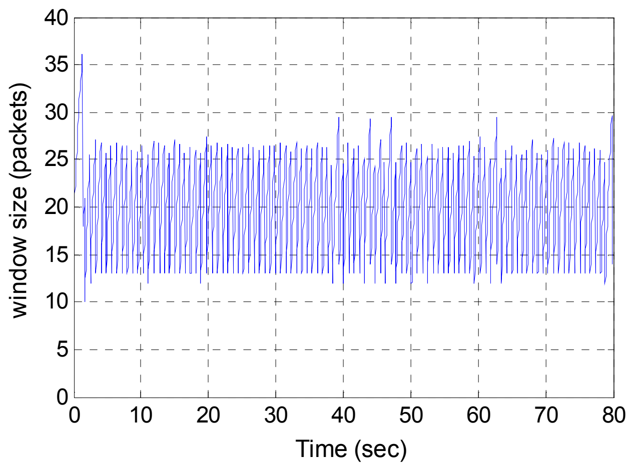
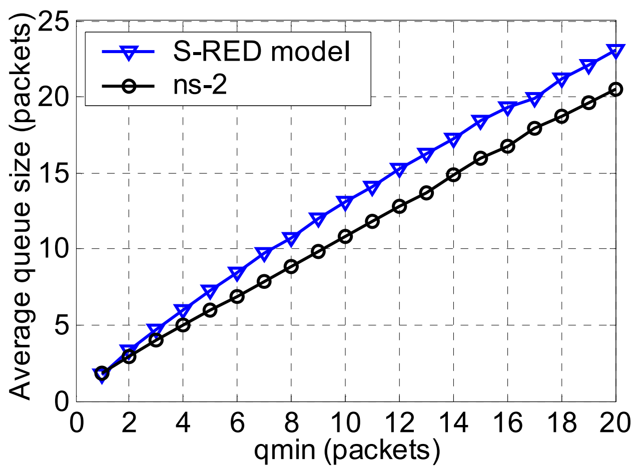


Figure 19. Comparison of the average queue size for various q_{min} and q_{max}.



The statistical analysis for the window size and the average queue size are given in Tables 9 and 10.

Table 9. Statistical analysis of the window size for various q_{max} *and* q_{min}.

Parameters	Average window size (packets)			Variance: $\frac{n\sum x^2-(\sum x)^2}{n(n-1)}$	
q_{min} (packets)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
3	12.75	13.03	-2.14	8.44	8.53
5	15.35	15.01	2.25	11.27	10.71
10	21.00	19.61	7.08	18.72	17.28
15	26.00	24.28	7.08	30.34	31.35
20	30.52	29.06	5.02	40.31	38.64

Table 10. Statistical analysis of the average queue size for various q_{max} *and* q_{min}.

Parameters	Average queue size (packets)			Variance: $\frac{n\sum x^2-(\sum x)^2}{n(n-1)}$	
q_{min} (packets)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2
3	4.67	3.97	17.63	0.013	0.023
5	7.24	5.95	21.63	0.027	0.046
10	13.12	10.77	21.81	0.123	0.175
15	18.39	15.86	15.95	0.474	1.128
20	23.05	20.55	12.16	1.113	1.650

When q_{max} and q_{min} increase, which implies that more packets can be admitted to the queue without being dropped and marked, the window size, instantaneous queue size, and average queue size, together with their variances, increase.

As shown in Table 11, the average RTT increases significantly as a result of the increase in instantaneous queue length in the buffer, when the thresholds q_{max} and q_{min} increase. At the same time, the drop rate reduces and the sending rate increases slowly. Hence, if the buffer is large enough, increasing the two thresholds can help to increase the sending rate and decrease the drop rate; however, this is at the cost of long delays and bursty traffic.

Table 11. Statistic analysis of the average queue size for various w_q.

	Average RTT (msec)			Sending rate (packets/sec)			Drop rate (%)		
q_{min} (packets)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)	S-RED model	ns-2	Δ (%)
3	33.4	31.1	7.40	383.22	382.44	0.20	0.78	0.71	10.01
5	39.9	36.0	10.83	384.98	384.77	0.06	0.56	0.55	2.56
10	54.7	48.1	13.72	385.10	384.85	0.06	0.31	0.33	-6.34
15	67.7	60.3	12.27	385.06	384.83	0.06	0.20	0.22	-10.71
20	79.1	73.0	8.36	385.30	384.95	0.09	0.15	0.16	-5.66

4.3.5 Validation summary

We evaluated the S-RED model using MATLAB and compared its performance to the ns-2 simulation results. First we evaluated the waveforms of the two state variables: window size and average queue size. While the window sizes matched quite well, there were differences in the average queue sizes during the steady-state. The difference is due to the simplifications to the RED's dropping algorithm that we made in the S-RED model. Nevertheless, the average queue size in the S-RED model and ns-2 results have the same trend when system parameters vary. We have also performed three sets of tests based on various system parameters: w_q, p_{max}, q_{min} and q_{max}. System variables RTT, sending rate, and drop rate have been evaluated. The comparisons with the ns-2 simulation results show good agreement.

4.4 Comparison of S-RED model and M-model

In this section, we compare our S-RED model with the Maryland model, termed M-model in this thesis, which is also a discrete nonlinear dynamical model of TCP Reno with RED gateway [24], [39], [40]. A brief description of the M-model is given in Section 3.2. We have compared S-RED model with ns-2 results in Section 4.3. In this section, we compare S-RED model, M-model, and ns-2 simulation results. S-RED model and M-model are evaluated using MATLAB.

