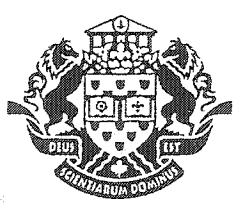



Université d'Ottawa • University of Ottawa

# Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Yiqun DING

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. Sc.(Systems Science)

GRADE - DEGREE

Systems Science program

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Solver for TCP Flows in a Network

D. McDonald

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

A. Dabrowski

F. Theberge

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

# SOLVER FOR TCP FLOWS IN A NETWORK

By
Yiqun Ding
May 2004

A Thesis
submitted to Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of
Master of Science in System Science

*Your file Votre référence*
*ISBN: 0-494-01459-8*
*Our file Notre référence*
*ISBN: 0-494-01459-8*

# Abstract

Quality of Service (QOS) of a congested network can be evaluated by some key factors such as packets discarding rate, queueing delay and the throughput of a bottleneck buffer. A network designer needs to predict these parameters in order to dimension a network properly.

This thesis developed a solver for predicting the stability of a TCP network when servers implement Active Queue Management(AQM) schemes. The solver analyzes two AQM schemes. One is Random Early Detection (RED) and the other is Dynamic Random Early Detection (DRED). We verify the analysis of the solver through the simulation of a small network with multiple bottlenecks carrying a large number of TCP connections.

# Acknowledgements

I would like to thank my supervisor Professor David McDonald took me under his wing and for all his priceless guidance and advice.

I would also like to thank Micheal Maskery for his many kindness help in OPNET simulation.

# Dedication

To Jun. For your enthusiastic help and support.

To My Parents and Parents in law. For the love they give me and my son.

# Contents

# List of Figures

# Chapter 1

# Introduction

The fast expansion of networks such as the internet is a consequence of user demand for higher speed communication services. Higher data transfer rates cause congestion or packet queueing in the buffers of the network. Congestion, therefore, may cause excessive delay in the network and loss of packets, and will degrade the quality of service provided to the user. Congestion is a function of several network characteristics including the number of nodes, the flow control policy, the retransmission policy, the discard strategy, router selection, propagation delay, processing delay and so on. The purpose of this thesis is to predict the congestion of TCP/IP networks using RED (Random Early Detection) or DRED (Dynamic Random Early Detection).

TCP, one of the most popular transmission protocols, controls the sources' packets sending rates by its congestion control algorithms. The four intertwined algorithms are slow start, congestion avoidance, fast retransmit and fast recovery[1]. However, when many TCP connections share network resources, these algorithms lead to synchronization, which means that the sources increase and decrease their window sizes simultaneously. Thus, the queue size and transmission delay of the bottleneck buffer introduce oscillation. Furthermore the sources can not steadily send data and the network becomes unstable. Choosing a good active queueing management (AQM) scheme and suitable amount of buffer size in a router is useful in stabilizing the network, improving the performance of the sources.

To solve the problem of instability in TCP network, a mathematical model has

been developed in reference [2] for AQM schemes such as RED and DRED. This model is used to represent a number of TCP connections going through a bottleneck buffer. Under certain conditions, the sources approach a steady state. When the sources are stable, the amount of data in the network and the mean transmission rate are constant. Even in the buffer, the queue size and packet drop rate become stabilized. Thus, the buffer space can be limited at a lower size with better performance.

This thesis discusses the steady state of a network by extending the mean field model for TCP networks [2]. The TCP connections may go through two or more routers with different paths. To reach this goal, a solver is designed to estimate the long run packet drop probability, queue size in the buffer and queueing delay.

To start, a review of TCP and its related technical knowledge is presented.

## 1.1 A Brief Review of TCP

Before introducing TCP's data transfer behaviour, there are some important features that need to be introduced

- Round Trip Time (RTT)

  The total time elapsed between sending each packet and receiving an acknowledgement for the data in that packet is called *sample round trip time* [3]. Generally, RTT can be computed as the sum of the total transmission delay (or propagation delay) in network and the total queueing delay in the buffers. In a given network, the total transmission delay is usually fixed. This is the time it takes for a packet to travel between two nodes. It is determined by the network's physical media such as the links' type and distance between the sender and receiver. Queueing delays are more complex. When a network suffers congestion, queueing delay refers to the amount of time a packet waits in the buffer for services. It is variable and is affected by many factors such as packets dropping algorithm in the buffer, the TCP type, the AQM scheme in the router and even the congestion threshold.

- Packet Loss Rate

Packet loss rate is one of the Quality of Service (QoS) parameters used to measure the performance of the network for a given connection. Packet loss occurs because of the overrun of buffering resources due to simultaneous arrivals of bursts from different connections. The packet loss rate is defined as

$$PacketLossRate = \frac{LostPackets}{TotalTransmittedPackets}$$

- Window Size

  Window size is the number of packets that can be unacknowledged at any given time. For example, a window size of eight means that the sender is permitted to transmit eight packets before it receive an ACK.

- Throughput

  Throughput is defined as the rate at which data is transmitted from the sender to the receiver. The value of the throughput depends on the size of the congestion window. It can also be called the transmission rate. Thus, if the connection has $W(t)$ windows at time t, and the Round trip time for the packets at time t is $RTT(t)$, the throughput or transmission rate would be

$$TransmissionRate(t) = \frac{W(t)}{RTT(t)}$$

  Throughput is one of the most important measures of TCP connection performance.

TCP, standing for Transmission Control Protocol, is a transport layer protocol that provides reliable transport service. It supports transfer of over 90 percent of overall traffic on the internet today [4].

TCP is also a connection-oriented protocol that requires a connection to be set up before any host can send data. When two hosts wish to communicate with each other, they exchange connection setup packets, called "SYN". For example, Host A requests a connection with host B by sending the "SYN". Host B acknowledges the request by returning an ACK and sending out a "SYN" to request connection with

A. When host A receives the "SYN" from B, it also returns an ACK to confirm the connection, then the conversation begins. See Figure 1. To terminate the connection, the two end points exchange finish packets called "FIN".

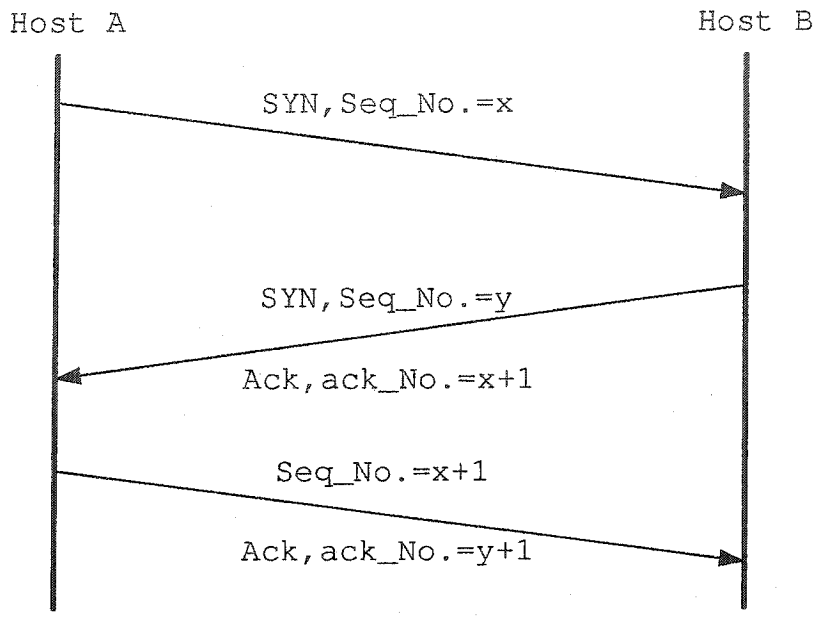



Figure 1: *TCP connection establishment.*

After connection setup, data transmission may begin. With the stream-oriented nature of TCP the data are divided into 8-bit octets called "bytes" for transmission. TCP uses an ACK scheme to report how many data packets have been accumulated by a remote receiver. A TCP acknowledgement specifies the sequence number of the next octet that the receiver expects to receive [3]. For example, host A sends a packet 1 to host B. Upon receiving this packet, B returns an acknowledgement (ACK) telling A that it has received packet 1 and expects packet 2. A then sends out packet 2 after receiving ACK 1. In order to enhance the utilization of the network, TCP uses a sliding window protocol. This window mechanism provides the sender credit to transmit data without receiving an acknowledgement. To continue the example just discussed, assume now a window size of five. In this case, ideally, the sender is allowed to send five packets in a row before receiving ACK 1, while in the meantime,

five ACKs are returned on the fly. However if packet 2 is lost during the transmission, packet 3, 4, and 5 are sent without packet 2 being acknowledged. Due to the loss of packet 2, upon receiving packet 3, 4, and 5, host B always returns the same ACK as ACK 1 saying that it has received packet 1 and is waiting for packet 2. Such ACK is called *duplicate ACK*. Every time sender sends a packet, TCP starts a timer and waits for an acknowledgement. If the timer expires before data in the segment has been acknowledged, TCP assumes that the packet was lost or corrupted and retransmits it.

On the other hand, TCP also uses a sliding window scheme to solve the end-to-end flow control problem. Because a well tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol [3]. However, if too many packets are sent, the network will experience congestion and even packet loss. The modern implementation of congestion control in TCP contains the four intertwined algorithms previously mentioned: Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery. They are described below:

- Slow Start

  Slow start is designed to dynamically maximize throughput and prevent congestion collapse. It requires two variables to maintain for each connection: a congestion window size and a congestion threshold size. Initialization for a given connection sets window size to one segment and threshold to 65535 bytes. When a connection is established, it sends one packet known as the initial window size and waits for an acknowledgement. Upon receiving the acknowledgement, the sender then increases the congestion window size to two, sends two packets and again waits for the ACKs. Upon receiving the ACKs, the window size doubles again to four. Now the sender sends four packets. After every round trip time the sender doubles the size of the congestion window by the number of ACKs received until the congestion window reaches its congestion threshold, at which point congestion avoidance takes place.

- Congestion Avoidance

If the congestion window reaches its threshold, it indicates that the bottleneck resource is running close to utilization. The sender slows down the transmit rate, then increases the congestion window size linearly: one packet for each round trip time(RTT). Although the congestion window size is increased one packet per RTT, the network is still reaching its threshold. When the sender does not receive an ACK within the time-out, or the sender receives duplicated ACKs within one RTT suggesting the packet is lost, the congestion threshold is decreased to one-half of the current window size. Additionally, the congestion window size is set to one segment. Slow start takes over again.

- Fast Retransmit

  After the TCP receiver sends ACKs for each of the remaining packets in the window, the sender starts retransmitting. In past versions, TCP would wait at least one second before retransmitting a lost packet. When the packet is lost, the receiver repeats the lost packet's sequence number to indicate that the packet was not received. In practice, fast retransmit, a new algorithm, has been introduced. If the sender receives three duplicate ACKs in a row, it considers the packet is lost and retransmits the lost packet immediately. At the same time the congestion window size is cut to half of the congestion threshold, and enters slow-start. See Figure 2.

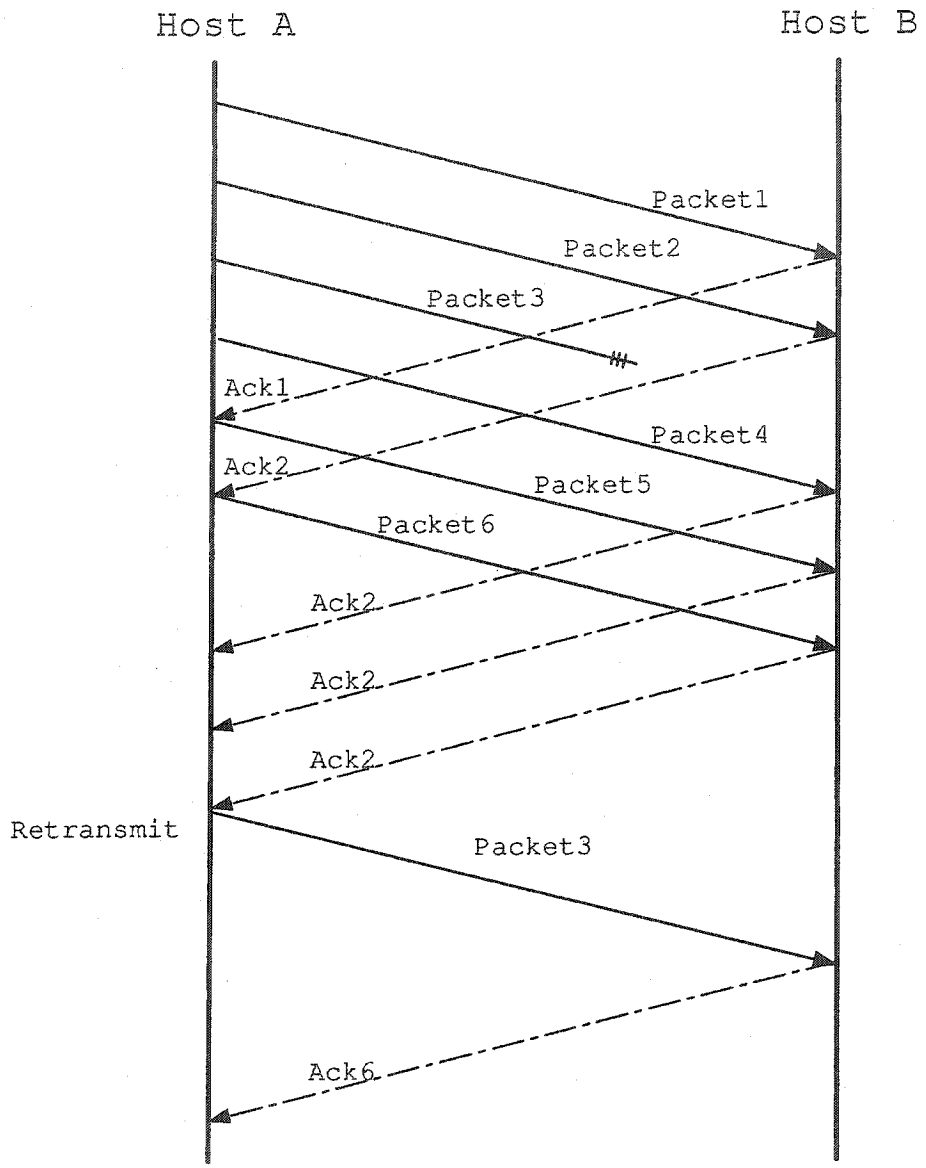



Figure 2: *Fast retransmit and fast recovery.*

When the TCP congestion window is less than four packets, TCP is no longer able to recover from a single packet because fast retransmit (fast recovery as well) needs at least three duplicate ACKs to be trigged. Thus, when the window size is below four, a single packet loss causes the TCP connection to timeout.

- Fast Recovery

  After retransmitting the lost packet, TCP does not enter slow start, but implements congestion avoidance. This is fast recovery. This improvement to the congestion control algorithm allows high throughput under moderate congestion, especially for large windows.

In older versions, TCP would only use slow start and congestion avoidance to control network congestion. With the development of networks, TCP added fast retransmission and fast recovery. These two algorithms allow TCP to retransmit the lost packet immediately after three duplicate ACKs without waiting for a retransmission timer to expire. This leads to higher utilization of the network. The algorithm that includes slow-start, congestion avoidance, and fast retransmit is called TCP Tahoe. TCP Reno includes slow start, congestion avoidance, but modifies the fast retransmit to the fast recovery algorithm. TCP Reno is better than Tahoe when the source has a large window size because it prevents increasing the window size too quickly after loss of a single packet. In this thesis, all sources in the models are TCP Reno.

The advantage of using an available window size is to provide a reliable transfer for computers to exchange data and acknowledgements. It avoids receiving more data than it can store, and even shrinks the window size to zero to stop the transmission. Also, when the buffer space becomes available, the receiver can trigger the flow of data again. The four algorithms described above are suggested by the TCP standard to avoid and control congestion, which are correlated and can be implemented easily.

## 1.2 A Brief Review of Random Early Detection Algorithm

As mentioned earlier, TCP uses a sliding window protocol to provide flow control. When a single TCP connection goes through a router, the router places each incoming packet in the queue of a buffer until it can be processed. If the data arrive slower than they can be forwarded, the queue shrinks until the queue reaches zero. When data arrive faster than they can be forwarded, the queue grows until it reaches the limit of the buffer. Usually the router software uses a *tail-drop* policy to manage queue overflow. " If the input queue is filled when a datagram arrives discard the datagram." [3]. After the packet is dropped, the TCP source detects the loss from the reception of duplicate ACKs. The connection cuts the window size into a half, and enter slow-start or congestion avoidance, so that the throughput is reduced.

If multiple TCP connections share one output link, a server problem occurs. It is known that the flow per connection decreases as the number of connections increases because studies have shown TCPs is able to share a bottleneck fairly and efficiently. For example, the data arriving in the router are from different TCP connections. When the queue is filled, the router discards the packets regardless of where they are coming from. *Tail-drop* causes the losses at the same time, and the sources cut the window size and increase the congestion windows simultaneously. This phenomenon is called *global synchronization.*

The queue size in the router oscillates. To solve this problem, Active Queue Management (AQM) and in particular Random Early Detection (RED) schemes are suggested. With RED, an arriving packet is killed with a probability which increases with the queue size of the buffer [2]. The main idea of RED is to drop the packets before the buffer overflows, and to set the packet random dropping probability linearly with respect to the queue size of the buffer. The larger the queue size, the more the packets are dropped. Thus, before the buffer fills up, some packets are dropped, and some sources have already cut the window size.

RED is a good method to control the overload of the buffer and avoid synchronization. In order to implement RED, there are four parameters that should be

mentioned: $Q_{min}$ and $Q_{max}$ for two acceptable thresholds, the maximum dropping probability $P_{max}$ and the minimum dropping probability$P_{min}$. When a packet arrives at a router which is implementing RED, it reads the current average queue size $q_{size}$. If $q_{size} < Q_{min}$, the router queues the packet. If $Q_{min} \leq q_{size} < Q_{max}$, the router drops the packet with a probability proportional to the average queue length:

$$P_{droprate} = \frac{(q_{size} - Q_{min}) \times (P_{max} - P_{min})}{Q_{max} - Q_{min}} + P_{min} \tag{1.1}$$

If $q_{size} \geq Q_{max}$, the router drops the packet.

The basic RED scheme maintains an average of the queue length, provides a more equitable distribution of packet loss, avoids the synchronization of flows, and improves the utilization of the network.

# Chapter 2

# Solver for Buffers Implementing RED

This chapter describes a solver which can predict the long run steady state of a small network under certain given conditions. The solver allows the users themselves to define the parameters of the network , and the system generates equations for this network by using the mean field approximation model for the RED algorithm [2] . Finally, the solver computes out a numerical solution for the window size, the average queue size in a buffer and the average loss rate per connection in the small network. Another algorithm called constant drop rate is demonstrated in the next chapter.

Before introducing the algorithm, here are some assumptions:

- All the connections in the network are TCP Renos that execute congestion avoidance.
- The buffers in the network all implement RED mechanism.
- Each buffer has a limited size that can be defined by the user. Once the space is exhausted the arriving packets are dropped.
- The buffer follows a First In First Out (FIFO) scheme.
- There are no transmission losses or timeouts. The only losses are generated by AQM tail-drop.

## 2.1 A Mean-Field Model for TCP

In this section, a review is given of the mean-field model for a buffer multiplexing a large number of TCP flows found in [2].

In this model there are a large number $N$ of TCP/IP sources executing congestion avoidance. They are all routed through a bottleneck queue. Assume that $q(t)$ is the relative queue size per active connection, and that the packets from all connections join the queue $Q(t)$ in the buffer. Hence, the flows can be treated as a fluid.

$$Q(t) = N \times Q_n(t) \tag{2.2}$$

Furthermore, define the following:

- $W_n(t)$ is the window size of connections $n$ in congestion avoidance at the time t.
- $R_n(t)$ is the round trip time of source $n$ at time $t$.

$$R_n(t) = T + \frac{Q_n(t - R_n(t))}{C} \tag{2.3}$$

  where $C$ is the link rate per source in packets per second per active source. Thus, by Little's formula $\frac{Q_n(t)}{C}$ is the queueing delay of the packet's arrival in the buffer at time $t$, and $T$ is the common transmission delay in the network.

- $k(t)$ is the packets drop rate at time $t$.
- $F(Q_n(t))$ is a distribution function to the drop probability of RED. $F(Q_n(t))$ is zero below $Q_{nmin} = Q_{min}/N$, but linearly rises to $p_{max}$ at $Q_{nmax} = Q_{max}/N$ and further to one when $Q_n(t)$ reaches the buffer size $Q_{nmax}$.

Equation (2.3) defines the round-trip time based on the past queue size, so that there is an implicit delay built into the model. Assume that a TCP source receives an acknowledgement from the destination at time $t$. Let $R_n(t)$ be the round trip time of the packet from source $n$, which experiences packet generation, delay and acknowledgement. The packet, however, reaches the router in advance by approximately

one $R_n(t)$, making its queueing delay $\frac{Q_n(t-R_n(t))}{C}$. Thus, $R_n(t)$ can be viewed as the round-trip time experienced by packets *arriving* to source $n$ at time $t$, instead of the round-trip time that is experienced by packets *leaving* the source at time $t$. It may be imagined that the source writes this value into the new packet being transmitted at time $t$.

According to Equation (3.1) of [2], the evolution of the window size of connection $n$ is described by the following stochastic differential equation.

$$dW_n(t) = \frac{1}{R_n(t)} \cdot (1 - \chi_{S_n(t)})dt - \frac{W_n(t^-)}{2} dN_n(\Lambda_n(t)) \tag{2.4}$$

This equation shows the window size's change at time $t$. At time $t$, the window size is either increasing linearly with the rate $1/R_n(t)$ or being decreased to $W_n(t^-)/2$, half of the current window size, by finding a packet loss. The value of $\chi_{S_n(t)}$ is 1, when source $n$ simply finds the loss, and changes to fast recovery. In that case, the window size does not follow a linear increase, and no losses can be detected. Since the fast recovery periods are relatively short, this term can be ignored without significantly changing the dynamics. As in [2], $N_n(t)$ is a Poisson process with intensity 1, and

$$\Lambda_n(t) = \int_0^t \frac{W_n(s - R_n(s))}{R_n(s - R_n(s))} F(Q_n(s - R_n(s)))ds \tag{2.5}$$

is the stochastic intensity for the Poisson point process of the losses of connection $n$. Hence, the losses of connection $n$ occur according to the time changed Poison process $N_n(\Lambda_n(t))$.

According to (3.2) of [2], the rate of change of the fluid buffer is given by the following differential equation.

$$\frac{dQ_n(t)}{dt} = \frac{1}{N}\sum_{n=1}^{N}\frac{W_n(t)}{R_n(t)}(1 - F(Q_n(t))) - C - \left(\frac{1}{N}\sum_{n=1}^{N}\frac{W_n(t)}{R_n(t)}(1 - F(Q_n(t))) - C\right)^{-}\chi\{q(t) = 0\} \quad (2.6)$$

This equation shows that the rate of change in the relative queue size is the difference between the rate per source at which fluid arrives minus the link rate per source $C$. The mean arrival rate per source is the mean window size $\frac{1}{N}\sum_{n=1}^{N} W_n(t)$ divided by the round trip time $R_n(t)$ times the proportion $(1 - k(t))$, which is not discarded. The second term simply keeps the queue from becoming negative.