The comparison study is also divided into four stages: with default RED parameters, with various queue weights w_q, with various drop probabilities p_{max}, and with various thresholds q_{min} and q_{max}.

The default RED parameters are listed in Table 1. Other system parameters corresponding to S-RED model and M-model are summarized in Table 12.

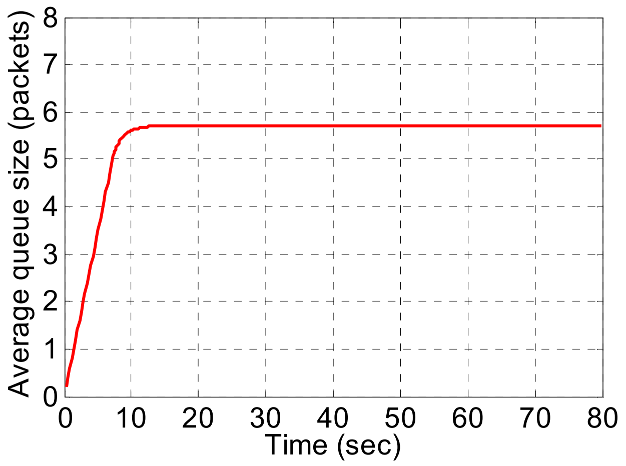
Table 12. System parameters.

	S-RED model	M-model
Link capacity	1.54e+6	1.54e+6
Packet size	500	500
Round-trip delay	0.0228	0.0228
Buffer size	_	100
Slow start	20	_
Number of TCP	1	1
Constant: K	_	$\sqrt{3/2}$

4.4.1 Comparison for default RED parameters

The M-model [24], [39], [40] is a first-order discrete dynamical model whose state variable is the average queue size. Hence, we can only observe the time waveform of the average queue size (Figure 20).

Figure 20. M-model: evolution of the average queue size with default RED parameters.

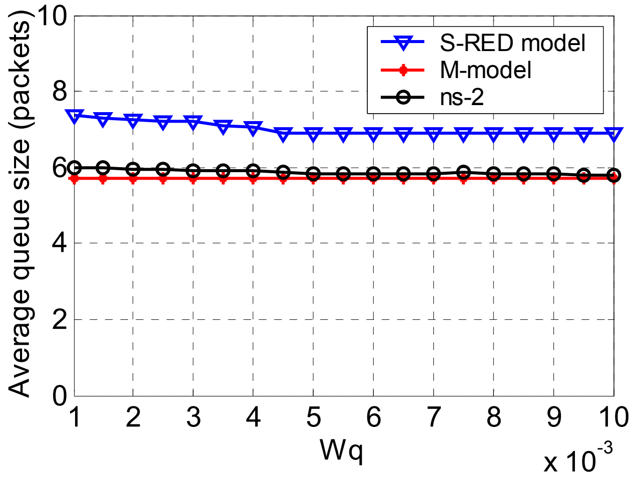


The value of the average queue size of the M-model closely matches the ns-2 results shown in Figures 13 (c) and (d). After the transient state, the average queue size reaches a constant value (5.71 packets), while the average queue size for the ns-2 simulations varies around its fixed point. Conversely, the proposed S-RED model captures the dynamical characteristics of the average queue size.

4.4.2 Comparison for various queue weights w_q

In this simulation set, we varied the queue weight w_q from 0.001 to 0.01, while keeping the parameters listed in Table 1 and Table 12 fixed. The simulation results of the average queue size for various w_q are shown in Figure 21 with the results of S-RED model and ns-2.

Figure 21. Comparison of the average queue size for various w_q.



The average values of the average queue size of M-model are closer to ns-2 than those of S-RED model with various w_q. However, for each fixed w_q , the average queue size of M-model keeps constant during steady state and can not show the dynamical behaviour of the system.

The M-model was then evaluated for other system variables, namely the average RTT, sending rate and packet drop rate. Results are summarized in Table 13 with the results obtained from S-RED model and ns-2 simulations. Δ is the difference between S-RED model with ns-2 and between M-model with ns-2.

Table 13. System variables for various w_q.

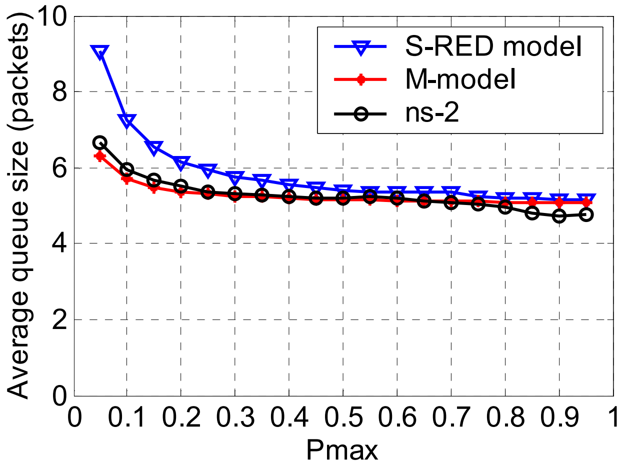
Parameters	Average RTT (msec)				
weight (w_q)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.001	40.3	11.63	45.8	26.87	36.1
0.002	39.9	10.83	41.3	14.72	36.0
0.004	39.4	8.8	39.4	8.84	36.2
0.006	39.0	8.93	38.8	8.38	35.8
0.008	39.0	8.9	38.5	7.54	35.8
0.010	38.9	8.96	38.3	7.28	35.7
Parameters	Sending rate (packets/sec)				
weight (wq)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.001	384.99	0.07	325.89	-15.29	384.71
0.002	384.98	0.06	355.26	-7.67	384.77
0.004	385.11	0.08	370.04	-3.83	384.79
0.006	385.08	0.09	374.90	-2.55	384.73
0.008	385.10	0.11	377.25	-1.93	384.67
0.010	385.02	0.08	378.37	-1.65	384.70
Parameters	Drop rate (%)				
weight (w_q)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.001	0.55	1.29	0.67	23.39	0.54
0.002	0.56	2.56	0.70	28.21	0.55
0.004	0.59	6.12	0.71	27.70	0.56
0.006	0.60	7.91	0.71	27.70	0.56
0.008	0.61	11.11	0.71	29.33	0.55
0.010	0.61	11.72	0.71	30.04	0.55

As shown in Table 13, we also evaluated the M-model for the same system variables: average RTT, sending rate, and packet drop rate. Except for w_q = 0.001 and 0.002, when the S-RED model performs better, the discrepancy in predicting the RTT for the two models are similar. In all cases, the S-RED model more accurately predicts the sending and drop rates.

4.4.3 Comparison for various drop probabilities p_{max}

In this comparison study, we compare the M-model, the S-RED model, and ns-2 simulation results for various drop probabilities p_{max}. Other parameters remain as listed in Tables 1 and 13. The average queue size for various p_{max} is shown in Figure 22.

Figure 22. Comparison of the average queue size for various p_{max}.



The average values of the average queue size of M-model match better the ns-2 results than the of S-RED model for various p_{max}. Similar to Figure 20, for each fixed p_{max}, the average queue size of M-model is constant during steady state. It is not able to show the detail dynamics of the system.

The average RTT, sending rate, and drop rate for the M-model are also summarized in Table 14. With various drop probabilities p_{max}, the S-RED model better estimates the average RTT, the sending rate, and the drop rate.

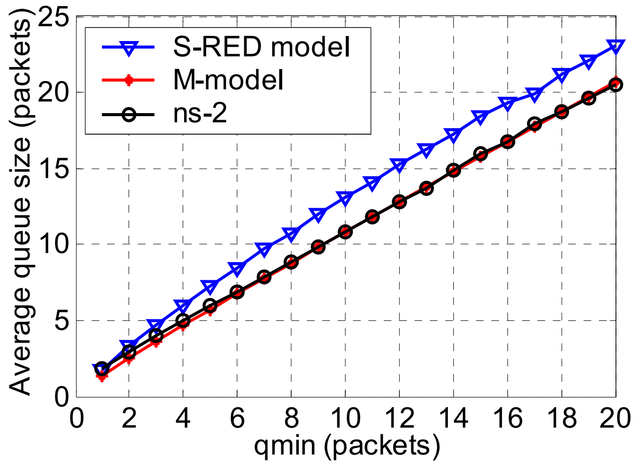
Table 14. System variables for various p_{max}.

Parameters	Average RTT (msec)				
p_{max}	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.05	44.3	16.27	43.4	13.91	38.1
0.10	39.9	10.83	41.3	14.72	36.0
0.25	36.5	5.80	39.9	15.65	34.5
0.50	35.3	3.80	39.4	15.88	34.0
0.75	34.8	-0.85	39.2	11.68	35.1
Parameters	Sending rate (packets/sec)				
p_{max}	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.05	385.13	0.11	354.23	-7.92	384.70
0.10	384.98	0.06	355.26	-7.67	384.77
0.25	384.93	0.05	356.02	-7.46	384.73
0.50	384.98	1.48	356.27	-6.09	379.37
0.75	384.63	7.60	356.33	-0.34	357.55
Parameters	Drop rate (%)				
p_{max}	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
0.05	0.45	-11.76	0.63	23.53	0.51
0.10	0.56	2.56	0.70	28.21	0.55
0.25	0.65	11.28	0.74	26.50	0.59
0.50	0.73	19.09	0.76	23.98	0.61
0.75	0.74	14.37	0.77	19.01	0.65

4.4.4 Comparison for various thresholds q_{min} and q_{max}

In this subsection, we investigate the comparison of the S-RED model, M-model, and ns-2 results for various queue thresholds. We change the values of the thresholds q_{min} and q_{max} simultaneously, keeping the ratio q_{max}/q_{min} fixed at 3, as recommended in [2], and setting the other parameters to the values in Tables 1 and 12. The average queue size for various q_{min} and q_{max} is shown in Figure 23. The sending rate, RTT, and drop rate are summarized in Table 15.

Figure 23. Comparison of the average queue size for various q_{min} and q_{max}.



A comparison with the M-model suggests that the proposed S-RED model is more accurate in predicting average RTT, sending rate, and drop rate. However, M-model has a better prediction for the average queue size during steady state. Similar to Figure 20, for each fixed q_{min} and q_{max}, the average queue size of M-model stays constant. Therefore, S-RED model can capture more dynamical details of the system than M-model.

Table 15. System parameters with various q_{min} and q_{max}.