When $N$ becomes large, $W_n(t) \to w$, $R_n(t) \to r(t)$, $Q_n(t) \to q(t)$, the expressions in [2] lead Equation (2.4) and Equation (2.6) to the mean-field limit equations:

$$\frac{\partial p(t,w)}{\partial t} = -\frac{1}{r(t)}\frac{\partial p(t,w)}{\partial w} + (4wp(t,2w) - wp(t,w))\frac{k(t - r(t))}{r(t - r(t))} \quad (2.7)$$

and

$$\frac{dq(t)}{dt} = \int_w \frac{w}{r(t)}p(t,w)dw(1 - k(t)) - C - (\int_w \frac{w}{r(t)}p(t,w)dw(1 - k(t)) - C)^{-}\chi\{q = 0\} \quad (2.8)$$

In these equations, $p(t,w)$ is the histogram of the window size of the connections in congestion avoidance at time $t$. $N$ is the number of sources executing congestion avoidance, $k(t)$ and $C$ have been previously defined.

The equations can be interpreted by integrating both sides of Equation (2.7). The term $p(t,w)$ is the density of the window size for all time t, and $\frac{\partial p(t,w)}{\partial t}dw$ is equal to the rate at which mass flows into and out of a infinitesimal slice of windows $(w, w + dw]$. Due to the additive increase of TCP, every RTT the window increases by one, so on the right side, the first term times $dw$ is equal to $-\frac{1}{r(t)}\frac{\partial p(t,w)}{\partial w}dw$. This shows that the

rate at which the mass in the slice $(w, w+dw]$ grows by an additive increase. Because the quantity of fluid that pours into the slice in time $dt$ is $p(t,w)dt/r(t)$ to first order, the quantity that pours out is $p(t,w+dw)dt/r(t)$ to first order. Thus the difference is $-\frac{dt}{r(t)}\frac{\partial p(t,w)}{\partial w}dw$. The second term of Equation (2.7) is due to the multiplicative decrease. When the window size is $2w$, the transmission rate at time $t$ is $\frac{2w}{r(t-r(t))}$, and at that time the kill rate is $k(t-r(t))$. When the loss occurs at time $t$, the mass that flows out from the window is $p(t,2w)2dw$. Thus the rate at which the mass exits the window is $p(t,2w)\frac{2w}{r(t-r(t))}2dw$. Also, when the window size is $w$ at time $t$, when the loss is detected, the rate at which the mass needs to be subtracted from this slice is $p(t,w)\frac{w}{r(t-r(t))}dw$. Equation (2.7), therefore describes the net effect of the flows.

Equation (2.8) is very clear. Since $N$ is a very large number, and all sources have the same RTT, the mean window size can be calculated as $\int_w wp(t,w)dw$ for each connection. The transmission rate is the mean window size divided by the round trip time $r(t)$. As described before, Equation (2.8 gives the relative queue size's change rate, which is the difference between the packets arrival rate times $(1-k(t))$ and the link rate $C$.

When the RTT is sufficiently small and the loss rate $k(t)$ is well chosen, the system defined by Equation (2.7) and Equation (2.8) may stabilize. The term $q(t)$ tends to a constant value $q$ and the loss rate $k(t)$ tends to $k$. The window distribution stabilizes to a fixed distribution $f_k$ and RTT $r(t)$ tends to a constant $r$. The solutions for the stable systems (2.7) and (2.8) are:

$$\frac{df_k(w)}{dw} = k\left(2(2w)f_k(2w) - wf_k(w)\right) \tag{2.9}$$

$$C = (1-k)\frac{1}{r}\int_w wf_k(w)dw. \tag{2.10}$$

Equation (2.10) is simply Little's formula since the right hand side represents the throughput as the average window size divided by the RTT and multiplied by the proportion of packets that are not killed.

We also can find a theorem in [2].

**Theorem 2.1** *Let* $\Psi = \sum_{i=0}^{\infty} \frac{2^i}{\prod_{j=1}^{i}(1-4^j)}$ *(*$\Psi \approx 0.4194$*). The unique density* $f_k(w)$ *solving (2.9) is given by*

$$f_k(w) = \sum_{i=0}^{\infty} a_i \exp(-k4^i \frac{w^2}{2}) \tag{2.11}$$

$$a_0 = \sqrt{\frac{2}{\pi}} \frac{1}{\Psi} \sqrt{k} \; ; \; a_i = a_{i-1} \frac{4}{1-4^i} = a_0 \frac{4^i}{\prod_{j=1}^{i}(1-4^j)}. \tag{2.12}$$

The mean window size is

$$\int_w w f_k(w) dw \;\; = \;\; \alpha \sqrt{\frac{1}{k}} \tag{2.13}$$

where $\alpha \approx 1.310$. Combining Equation (2.13) with Equation (2.10) yields

$$\frac{(1-k)^2}{k} \;\; = \;\; \left(\frac{rC}{\alpha}\right)^2 \tag{2.14}$$

In this equation, the function $(1-k)^2/k$ is monotonically decreasing. It follows that there is a unique point $k$ satisfying (2.14).

As mentioned in Chapter1, the packet drop rate$k$ is a function of queue size $q$. The Equation (2.14) can be reformed to

$$\frac{(1-F(q))^2}{F(q)} = \left(\frac{rC}{\alpha}\right)^2$$

Since $(1-F(q))^2/F(q)$ is decreasing in $q$, the stable queue size is determined by the unique solution to

$$\frac{(1-F(q))^2}{F(q)} = (\frac{(T+q/C)C}{\alpha})^2 \tag{2.15}$$

If the solution to Equation (2.15) is less than $q_{max}$ then $k = F(q)$ gives the stable point and $r = T + q/C$. On the other hand if the solution to Equation (2.15) is equal or greater than $q_{max}$ then $q = q_{max}$ and $k$ is equal to one.

## 2.2 Model for Buffers Using RED in a Network

As we can see in last section, the stable solution to Equation (2.7) and Equation (2.8) is under the condition that a number of sources go through a same bottleneck buffer, and that they all have the same RTT. However in practice, this kind of extreme situations are very rare. Usually, some sources go through two or more buffers instead of only one buffer. Hence, packet loss may happen in all bottleneck buffers they passed. Moreover, the sources may go to different destinations, and so they might have different RTTs. In order to develop a more practical solver, this model needs to be extended because the above mean field arguments are also applicable even if the RTT of different sources varies. If the loss rates $k_n(t)$ of different sources are well chosen, then the queue size may still stabilize at $q_n$.

### 2.2.1 Mathematical Equations for the Model

In order to further clarify the model, assume that there is a small network with several routers. The sources in the network are grouped. Each group has different destinations, so that they have different RTTs, and also their data path are different. The configuration is shown as Figure 3.

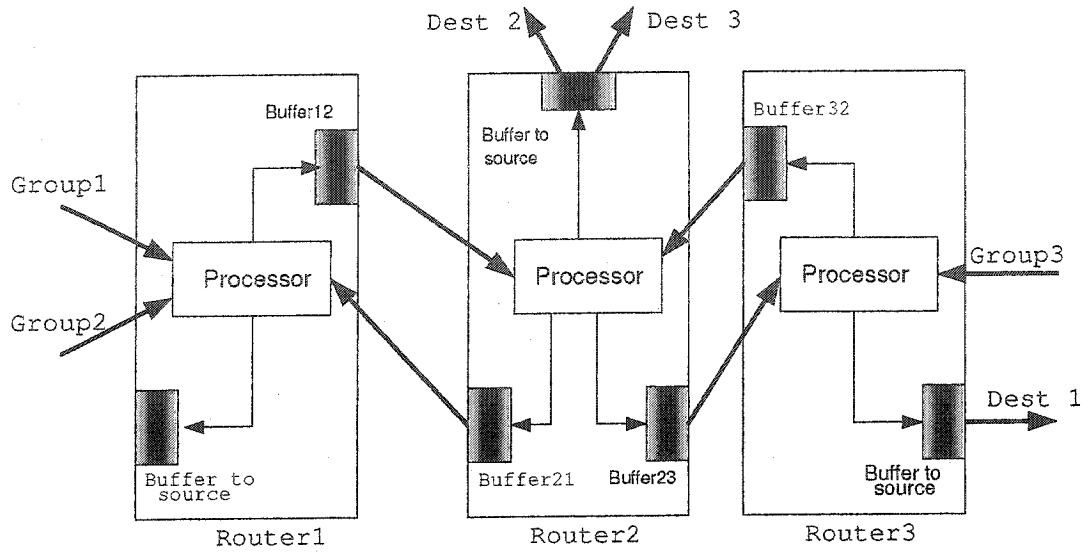



Figure 3: *An example of a small network.*

The total number of routers in the network is three, and there are three groups of sources, each with $N_n$ users. All the users send a large volume of data. Group 1 goes through router 1, router 2 and router 3 to destination 1. Group 2 goes through router 1 and router 2 to destination 2, and group 3 goes through router 3, and router 2 to destination 3.

In this network we can see that not all buffers suffer from congestion. For group 1 and group 2, at the beginning the data share buffer 12 from router 1 to router 2. It can be considered that $N_1 + N_2$ sources share a buffer to switch the packets. If the traffic is very busy and the bandwidth of link between router 1 and router 2 is not large enough, then congestion may happen in this buffer. When data pass through router 2, the packets from group 2 are switched to the destination, while the packets from group 1 must continue to router 3. Consequently, if the link between router 2 and router 3 has the same transmission rate as the link 1 to 2, the queue may not exist in buffer 23 because the traffic is lower than that of buffer 12. However, if the bandwidth of the link is narrow, packets can still be lost in buffer 23. For group 3,

the packets may be killed in buffer 32. It should be noted that ACKs may also be lost in addition to the loss of packets. For example, when destination 1 receives packets from group 1, it sends out ACKs to acknowledge them, and the ACKs return along the same path that the packets are sent. As a result, the ACKs from destination 1 shares the buffer with the packets of group 3. In this way the ACKs may be lost. However, the lost ACKs do not affect the flow control. For example, if the source receives ACK 4 after ACK 2, the TCP source considers that the receiver has received packet 3 and sends the ACK 3. Due to the network congestion, the ACK is lost, not the packet. Hence, the source does not retransmit the packet; however the time ACKs wait in the queue extends the round trip time.

From above it can be seen that the round trip time of each group is different. Assume that the transmission delay and bandwidth from router 1 to router 2 are $T_{12}$, $B_{12}$, and that the transmission delay and bandwidth from router 2 to 3 are $T_{23}$, $B_{23}$. Thus, $T_{21}$, $B_{21}$, $T_{32}$, and $B_{32}$ also illustrate the corresponding transmission delay and bandwidth. Queue sizes of buffer 12, buffer 21, buffer 23 and buffer 32, respectively, are $q_{12}$, $q_{21}$, $q_{23}$, and $q_{32}$.

Consequently the round trip time of group 1:

$$RTT_1 = T_{12} + T_{23} + T_{32} + T_{21} + \frac{q_{12}}{B_{12}} + \frac{q_{23}}{B_{23}} + \frac{q_{32}}{B_{32}} + \frac{q_{21}}{B_{21}} \tag{2.16}$$

for group 2:

$$RTT_2 = T_{12} + T_{21} + \frac{q_{12}}{B_{12}} + \frac{q_{21}}{B_{21}} \tag{2.17}$$

for group 3:

$$RTT_3 = T_{32} + T_{23} + \frac{q_{32}}{B_{32}} + \frac{q_{23}}{B_{23}} \tag{2.18}$$

Moreover, the proportion of packets lost by connections of group 1 is $k_12 + (1 - k_12)k_23 \approx k_12 + k_23$; while the proportions lost by group 2 and group 3 connections are $k_12$ and $k32$ respectively.

Equation (2.13)shows the mean window size of each source. The average window size divided by RTT is the transmission rate of each relative queue in packets per second. In this small network, some sources with different RTT and window size share one buffer. Thus the total rate flowing into the buffer should be the sum of the transmission rate of each relative queue. The total transmission rate in bytes per second times $(1-k)$ is the portion that has not been dropped by the buffer. This remaining portion should be less or equal to the egress link. A group of equations can thus be established for this small network.

$$\left[\left(N_1\frac{\alpha\sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1}+N_2\frac{\alpha\sqrt{\frac{1}{k_{12}}}}{RTT_2}\right)\times PacketSize\right]\times(1-k_{12}) \leq B_{12} \quad (2.19)$$

$$\left[\left(N_1\frac{\alpha\sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1}(1-k_{12})(1-k_{23})(1-k_{32})+N_2\frac{\alpha\sqrt{\frac{1}{k_{12}}}}{RTT_2}(1-k_{12})\right)\times AckSize\right] \times(1-k_{21}) \leq B_{21} \quad (2.20)$$

$$\left[\left(N_1\frac{\alpha\sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1}(1-k_{12})\right)\times PacketSize+\left(N_3\frac{\alpha\sqrt{\frac{1}{k_{32}}}}{RTT_3}(1-k_{32})\right)\times AckSize\right] \times(1-k_{23}) \leq B_{23} \quad (2.21)$$

$$\left[\left(N_3\frac{\alpha\sqrt{\frac{1}{k_{32}}}}{RTT_3}\right)\times PacketSize+\left(N_1\frac{\alpha\sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1}(1-k_{12})(1-k_{23})\right)\times AckSize\right] \times(1-k_{32}) \leq B_{32} \quad (2.22)$$

Here, $k_{12}$, $k_{21}$, $k_{23}$, and $k_{32}$ respectively represent the packet drop rate in buffer 12, buffer 21, buffer 23 and buffer 32. Since the buffers are running RED, the packets' kill rates in the buffer are functions of the queue size. In the solver, RED is defined as

$$k = F(q_{size}) = \begin{cases} 0, & if \quad q_{size} \leq Q_{min}; \\ \frac{(q_{size}-Q_{min})\times(k_{max}-k_{min})}{(Q_{max}-Q_{min})} + k_{min}, & if \quad Q_{min} < q_{size} < Q_{max}; \\ 1, & if \quad q_{size} \geq Q_{max}. \end{cases} \tag{2.23}$$

In this group of equations $k_{12}$, $k_{21}$, $k_{23}$, and $k_{32}$ can be substituted into $F(q_{12})$, $F(q_{21})$, $F(q_{23})$, and $F(q_{32})$. So the equations then become:

$$\left[\left(N_1\frac{\alpha\sqrt{\frac{1}{F(q_{12})+F(q_{23})}}}{RTT_1} + N_2\frac{\alpha\sqrt{\frac{1}{F(q_{12})}}}{RTT_2}\right) \times PacketSize\right] \times (1 - F(q_{12})) \leq B_{12} \tag{2.24}$$

$$[(N_1\frac{\alpha\sqrt{\frac{1}{F(q_{12})+F(q_{23})}}}{RTT_1}(1 - F(q_{12})(1 - F(q_{23})(1 - F(q_{32}) + N_2\frac{\alpha\sqrt{\frac{1}{F(q_{12})}}}{RTT_2}(1 - F(q_{21})) \times AckSize] \times (1 - F(q_{21})) \leq B_{21} \tag{2.25}$$

$$[\left(N_1\frac{\alpha\sqrt{\frac{1}{F(q_{12})+F(q_{23})}}}{RTT_1}(1 - F(q_{12}))\right) \times PacketSize + \left(N_3\frac{\alpha\sqrt{\frac{1}{F(q_{32})}}}{RTT_3}(1 - F(q_{32})\right) \times AckSize] \times (1 - F(q_{23})) \leq B_{23} \tag{2.26}$$

$$[(N_3\frac{\alpha\sqrt{\frac{1}{F(q_{32})}}}{RTT_3}) \times PacketSize + \left( N_1\frac{\alpha\sqrt{\frac{1}{F(q_{12})+F(q_{23})}}}{RTT_1}(1-F(q_{12})(1-F(q_{23})\right) \times AckSize] \times (1-F(q_{32})) \leq B_{32} \tag{2.27}$$

Now it is visible that there are only four unknown variables and four equations. Matlab can be used to solve these equations. The results of $q_{12}$, $q_{21}$, $q_{23}$, and $q_{32}$ are the fixed points of the mean-field equations for this small network.

### 2.2.2 Introduction of the Solver

Based on the algorithm in the previous section, a solver has been developed in Matlab to calculate the mean-field long run steady state of the buffers running RED for a small network. The programming flow diagrams is shown as follows.

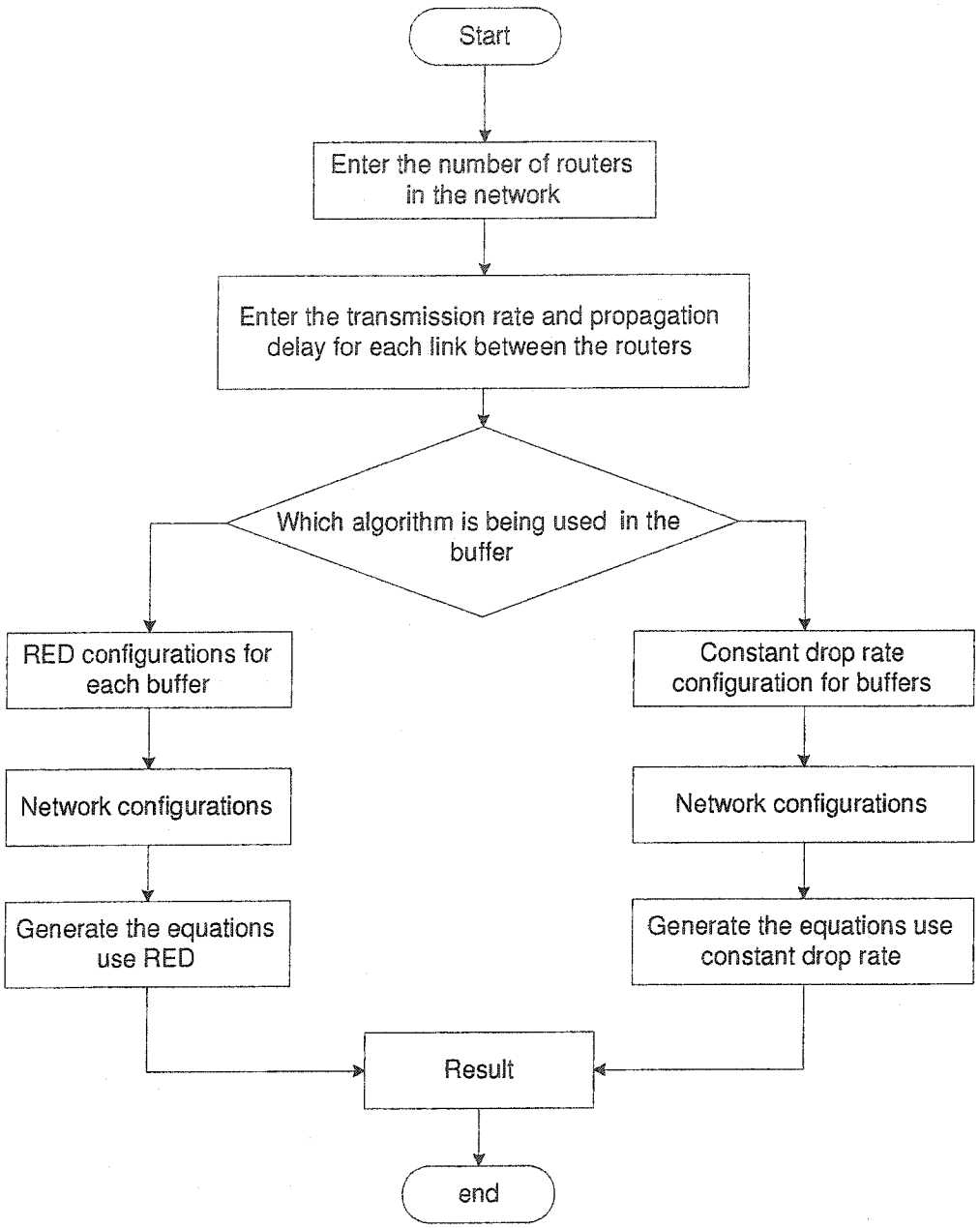



Figure 4: *Programming flow chart.*

In Figure 4, the solver can be divided into two portions that are described below:

- Network definition

In this part, the solver allows the user to define some parameters of the network. The first step is to define how many routers are in the network.

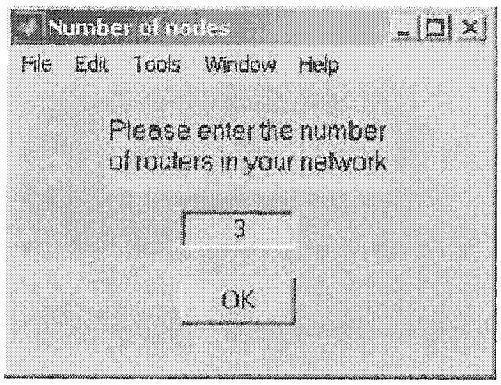



Figure 5: *Setting the number of nodes.*

Next, the solver asks the user to define the transmission rate and propagation delay between the routers. In order to simplify the calculation, It is suggested that all the links between the sources and the routers are T3 lines (44.74Mbps), and that there is no transmission delay between the sources and routers. The only delay exists between routers. The interface for the user to set the bandwidth and delay between the routers is shown in the following figure.

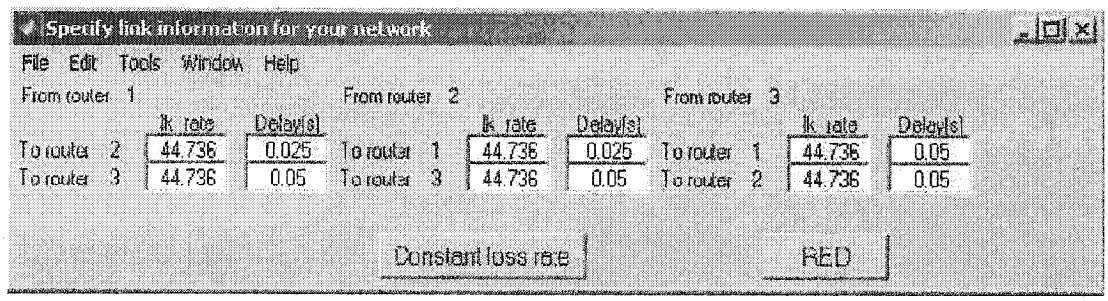



Figure 6: *Link configuration.*

After the user chooses the RED algorithm, which is used in the buffer, the next step is to define the parameters of RED for all buffers, i.e. the maximum

buffer size and the maximum packet dropping rate. Here, the minimum queue thresholds are automatically set to zero and the minimum packet dropping probabilities are also zero. See Figure 7.

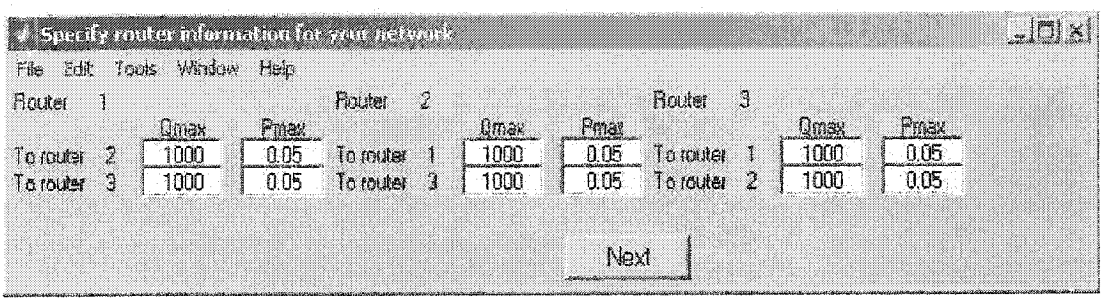



Figure 7: *Buffer configuration.*

The last step in this part is to configure the topology of the network, i.e. the number of sources in each group, and the path of each group. The path of the each group is fixed during the whole transmission procedure. The routers are working as layer 2 switches without routing capability. See Figure 8. Due to the limitation of the computer, the maximum number of groups is 20.