Parameters	Average RTT (msec)				
q_{min} (packets)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
3	33.4	7.4	34.1	9.65	31.1
5	39.9	10.83	41.3	14.72	36.0
10	54.7	13.72	60.8	26.40	48.1
15	67.7	12.27	83.1	37.81	60.3
20	79.1	8.36	109.1	49.45	73.0
Parameters	Sending rate (packets/sec)				
q_{min} (packets)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
3	383.22	0.20	366.13	-4.26	382.44
5	384.98	0.06	355.26	-7.76	384.77
10	385.096	0.06	330.94	-14.01	384.85
15	385.06	0.06	311.01	-19.19	384.83
20	385.30	0.09	296.27	-23.04	384.95
Parameters	Drop rate (%)				
q_{min} (packets)	S-RED model	Δ (%)	M-model	Δ (%)	ns-2
3	0.78	10.01	0.96	35.40	0.71
5	0.56	2.56	0.70	28.21	0.55
10	0.31	-6.34	0.37	11.78	0.33
15	0.20	-10.71	0.22	-1.79	0.22
20	0.15	-5.66	0.14	-11.95	0.16

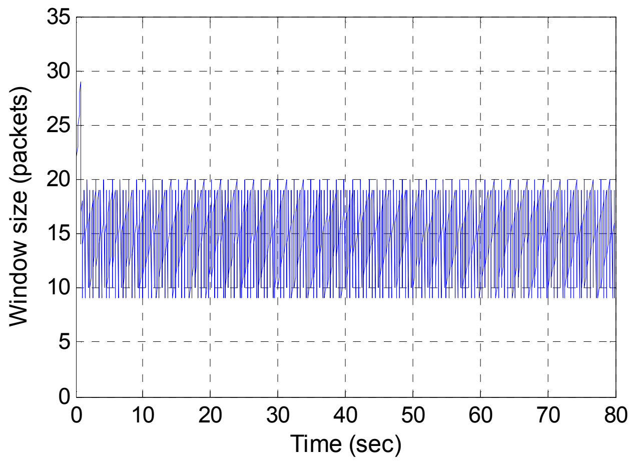
4.5 Model modification

The difference in the average queue size between the S-RED model and ns-2 is due to the simplifications to the RED's packet discarding algorithm. The S-RED model employs the probability p_b (Eq. 4) as the final drop probability, while RED in ns-2 uses p_a (Eq. 5). If a modified drop probability $p_a = \alpha \cdot p_b$ is used, the window size and the average queue size would evolve as shown in Figure 24 (a) and (b), respectively (with the default RED parameter settings).

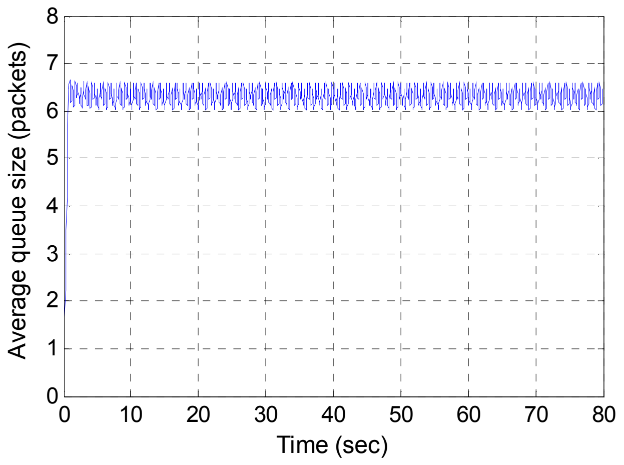
Comparisons show that the average queue size well matches the ns-2 simulation results for modified drop probabilities with $\alpha = 1.8$.

Figure 24. S-RED model with modification: (a) evolution of the window size and (b) the average queue size.

(a)



(b)



For further detail, we show a comparison of the average queue size of the modified model with the original S-RED model, modified S-RED model, and ns-2, for various w_q, p_{max}, and q_{min} in Figures 25, 26, and 27, respectively. From the following figures, we observed that after modifying the drop probability, the modified model has better agreement with ns-2 simulation results.

Figure 25. Comparison of the average queue size for various w_q.

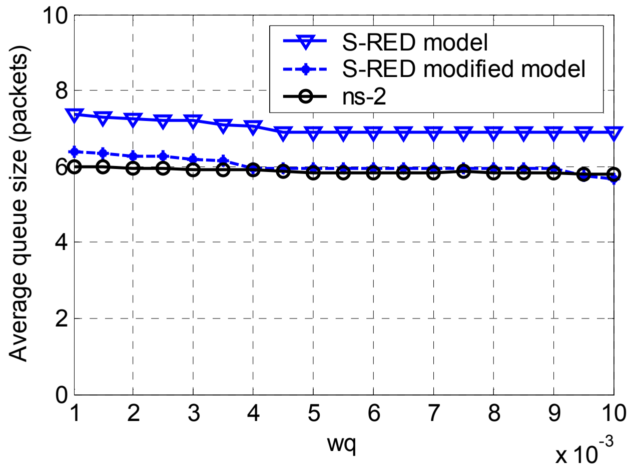


Figure 26. Comparison of the average queue size for various p_{max}.

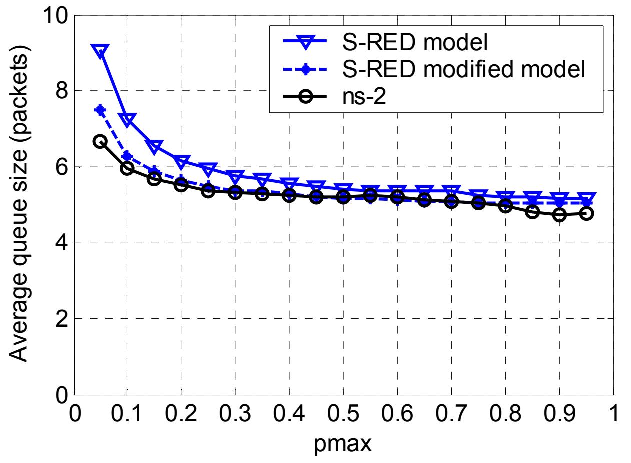
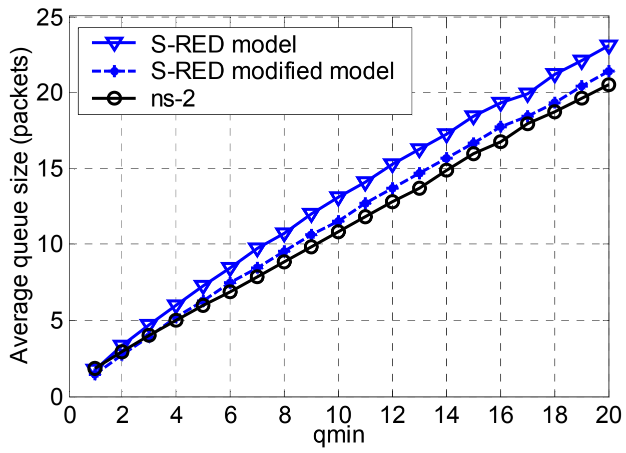


Figure 27. Comparison of average queue size for various q_{min} and q_{max}.



4.6 Model extension: TCP with Adaptive RED (S-ARED model)

Adaptive RED (ARED) [15] is an extension to the RED algorithm proposed for improving RED's robustness. One of the main weaknesses of RED is that the average queue size is too sensitive to the parameter settings and the level of congestion. When the link load is light or p_{max} is high, the average queue size is closer to q_{min}. On the other hand, if the link load is heavy or p_{max} is low, then the average queue size is near or even greater than q_{max} [15]. As queuing delay is determined by the average queue size, the delay is also very sensitive to the parameters and is not easy to predict and control. Thus, the goal of ARED is to reduce the sensitivity of the parameters that affect RED performance and achieve a stable and predictable average queue size based on the RED algorithm. The main idea of this approach is to adjust p_{max} in response to dynamical changes of the average queue size. The robustness of this algorithm depends on the frequency of adjusting p_{max}. The pseudo-code of ARED algorithm was given in Section 2.2.4.

The ARED algorithm is an extension to the RED algorithm with the modification of p_{max}. We derived a TCP/ARED model based on our S-RED model discussed in Section 4.1. We call it S-ARED model. The pseudo-code for the S-ARED model is described in Algorithm 4. We kept most of the concepts and assumptions of the proposed S-RED model and incorporated the modification of the system parameter p_{max}. The value of the interval, which refers to the frequency of updating the value of p_{max}, is a function of $C \cdot RTT$, where C is a constant.

Algorithm 4: The pseudo-code for the S-ARED model.

```
Initialization:
        $\overline{q}_0 \leftarrow 0;$
        $q_0 \leftarrow 0;$
        $p_0 \leftarrow 0;$
        $sample\_interval \leftarrow C;$
        $part \leftarrow 0.4(q_{max} - q_{min});$
        $\alpha \leftarrow min(0.01, p_{max}/4);$
        $\beta \leftarrow 0.9;$
for every round
{
        if $sample\_interval == 0$
        {
                if $\overline{q} > q_{max} - part$ && $p_{max} <= 0.05$
                {
                        $p_{max} \leftarrow p_{max} + \alpha$
                }
                else if $\overline{q} < q_{min} + part$ && $p_{max} >= 0.01$
                {
                        $p_{max} \leftarrow p_{max} \cdot \beta$
                }
                $sample\_interval = C$
        }
        else
        {
                $sample\_interval = C-1;$
        }
        calculate $p_k$
        calculate the product of $p_k \cdot W_k$
        if $p_k \cdot W_k < 0.5$
        {
                compare $W_k$ with ssthresh
                if $W_k$ < ssthresh
                {
                    $W_{k+1} \leftarrow \min(2W_k, ssthresh)$
                    $\overline{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \overline{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                }
                else
                {
                    $W_{k+1} \leftarrow \min(W_k + 1, rwnd)$
                    $\overline{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \overline{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                }
                calculate the drop probability using Eq. (4)
        }
        elseif $0.5 \le p_k \cdot W_k < 1.5$
        {
                $W_{k+1} \leftarrow \frac{1}{2} W_k$
                $\overline{q}_{k+1} = (1-w_q)^{W_{k+1}} \cdot \overline{q}_k + (1-(1-w_q)^{W_{k+1}}) \cdot \max(W_{k+1} - \frac{C \cdot d}{M}, 0)$
                calculate the drop probability using Eq. (4)
        }

        else
        {
                $W_{k+1} \leftarrow 0$
                $\overline{q}_{k+1} \leftarrow \overline{q}_k$ .
              calculate the drop probability using Eq. (4)

        }
}
```

4.6.1 S-ARED model validation

In order to validate the S-ARED model, we evaluated the model using MATLAB and compared its performance with the ns-2 simulation results. For consistency, we used the same simulation topology as shown in Figure 8.