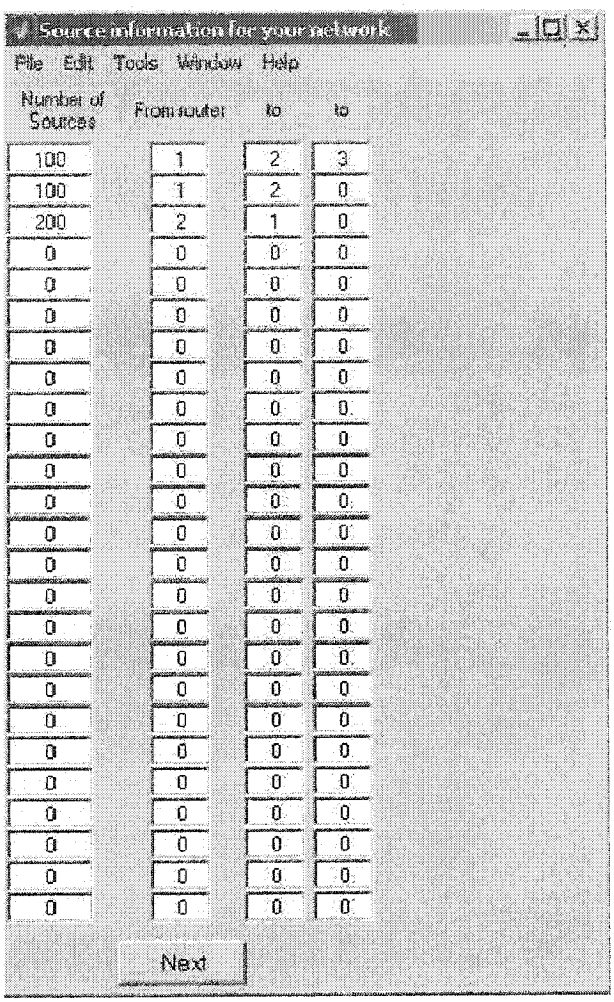



Figure 8: *Topology configuration.*

- Calculation

  Following the method shown in Chapter 2.2.1, the solver combines and extends Equation(2.13) with Equation(2.10) for each queue. After it generates a group of equations for the user-defined model, the solver calculates the steady state for the system. The solution is the stable queue size for the mean-field model in the buffers that implement RED. Since the RED mechanism drops the incoming packets with a probability which increases as the queue size increases, it is possible to calculate the packet drop probabilities of the buffers by Equation

(2.23) and queueing delay of each buffer from the stable queue size by Little's Law. Furthermore, it is possible to estimate RTTs for each group of sources by Equation(2.3) when the system is stabilized.

## 2.3 Analysis of the Model

RED is a good active queue management scheme that balances a buffer fill by dropping packets to adapt to the aggregation of ingress to the link rate of egress. Based on the mean field model described before for buffer implementing RED, the solver built in Matlab generates the equations, and finally calculates the solutions for the equations. They are the long run balanced queue sizes and packets dropping probabilities for the buffers.

After building the solver, it needs to be verified. Thus, the same model is in Matlab and OPNET, and then the mathematical results from the Matlab are compared with with the simulation results from OPNET. As described before, a very simple and small network has been designed with the following components:

1. There are three routers in the network.

2. The links between the routers are all $T3$ line. The propagation delay between router 1 and router 2 is 0.025 seconds, and 0.05 seconds between router 2 and router 3 . The buffers run RED. All the buffers have the same buffer size and maximum packet dropping rate.

3. All the buffers are same: the maximum number of packets that can be buffered is 1000; the maximum packet dropping rate is 0.05; the minimum packet dropping rate is 0, the minimum threshold is zero; and the maximum threshold is 1000.

4. There are three groups going through the network with their paths fixed. Group 1 goes through router 1, router 2, and router 3 to destination 1. Group 2 passes router 1 and router 2 to destination 2. There are 100 sources in group 1 and group 2 and 200 sources in group 3. Group 3 goes from router 2 to router 1 to destination 3.

### 2.3.1 Numerical Results From Matlab

After entering all the data into the solver, the solver calculates the results shown in the following figure.

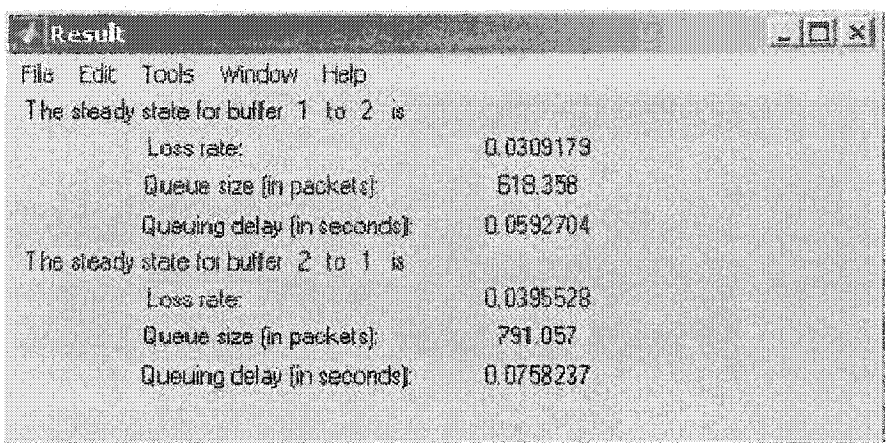



Figure 9: *Steady state for the buffers.*

From the figure, it can be seen that the packets are queued in buffer 12 and buffer 21. Although the packets go through the buffer 23 and the ACKs go through the buffer 32, there no queues exist. This means that buffer 23 and buffer 32 are not bottlenecks.

### 2.3.2 Simulation Results From OPNET

To validate the solver, The same TCP network has been built in OPNET. The topology of the network implemented is shown in Figure 10.

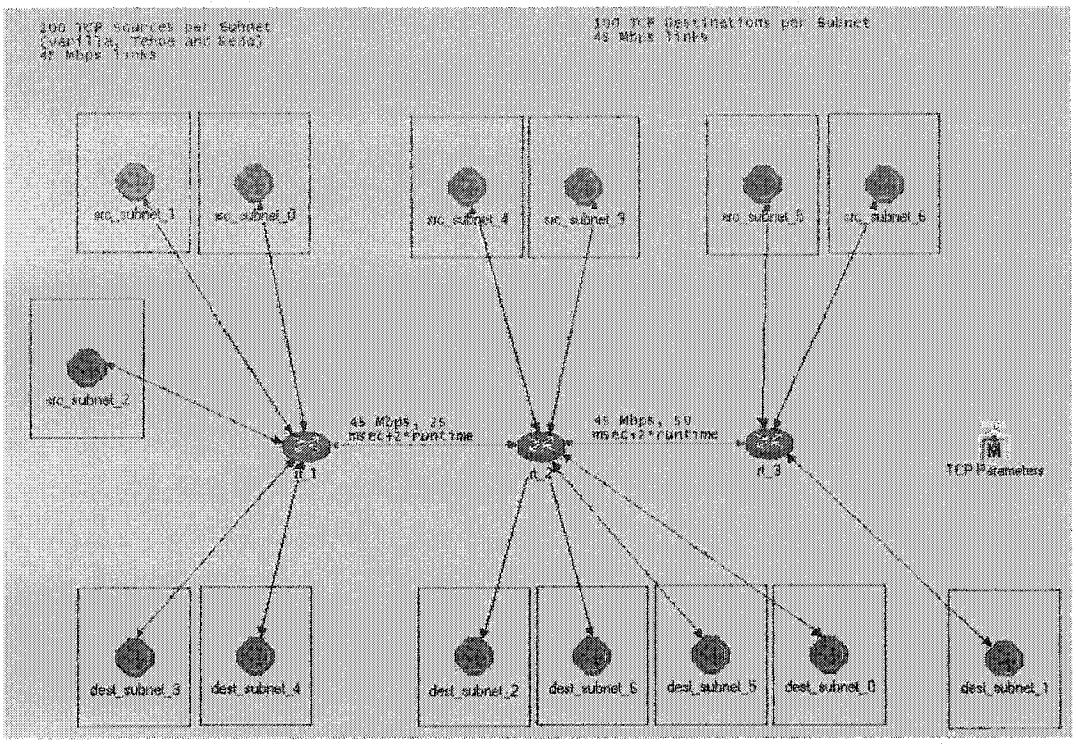
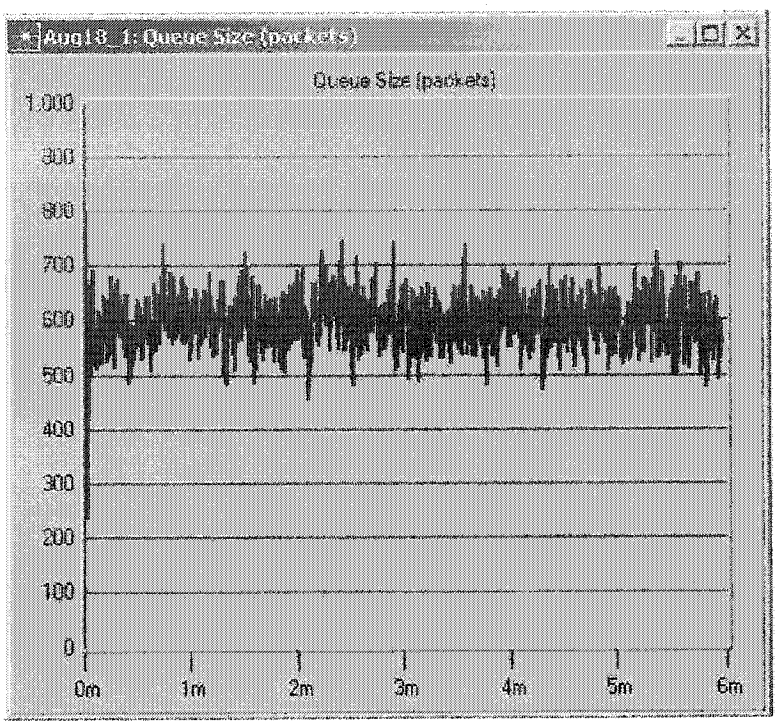
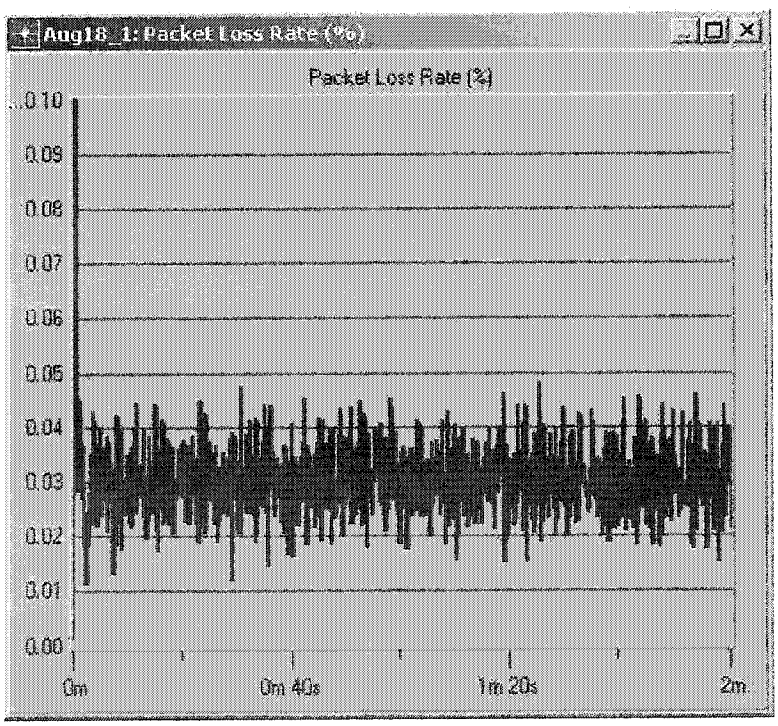



Figure 10: *A small network.*

In this network, each sub-net has 100 TCP sources. They run TCP Reno and the routers use RED as their packet dropping mechanism. In this network, only sub-net 1, sub-net 2, sub-net 3 and sub-net 4 send data, the others are set to idle. All the parameters such as link bandwidth, propagation delay, maximum buffer size are set to the same as in Matlab simulation. The simulation runs for 360 seconds. The primary performance metrics in the simulation experiments are the queue size and drop probability. The queue size is sampled and plotted every 10ms, and the drop probability is plotted every $\Delta t = 0.1s$. The reports of the queue size and packets dropping rate for buffer 12 and buffer 21 are shown in Figure 11 and Figure 12 respectively. Figure 13 shows that the queues size in buffer 32 and buffer 23 are zero.

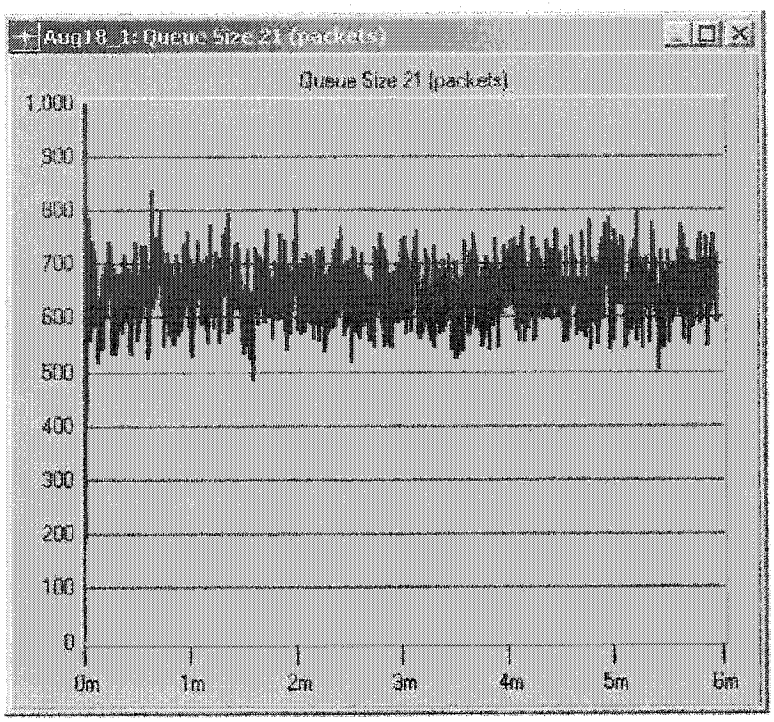



(a) Queue size

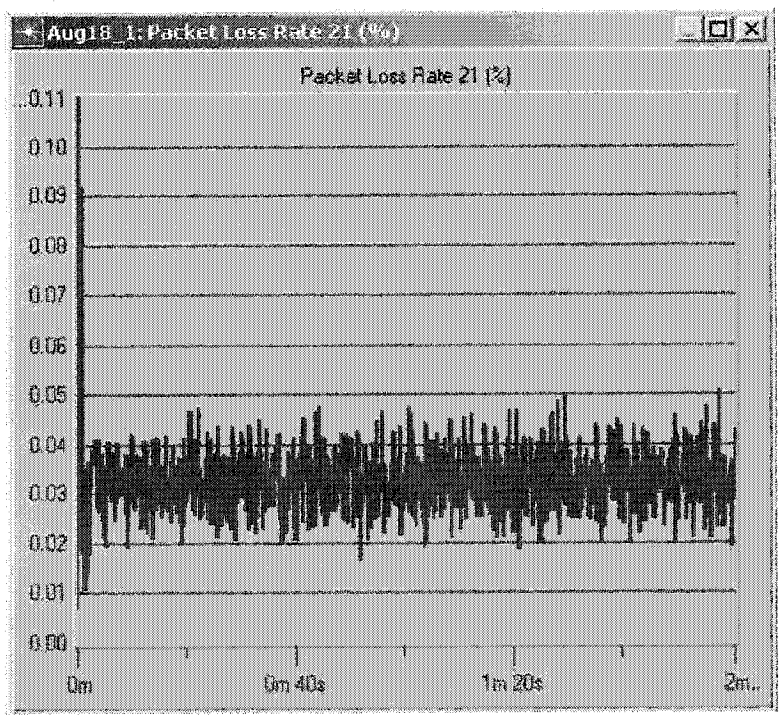



(b) Packet loss rate

Figure 11: *Queue size and packet loss rate in buffer* 12.

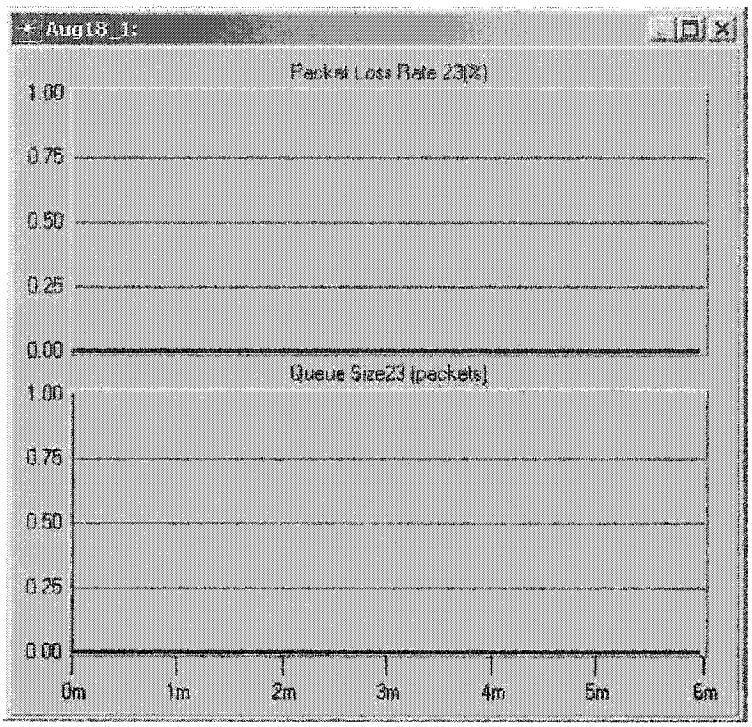



(a) Queue Size

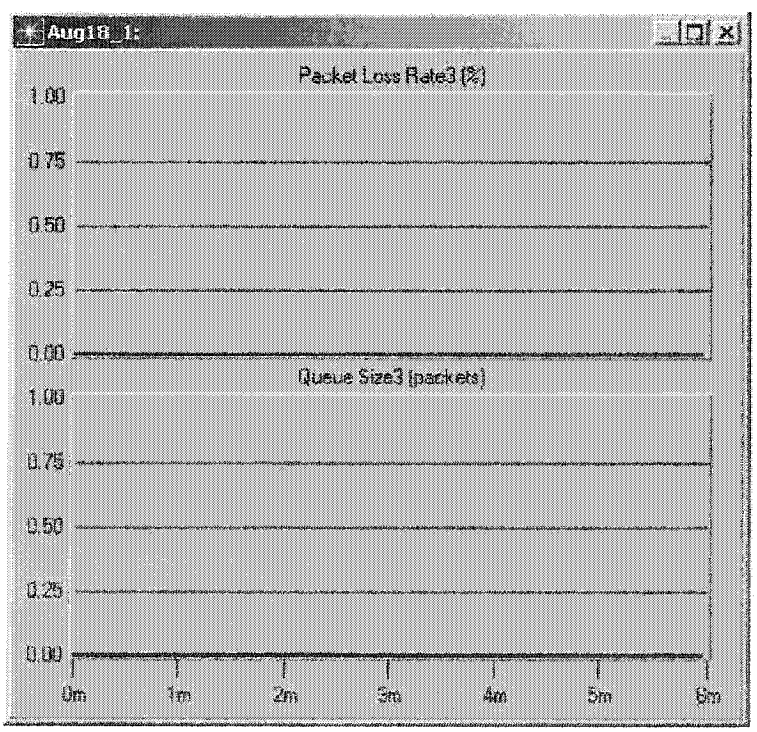



(b) Packet loss rate

Figure 12: *Queue size and packet loss rate in buffer* 21.




(a) In Buffer23




(b) In Buffer32

Figure 13: *Queue size and packet loss rate in buffer* 23 *and buffer* 32.

By exporting the data sheets from the OPNET, there are 3600 sample points. After the queues become stable ( approximately after 6 second), the average queue size and average drop rate were calculated. See the table below:

| After 6 sec., queues are stable | Mean | Standard Deviation |
|---|---|---|
| Queue size in buffer 12 | 596.4284 packets | 38.2234 packets |
| Queue size in buffer 21 | 647.1743 packets | 42.2935 packets |
| Packet drop rate in buffer 12 | 0.0299 | 0.00575 |
| Packet drop rate in buffer 21 | 0.0333 | 0.00569 |

## 2.4 Summary

The basic idea behind active queue management schemes such as RED is to detect incipient congestion early, and to convey congestion notification to the end systems, having them reduce their transmission rates before the queue in the network overflows and excessive packet loss occurs. Well tuned RED parameters can lead to a stable network.

From Figure 11 and Figure 12, it can be seen after six seconds that the queue size in buffer 12 tends to stable around 597, and the queue in buffer 21 tends to around 648. According to the RED schemes, the incoming packets are dropped in a random probabilistic manner where the probability is a function of the recent buffer fill history. Thus, the corresponding packet loss rate is around 0.0299 and 0.033 respectively. In Figure 13, there are no queues in buffer 23 and buffer 32, which matches the solution of the solver, because those buffers are not bottleneck buffers. There is no packet loss in these buffers either.

To analysis simulation data more precisely, statistics method is being used. From Figure 11, the relationship between the time and the packet drop rate can be estimated as a simple linear regression model. Summary statistics for the data given in the figure are($n = 3540$):

$$\bar{t} = 182.95 \qquad \sum t = 647643 \quad \sum t^2 = 155454503.9$$
$$\sum loss = 105.85078 \quad \bar{loss} = 0.0299 \quad \sum loss^2 = 3.28224$$

The Least-squares estimates for the slope $b_1$ and intercept $b_0$ are:

$$\begin{aligned} b_1 &= \frac{n\sum t_i loss_i - (\sum t_i)(\sum loss_i)}{n\sum t_i^2 - (\sum t_i)^2} \\ &= 0.000007 \\ b_0 &= \bar{loss} - b_1\bar{t} \\ &= 0.029907 \end{aligned}$$

The slope $b_1$ close to zero shows that the packet drop rate is close to a constant. This also verified the queue in the buffer running RED is almost constant too. For these data

$$\begin{aligned} S_{tt} &= \left[n\sum t^2 - \left(\sum t\right)^2\right]/n \\ &= [3540(155454503.9) - (647643)^2]/3540 \\ &= 36968217.05 \\ S_{ll} &= \left[n\sum loss^2 - \left(\sum loss\right)^2\right]/n \\ &= [3540(3.282243704) - (105.8507876)^2]/3540 \\ &= 0.117161999 \end{aligned}$$

Using these data, we obtain

$$\begin{aligned} SSE &= S_{ll} - b_1 S_{tl} \\ &= 0.117161999 \end{aligned}$$

Hence

$$\begin{aligned} S^2 &= SSE/(n-2) \\ &= 0.117161999/3538 \\ &= 0.0000331153 \end{aligned}$$

A 95 *percent* confidence intervals on the slope of this regression line is given by,

$$\begin{aligned} & b_1 \pm t_{0.025}\frac{S}{\sqrt{S_{tt}}} \\ = \; & 0.000007 \pm 1.96\frac{\sqrt{0.0000331153}}{\sqrt{36968217.05}} \\ = \; & 0.000007 \pm 0.000002 \end{aligned}$$

A 95 *percent* confidence intervals on the intercept of this regression line is given by,

$$\begin{aligned} & b_0 \pm t_{0.025}\frac{S\sqrt{\sum t^2}}{\sqrt{nS_{tt}}} \\ = \; & 0.0299 \pm 1.96\frac{\sqrt{0.0000331153}\sqrt{155454503.9}}{\sqrt{3540(36968217.05)}} \\ = \; & 0.0299 \pm 0.0004 \end{aligned}$$

Now we can be 95 per cent confident that the true regression line crosses the $y$ axis between the points $y = 0.0295$ and $y = 0.0303$. Comparing the statistical result with the matlab solutions, the results match well. We can use the same method to analysis the queue sizes and packet loss rates in each buffer. Matlab solutions are close to the OPNET results. But OPNET has a lower loss rate combined with a lower mean queue size. In the mean-field algorithm, the timeouts have not been accounted for. If timeouts are counted(see [2]), the solver can predict the loss rate and stable queue size much better.

# Chapter 3

# Solver for Buffers Implementing CDR

This chapter explains the other function of the solver. The solver can solve the stabilized queue size and loss rate of the buffers that drop packets at a constant rate.

First, a review is presented of the Dynamic RED scheme, which is a new AQM developed by Nortel networks. As can be found in the following section, the essence of DRED is to drop packets at a constant rate while a queue is under a dynamic threshold. The solver can, therefore, be used to predict the long run state for the buffers that implement DRED algorithm.

## 3.1 DRED Strategy

DRED is an improved algorithm for active queue management, and is capable of stabilizing a router queue at a pre-specified level independent of the number of active connections. DRED enhances the effectiveness of RED schemes by dynamically changing the threshold setting as the number of connection changes. A block diagram of the this control techniques is shown in Figure 14.(see [5]).

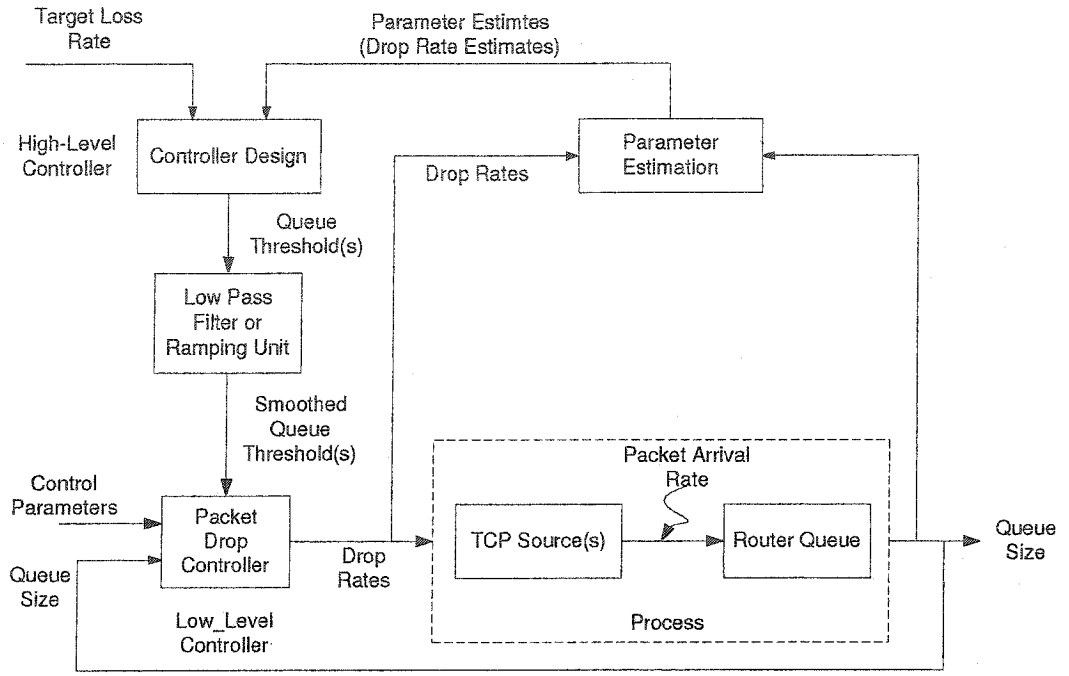



Figure 14: *Technique for online adjustment of queue thresholds.*

The drop rate of the process is estimated based on observations of packet drop rates and queue size. The high level controller, or controller design, computes the queue threshold adjustments from the estimated drop rate and the target loss rate.

For example, the actual packet loss rate is approximated by $p_1$, and the target loss rate is $\theta_{max}$. If $|p_1 - \theta_{max}| > \varepsilon$ for $\delta$ seconds, then the threshold $T$ is changed to $[T + \Delta T sgn[p_1 - \theta_{max}]]_{T_{min}}^{T_{max}}$. $\varepsilon$ is a tolerance value to eliminate unnecessary updates of $T$; $\delta$ is an elapse time used to check the loss rate mismatch; $\Delta T$ is the control step size and is equal to $B/K$ (the buffer size $B$ divided into $K$ bands); $sgn[.]$ denotes the sign of [.], which means $T$ that is either minus or plus $\Delta T$ based on whether $p_1$ is greater or less than $\theta_{max}$; $T_{max}$ is an upper bound on $T$, since $T$ cannot be allowed to be close to $B$ resulting in drop-tail behavior; $T_{min}$ is a lower bound on $T$ in order to maintain high link utilization since $T$ should not be allowed to be close to zero.

Before the queue thresholds adjustment enters the low-level controller, it is fed through a low-pass filter or a ramping module. In this way, the step function can be

made smoother.

There is also a low-level controller called packet drop controller. It computes the packet drop rate based on the queue threshold, the smoothed queue threshold adjustments which is from the high-level controller, and the actual queue size.

Continuing the previous example, the DRED computations can be summarized as follows [5]:

At time $n$, sample queue size is $q(n)$, and together with the slow varying threshold $T(n)$, the current drop probability can be computed by

$$P_d(n) = [P_d(n-1) + \alpha \frac{\hat{e}(n)}{2T(n)}]_0^{P_{max} \leq 1} \tag{3.28}$$

where $\hat{e}(n) = (1-\beta)\hat{e}(n-1) + \beta(q(n) - Tn))$. $\alpha$ is the control gain, and $\beta$ is the filter gain. The $2T(n)$ term in the above computation is simply a normalization parameter, so that the chosen control gain $\alpha$ can be preserved throughout the control process.

The buffer uses $P_d(n)$ as its drop probability until time $n+1$, when a new $P_d$ is to be computed again, and $\hat{e}(n)$, $P_d(n)$ are stored and used at time $n+1$.

From here, it can be seen, in DRED, that the packet drop rate remains constant until the arrival of the next time slot,regardless of the queue size. Furthermore, the packet drop rate converges asymptotically to the actual loss rate. Thus the packet drop rate $P_d(n)$ approximates the actual loss rate very well.

## 3.2 Model for Buffers Using Constant Drop Rate

When the buffer drops packets with a constant random probability, the queue size under a certain set threshold no longer effect on the kill rate any more. However in the steady state, the loss rate is still directly related to the mean window size of the source in the network by the mean field limit Equation (2.13)

$$\int_w w f_k(w) dw = \alpha \sqrt{\frac{1}{k}}$$

where $\alpha \approx 1.310$. The RTTs are still equal to the propagation delay in the wire plus the queueing delay in the buffer. The network specified in Chapter 2, Figure 3 is used here. The RTTs for each group are the same as Equation (2.16), Equation (2.17), Equation (2.18). The stable point of the queue is also satisfied with Equation (2.19), Equation (2.21), Equation (2.22). In traditional RED, $k_{12}$, $k_{23}$, and $k_{32}$ are functions of the queue size of each buffer, while in the constant drop rate algorithm, they are known numbers, and are set by the router administration. Choice of the kill rates for the buffer are explained in next section. The equations are thus:

$$\left[ \left( N_1 \frac{\alpha \sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1} + N_2 \frac{\alpha \sqrt{\frac{1}{k_{12}}}}{RTT_2} \right) \times PacketSize \right] \times (1 - k_{12}) \leq B_{12}$$

$$\left[ \left( N_1 \frac{\alpha \sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1} (1 - k_{12})(1 - k_{23})(1 - k_{32}) + N_2 \frac{\alpha \sqrt{\frac{1}{k_{12}}}}{RTT_2} (1 - k_{12}) \right) \times AckSize \right] \times (1 - k_{21}) \leq B_{21}$$

$$\left[ \left( N_1 \frac{\alpha \sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1} (1 - k_{12}) \right) \times PacketSize + \left( N_3 \frac{\alpha \sqrt{\frac{1}{k_{32}}}}{RTT_3} (1 - k_{32}) \right) \times AckSize \right] \times (1 - k_{23}) \leq B_{23}$$

$$\left[\left(N_3\frac{\alpha\sqrt{\frac{1}{k_{32}}}}{RTT_3}\right)\times PacketSize+\left(N_1\frac{\alpha\sqrt{\frac{1}{k_{12}+k_{23}}}}{RTT_1}(1-k_{12})(1-k_{23})\right)\times AckSize\right]\times(1-k_{32}) \leq B_{32}$$

where the RTTs

for group 1:

$$RTT_1 = T_{12}+T_{23}+T_{32}+T_{21}+\frac{q_{12}}{B_{12}}+\frac{q_{23}}{B_{23}}+\frac{q_{32}}{B_{32}}+\frac{q_{21}}{B_{21}}$$

for group 2:

$$RTT_2 = T_{12}+T_{21}+\frac{q_{12}}{B_{12}}+\frac{q_{21}}{B_{21}}$$

for group 3:

$$RTT_3 = T_{32}+T_{23}+\frac{q_{32}}{B_{32}}+\frac{q_{23}}{B_{23}}$$

There are four equations and four unknown variables $q_{12}$, $q_{21}$, $q_{23}$, and $q_{32}$. Matlab can be used to solve these mean-field mathematical equations. The solution of the equations are the steady states of the queue size in the buffers of this small network.

## 3.3 Analysis of the Model

According to the mathematical equations described in the previous section, the solver is designed to have one more function. Not only can it predict the steady state for the buffers using RED algorithm as their active queueing management scheme, it can also predict the steady state for the buffers using the constant drop rate algorithm. The solver generates the mean-field equations after the user chooses the constant drop rate as the AQM scheme in the router, and finally calculates the solution for the equations. The result shows the long-run steady state of the queue size in each buffer.

After the solver is built , it needs to be verified. A small network is assumed in both Matlab and in OPNET, the mathematical results from the Matlab and the simulation results from OPNET are compared. In order to have a clear comparison with the RED AQM scheme in Chapter 2, the topology and some parameters of these two small network are made the same.

1. There are three routers in the network

2. The links between the routers are all $T3$ line. The propagation delay between router 1 and router 2 is 0.025 second, and 0.05 seconds between router 2 and router 3. The buffers run RED, All the buffers have the same buffer size and maximum packet dropping rate.

3. All the buffers are same: they can all buffer 4000 packets in their memory; the packets dropping rates are fixed at 0.02, and the minimum thresholds are zero, and the maximum threshold is 4000.

4. There are three groups going through the network with their paths fixed. Group 1 goes through router 1, router 2, and router 3 to destination 1. Group 2 passes router 1 and router 2 to destination 2. There are 100 sources in group 1 and group 2, 200 sources in group 3. Group 3 goes from router 2 to router 1 to destination 3.

### 3.3.1 Numerical Results From Matlab

In Figure 6, there are two algorithms that can be chosen: *RED* and *Constant Drop Rate*. After choosing the *Constant Drop Rate*, the user assigns the constant drop scheme to all the buffers in the network. The next step for the user is to define the parameters of this algorithm for all the buffers: the maximum buffer size and the constant packet dropping rate. Here the minimum queue thresholds are automatically set to zero. See Figure 15.

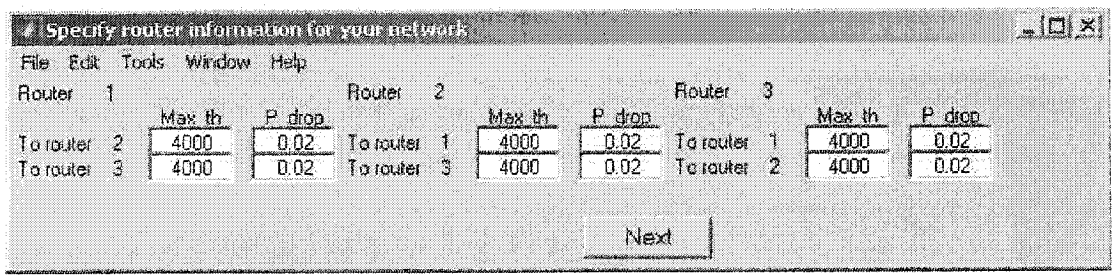



Figure 15: *Buffer configuration for constant drop rate.*

The next step is the same as for RED. The user must configure the topology of the network. See Figure 8.

After filling all the tables in the solver, the system generates a set of equations according to the constant drop rate scheme and the parameters that are set in the beginning of this section. The result is shown in Figure 16.

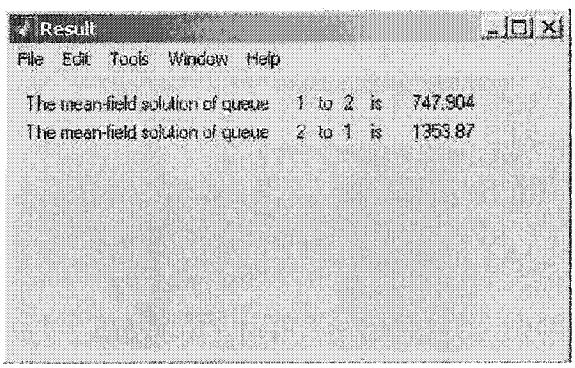



Figure 16: *Steady state for the buffer using constant drop rate.*

Thus, when the network reaches the stable steady state, queue size in buffer 12 is 747.904, and queue size in buffer 21 is 1353.87, no queues exist in buffer 23 and buffer 32.

### 3.3.2 Simulation Results by OPNET

Here, OPNET is introduced to verify the solver in this chapter. The same network for RED is used here. The network configuration is shown in Figure 10. Most parameters are set to the same as for RED. In this network, each sub-net has 100 TCP sources, and each source is running TCP Reno. Only sub-net 1, sub-net 2, sub-net 3 and sub-net 4 send data; the others are set to idle. The link bandwidth, propagation delay, maximum buffer size(maximum threshold), etc. are kept the same. The routers still use RED as their packets dropping mechanism. The maximum dropping probability, however is set to equal to the minimum dropping probability. Thus, according to the Equation (1.1), the router drops the packets at a fixed rate between the maximum threshold and the minimum threshold. The dropping probability is no longer related to the queue size; it is now equal to minimum dropping probability.

Another important setting is the minimum threshold. As introduced in the beginning of this chapter, DRED drops packets at a constant rate if the queue does not exceed a specified threshold $T$, until the next time slot arrives. In the next time slot, if the actual loss rate $p_1$ and the target loss rate $\theta_{max}$ are satisfied with the equation $|p_1 - \theta_{max}| < \varepsilon$, then the threshold $T$ is unchanged. $T$ is changeable between $T_{max}$ and $T_{min}$. $T_{max}$ is an upper bound of $T$, and it cannot be allowed to be close to $B$ (the bandwidth of the link); $T_{min}$ is a lower bound of $T$. In order to maintain high link utilization $T_{min}$ should not be allowed to be close to zero. Therefore, the minimum threshold in all buffers is set to 100 packets. The buffer does not drop packets until it reaches the minimum threshold.

The simulation runs for 360 seconds. The primary performance metrics in the constant dropping rate algorithm simulation experiments is the queue size. The queue size is sampled and plotted every 10ms. The respective queue sizes for buffer 12 and buffer 21 are shown in Figure 17 and Figure 18. Figure 19 shows that the

queues size in buffer 32 and buffer 23 are zero.

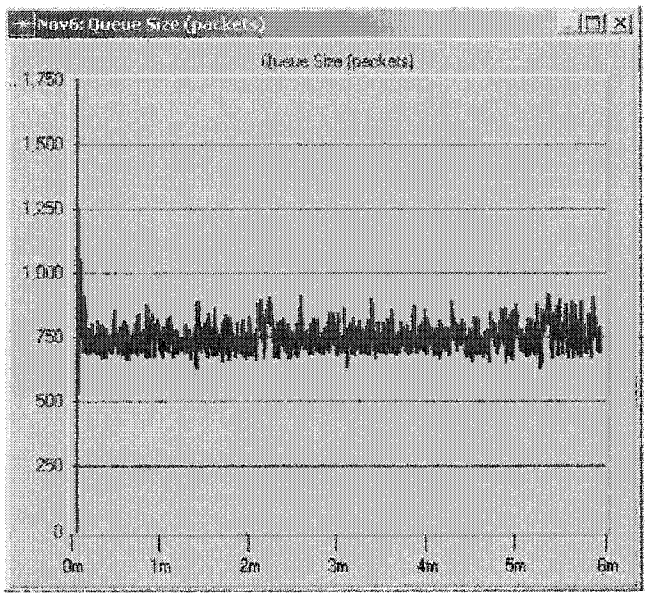



Figure 17: *Queue size in buffer* 12*(constant drop rate).*

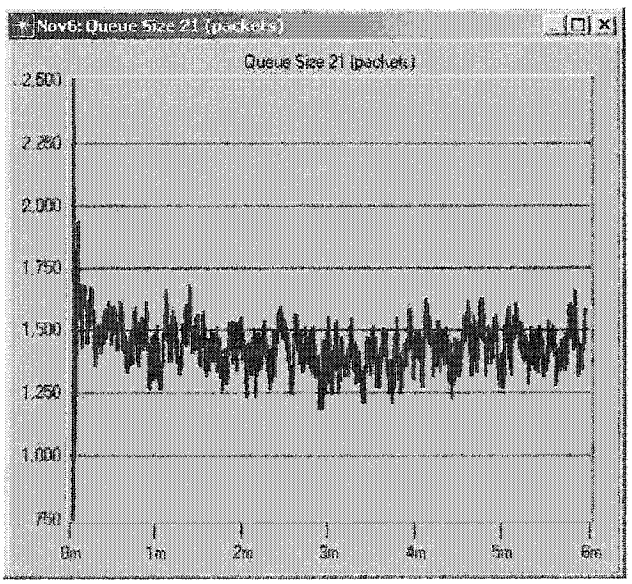



Figure 18: *Queue size in buffer* 21*(constant drop rate).*

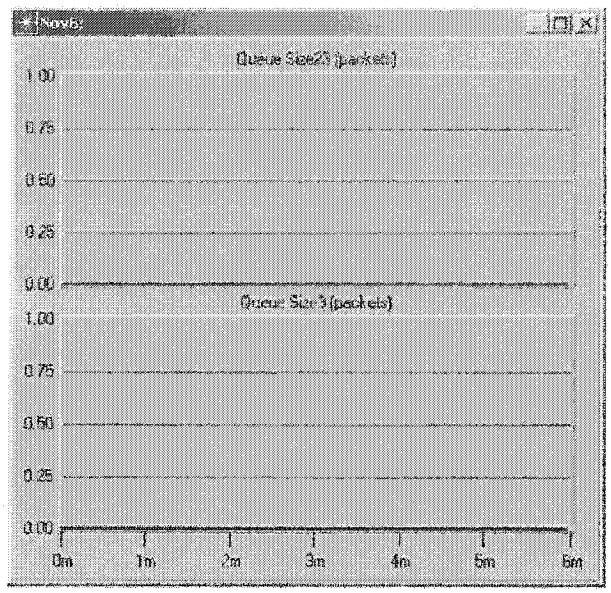



Figure 19: *Queue size in buffer* 23 *and buffer* 32 *(constant Drop Rate).*

By exporting the data sheets from OPNET, the average queue size and average drop rate after the queue becomes stable can be shown as in the following table:

| After 12 seconds, queues are stable | Mean | Standard Deviation |
|---|---|---|
| Queue size in buffer 12 | 752.5957 packets | 45.45612 packets |
| Queue size in buffer 21 | 1430.65 packets | 77.40345 packets |
| Packet drop rate in buffer 12 | 0.01848 | 0.00598 |
| Packet drop rate in buffer 21 | 0.02005 | 0.00437 |

## 3.4 Summary

Reference [2] asserts that the steady state distribution of a window size has a unique density. Also reference [6] argues that steady state $\Pi$ and the $t$-marginals of $\Pi - \Pi_t$ are unique and concludes that in a TCP network with a large constant number of connections and any constant round trip time and bandwidth, there exists a constant loss rate $k$, such that a router that randomly drops each in coming packet with probability $k$ causes the network to reach a stable steady state.

In a small TCP network, if each connection has a constant round trip time and bandwidth, the constant drop rates in the routers cause the routers to reach a steady state. Based on Equation (2.3), all RTT equations are simple multivariate equations with unique solutions.

Figure 17, Figure 18 and Figure 19 shows that the network can be stabilized with the constant drop rates of $k = 0.02$ in all the buffers. Exporting all data between 12 seconds to 360 seconds from OPNET to do precisely statistical analysis which is the same method as in the previous chapter, the calculation verified that the network is stabled. The queue 12 is stabilized around 750 packets and the queue 21 is stabilized around 1430 packets, The results match the solution of the mean-field equation built in Matlab.

# Chapter 4

# Technical Details about OPNET simulation

## 4.1 Introduction of Opnet

OPNET is a powerful network simulation tool that can be used to model communication systems and predict network performance. Opnet defines a model using a hierarchical structure. At the top there is the network model level, which is constructed from the node model level, which in turn is made from the process model level.

In this section, in order to explore the operation of each modelling level, a top-down approach is being presented. Furthermore, a briefly description for other facilities (such as Simulation Tool, Data Collection Tool and Data Analysis Tool) provided by OPNET is also introduced.

## 4.2 Network Design Details

### 4.2.1 Network Model Level

The network model contains a top level network. This top level network may consist of none or some sub-nets. A network model can be constructed with project editor by using network objets in OPNET's built-in object palette. The network palette contains network objects(e.g. subnetworks, network devices, links, etc.) for building a network model. By dragging and dropping objects from an object palette, a small network model is created, see Figure 10. In the scenario , there are seven source sub-nets, seven destination sub-nets, three routers. There are a hundred terminals link with a mux in each sub-net, see Figure 20. All links between routers and muxs are T3s. Each source and destination sub-net is the same.