In order to evaluate how accurately the S-ARED model can capture the dynamical behaviour of the system, we studied the time waveforms of the two state variables: window size and average queue size. The ns-2 simulation results are obtained using default ARED parameters, listed in Table 16.

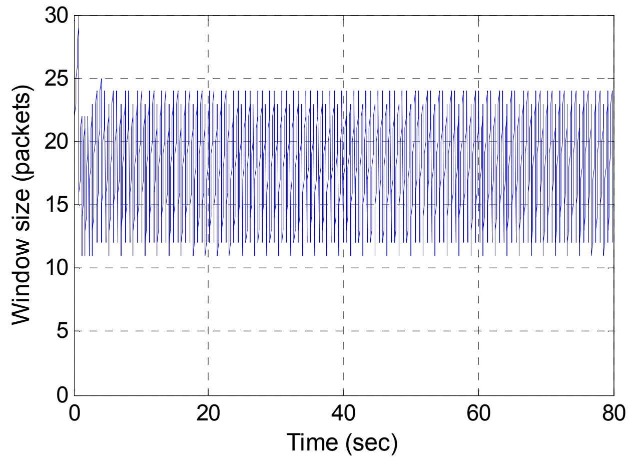
Table 16. Default ARED parameters in ns-2.

Packet size M (bytes)	500
Maximum drop probability p_{max}	0.1
Minimum queue threshold q_{min} (packets)	5
Maximum queue threshold q_{max} (packets)	15
Queue weight w_q	0.002
Sample_interval C	10

Figures 28 and 29 show the waveforms of the simulation results for the window size and the average queue size for the S-ARED model and ns-2, respectively.

Figure 28. Evolution of the window size: (a) S-ARED and (b) ns-2 simulation results.

(a)



(b)

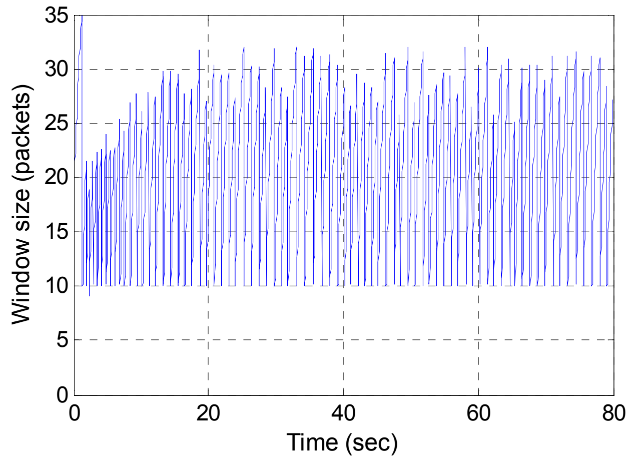
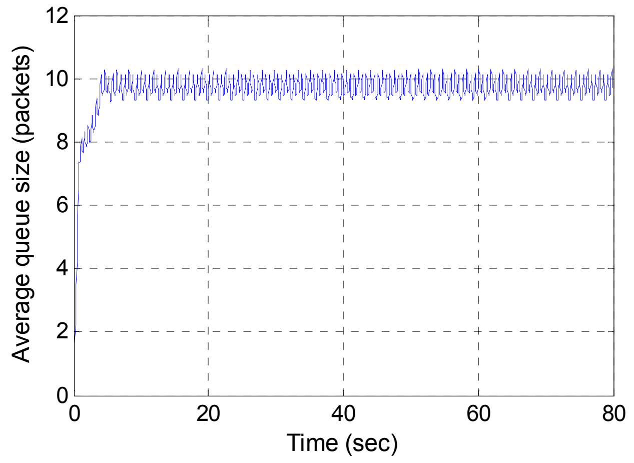
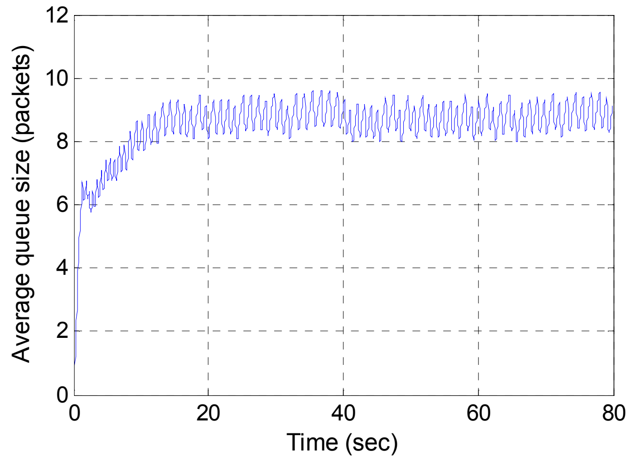


Figure 29. Evolution of the average queue size: (a) S-ARED and (b) ns-2 simulation results.

(a)



(b)



The waveforms for the window size and the average queue size in ns-2 simulations oscillate more severe than those of S-ARED. A detailed comparison of simulation results of S-ARED and ns-2, for the two state variables and three system parameters (RTT, sending rate, and drop rate) are listed in Table 17.

Table 17. Average values of the window size, average queue size, and system parameters.

	S-ARED model	ns-2	Δ (%)
Window size (packets)	17.66	20.10	-12.13
Average queue size (packets)	9.69	8.69	11.50
RTT (msec)	45.90	44.20	3.84
Sending rate (packets/sec)	385.03	377.41	2.02
Drop probability (%)	0.44	0.26	69.23

The RTT and sending rate of the S-ARED model and ns-2 match quite well. The difference in the average queue size remains to be one packet. Due to the different drop probability, the difference in the average value of the window size is more than two packets. The S-ARED model still captures the dynamical behaviour of the system

4.6.2 Model evaluation

The purpose of the ARED algorithm is to maintain the average queue size at a specified target value which is approximately $(q_{max} + q_{min})/2$. From the results, the average queue size in the S-ARED model is 9.69 packets. This is close to the target value of 10 packets. Compared to the results of 7.24 packets from the S-RED model (Table 2), the average queue size is less dependent on the parameter q_{min} in ARED algorithm than in RED algorithm. Also, from the evaluation results, S-ARED model exhibits an improved sending rate and the drop probability. However, S-ARED model is quite sensitive to the sampling interval of p_{max}. If the sampling interval is small, the TCP/ARED system can have a faster response to the network. As a result, it is less stable. Conversely, if the sampling interval is too long, the system behaves like TCP/RED.

CHAPTER FIVE: CONCLUSIONS AND FUTURE WORK

The TCP/RED system can be viewed as a complex feedback control system, where TCP adjusts its sending rate depending on the packet loss probability determined by RED. In this thesis, we have introduced a second-order discrete-time analytical model for the interaction between TCP Reno and RED algorithms. We used an iterative map to construct the discrete-time model of the system. The S-RED model captures the dynamical behaviour of TCP/RED and may be used to study its nonlinear behaviour. Unlike other models, it takes into account the TCP slow start and timeout events.

We evaluated the model by comparing its performance to ns-2 simulations. Validation of the S-RED model illustrates the performance of the model for various RED parameters. The comparison shows that our S-RED model leads to similar results as ns-2, particularly for the window size, sending rate, RTT, and drop rate. However, there is difference in the average queue size due to the model simplification. In Section 4.5, after modifying the S-RED model, both state variables, the window size and the average queue size, match well the ns-2 simulations.

We compared the S-RED model to the M-model, an existing first-order discrete-time analytical TCP/RED model. Although the average value of the average queue size of the M-model matches the ns-2 simulation results better than the S-RED model, the S-RED model captures more dynamical details of the TCP and RED algorithms. Moreover, for the window size, sending rate, RTT, and drop rate, the S-RED model outperforms the M-model.

We also discussed a discrete-time model for TCP and ARED (S-ARED) based on the proposed S-RED model. The S-ARED model can be considered as an extension to the S-RED model. The S-ARED model is validated by comparing it to the ns-2 simulation results. From the

S-ARED model, we find that the ARED algorithm can help reduce some weaknesses of the RED algorithm. The average queue size is less dependent on the system parameter q_{min}, especially when the load of the network is light. Thus, it is easier to predict the average queue size as well as the link delay. The S-ARED model can achieve a higher throughput and a lower drop rate. However, the sampling interval of p_{max} can affect the stability of the system.

Based on the comparisons of the results, we conclude that the S-RED and S-ARED models can capture the dynamical behaviour of TCP/RED system and provide reasonable predictions of system parameters.

There are two avenues for future work. Firstly, the new discrete-time model may be used to study the non-linear phenomena of the system, such as bifurcation and chaos. Secondly, given the flexibility of the S-RED model, it can be used to evaluate TCP/RED based AQM schemes and develop new and improved algorithms.

REFERENCE LIST

[1] M. Allman, V. Paxson, and W. Steven, "TCP Congestion Control," *Request for Comment (RFC) 2581*, Apr. 1999.

[2] B. Barden et al., "Recommendations on queue management and congestion avoidance in the Internet," *Request for Comment (RFC) 2309*, Apr. 1998.

[3] M. di Bernardo and C. K. Tse, "Chaos in power electronics: an overview," in *Chaos in Circuits and Systems*, New York: World Scientific, pp. 317-340, 2002.

[4] R. Braden, "Requirements for Internet hosts: communication layers," *Request for Comment (RFC) 1122*, Oct. 1989.

[5] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 8, pp. 1465-1480, Oct. 1995.

[6] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proc. IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, vol. 3, pp. 1742-1751.

[7] C. Casetti, M. Gerla, S. Lee, S. Mascolo, and M. Sanadidi, "TCP with faster recovery," in *Proc. MILCOM 2000*, Los Angeles, CA, USA, Oct. 2000, vol. 1, pp. 320-324.

[8] C. Casetti and M. Meo, "A new approach to model the stationary behavior of TCP connections," in *Proc. INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, vol. 1, pp. 367-375.

[9] W. C. Y. Chan and C. K. Tse, "Study of bifurcation in current-programmed DC/DC boost converters: from quasi-periodicity to period-doubling," *IEEE Trans. Circuit and Systems I*, vol. 44, no. 12, pp. 1129-1142, Dec. 1997.

[10] K. W. E. Cheng, M. Liu, and J. Wu, "Chaos study and parameter-space analysis of the DC-DC buck-boost converter," *IEE Proceedings-Electric Power Applications*, vol. 150, pp. 126-138, Mar. 2003.

[11] K. Fall and S. Floyd, "Simulation-based comparison of Tahoe, Reno, and SACK TCP," *ACM Communication Review*, vol. 26, no. 3, pp. 5-21, July 1996.