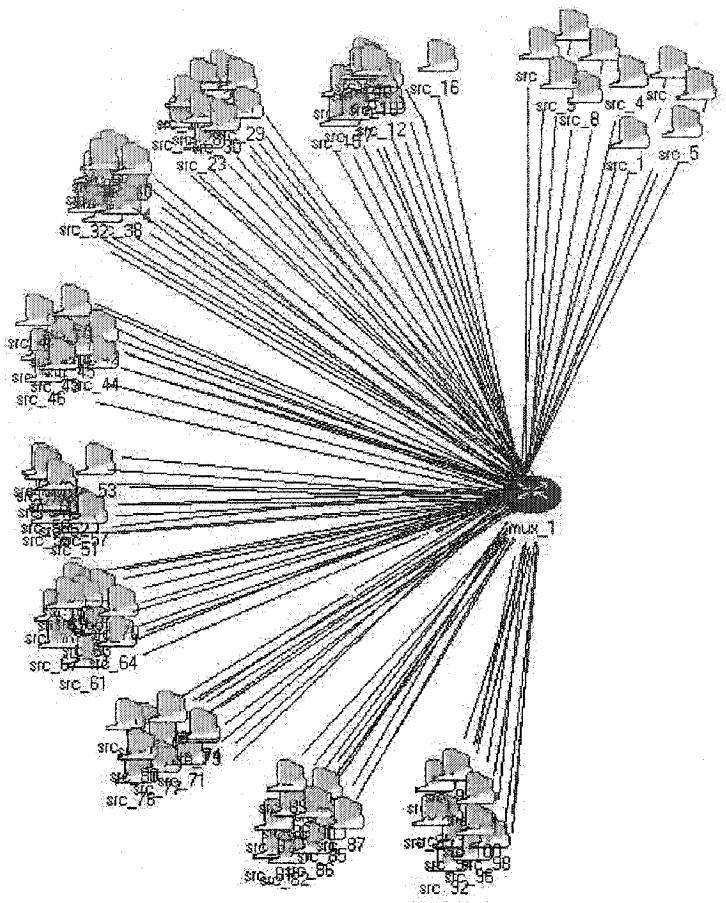



Figure 20: *subnet*

After the network model setup, each network object is characterized by setting the object attributes. Attributes specify the node model of a network object and a number of parameters associated with the network object. Those parameters determine the behavior of the nodes.

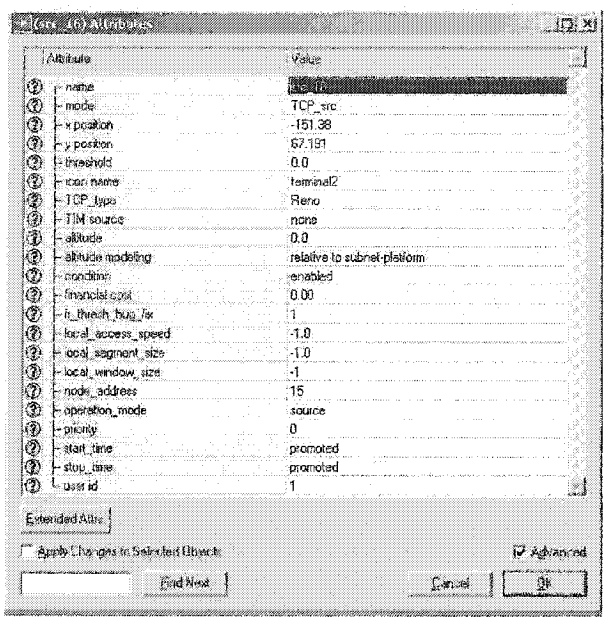



Figure 21: *Attributes of one source at network layer*

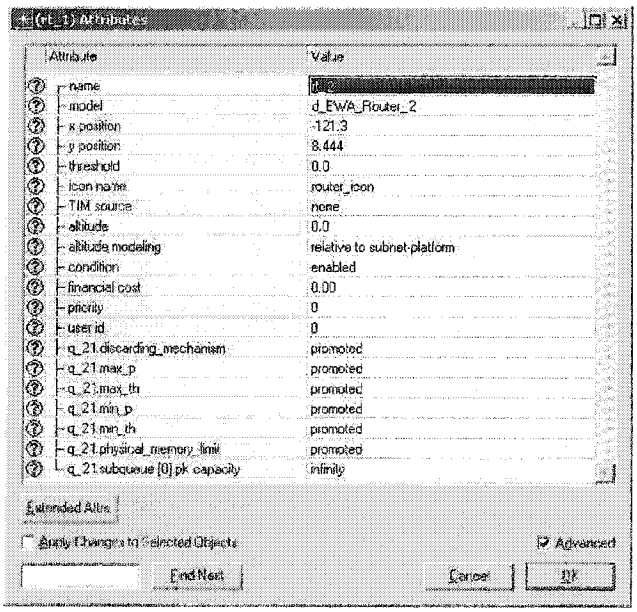



Figure 22: *Attributes of router* 2 *at network layer*

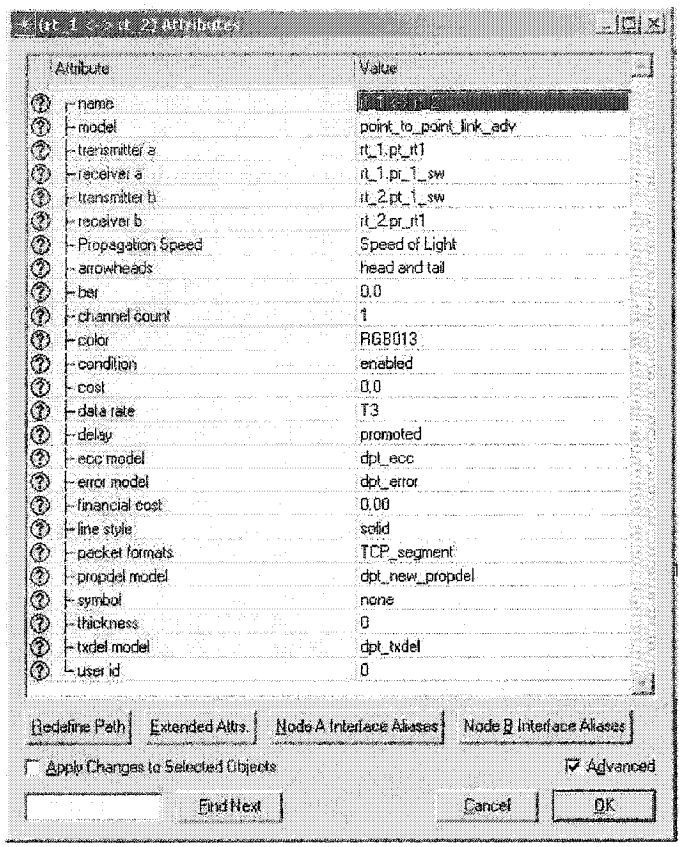



Figure 23: *Attributes of link* 12

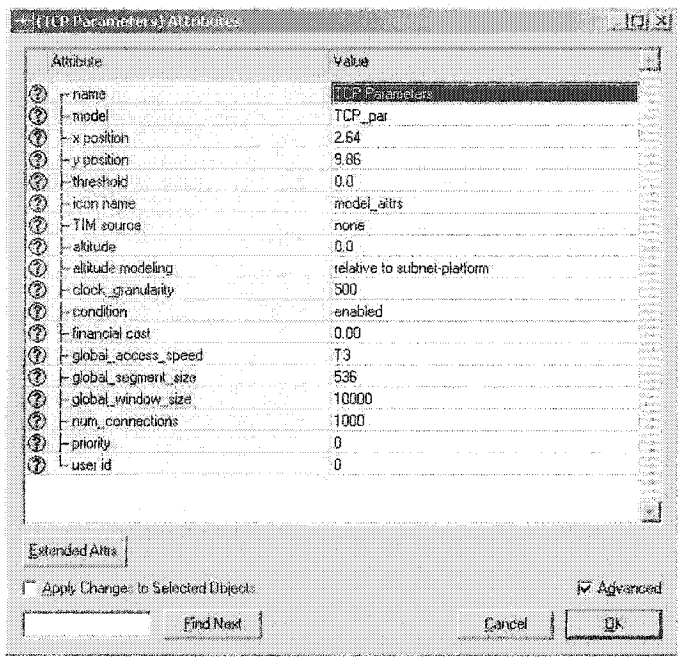



Figure 24: *TCP parameters.*

By right clicking on the object, the attributes of an object can be viewed and modified. Figure 21 shows the attributes of terminal 16 in sub-net 1. The node model of this terminal is *TCP-src*, the sub-net id is *user id* which is 1 and the node address is 16 which represents the number of the terminal. Figure 22 shows the attributes of router 2. The node model of router 2 is named as *d-EWA-Router-2*. In this table , the parameters (such as maximum threshold, minimum threshold, max-p, min-p etc.) are set to promoted. Those parameters are promoted to a high level layer– *simulation sequence.*

*Simulation Sequence* is the name of the OPNET simulation tool. In this tool, the user is allowed to set the simulation time, seeds, and global variables and object attribute settings and so on.

In Figure 23, the node model of the link is "point-to-point-link-adv". The propagation speed is set to the speed of light. Data transmission rate is $T3$. The traffic on the line is "TCP-segment". The transmission delay is promoted to simulation sequence.

The object "TCP-parameter" in the network model is used to set the global attributes. For example (Figure 24), the global segment size is 536, and the clock granularity is 500.

Here, four objects attributes setting table are presented . For other objects in the network, also, there is an attributes configuration for each of them.

### 4.2.2 Node Model Level

Node model is used to model the internal structure of a network node(e.g. router and mux, etc.). A node model is composed of a number of modules which are the fundamental "building blocks". At the node model level, each module models some aspects of the behaviours of a node. There are five different types of modules available in Node editor. They are processor, generator, queue, transmitter and receiver.

Each type of module is used to model a particular function of the operation of a network node. A processor module represents a generic process. A generator module is used to model packet generation. The queue module models a queueing process. A queue module may consist of a number of subqueues. Transmitter modules and receiver modules are used to model packet transmission and packet reception respectively.

By double clicking the object in network, the node model is shown in a new window. The node model of TCP source is shown in Figure 25 whose name is *TCP-src* in network model Figure 21.

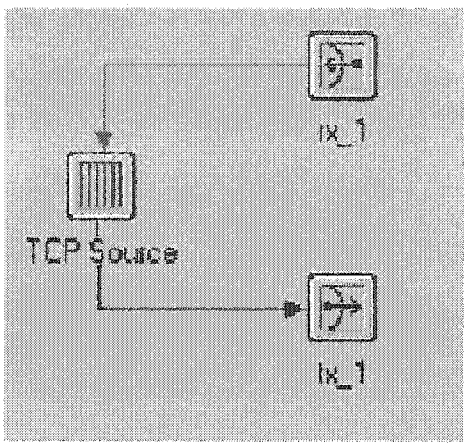



Figure 25: *Node model of TCP source.*

In this model, there are three modules, a generator, a transmitter and a receiver. The generator module can generate packets as per specified *probability Density Function*(PDF) and packet format. The transmitter sends the data out for the generator. And the receiver is used to receive the ACK from the destination for the generator.

In this thesis, the most important nodes in the network are the routers. The routers receive the packets from the sources and other routers, then switch them to the egress queue. The packets wait in the queue until they are ready for delivery.

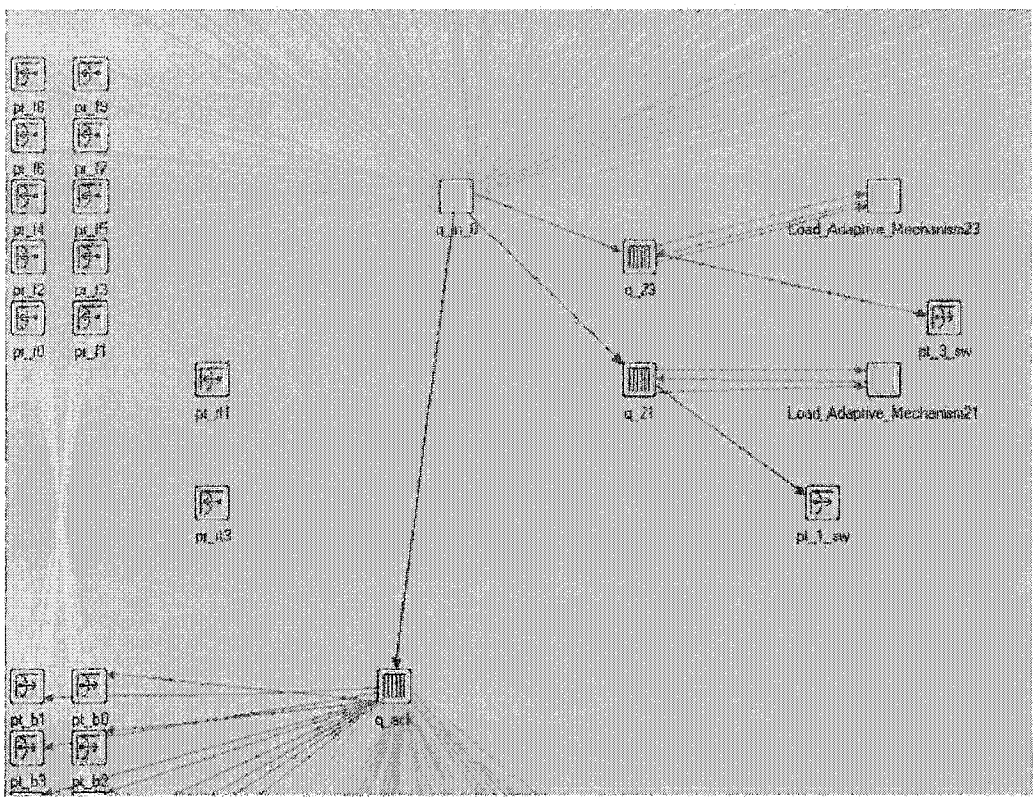



Figure 26: *Node model of router* 2.

Figure 26 is the node model of router 2 whose name is *d-EWA-Router-2* in Figure 22. The functions of this router are listed as below:

- Receive the packets including data and ACKs from router 1, router 3 and the source and destination sub-nets. So the receivers named "pr-rt1" and "pr-rt3" are respectively receiving the packets from those two routers. The receivers from "pr-f0" to "pr-f6" are receiving the packets from different sub-net.
- All packets go through the processor, which is called "q-in-0" to be switched to the outgoing queue.
- The queue "q-21" queues the packets to be transmitted to the router 1, and "q-23" is the queue to router 3. "q-ack"is for the packets to be transmitted to either destinations or sources.

- The flow control is executed in the egress queues. The survivors are transmitted by the transmitter to different paths.

The structure of the router 1 and router 3's node models is the same as router 2. To define the attributes of each module in the node level, right click on the module, an attribute window is shown below:

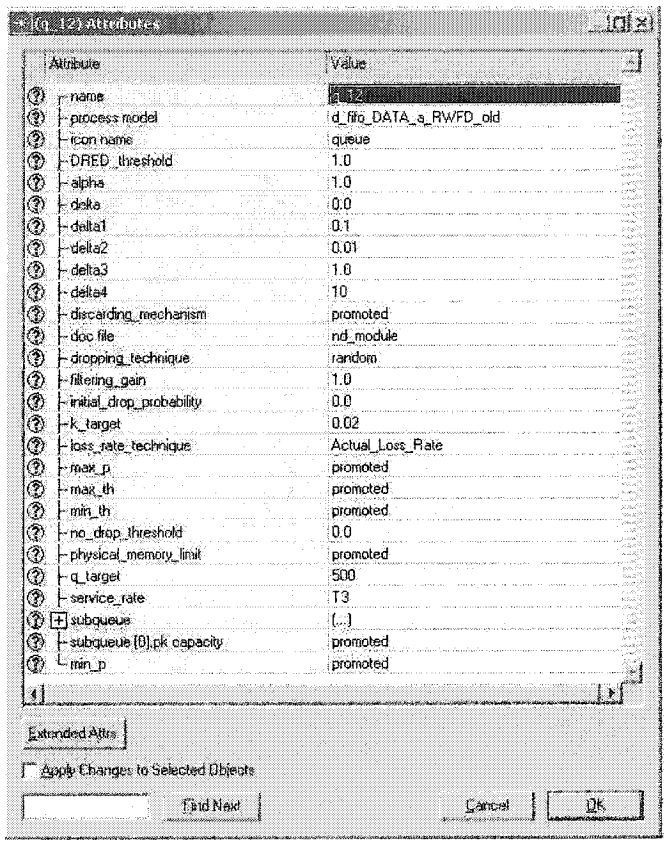



Figure 27: *Attributes of q12 in node level*

The process model and some other useful parameters are used as attributes of a node model in OPNET's modelling hierarchy.

### 4.2.3 Process Modelling

By double clicking the module in the node model level, the process model of this module is opened. A process defines a specific operation of a system, e.g. packet

generation, queueing, etc. It represents functionality that can be implemented in software. Process models are the lowest level of objects in OPNET. Each protocol or algorithm is modelled as a process. OPNET depicts processes graphically in forms of FSM (Finite State Machines). A process consists of a number of states, transitions, and conditions. States are mutually exclusive, which means that a process can only be in one state at any given time. Actions taken in a state are called *executives* in OPNET's terminology. Executives of a state are split into two parts – *enter executives* and *exit executives*. The enter executives of a state are executed when a process enters the state, and the exit executives are executed when the process leaves the state.

The process model of our flow control algorithm – *RED* is shown in Figure 28:

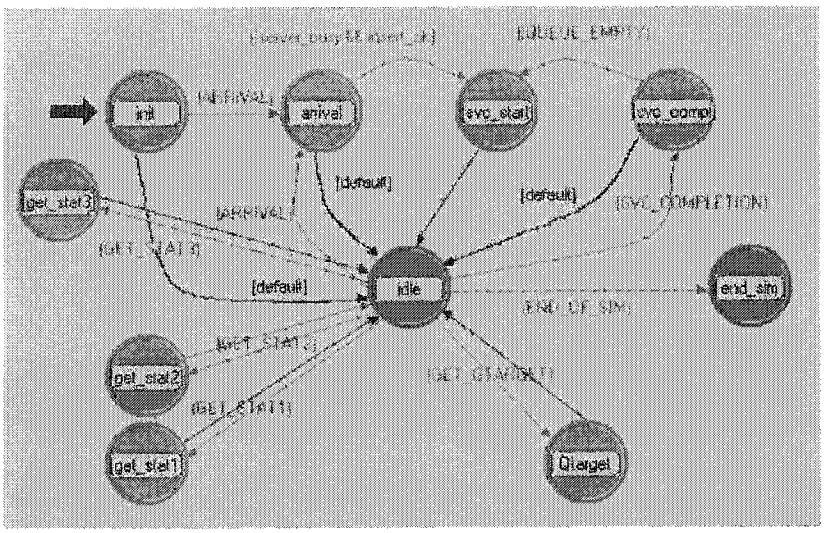



Figure 28: *Queue's process model.*

The state *init* is an initial state to define some initial values for some parameters. "ARRIVAL" is a condition that controls the transition between states "*init*" and "*arrival*". The state *arrival* is a forced state. The process retains control of the simulation until it has completed all its tasks. RED algorithm is working in this state. For each incoming packet, the router randomly drops it at a specified rate. The survivors are successfully inserted into the queue for further process. Then the process will transit to an unforced state "*idle*". Or the process will transit to "*svc-start*" if the server is not busy.

In the state "*svc-start*", the CPU looks up a route and changes header, decides which interface packet should be sent through [8]. So the service time or processing delay on the other hand is dependent upon the processor architecture and independent of the traffic. For a specified router, the processing time to serve a packet is fixed regardless of the size of the packets. Here we set the "pkt-svc-time" is set to "global-packet-size" divided by "service-rate".

The process continues serving the packets until it has completed the tasks and moves into *idle* again. The unforced state can be interrupted by an interrupt "SVC-COMPLETION" which is a Boolean condition, if TRUE, causes a transition to another state *svc-compl.* In this state, the packet in the front of the queue is being sent out bit by bit.

In the whole process, from the packet entering the router to being sent out, the total delay consists of three components: process delay which is fixed in the router, transmission delay which is the delay between the time that the first and last bits of the packet are transmitted, and the queueing delay which is the delay that a packet waits for transmission. The queueing delay varies by the packet length, number of packets in the queue, ingress and egress rate. This is what the model is interested in, and the algorithms work on.

### 4.2.4 Data Collection Tool

In OPNET, advanced data collection is supported by Probe Editor which is activated by selecting *Choose Statistics(Advanced).* The probe editor is used to create *probe objects* which specify where to collect statistics and what statistics to be collected. Like other OPNET objects, a probe object also has a set of attributes which can be configured to specify the name of the probe, statistic, and the detail location for data collection, etc. The probe objects are saved in a *probe file* which is an attribute of a simulation object.

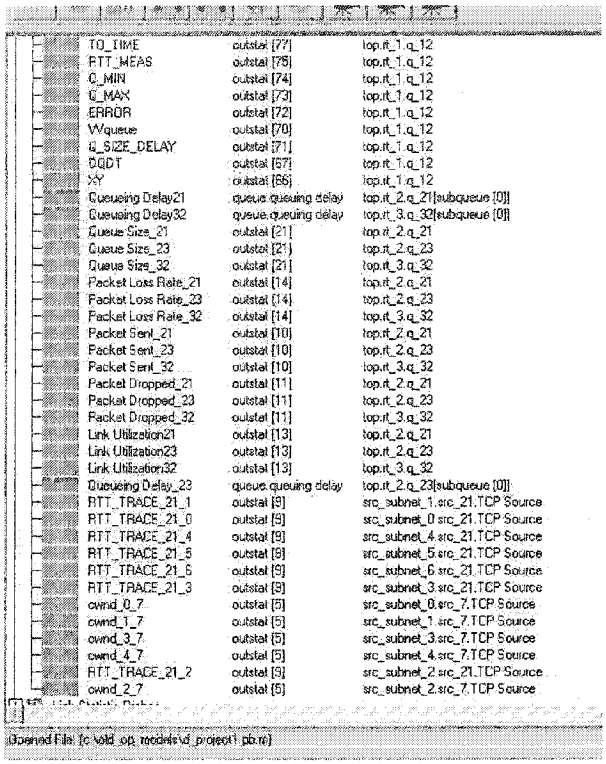



Figure 29: *Probe model.*

### 4.2.5 Simulation Tool

Simulation sequences is an advanced simulation tool which is used to control the execution of the simulation. The simulation sequences may consist of one or more simulation objects. Each simulation object has a set of attributes.

The promoted object attributes are defined in this attributes set. This is a very good feature for some parameters changing frequently. For example when simulating two or more similar networks, most parameters are the same except for a few attributes. Those attributes can e set to promoted. The user can change them only in this level simultaneously. It is not necessary to set them in network model, node model or process model level.

The probe model, network model which needs to be simulated, and the simulation result output vector file are also defined in attributes set.

### 4.2.6 Data Analysis Tool

After the simulation, the results are saved in the *output vector file*. Analysis tool extracts data from output vector file and displays it in *analysis configuration*. The analysis tool is usually used to process statistics specified in probe objects which are created in the probe model.

### 4.2.7 Summary

OPNET is a very complex but helpful network simulation tool. This is a simple introduction. Usually, a big network can be modelled by this top-down approach. This approach shows the network hierarchy clearly and understandably. OPNET can simulate much more complex network than this project. However, more features need to be explored.

# Chapter 5

# Conclusions

The contribution of this thesis is to design of a solver in Matlab in order to predict the stable steady state of a TCP network when the routers implement two AQM schemes: RED and constant drop rate. To a system designer, the parameters' selection for RED and constant drop rate algorithm can lead to some valuable insight into the appropriate choice for the packet drop rate and queue size. A network designer usually expects to have a good guide before choosing parameters for the network. The solver can suggest suitable parameters to achieve an expected packet drop rate and queue size.