[12] K. Fall and K. Varadhan, "The ns Manual," UC Berkeley, LBL, USC/ISI, and Xerox PARC, June 2003: http://www.isi.edu/nsnam/ns/doc/ns doc.pdf.

[13] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The BLUE Active Queue Management algorithm", *IEEE/ACM Trans. Networking*, vol. 10, no. 4, pp. 513-528, Aug. 2002.

[14] V. Firoiu and M. Borden, "A study of active queue management for congestion control," *in Proc. IEEE INFOCOM 2000*, Tel-Aviv, Israel, Mar. 2000, vol. 3, pp. 1435-1444.

[15] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: an algorithm for increasing the robustness of RED's Active Queue Management," Aug. 2001:

http://www.icir.org/floyd/papers/.

[16] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397-413, Aug. 1993.

[17] S. Floyd, "RED: discussions of setting parameters," Nov. 1997:

http://www.icir.org/floyd/REDparameters.txt.

[18] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "A control theoretic analysis of RED," in *Proc. IEEE INFOCOM 2001*, Anchorage, AK, Apr. 2001, vol. 3, pp. 1510-1519.

[19] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Trans. on Automatic Control*, vol. 47, no. 6, pp. 945-959, June 2002.

[20] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, no. 4, pp. 314-329, Aug. 1988.

[21] V. Jacobson, "Modifed TCP congestion avoidance algorithm,"

ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail, Apr. 1990.

[22] I. Khalifa and Lj. Trajković, "An overview and comparison of analytical TCP models," in *Proc. IEEE International Symposium on Circuits and Systems*, Vancouver, BC, Canada, May 2004, vol. V, pp. 469-472.

[23] P. Kuusela, P. Lassila, J. Virtamo, and P. Key, "Modeling RED with idealized TCP sources," in *9th IFIP Conference on Performance Modeling and Evaluation of ATM and IP Networks 2001*, Budapest, Hungary, June 2001, pp. 155-166.

[24] R. J. La, P. Ranjan, and E. H. Abed, "Nonlinearity of TCP and instability with RED," in *Proc. SPIE ITCom*, Boston, MA, USA, July 2002, pp. 283-294.

[25] Y. Lai and C. Yao, "TCP congestion control algorithms and a performance comparison," in *Proc. 10th International Conference on Computer Communications and Networks*, Scottsdale, AZ, USA., Oct. 2001, pp. 523-526.

[26] S. H. Low and D. E. Lapsley, "Optimization flow control I: basic algorithm and convergence," *IEEE/ACM Trans. Networking*, vol. 7, no. 6, pp. 861-874, Dec. 1999.

[27] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of TCP/RED and a scalable control," in *Proc. IEEE INFOCOM 2002*, New York, NY, USA., June 2002, vol. 1, pp. 239-248.

[28] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, pp. 67-82, July 1997.

[29] MATLAB 7: http://www.mathworks.com.

[30] M. Mellia, I. Stoica, and H. Zhang, "TCP model for short lived flows," *IEEE Communications Letters*, vol. 6, no. 2, pp. 85-87, Feb. 2002.

[31] V. Misra, W. B. Gong, and D. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *Proc. ACM SIGCOMM 2000*, Stockholm, Sweden, Aug. 2000, pp. 151-160.

[32] A. Misra, J. Baras, and T. Ott, "Generalized TCP congestion avoidance and its erect on bandwidth sharing and variability," in *Proc. IEEE GLOBECOM*, San Francisco, CA, USA., Nov. 2000, vol. 1, pp. 329-337.

[33] ns-2 Network Simulator: http://www.isi.edu/nsnam/ns.

[34] J. Padhye, V. Firoiu, and D. F. Towsley, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Networking*, vol. 8, no. 2, pp. 133-145, Apr. 2000.

[35] J. Padhye and S. Floyd, "On inferring TCP behavior," in *Proc. ACM SIGCOMM 2001*, San Diego, CA, USA, Aug. 2001, pp. 287-298.

[36] R. Pan, K. Psounis, and B. Prabhakar, "CHOKe, A stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, vol. 2, pp. 942-951.

[37] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," *Request for Comment (RFC) 2988*, Nov. 2000.

[38] J. Postel, "Transmission control protocol," *Request for Comment (RFC) 793*, Sept. 1981.

[39] P. Ranjan, E. H. Abed, and Richard J. La, "Nonlinear instabilities in TCP-RED," in *Proc. IEEE INFOCOM 2002*, New York, NY, USA, June 2002, vol. 1, pp. 249-258.

[40] P. Ranjan, R. J. La, and E. H. Abed, "Bifurcations of TCP and UDP traces under RED," in *Proc. 10 th Mediterranean Conference on Control and Automation (MED) 2002*, Lisbon, Portugal, July 2002.

[41] R. Roy, R. C. Mudumbai, and S. S. Panwar, "Analysis of TCP congestion control using a fluid model," in *Proc. IEEE ICC 2001*, Helsinki, Finland, June 2001, vol. 8, pp. 2396-2403.

[42] W. R. Stevens, *TCP/IP Illustrated*, *Volume 1: The protocols*. New York: Addison-Wesley, 1994.

[43] C. K. Tse and M. D. Bernardo, "Complex behavior in switching power converters," *Proceedings of the IEEE*, vol. 90, no. 5, pp. 768-781, May 2002.