In chapter 2, it is already known that in the steady state, the window size is relate to the loss rate in the buffers.

$$\int_w w f_k(w) dw = \alpha \sqrt{\frac{1}{k}}$$

where $\alpha \approx 1.310$. Thus, for each group of $N_n$ sources, the number of packets in the network is given by

$$N_n \alpha \sqrt{\frac{1}{k_1 + k_2 + ... + k_m}}$$

where $k_1$, $k_2$,..., $k_m$ are the loss rates of the buffers that the packets go through. Thus by the definition of transmission rate, theoretically, each group can transmit

$$\frac{N_n \alpha \sqrt{\frac{1}{k_1+k_2+\ldots+k_m}}}{RTT_n}$$

in packets per second. When some flows share one egress link, the total transmission rate times the portion that is not dropped should not exceed the bandwidth of the output link, this is the main idea developed in this thesis. This principle can handle much larger systems: a few groups with different round trip time and different path.

It can be seen from Chapter 2 and Chapter 3, the OPNET simulation results verifies the matlab results, thus the solver has been proven useful.

From OPNET simulation, it also can be seen that not all types of combinations of the parameters can make the network stable. The solver can suggest a suitable parameter set for the network designer.

On the other hand, the solver also can be used to predict whether or not a small TCP network can be stabilized or not by randomly dropping packets with a chosen constant probability or by RED parameters.

# Chapter 6

# Future work

There exits other Active Queueing Management Schemes and networks that can be more complex than the model studied here. The prediction to the steady state of a network is helpful to the network designer in designing a network of high throughput and high utilization. The solver can not find a steady state for some complicated networks due to the limitation of time and calculation abilities of Matlab.

The other limitation of the solver is that the data path in the solver is fixed. The router acts much more like a layer two switch than a router. In real TCP networks, data may go different ways; the solver discussed here does not apply.

The above are interesting topics for further study.

# Bibliography


[1] STEVENS, W., NOAO(January 1997). TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms [RCF2001]

[2] BACCELLI, F., MCDONALD, D. R., REYNIER, J.(2002). A mean-field model for multiple TCP connections through a buffer implementing RED

[3] COMER, DOUGLAS E.(2000). Internetworking with TCP/IP principles, protocols, and architectures

[4] www.opnet.com

[5] J.AWEYA, M.OUELLETTE, D.Y.MONTUNO, A.CHAPMAN(January 2001). Aload adaptive mechnism for buffer management

[6] MICHAEL MASKERY(May 2003). Stochastic Stability of TCP Networks under Random Packet Dropping Schemes

[7] BRIAN R. HUNT, RONALD L. LIPSMAN, JONATHAN M. ROSENBERG(2001).A Guide to MATLAB

[8] WAHEED UZ ZAMAN, SYED AFFAN AHMAD, ADEEL ABBAS, ABID QADEER(2002).Modeling Of Processor Delay And Oerall Reduction In Network Latencies For Real Time, Interactive Applications.

[9] J. SUSAN MILTON, JESSE C. ARNOLD(2003). Introduction To Proability And Statstics.

# Appendix A

# Program and Figure Generate Platform

All program and figures were generated in the operation system windows 2000.

Figure 1, Figure 2, Figure 3, Figure 4, Figure 14 were generated by Microsoft Visio 2000.

The solver was built in Matlab Version 5.3.0 Release 11.

The simulation was in OPNET Version 9.0

# Appendix B

# Code List of Process Model of $q_{21}$

```
--------------------------------------------------------------------------
File: Start1.m
--------------------------------------------------------------------------

Init1;      % Initial all parameter

Figure1;    % get the number of the node

--------------------------------------------------------------------------
File: Figure1.m
--------------------------------------------------------------------------

%Create a GUI to indicate the user to enter the number of Nodes for one
%specified networks and give this positive integer number to an available N
%function test

h_fig = figure( 'position',[300 200 250 150],...
   'resize', 'on',...
   'name', 'Number of nodes',...
   'numbertitle', 'off');

h_frame = uicontrol (h_fig,...
   'Position', [0 0 1 1], ...
   'Style','frame');

uicontrol (h_fig,...
   'Position', [50 95 150 40],...
   'String', 'Please enter the number of routers in your network',...
   'Style', 'text',...
   'FontSize', 10);

h_edit = uicontrol (h_fig,...
   'style','edit',...
   'position',[90 65 60 20],...
   'callback','get_number',...
   'string', '3');

h_next= uicontrol (h_fig,...
   'Position', [ 90 25 60 25],...
   'Style', 'pushbutton',...
   'String', 'OK',...
   'FontSize', 10,...
   'callback','verify');
```

```
Handel1=[h_edit];
set(h_fig,'Visible','on', ...
   'UserData',Handel1);
get_number;
```

---

**File: get_number.m**

---

```
Handel1=get(gcf,'UserData');
x=Handel1(1);
N=str2num(get(x,'String'));
```

---

**File: verify.m**

---

```
x=floor(N);
if  N>=2
   if  N == x        %verify if the number is legal
      link_info;
   else
      h_error_fig = errordlg (' Not a valid input','Input Error','on');
   end
else
   h_error_fig = errordlg (' Not a valid input','Input Error','on');
end
```

---

**File: link_info.m**

---

```
% specify link information for the network

global linkrate delay

   b=N;
   x=1;

   while b>4
      b=b-4
      x=x+1;
   end;
   h_link = figure( 'units','pixels',...
      'position',[10 35 785 x^2*70+29+12*x],...
      'resize', 'on',...
      'name', 'Specify link information for your network',...
      'numbertitle', 'off');

   h_next1= uicontrol (h_link,...
       'units','pixels',...
       'Position', [220 5 125 25],...
       'Style', 'pushbutton',...
       'String', 'Constant loss rate',...
       'FontSize', 10,...
       'callback','cred_info');

   h_next2= uicontrol (h_link,...
       'units','pixels',...
```

```
        'Position', [450 5 75 25],...
        'Style', 'pushbutton',...
        'String', 'RED',...
        'FontSize', 10,...
        'callback','red_info');

for b=1:x

for a=0:3

   m=(b-1)*4+a+1;

if m<=N
uicontrol('Parent',h_link,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position', ...
       [a*192 (x^2*70+29+12*x-15*(2*b-1)-(4*x-1)*18*(b-1)) 65 15], ...
      'String','From router', ...
      'Style','text', ...
      'Tag','StaticText12');

uicontrol('Parent',h_link,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position', ...
      [65+a*192 (x^2*70+29+12*x-15*(2*b-1)-(4*x-1)*18*(b-1)) 15 15],...
      'String', m, ...
      'Style','text', ...
      'Tag','StaticText12');

uicontrol('Parent',h_link,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position', ...
      [80+a*192 (x^2*70+29+12*x-30*b-(4*x-1)*18*(b-1)) 50 15], ...
      'String','lk_rate', ...
      'tooltipString','link ratein Mbps', ...
      'Style','text', ...
      'Tag','StaticText12');
uicontrol('Parent',h_link,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position', ...
      [140+a*192 (x^2*70+29+12*x-30*b-(4*x-1)*18*(b-1)) 50 15], ...
      'String', 'Delay(s)', ...
      'Style','text', ...
      'Tag','StaticText12');

Edit3(m,m) = uicontrol('Parent',h_link,'Units','pixels','BackgroundColor', ...
   [1 1 1], 'Callback','get_link', ...
   'ListboxTop',0, ...
   'Position',[800 750 50 18], ...
   'String','0', ...
   'Style','edit','Tag','EditText1');

-----------------------------------------------------------------------
```

**File: get_link.m**

```
linkrate=zeros(N^2,1);
delay=zeros(N^2,1);

Handle4=get(gcf,'UserData');

for n1=1:N
   for n2=1:N
   x= Handle4((n1-1)*N+n2);
   linkrate((n1-1)*N+n2)=max(0,eval(get(x,'String')));

   x= Handle4(N^2+(n1-1)*N+n2);
   delay((n1-1)*N+n2)= max(0,eval(get(x,'String')));
   end;
end;
```

**File: red_info.m**

```
% specify router information for the network

global Pmax Qmax

b=N;
x=1;

while b>4
   b=b-4
   x=x+1;
end;

h_table = figure( 'units','pixels',...
      'position',[10 35 785 x^2*70+29+12*x],...
      'resize', 'on',...
      'name', 'Specify router information for your network',...
      'numbertitle', 'off');

  h_next= uicontrol (h_table,...
       'units','pixels',...
       'Position', [335 5 75 25],...
       'Style', 'pushbutton',...
       'String', 'Next',...
       'FontSize', 10,...
       'callback','source_info');


for b=1:x

for a=0:3

   m=(b-1)*4+a+1;

   if m<=N
   uicontrol('Parent',h_table,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
```

```
        'ListboxTop',0, ...
        'Position', ...
        [a*192 (x^2*70+29+12*x-15*(2*b-1)-(4*x-1)*18*(b-1)) 40 15], ...
        'String','Router', ...
        'Style','text', ...
        'Tag','StaticText12');

    uicontrol('Parent',h_table,'Units','pixels','BackgroundColor',...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position', ...
        [41+a*192 (x^2*70+29+12*x-15*(2*b-1)-(4*x-1)*18*(b-1)) 35 15],...
        'String', m, ...
        'Style','text', ...
        'Tag','StaticText12');

    uicontrol('Parent',h_table,'Units','pixels','BackgroundColor',...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position', ...
        [80+a*192 (x^2*70+29+12*x-30*b-(4*x-1)*18*(b-1)) 50 15], ...
        'String','Qmax', ...
        'Style','text', ...
        'Tag','StaticText12');

    uicontrol('Parent',h_table,'Units','pixels','BackgroundColor',...
         [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position', ...
        [140+a*192 (x^2*70+29+12*x-30*b-(4*x-1)*18*(b-1)) 50 15], ...
        'String', 'Pmax', ...
        'Style','text', ...
        'Tag','StaticText12');

Edit1(m,m) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position',[800,750,50,18], ...
    'String','0', ...
    'Style','edit','Tag','EditText1');

Edit2(m,m) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position',[800,750,50,18], ...
    'String','0', ...
    'Style','edit','Tag','EditText2');

for n1=1:m-1 uicontrol('Parent',h_table,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position', ...
    [1+a*192 (x^2*70+29+12*x-15*(2*b+n1)-(4*x-1)*18*(b-1)) 50 15], ...
    'String', 'To router', ...
    'Style','text', ...
```

```
    'Tag','StaticText12');

uicontrol('Parent',h_table,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position', ...
    [50+a*192 (x^2*70+29+12*x-15*(2*b+n1)-(4*x-1)*18*(b-1)) 25 15], ...
    'String', n1, ...
    'Style','text', ...
    'Tag','StaticText12');

Edit1(m,n1) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position', ...
    [80+a*192 (x^2*70+29+12*x-15*(2*b+n1)-(4*x-1)*18*(b-1)) 50 18], ...
    'String','1000', ...
    'Style','edit','Tag','EditText3');

Edit2(m,n1) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position', ...
    [140+a*192 (x^2*70+29+12*x-15*(2*b+n1)-(4*x-1)*18*(b-1)) 50 18], ...
    'String','0.05', ...
    'Style','edit','Tag','EditText4');

end;

for n2=m+1:N
 uicontrol('Parent',h_table,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.7843137254901961], ...
    'ListboxTop',0, ...
    'Position', ...
    [1+a*192 (x^2*70+29+12*x-15*(2*b+n2-1)-(4*x-1)*18*(b-1)) 50 15], ...
    'String', 'To router', ...
    'Style','text', ...
    'Tag','StaticText12');

uicontrol('Parent',h_table,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position', ...
    [50+a*192 (x^2*70+29+12*x-15*(2*b+n2-1)-(4*x-1)*18*(b-1)) 25 15],...
    'String', n2, ...
    'Style','text', ...
    'Tag','StaticText12');

Edit1(m,n2) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position', ...
    [80+a*192 (x^2*70+29+12*x-15*(2*b+n2-1)-(4*x-1)*18*(b-1)) 50 18],...
    'String','1000', ...
```

```
    'Style','edit','Tag','EditText5');

Edit2(m,n2) = uicontrol('Parent',h_table,'Units','pixels', ...
    'BackgroundColor',[1 1 1], ...
    'Callback','get_red', ...
    'ListboxTop',0, ...
    'Position', ...
    [140+a*192 (x^2*70+29+12*x-15*(2*b+n2-1)-(4*x-1)*18*(b-1)) 50 18],...
    'String','0.05', ...
    'Style','edit','Tag','EditText6');

end;
end;
end;
end;

for n1=1:N
   for n2=1:N
      handle3((n1-1)*N+n2)=Edit1(n1,n2);
      handle3(N^2+(n1-1)*N+n2)=Edit2(n1,n2);
   end;
end;

set(h_table, 'Visible','on', ...
   'UserData',handle3);
get_red;
```

---

**File: get_red.m**

---

```
   Pmax=zeros(N^2,1);
   Qmax=zeros(N^2,1);

Handle3=get(gcf,'UserData');

for n1=1:N
   for n2=1:N
   x=Handle3((n1-1)*N+n2);
   Qmax((n1-1)*N+n2)=max(0,eval(get(x,'String')));

   x=Handle3(N^2+(n1-1)*N+n2);
   Pmax((n1-1)*N+n2)=max(0,eval(get(x,'String')));
   end;
end;
```

---

**File: source_info.m**

---

```
% specify source information for the network

global  number_sc  path

   h_source = figure( 'units','pixels',...
      'position',[10 35 785 525],...
      'resize', 'on',...
      'name','Source information for your network',...
      'numbertitle', 'off');
```

```
   h_next= uicontrol (h_source,...
       'units','pixels',...
       'Position', [335 5 75 25],...
       'Style', 'pushbutton',...
       'String', 'Next',...
       'FontSize', 10,...
       'callback','sim_info');

   uicontrol('Parent',h_source,'Units','pixels','BackgroundColor', ...
         [0.831372549019608 0.815686274509804 0.784313725490196], ...
         'ListboxTop',0, ...
         'Position',[1 492 65 30], ...
         'String','Number of Sources', ...
         'Style','text', ...
         'Tag','StaticText12');

   uicontrol('Parent',h_source,'Units','pixels','BackgroundColor', ...
         [0.831372549019608 0.815686274509804 0.784313725490196], ...
         'ListboxTop',0, ...
         'Position',[70 500 65 15], ...
         'String','From router', ...
         'Style','text', ...
         'Tag','StaticText12');

for b=2:N
  uicontrol('Parent',h_source,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
   'Position',[140+(b-2)*40 500 35 15], ...
   'String','to', ...
   'Style','text', ...
   'Tag','StaticText12');

end;

for a=1:25
Edit5(a) = uicontrol('Parent',h_source,'Units','pixels', ...
   'BackgroundColor',[1 1 1], ...
   'Callback','get_source', ...
   'ListboxTop',0, ...
   'Position',[1 490-a*18 50 18], ...
   'String','0', ...
   'Style','edit','Tag','EditText1');

Edit6(a,1)=  uicontrol('Parent',h_source,'Units','pixels', ...
   'BackgroundColor',[1 1 1], ...
   'Callback','get_source', ...
   'ListboxTop',0, ...
   'Position',[85 490-a*18 35 18], ...
   'String','0', ...
   'Style','edit', ...
   'Tag','StaticText12');

   for b=2:N

       Edit6(a,b) = uicontrol('Parent',h_source,'Units','pixels', ...
```

```
          'BackgroundColor',[1 1 1], ...
          'Callback','get_source', ...
          'ListboxTop',0, ...
          'Position',[140+(b-2)*40 490-a*18 35 18], ...
          'String','0', ...
          'Style','edit','Tag','EditText2');
   end;
end;

for n1=1:25
    handle5(n1)=Edit5(n1);

    for n2=1:N
      handle5(25+(n1-1)*N+n2)=Edit6(n1,n2);
    end;
end;

set(h_source, 'Visible','on', ...
   'UserData',handle5);
get_source;
```

---

File: get_source.m

---

```
number_sc=zeros(1,N);
path=zeros(25,N);

Handle5=get(gcf,'UserData');
for n1=1:25
  x1=Handle5(n1);
  number_sc(n1)=max(0,eval(get(x1,'String')));

  for n2=1:N
    x2=Handle5(25+(n1-1)*N+n2);
    path(n1,n2)=max(0,eval(get(x2,'String')));
   end;
  end;
```

---

**File: sim_info.m**

---

```
%_____________________________________________________________________________%
%####################| Main program for calculation |#######################%
%^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^%
global B number_step buffer

Psi=sqrt(pi/2)*
double(sym(maple('sum(2^zozo/(product(1-4^judo,judo=1..zozo)),zozo=0..10)')));
Xi=double(sym(maple('sum(1/(product(1-4^judo,judo=1..zozo)),zozo=0..10)')));
Alpha=Xi/Psi;              %Alpha

B=linkrate*1000*1000/8;
packet=536;
ack=48;

for i=1:25
```

```
   if path(i,1)~=0
      number_path=i;
   end;
end;

buffer=zeros(N^2,number_path);

for i=1:25
   j=1;
   n=1;
   if path(i,1)~=0

      while path(i,j+1)~=0
          while buffer((path(i,j)-1)*N+path(i,j+1),n)~=0
             n=n+1;
          end;
         buffer((path(i,j)-1)*N+path(i,j+1),n)=i;
         n=1;
         j=j+1;
         if j>=N
            break;
         end;
      end;

      number_step(i)=j-1;

    end;
end;

syms k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14 k15 k16 k17 k18 positive;
syms k19 k20 k21 k22 k23 k24 k25 positive;
ktemp=[k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13,k14,k15,k16,k17,k18,k19,k20,
      k21,k22,k23,k24,k25];
k=ktemp(1:N^2);
redo=0;
sc=zeros(1,N^2);

for i=1:N^2
   if buffer(i,1)~=0
      for j=1:number_path
         if buffer(i,j)~=0
            sc(i)=number_sc(buffer(i,j))+sc(i);
         end;
      end;
   end;
end;

for i=1:N^2
   bneck=0;
   sign=0;
   for j=1:number_path
      if buffer(i,j)~=0
         if((path(buffer(i,j),1)-1)*N+path(buffer(i,j),2))==i
            bneck=bneck+number_sc(buffer(i,j));
            sign=3;
         else
            sign=1;
```

```
            j1=j;
         end;
      end;
      if buffer(inv_index(i),j)~=0
         if
((path(buffer(inv_index(i),j),(number_step(buffer(inv_index(i),j))+1))-1)
*N+path(buffer(inv_index(i),j),number_step(buffer(inv_index(i),j))))==i

            bneck=bneck+number_sc(buffer(inv_index(i),j))*0.09;

         else
            sign=2;
            j1=j;
         end;
      end;
   end;

 if sign==1
 bneck=bneck*0.44736+linkrate((path(buffer(i,j1),1)-1)*N+path(buffer(i,j1),2));

 elseif sign==2
 bneck=bneck*0.44736+linkrate((path(buffer(inv_index(i),j1),
   (number_step(buffer(inv_index(i),j1))+1))-1)*N+path(buffer(inv_index(i),j1),
   number_step(buffer(inv_index(i),j1))))*0.09;

 elseif sign ==3
       bneck=bneck*44.736

 else
       bneck=bneck*0.44736;
    end;
    if bneck <= linkrate(i)
        k(i)=0;
   end;
end;

while redo==0,
   redo=1;
   clear temp_k
   for i=1: number_path
      for j=1:number_step(i)
         x=(path(i,j)-1)*N+path(i,j+1);
         y=inv_index(x);
         trans_d(i)=delay(x)+delay(y);

      queue_d(i)=Inverqueue(k(x),Pmax(x),Qmax(x))*packet/B(x)+Inverqueue(k(y),
                 Pmax(y),Qmax(y))*ack/B(y);

         if j==1
            rtt(i)=trans_d(i)+queue_d(i);
         else
            rtt(i)=rtt(i)+trans_d(i)+queue_d(i);
         end;
      end;
   end;

number_equ=1;
```

```
inv_trans(1)= k2;
for i=1:N^2
   if buffer(i,1)~=0
     for j=1:number_path
       if buffer(i,j)~=0
         x=buffer(i,j);
         for l=1:number_step(x)
            z=(path(x,l)-1)*N+path(x,l+1);
            if l==1
              y=k(z);
            else
               y=k(z)+y;
            end;
         end;

 if j==1 & ((path(x,1)-1)*N+path(x,2))==i
    trans_rt(number_equ)=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k(i))/rtt(x);
 elseif j==1 & ((path(x,1)-1)*N+path(x,2))~=i
  trans_rt(number_equ)=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k((path(x,1)-
                              1)*N+path(x,2)))/rtt(x)
  for m=2:number_step(x)
  trans_rt(number_equ)=trans_rt(number_equ)*(1-k((path(x,m)-1)*N+path(x,m+1)));
   if path(x,m)==i
     break;
   end;
  end;
 elseif j>=2 & ((path(x,1)-1)*N+path(x,2))==i

      trans_rt(number_equ)=trans_rt(number_equ)+number_sc(x)*packet*Alpha
                              *sqrt(1/y)*(1-k(i))/rtt(x);
   else
      temp=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k((path(x,1)-1)*N
           +path(x,2)))/rtt(x)
      for m=2:number_step(x)
         temp=temp*(1-k((path(x,m)-1)*N+path(x,m+1)));
         if path(x,m)==i
            break;
         end;
      end;
         trans_rt(number_equ)=trans_rt(number_equ)+temp;
    end;
    end;
   end;

   if buffer(inv_index(i),1)~=0
      for j1=1:number_path
         if buffer(inv_index(i),j1)~=0
            x1=buffer(inv_index(i),j1);
            for m=1:number_step(x1)
               z1=(path(x1,m)-1)*N+path(x1,m+1);
               if m==1
                  y1=k(z1);
               else
                  y1=k(z1)+y1;
               end;
            end;
```

```
    if j1==1
      inv_trans(number_equ)=number_sc(x1)*ack*Alpha*sqrt(1/y1)*(1-k(i))/rtt(x1);

    elseif j1>=2
    inv_trans(number_equ)=inv_trans(number_equ)+number_sc(x1)*ack*Alpha
                          *sqrt(1/y1)*(1-k(i))/rtt(x);
    end;
    end;
    end;
    else
       inv_trans(number_equ)=0;
    end;

    if k(i)~=0
       temp_k(number_equ,i)=k(i);
       temp_k(number_equ,(i+1):N^2)=0;

       steadystate(number_equ)=trans_rt(number_equ)+inv_trans(number_equ)-B(i);
       number_equ=1+number_equ;
     end;
 end;
end;

c=char(steadystate(1));

for i=2:(number_equ-1)
   c=strcat(c,',',char(steadystate(i)));
end;
sol=solve(c);

krate=zeros(1,N^2)
if size(temp_k,1)==1
   for j=1:size(temp_k,2)
      if temp_k(1,j)~=0
         for i=1:size(sol,1)
            if double(sol(i))>0
               krate(j)=double(sol(i));
            else
               if krate(j)==0
                  krate(j)=0;
               end;
            end;
          end;

      end;
   end;
else
   for i=1:size(temp_k,1)
      for j=1:size(temp_k,2)
         if temp_k(i,j)~=0
            x=getfield(sol,char(temp_k(i,j)))
            for m=1:size(x,1)
               if isreal(getfield(sol,char(temp_k(i,j)),{m}))~=0
                  krate(j)=double(getfield(sol,char(temp_k(i,j)),{m}));
                  if krate(j)<0
                     redo=0;
                     k(j)=0;
```

```
                end;
             end;
          end;
       end;
    end;
end;
end;
continu_sim;
```

---

**File: continu_sim.m**

---

```
global sim_indx

sim_indx=2;

h_sim = figure('Units','pixels',...
   'Position',[50 100 700 370], ...
   'Resize','on',...
   'name','Result',...
   'numbertitle', 'off');

if krate == zeros(1,N^2)
   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
         [0.831372549019608 0.815686274509804 0.784313725490196], ...
         'ListboxTop',0, ...
         'Position',[5 155 200 15], ...
         'String','There is no congestions in all buffers',...
         'Style','text','Tag','StaticText12');
end;
x=0
for i=1:N^2
   if krate(i)~=0
      j=i
      y=0
      while j>N
         j=j-N;
         y=y+1;
      end;
   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
         [0.831372549019608 0.815686274509804 0.784313725490196], ...
         'ListboxTop',0, ...
         'Position',[5 355-18*x 130 15], ...
         'String','The steady state for buffer',...
         'Style','text','Tag','StaticText12');

   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[135 355-18*x 15 15], ...
      'String',y+1,...
      'Style','text','Tag','StaticText12');

   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
```

```
    'ListboxTop',0, ...
    'Position',[150 355-18*x 15 15], ...
    'String','to',...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[165 355-18*x 15 15], ...
    'String',j,...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[180 355-18*x 15 15], ...
    'String','is',...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[60 337-18*x 60 15], ...
    'String','Loss rate:',...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[230 337-18*x 50 15], ...
    'String',krate(i),...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
     [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[60 319-18*x 125 15], ...
    'String','Queue size (in packets):',...
    'Style','text','Tag','StaticText12');

q=(krate(i)/Pmax)*Qmax;
uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[230 319-18*x 50 15], ...
    'String',q,...
    'Style','text','Tag','StaticText12');

uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
    [0.831372549019608 0.815686274509804 0.784313725490196], ...
    'ListboxTop',0, ...
    'Position',[60 300-18*x 140 15], ...
    'String','Queuing delay (in seconds):',...
    'Style','text','Tag','StaticText12');

t=q*536/B(i);
uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
```

```
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[230 300-18*x 50 15], ...
      'String',t,...
      'Style','text','Tag','StaticText12');

      x=x+4;
   end;
end;
```

---

**File: sim_info_cons.m**

---

```
%_____________________________________________________________________________%
%######################| Main program for calculation |######################%
%^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^%
global B number_step buffer

Psi=sqrt(pi/2)*double(sym(maple('sum(2^zozo/(product(1-
4^judo,judo=1..zozo)),zozo=0..10)')));
Xi=double(sym(maple('sum(1/(product(1-4^judo,judo=1..zozo)),zozo=0..10)')));
Alpha=Xi/Psi;              %Alpha

B=linkrate*1000*1000/8;
packet=536;
ack=48;

for i=1:25
   if path(i,1)~=0
      number_path=i;
   end;
end;

buffer=zeros(N^2,number_path);

for i=1:25
   j=1;
   n=1;
   if path(i,1)~=0

      while path(i,j+1)~=0
          while buffer((path(i,j)-1)*N+path(i,j+1),n)~=0
             n=n+1;
          end;
         buffer((path(i,j)-1)*N+path(i,j+1),n)=i;
         n=1;
         j=j+1;
         if j>=N
            break;
         end;
      end;

      number_step(i)=j-1;

    end;
end;
syms q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 q12 q13 q14 q15 q16 q17 q18 positive;
```

```
syms q19 q20 q21 q22 q23 q24 q25 positive;
qtemp=[q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14,q15,q16,q17,q18,q19,q20,
      q21,q22,q23,q24,q25];
q=qtemp(1:N^2);

k=Pmax;
redo=0;
sc=zeros(1,N^2);

for i=1:N^2
   if buffer(i,1)~=0
      for j=1:number_path
         if buffer(i,j)~=0
            sc(i)=number_sc(buffer(i,j))+sc(i);
         end;
      end;
   end;
end;

for i=1:N^2
   bneck=0;
   sign=0;
   for j=1:number_path
      if buffer(i,j)~=0
         if((path(buffer(i,j),1)-1)*N+path(buffer(i,j),2))==i
            bneck=bneck+number_sc(buffer(i,j));
            sign=3;
         else
            sign=1;
            j1=j;
         end;
end;
if buffer(inv_index(i),j)~=0
  if ((path(buffer(inv_index(i),j),(number_step(buffer(inv_index(i),j))+1))-1)*N
      +path(buffer(inv_index(i),j),number_step(buffer(inv_index(i),j))))==i

      bneck=bneck+number_sc(buffer(inv_index(i),j))*0.09;
  else
      sign=2;
      j1=j;
  end;
end;
end;
if sign==1

  bneck=bneck*0.44736+linkrate((path(buffer(i,j1),1)-1)*N+path(buffer(i,j1),2));
elseif sign==2
  bneck=bneck*0.44736+linkrate((path(buffer(inv_index(i),j1),
      (number_step(buffer(inv_index(i),j1))+1))-1)*N+
      path(buffer(inv_index(i),j1), number_step(buffer(inv_index(i),j1))))*0.09;
elseif sign ==3
  bneck=bneck*44.736
else
  bneck=bneck*0.44736;
end;
   if bneck <= linkrate(i)
        q(i)=0;
```

```
   end;
end;

while redo==0,
  redo=1;
  clear temp_q
   for i=1: number_path
      for j=1:number_step(i)
         x=(path(i,j)-1)*N+path(i,j+1);
         y=inv_index(x);
         trans_d(i)=delay(x)+delay(y);
         queue_d(i)=q(x)*packet/B(x)+q(y)*ack/B(y);

         if j==1
            rtt(i)=trans_d(i)+queue_d(i);
         else
            rtt(i)=rtt(i)+trans_d(i)+queue_d(i);
         end;
      end;
     end;

number_equ=1;
inv_trans(1)= q2;
for i=1:N^2
   if buffer(i,1)~=0
     for j=1:number_path
       if buffer(i,j)~=0
         x=buffer(i,j);
         for l=1:number_step(x)
            z=(path(x,l)-1)*N+path(x,l+1);
            if l==1
              y=k(z);
            else
               y=k(z)+y;
            end;
         end;

 if j==1 & ((path(x,1)-1)*N+path(x,2))==i
    trans_rt(number_equ)=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k(i))/rtt(x);
 elseif j==1 & ((path(x,1)-1)*N+path(x,2))~=i
    trans_rt(number_equ)=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k((path(x,1)-
                         1)*N+path(x,2)))/rtt(x)
   for m=2:number_step(x)
   trans_rt(number_equ)=trans_rt(number_equ)*(1-k((path(x,m)-1)*N+path(x,m+1)));
   if path(x,m)==i
       break;
   end;
   end;
  elseif j>=2 & ((path(x,1)-1)*N+path(x,2))==i

trans_rt(number_equ)=trans_rt(number_equ)+number_sc(x)*packet*Alpha*sqrt(1/y)
                     *(1-k(i))/rtt(x);
  else
temp=number_sc(x)*packet*Alpha*sqrt(1/y)*(1-k((path(x,1)-1)*N+path(x,2)))/rtt(x)
  for m=2:number_step(x)
        temp=temp*(1-k((path(x,m)-1)*N+path(x,m+1)));
        if path(x,m)==i
```

```
            break;
        end;
  end;
   trans_rt(number_equ)=trans_rt(number_equ)+temp;
  end;
  end;
end;

   if buffer(inv_index(i),1)~=0
      for j1=1:number_path
         if buffer(inv_index(i),j1)~=0
            x1=buffer(inv_index(i),j1);
            for m=1:number_step(x1)
               z1=(path(x1,m)-1)*N+path(x1,m+1);
               if m==1
                  y1=k(z1);
               else
                  y1=k(z1)+y1;
               end;
            end;

if j1==1
  inv_trans(number_equ)=number_sc(x1)*ack*Alpha*sqrt(1/y1)*(1-k(i))/rtt(x1);
elseif j1>=2

inv_trans(number_equ)=inv_trans(number_equ)+number_sc(x1)*ack*Alpha*sqrt(1/y1)
                      *(1-k(i))/rtt(x);
end;
end;
end;
else
   inv_trans(number_equ)=0;
end;

   if q(i)~=0
      temp_q(number_equ,i)=q(i);
      temp_q(number_equ,(i+1):N^2)=0;

      steadystate(number_equ)=trans_rt(number_equ)+inv_trans(number_equ)-B(i);
      number_equ=1+number_equ;
    end;
 end;
end;

c=char(steadystate(1));

for i=2:(number_equ-1)
   c=strcat(c,',',char(steadystate(i)));
end;
sol=solve(c);

t=1;
qsize=zeros(1,N^2)
if size(temp_q,1)==1
   for j=1:size(temp_q,2)
      if temp_q(1,j)~=0
         for i=1:size(sol,1)
```

```
            if double(sol(i))>0
               qsize(j)=double(sol(i));
            else
               if qsize(j)==0
                  qsize(j)=0;
               end;
            end;
         end;
      end;
   end;
else
   for i=1:size(temp_q,1)
      for j=1:size(temp_q,2)
         if i==1
            if temp_q(i,j)~=0
               x=getfield(sol,char(temp_q(i,j)))
               for m=1:size(x,1)
                  if isreal(getfield(sol,char(temp_q(i,j)),{m}))~=0
                     qsize(j)=double(getfield(sol,char(temp_q(i,j)),{m}));
                     t=1;
                     if qsize(j)<0
                        redo=0;
                        t=0;
                        q(j)=0;
                     elseif qsize(j) >= Qmax(j)
                        redo=0;
                        t=0;
                        q(j)=Qmax(j);
                     elseif t~=0
                          redo=1;
                          number_sol = m;
                     end;
                  end;
               end;
            end;
           elseif redo == 1
              if temp_q(i,j)~=0
           if isreal(getfield(sol,char(temp_q(i,j)),{number_sol}))~=0
              qsize(j)=double(getfield(sol,char(temp_q(i,j)),{number_sol}));
              if qsize(j)<0
                 redo=0;
                 t=0;
                 q(j)=0;
              elseif qsize(j) >= Qmax(j)
                 redo=0;
                 t=0;
                 q(j)=Qmax(j);
              elseif t~=0
                 redo=1;
              end;
           else
              qsize=zeros(1,N^2);
              redo=1;
           end;
           end;
         end;
      end;
```

```
   end;
end;
end;
continu_sim_cons;
```

---

**File: continu_sim_cons.m**

---

```
global sim_indx

sim_indx=2;

h_sim = figure('Units','pixels',...
   'Position',[100 100 350 180], ...
   'Resize','on',...
   'name','Result',...
   'numbertitle', 'off');

x=0
if no_stable == 1
   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[5 155 200 15], ...
      'String','Queues can not be stabilized',...
      'Style','text','Tag','StaticText12');
end;

if and((qsize == zeros(1,N^2)),no_stable==0)
  uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[5 155 200 15], ...
      'String','There is no congestions in all buffers',...
      'Style','text','Tag','StaticText12');
end;

for i=1:N^2
   if qsize(i)~=0
      j=i
      y=0
      while j>N
         j=j-N;
         y=y+1;
      end;
   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[5 155-18*x 170 15], ...
      'String','The mean-field solution of queue',...
      'Style','text','Tag','StaticText12');
   uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
      [0.831372549019608 0.815686274509804 0.784313725490196], ...
      'ListboxTop',0, ...
      'Position',[180 155-18*x 15 15], ...
      'String',y+1,...
```

```
        'Style','text','Tag','StaticText12');

    uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position',[195 155-18*x 15 15], ...
        'String','to',...
        'Style','text','Tag','StaticText12');

    uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position',[210 155-18*x 15 15], ...
        'String',j,...
        'Style','text','Tag','StaticText12');

    uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position',[225 155-18*x 20 15], ...
        'String','is',...
        'Style','text','Tag','StaticText12');

    uicontrol('Parent',h_sim,'Units','pixels','BackgroundColor', ...
        [0.831372549019608 0.815686274509804 0.784313725490196], ...
        'ListboxTop',0, ...
        'Position',[250 155-18*x 50 15], ...
        'String',qsize(i),...
        'Style','text','Tag','StaticText12');

      x=x+1;
   end;
end;
```

# Appendix C

# Matlab Code

```
/* Process model C form file: d_fifo_DATA_a_RWFD21.pr.c */
/* Portions of this file copyright 1992-2002 by OPNET Technologies, Inc. */


/* This variable carries the header into the object file */
static const char d_fifo_DATA_a_RWFD21_pr_c [] = "MIL_3_Tfile_Hdr_ 90A 30A modeler 7
3FDDDCCE 3FDDDCCE 1 MATH4 Administrator 0 0 none none 0 0 none 0 0 0 0 0 0
";
#include <string.h>


/* OPNET system definitions */
#include <opnet.h>

#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif


/* Header Block */

#include "math.h"
#include "stdio.h"

#define NOW                op_sim_time()
#define QUEUE_EMPTY        (op_q_empty ())
#define SVC_COMPLETION     ((op_intrpt_type () == OPC_INTRPT_SELF) && \
                             (op_intrpt_code() == Q_SVC))
#define ARRIVAL            op_intrpt_type () == OPC_INTRPT_STRM
#define END_OF_SIM         op_intrpt_type() == OPC_INTRPT_ENDSIM

#define Q_SVC                                0
#define DATA_GET3                            2
#define STAT1                                10
#define STAT2                                20
#define STAT3                                25
#define QTARGET                              27
#define DRED_DELTA                           30
#define S_RATE                               37
#define DQDT                                 67
#define XY                                   66
```

```
#define Q_DELAY                                  71
#define       ERROR                              72
#define Q_MAX                                    73
#define       Q_MIN                              74
#define RTT_MEAS                                 75
#define Q_AVG                                    76
#define TO_TIME                                  77
#define Q_TARGET                                 78
#define N_EFFT                                   79
#define R0                                       80
#define K_EQU                                    81
#define K_AVG                                    84
#define N_CNT                                    85
#define N_EFF                                    87
#define RTT_EFF                                  88
#define RTT_EST_21                               89
#define FILT_V_QUEUE                             86
#define GHOSTSEG_21                              93
#define GHOSTSEG2_21                             91
#define N_TO                                     92
#define GHOSTLIST_21                             94
#define INT_ARRIVAL_21                           90
#define BUFFER_OVERFLOW_PATTERN                  99
#define EARLY_LOSS_PATTERN                       98
#define MAX_THRESHOLD_OVERFLOW_PATTERN           97
#define RED_PROBABILITY_DROPPING                 96
#define RED_INTERMEDIATE_PROBABILITY_DROPPING    95

#define GET_STAT1    ( (op_intrpt_type() == OPC_INTRPT_SELF) && \
                                (op_intrpt_code() == STAT1) )
#define GET_STAT2    ( (op_intrpt_type() == OPC_INTRPT_SELF) && \
                                (op_intrpt_code() == STAT2) )
#define GET_STAT3    ( (op_intrpt_type() == OPC_INTRPT_SELF) && \
                                (op_intrpt_code() == STAT3) )
#define GET_QTARGET  ( (op_intrpt_type() == OPC_INTRPT_SELF) && \
                                (op_intrpt_code() == QTARGET) )

#define MIN_P                                    58
#define MAX_P                                    59
#define RHO                                      57

/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef BIN
#undef BOUT
#define       BIN    FIN_LOCAL_FIELD(last_line_passed)=__LINE__ - _block_origin;
#define       BOUT   BIN
#define       BINIT  FIN_LOCAL_FIELD(last_line_passed)=0; _block_origin = __LINE__;
#else
#define       BINIT
#endif /* #if !defined (VOSD_NO_FIN) */

/* State variable definitions */
typedef struct
        {
        /* Internal state tracking for FSM */
        FSM_SYS_STATE
        /* State Variables */
        int                             server_busy;
        double                          alpha;
        double                          drop_probability;
        double                          no_drop_threshold;
        double                          threshold;
```

```
double                  error;
double                  service_rate;
Objid                   own_id;
double                  pkt_total;
double                  pkt_total_all;
double                  pkt_drop;
double                  pkt_drop_all;
double                  pkt_send;
double                  pkt_send_all;
double                  buffer_overflow_pattern;
double                  early_loss_pattern;
double                  max_th_overflow_pattern;
int                     pkt_len;
int                     discarding_mechanism;
double                  q_size;
double                  ave_q;
double                  w_q;
double                  q_time;
double                  min_th;
double                  max_th;
double                  max_p;
double                  R;
double                  prob_b;
double                  prob_a;
Distribution*           uniform_dist;
double                  s;
double                  m;
double                  count;
Stathandle              red_ave_q;
double                  packet_loss_rate;
double                  physical_memory_limit;
double                  delta1;
double                  delta2;
double                  beta_max_p;
double                  ave_max_p;
double                  q_size_last;
double                  filtered_drop_probability;
double                  drop_probability_filter_gain;
Stathandle              filtered_pd;
Stathandle              filtered_actual_loss_rate;
int                     loss_rate_technique;
double                  filtered_packet_loss_rate;
double                  actual_loss_rate_filter_gain;
double                  filtered_prob_a;
double                  prob_a_filter_gain;
double                  last_statistic_sent_time;
double                  stat_wire_delta;
int                     dropping_technique;
double                  filtered_q_size;
int                     use_prob_b;
double                  time_21;
int                     ipkt_total;
double                  rttlist[100][10];
double                  rtt_eff;
double                  N_eff;
double                  k_equ;
double                  r0;
int                     use_dyn_k;
int                     Nlist[2000];
int                     Nlistcnt;
double                  t_last;
Distribution *          uniform_int_dist;
int                     droplist[100][10][3];
```

```
        double                          q_target;
        double                          delta3;
        double                          delta4;
        double                          k_avg;
        double                          q_min;
        double                          q_max;
        int                             q_avg;
        double                          rtt_meas;
        double                          k_target;
        double                          q_rec[1000];
        double                          dqdt;
        double                          cx[8];
        int                             cy[8];
        double                          hcx[8];
        double                          xy;
        double                          a[8];
        double                          q_avg_old;
        double                          min_p;
        double                          tserv_start;
        double                          serv_cnt;
        double                          rho;
        double                          rho_target;
        int                             zero_buffer;
        double                          inter_drop_time;
        double                          last_drop;
        int                             data_get3;
        int                             pk_type;
        } d_fifo_DATA_a_RWFD21_state;

#define pr_state_ptr                ((d_fifo_DATA_a_RWFD21_state*) SimI_Mod_State_Ptr)
#define server_busy                 pr_state_ptr->server_busy
#define alpha                       pr_state_ptr->alpha
#define drop_probability            pr_state_ptr->drop_probability
#define no_drop_threshold           pr_state_ptr->no_drop_threshold
#define threshold                   pr_state_ptr->threshold
#define error                       pr_state_ptr->error
#define service_rate                pr_state_ptr->service_rate
#define own_id                      pr_state_ptr->own_id
#define pkt_total                   pr_state_ptr->pkt_total
#define pkt_total_all               pr_state_ptr->pkt_total_all
#define pkt_drop                    pr_state_ptr->pkt_drop
#define pkt_drop_all                pr_state_ptr->pkt_drop_all
#define pkt_send                    pr_state_ptr->pkt_send
#define pkt_send_all                pr_state_ptr->pkt_send_all
#define buffer_overflow_pattern     pr_state_ptr->buffer_overflow_pattern
#define early_loss_pattern          pr_state_ptr->early_loss_pattern
#define max_th_overflow_pattern     pr_state_ptr->max_th_overflow_pattern
#define pkt_len                     pr_state_ptr->pkt_len
#define discarding_mechanism        pr_state_ptr->discarding_mechanism
#define q_size                      pr_state_ptr->q_size
#define ave_q                       pr_state_ptr->ave_q
#define w_q                         pr_state_ptr->w_q
#define q_time                      pr_state_ptr->q_time
#define min_th                      pr_state_ptr->min_th
#define max_th                      pr_state_ptr->max_th
#define max_p                       pr_state_ptr->max_p
#define R                           pr_state_ptr->R
#define prob_b                      pr_state_ptr->prob_b
#define prob_a                      pr_state_ptr->prob_a
#define uniform_dist                pr_state_ptr->uniform_dist
#define s                           pr_state_ptr->s
#define m                           pr_state_ptr->m
#define count                       pr_state_ptr->count
```

```
#define red_ave_q                    pr_state_ptr->red_ave_q
#define packet_loss_rate             pr_state_ptr->packet_loss_rate
#define physical_memory_limit        pr_state_ptr->physical_memory_limit
#define delta1                       pr_state_ptr->delta1
#define delta2                       pr_state_ptr->delta2
#define beta_max_p                   pr_state_ptr->beta_max_p
#define ave_max_p                    pr_state_ptr->ave_max_p
#define q_size_last                  pr_state_ptr->q_size_last
#define filtered_drop_probability    pr_state_ptr->filtered_drop_probability
#define drop_probability_filter_gain pr_state_ptr->drop_probability_filter_gain
#define filtered_pd                  pr_state_ptr->filtered_pd
#define filtered_actual_loss_rate    pr_state_ptr->filtered_actual_loss_rate
#define loss_rate_technique          pr_state_ptr->loss_rate_technique
#define filtered_packet_loss_rate    pr_state_ptr->filtered_packet_loss_rate
#define actual_loss_rate_filter_gain pr_state_ptr->actual_loss_rate_filter_gain
#define filtered_prob_a              pr_state_ptr->filtered_prob_a
#define prob_a_filter_gain           pr_state_ptr->prob_a_filter_gain
#define last_statistic_sent_time     pr_state_ptr->last_statistic_sent_time
#define stat_wire_delta              pr_state_ptr->stat_wire_delta
#define dropping_technique           pr_state_ptr->dropping_technique
#define filtered_q_size              pr_state_ptr->filtered_q_size
#define use_prob_b                   pr_state_ptr->use_prob_b
#define time_21                      pr_state_ptr->time_21
#define ipkt_total                   pr_state_ptr->ipkt_total
#define rttlist                      pr_state_ptr->rttlist
#define rtt_eff                      pr_state_ptr->rtt_eff
#define N_eff                        pr_state_ptr->N_eff
#define k_equ                        pr_state_ptr->k_equ
#define r0                           pr_state_ptr->r0
#define use_dyn_k                    pr_state_ptr->use_dyn_k
#define Nlist                        pr_state_ptr->Nlist
#define Nlistcnt                     pr_state_ptr->Nlistcnt
#define t_last                       pr_state_ptr->t_last
#define uniform_int_dist             pr_state_ptr->uniform_int_dist
#define droplist                     pr_state_ptr->droplist
#define q_target                     pr_state_ptr->q_target
#define delta3                       pr_state_ptr->delta3
#define delta4                       pr_state_ptr->delta4
#define k_avg                        pr_state_ptr->k_avg
#define q_min                        pr_state_ptr->q_min
#define q_max                        pr_state_ptr->q_max
#define q_avg                        pr_state_ptr->q_avg
#define rtt_meas                     pr_state_ptr->rtt_meas
#define k_target                     pr_state_ptr->k_target
#define q_rec                        pr_state_ptr->q_rec
#define dqdt                         pr_state_ptr->dqdt
#define cx                           pr_state_ptr->cx
#define cy                           pr_state_ptr->cy
#define hcx                          pr_state_ptr->hcx
#define xy                           pr_state_ptr->xy
#define a                            pr_state_ptr->a
#define q_avg_old                    pr_state_ptr->q_avg_old
#define min_p                        pr_state_ptr->min_p
#define tserv_start                  pr_state_ptr->tserv_start
#define serv_cnt                     pr_state_ptr->serv_cnt
#define rho                          pr_state_ptr->rho
#define rho_target                   pr_state_ptr->rho_target
#define zero_buffer                  pr_state_ptr->zero_buffer
#define inter_drop_time              pr_state_ptr->inter_drop_time
#define last_drop                    pr_state_ptr->last_drop
#define data_get3                    pr_state_ptr->data_get3
#define pk_type                      pr_state_ptr->pk_type
```

```
/* This macro definition will define a local variable called       */
/* "op_sv_ptr" in each function containing a FIN statement. */
/* This variable points to the state variable data structure,      */
/* and can be used from a C debugger to display their values.      */
#undef FIN_PREAMBLE
#define FIN_PREAMBLE       d_fifo_DATA_a_RWFD21_state *op_sv_ptr = pr_state_ptr;

/* Function Block */

enum { _block_origin = __LINE__ };


/* End of Function Block */

/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.              */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
       void d_fifo_DATA_a_RWFD21 (void);
       Compcode d_fifo_DATA_a_RWFD21_init (void **);
       void d_fifo_DATA_a_RWFD21_diag (void);
       void d_fifo_DATA_a_RWFD21_terminate (void);
       void d_fifo_DATA_a_RWFD21_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Process model interrupt handling procedure */

void
d_fifo_DATA_a_RWFD21 (void)
       {
       int _block_origin = 0;
       FIN (d_fifo_DATA_a_RWFD21 ());
       if (1)
              {
              Packet*              pkptr;
              Packet*              pkptr_tmp;
              double               pk_svc_time;
              int                  insert_ok;
              int                  stat_wire;
              double               random_index;
              double               temp;
              int                  ii;
              int                  jj;
              int                  seq_no;
              int                  source_addr;
              int                  subnet_id;
              int                  pkt_type;

              /* RTT Estimate */
              double               difft;

              /* Morris's Estimate for N */
              int                  h;
```

```
            int                 nclear;
            double              rand;
            double              N_efft;
            double              q_rectmp[1000];
            double              q_old;


            FSM_ENTER (d_fifo_DATA_a_RWFD21)

            FSM_BLOCK_SWITCH
{
/*---------------------------------------------------------*/
/** state (init) enter executives **/
FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "d_fifo_DATA_a_RWFD21 [init enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [init enter execs]", state0_enter_exec)
                        {
                        /*  initially the server is idle */
                        server_busy = 0;
                        /* get queue module's own object id     */
                        own_id = op_id_self ();

                        /* get assigned value of server  */
                        /* processing rate                                  */
                        op_ima_obj_attr_get (own_id, "service_rate", &service_rate);

                        /* get statistic interrupt interarrival time */
                        op_ima_obj_attr_get (own_id, "delta1", &delta1);
                        op_ima_obj_attr_get (own_id, "delta2", &delta2);
                        op_ima_obj_attr_get (own_id, "delta3", &delta3);
                        op_ima_obj_attr_get (own_id, "delta4", &delta4);

                        /* determine the packet length (in bits)             */
                        pkt_len = 500*8;    /* initially */
                        s = (pkt_len/service_rate);

                        /* Initialize Statistics */
                        pkt_total = 0.0;
                        ipkt_total = 0;
                        pkt_total_all = 0.0;
                        pkt_drop  = 0.0;
                        pkt_drop_all = 0.0;
                        pkt_send  = 0.0;
                        pkt_send_all = 0.0;
                        buffer_overflow_pattern = 0.0;
                        max_th_overflow_pattern = 0.0;
                        last_statistic_sent_time = 0.0;
                        stat_wire_delta = 1000;
                        time_21 = 0.0;
                        rtt_eff = 0.2;
                        rtt_meas = 0.2;
                        N_eff = 100;
                        use_dyn_k = 0;


      /* Obtain the discarding mechanism (Tail Drop, Drop From Front, RED, RWFD) */
      op_ima_obj_attr_get (own_id, "discarding_mechanism", &discarding_mechanism);

      /*****************************************/
      /****** RED parameter Initialization ******/
      /*****************************************/
      op_ima_obj_attr_get (own_id, "min_th", &min_th);
      op_ima_obj_attr_get (own_id, "max_th", &max_th);
      op_ima_obj_attr_get (own_id, "max_p", &max_p);
```

```
op_ima_obj_attr_get (own_id, "min_p", &min_p);
op_ima_obj_attr_get (own_id, "physical_memory_limit", &physical_memory_limit );
op_ima_obj_attr_get (own_id, "filtering_gain", &w_q);
k_equ = max_p/2;
uniform_dist = (op_dist_load ("uniform", 0.0, 1.0));
uniform_int_dist = (op_dist_load ("uniform_int", 1, 2000));
R = op_dist_outcome(uniform_dist); /* sample a random number */
ave_q = 0.0; /* RED average queue size */
q_time = 0.0; /* RED decay time */
count = -1.0;
q_min = min_th + 10;
q_max = max_th -10;

/*************************************************/
/******* RED with Load Adaptive Mechanism  *******/
/*************************************************/
prob_a_filter_gain = 0.00005;
op_ima_obj_attr_get (own_id, "initial_drop_probability", &drop_probability);
op_ima_obj_attr_get (own_id, "loss_rate_technique", &loss_rate_technique);
filtered_prob_a = drop_probability;


/*************************************************/
/**** Dynamic RED parameter initialization ******/
/*************************************************/
op_ima_obj_attr_get (own_id, "alpha", &alpha); /* control gain */
op_ima_obj_attr_get (own_id, "no_drop_threshold", &no_drop_threshold);
op_ima_obj_attr_get (own_id, "dropping_technique", &dropping_technique);
op_ima_obj_attr_get (own_id, "q_target", &q_target);
op_ima_obj_attr_get (own_id, "k_target", &k_target);


filtered_q_size = 0.0;
q_avg = q_target;
dqdt = 0;

/*************************************************/
/*********** Ghost RED Initialization ************/
/*************************************************/

for (ii = 0; ii < 100; ii++)
{
  for (jj=0; jj < 10; jj++)
       {
        rttlist[ii][jj] = 0.0; /* rtt start time  */
         droplist[ii][jj][0] = 0; /* Sequence number of dropped packet */
         droplist[ii][jj][1] = 0;
         droplist[ii][jj][2] = 0; /* Queue size on 3rd packet after drop */
       }
}

/* Schedule first statistic interupt */
op_intrpt_schedule_self( NOW, STAT1 ); /* to monitor performance metrics */
op_intrpt_schedule_self( NOW, STAT2 ); /* to monitor performance metrics */
op_intrpt_schedule_self( NOW + delta3, STAT3 ); /* to update target loss rate */
op_intrpt_schedule_self( NOW + delta4, QTARGET );/*to update target queue size */

use_prob_b = 1;

for (ii = 0; ii < 2000; ii++)
{
      Nlist[ii]=0;
}
```

```
	for (ii = 0; ii < 1000; ii++)
	{
		q_rec[ii]=0;
	}

	for (ii=0; ii < 8; ii++) /* Make sure ii is defined in temporary variables */
	{
		cx[ii] = 0.0; /* These are the arrival counters */
		cy[ii] = 0; /* These are the departure counters */
		hcx[ii] = 0.0; /* This is a memory for the cx counters */
	}
		a[0]=128.0/255; /* These are the weights for averaging the counters */
		a[1]=64.0/255;  /* They must always add to one */
		a[2]=32.0/255;
		a[3]=16.0/255;
		a[4]=8.0/255;
		a[5]=4.0/255;
		a[6]=2.0/255;
		a[7]=1.0/255;

		xy = 1; /* This state variable tracks cx/cy. */

		k_target = 0.02;

		q_avg_old = physical_memory_limit/2.0;

	}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [init enter execs]", state0_enter_exec)

/** state (init) exit executives **/
FSM_STATE_EXIT_FORCED (0, "init", "d_fifo_DATA_a_RWFD21 [init exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [init exit execs]", state0_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [init exit execs]", state0_exit_exec)
/** state (init) transition processing **/
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [init trans conditions]",
state0_trans_conds)
FSM_INIT_COND (ARRIVAL)
FSM_DFLT_COND
FSM_TEST_LOGIC ("init")
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [init trans conditions]",
state0_trans_conds)

FSM_TRANSIT_SWITCH
{
	FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;, "ARRIVAL", "", "init", "arrival")
	FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "", "init", "idle")
}
/*---------------------------------------------------------*/

/** state (arrival) enter executives **/
FSM_STATE_ENTER_FORCED (1, state1_enter_exec, "arrival", "d_fifo_DATA_a_RWFD21 [arrival
enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [arrival enter execs]", state1_enter_exec)
{
	/* acquire the arriving packet                          */
	/* multiple arriving streams are supported. */
	pkptr = op_pk_get (op_intrpt_strm ());

	if (discarding_mechanism == 0) /****** TAIL DROP ********/
```

```
        {
              /* attempt to enqueue the packet at tail       */
              /* of subqueue 0.                                          */

              ++pkt_total;

          if (op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK)
            {
              /* the inserton failed (due to to a     */
              /* full queue) deallocate the packet.   */

              /* To remove phase effects */
              random_index = op_dist_uniform ( (physical_memory_limit) );
              pkptr_tmp = op_subq_pk_remove(0, (int)random_index);
              op_pk_destroy (pkptr_tmp);
              op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL);
              /* set flag indicating insertion fail   */
              /* this flag is used to determine                */
              /* transition out of this state         */
              insert_ok = 0;

              /* count number of packet dropped to measure loss rate, etc. */
              ++pkt_drop;

              /* plot packet loss pattern over time */
              ++buffer_overflow_pattern;
              op_stat_local_write (BUFFER_OVERFLOW_PATTERN, buffer_overflow_pattern);
            }
        else
        {
              /* insertion was successful                    */
              insert_ok = 1;
        }
}

if (discarding_mechanism == 1) /****** DROP FROM FRONT ********/
{
        /* count number of total packets received */
        ++pkt_total;
        if (op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK)
        {
              /* if queue is completely full, remove a packet from head of queue */
              pkptr_tmp = op_subq_pk_remove (0, OPC_QPOS_HEAD);
              op_pk_destroy (pkptr_tmp);
              ++pkt_drop;

              /* queue arriving packet at tail of queue */
              op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL);

              /* insertion was successful                    */
              insert_ok = 1;
        }
        else
        {
              /* insertion was successful                    */
              insert_ok = 1;
        }
}

if (discarding_mechanism == 2) /******** RANDOM EARLY DETECTION (RED) **********/
{

        ++pkt_total;
```

```
        /* a stat for the interarrival time */
        op_pk_nfd_get( pkptr, "source_addr", &source_addr );
        op_pk_nfd_get( pkptr, "subnet_id", &subnet_id );
        op_pk_nfd_get( pkptr, "type", &pkt_type );

        if((subnet_id == 0) && (pkt_type == 1))
        {
               data_get3=data_get3+1;
               op_stat_local_write(DATA_GET3,data_get3);}

               if ((source_addr == 20) && (subnet_id == 3))
               {
                      op_stat_local_write(INT_ARRIVAL_21, op_sim_time() - time_21);
                      time_21=op_sim_time();
               }
               if ( op_subq_stat (0, OPC_QSTAT_PKSIZE) >= physical_memory_limit )
               {
                      /* the inserton failed (due to to a     */
                      /* full queue) deallocate the packet.   */
                      op_pk_destroy (pkptr);
                      /* set flag indicating insertion fail   */
                      /* this flag is used to determine              */
                      /* transition out of this state         */
                      insert_ok = 0;

       /* count number of packet dropped used to measure packet loss rates over time*/

                      ++pkt_drop;
                      /* plot packet loss pattern over time */
                      ++buffer_overflow_pattern;
              op_stat_local_write (BUFFER_OVERFLOW_PATTERN, buffer_overflow_pattern);
              }
              else
              {
                      q_size = op_subq_stat (0, OPC_QSTAT_PKSIZE);

                      /* compute the average queue size */
                      if (q_size != 0.0)
                      ave_q = ( (1-w_q)*ave_q + w_q*q_size);
                      else
                      ave_q = pow( (1-w_q), ( (op_sim_time() - q_time)/s) )*ave_q;

                      /* compute dropping probality and queue or dropmark appropriately */
                      if ( (ave_q >= min_th) && (ave_q < max_th) )
                      {
                             ++count;

                             R = op_dist_outcome (uniform_dist);

                     op_stat_local_write (RED_INTERMEDIATE_PROBABILITY_DROPPING, prob_b);

                             prob_a = prob_b/(1-count*prob_b);

                             /* bound probability */
                             if ( (prob_a > 1.0)  || (prob_a < 0.0) )
                                    prob_a = 1.0;

filtered_prob_a = (1-prob_a_filter_gain)*filtered_prob_a + prob_a_filter_gain*prob_a;

if (filtered_prob_a < 0.0 )
       filtered_prob_a = 0.0;

if (filtered_prob_a > 1.0)
```

```
        filtered_prob_a = 1.0;

if (loss_rate_technique == 1)
{
        if ( (op_sim_time() - last_statistic_sent_time) > stat_wire_delta )
        {
                op_stat_write ( filtered_pd, filtered_prob_a);
                last_statistic_sent_time = op_sim_time();
        }
}
        /* op_stat_local_write (RED_PROBABILITY_DROPPING, filtered_prob_a); */

if ((( R < prob_a ) && (use_prob_b == 0)) || (( R < prob_b ) && (use_prob_b==1)))
{
        /* randomly drop arriving packet */
        op_pk_destroy (pkptr);
        insert_ok = 0;
        count = 0.0;

        ++pkt_drop;
        /* plot early packet loss pattern over time */
        ++early_loss_pattern;
        op_stat_local_write (EARLY_LOSS_PATTERN, early_loss_pattern);
}
else
{
        /* queue arriving packet */
        op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL);
        insert_ok = 1;
}
}
else if ( ave_q >= max_th)
{
        /* drop arrving packet since exceeded maximum threshold */
        op_pk_destroy (pkptr);
        insert_ok = 0;
        count = 0.0;

        ++pkt_drop;

        /* plot packet loss pattern over time */
        ++max_th_overflow_pattern;
        op_stat_local_write (MAX_THRESHOLD_OVERFLOW_PATTERN, max_th_overflow_pattern);
}
else
{
        /* queue arriving packet */
        count = -1.0;
        op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL);
        insert_ok = 1;
}
}

}

if (discarding_mechanism == 3) /********  RWFD **********/
{
        ++pkt_total;
        ++ipkt_total;

        if ( op_subq_stat (0, OPC_QSTAT_PKSIZE) >= physical_memory_limit )
        {
                /* the inserton failed (due to to a      */
```

```
        /* full queue) deallocate the packet.   */
        op_pk_destroy (pkptr);

        /* set flag indicating insertion fail   */
        /* this flag is used to determine              */
        /* transition out of this state         */
        insert_ok = 0;

/* count number of packet dropped used to measure packet loss rates over time*/
        ++pkt_drop;

        /* plot packet loss pattern over time */
        ++buffer_overflow_pattern;
        op_stat_local_write (BUFFER_OVERFLOW_PATTERN, buffer_overflow_pattern);
   }
else
{
        /* read the packet details */
        op_pk_nfd_get( pkptr, "sequence", &seq_no );
        op_pk_nfd_get( pkptr, "source_addr", &source_addr );
        op_pk_nfd_get( pkptr, "subnet_id", &subnet_id );

if (droplist[source_addr][subnet_id][1] > 0.1)
{
droplist[source_addr][subnet_id][1] = droplist[source_addr][subnet_id][1] + 1;

        if (droplist[source_addr][subnet_id][1] == 4)
                {
                        rttlist[source_addr][subnet_id] = op_sim_time();
                        droplist[source_addr][subnet_id][2] = q_size;
                }
}

/* Morris' Estimate for the Number of Active Flows */

h = 100*subnet_id + source_addr;

if (Nlist[h] == 0)
{
        Nlist[h] = 1;
        ++Nlistcnt;
}

nclear = (2000*(op_sim_time()-t_last)/(1));
if (nclear > 0)
{
        t_last = op_sim_time();
        for (ii=0; ii < nclear; ii++)
        {
                rand = (op_dist_outcome (uniform_int_dist)) - 1.0;

                if (Nlist[(int)rand] == 1)
                {
                        Nlist[(int)rand] = 0;
                        Nlistcnt = Nlistcnt - 1;
                }
        }
}
if (Nlistcnt < 990)
        N_efft = 2000*(log(2000)-log(2000-Nlistcnt));
else
        N_efft = 1000;
        if (N_efft > 1000)
```

```
                N_efft = 1000;

        if (use_dyn_k == 1) /* Regular mode, otherwise do fast acquisition */
                N_eff = 0.9999*N_eff + 0.0001*N_efft;
        else
                N_eff = 0.99*N_eff + 0.01*N_efft;

        op_stat_local_write( N_EFFT, N_efft );


    /* End of Morris' Estimate */

    /* Update the RTT estimate on a retransmission, mod 256 for ease of
implementation & safety. */

    if ((seq_no % 256 == droplist[source_addr][subnet_id][0] % 256))
    {
        difft = op_sim_time() - rttlist[source_addr][subnet_id];
        if (difft < 0.95)
        {
            if (use_dyn_k == 1)
                rtt_meas = 0.98*rtt_meas + 0.02*difft;
            else
                rtt_meas = 0.9*rtt_meas + 0.1*difft;

    difft=difft+(q_target-droplist[source_addr][subnet_id][2])*pkt_len/service_rate;

        if (use_dyn_k == 1) /* Regular mode, otherwise do fast acquisition */
        {
            rtt_eff = 0.98*rtt_eff + 0.02*difft;
            q_avg = 0.98*q_avg + 0.02*op_subq_stat (0, OPC_QSTAT_PKSIZE);
        }
        else
        {
            rtt_eff = 0.9*rtt_eff + 0.1*difft;
            q_avg = 0.9*q_avg + 0.1*op_subq_stat (0, OPC_QSTAT_PKSIZE);
        }
            op_stat_local_write( RTT_EFF, rtt_eff );
            op_stat_local_write( RTT_MEAS, rtt_meas);

        }
        else if (difft > 0.95)
        {
            if (seq_no == droplist[source_addr][subnet_id][0] )
            {
                op_stat_local_write( TO_TIME, difft );
            }
        }

        droplist[source_addr][subnet_id][1] = 0;
    }

    /* End of RTT Estimate */

    q_size = op_subq_stat (0, OPC_QSTAT_PKSIZE);

    /* compute the average queue size */


    if (q_size != 0.0)
        ave_q = ( (1-w_q)*ave_q + w_q*q_size);
    else
        ave_q = pow( (1-w_q), ( (op_sim_time() - q_time)/s) )*ave_q;
```

```
                if (ave_q < q_min)
                        q_min = ave_q;
                else if (ave_q > q_max)
                        q_max = ave_q;

                /* compute dropping probality and queue or dropmark appropriately */
                if (ave_q>=0)
                {
                        ++count;

                        R = op_dist_outcome (uniform_dist);


                        if (use_dyn_k ==1)
                        {
                                //use the current rtt estimate to find q(t-r(t))

                                prob_b = k_target/(1+(1-k_target)*dqdt);
                        }
                        else
                                prob_b = k_target/(1+(1-k_target)*dqdt);

                        if (prob_b > 1.0)
                                prob_b = 1.0;
                        else if (prob_b < 0.0)
                                prob_b = 0.0;

                        op_stat_local_write (RED_INTERMEDIATE_PROBABILITY_DROPPING, prob_b);

                        k_avg = 0.99*k_avg + 0.01*prob_b;
                        prob_a = prob_b;

                        //prob_a = prob_b/(1-count*prob_b);

                        /* bound probability */

        /* Addition for the Load Adaptive Mechanism in conjuction with RED */
        /* One way is to filter the prob_a and apply it as input to the load adaptive
mechanism, in order to vary the max_th threshold */
        /* Another way would be to scrap this prob_a and use prob_b as a direct estimate
of the loss rate. One could filter it and use it to vary max_th */
        /* The idea to remove the random number generator could also be explored, as
explained in the DRED section below */

filtered_prob_a = (1-prob_a_filter_gain)*filtered_prob_a + prob_a_filter_gain*prob_a;

if (filtered_prob_a < 0.0 )
        filtered_prob_a = 0.0;
if (filtered_prob_a > 1.0)
        filtered_prob_a = 1.0;

if (loss_rate_technique == 1)
{
        if ( (op_sim_time() - last_statistic_sent_time) > stat_wire_delta )
        {
                op_stat_write ( filtered_pd, filtered_prob_a);
                last_statistic_sent_time = op_sim_time();
        }
}
        /* op_stat_local_write (RED_PROBABILITY_DROPPING, filtered_prob_a); */
```

```
if ((( R < prob_a ) && (use_prob_b == 0) && (ave_q>no_drop_threshold)) || (( R < prob_b
) && (use_prob_b == 1) && (ave_q>no_drop_threshold)))
{
	/* modify drop list and rtt list if an rtt estimate is not currently being
taken*/
	if (droplist[source_addr][subnet_id][1] == 0)
	{
		droplist[source_addr][subnet_id][0] = seq_no;
		droplist[source_addr][subnet_id][1] = 1;
		droplist[source_addr][subnet_id][2] = q_size; /* safety */

	rttlist[source_addr][subnet_id] = op_sim_time(); /* This is a safety reset in
case 3rd packet after is dropped */
	}

	/* randomly drop arriving packet */
	op_pk_destroy (pkptr);
	insert_ok = 0;
	count = 0.0;
	++pkt_drop;

	/* plot early packet loss pattern over time */
	++early_loss_pattern;
	op_stat_local_write (EARLY_LOSS_PATTERN, early_loss_pattern);
}
else
{
	/* queue arriving packet */
	op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL);
	insert_ok = 1;
	for (ii=0; ii < 8; ii++)
	{

		cx[ii]=cx[ii]+1.0; /* increment the arrival counters. */
	}
}
}
rttlist[source_addr][subnet_id] = op_sim_time();
/* write stats regardless of what happened */
if ((source_addr == 20) && (subnet_id == 0))
{
	op_stat_local_write(INT_ARRIVAL_21, op_sim_time() - time_21);
	time_21=op_sim_time();
	op_stat_local_write(RTT_EST_21, rttlist[source_addr][subnet_id]);
}
}
}

}


FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [arrival enter execs]",
state1_enter_exec)

/** state (arrival) exit executives **/
FSM_STATE_EXIT_FORCED (1, "arrival", "d_fifo_DATA_a_RWFD21 [arrival exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [arrival exit execs]", state1_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [arrival exit execs]", state1_exit_exec)

		/** state (arrival) transition processing **/
```

```
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [arrival trans conditions]",
state1_trans_conds)
FSM_INIT_COND (!server_busy && insert_ok)
FSM_DFLT_COND
FSM_TEST_LOGIC ("arrival")
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [arrival trans conditions]",
state1_trans_conds)

FSM_TRANSIT_SWITCH
{
FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;, "!server_busy && insert_ok", "",
"arrival", "svc_start")
FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "", "arrival", "idle")
}
/*---------------------------------------------------------*/



/** state (idle) enter executives **/
FSM_STATE_ENTER_UNFORCED (2, state2_enter_exec, "idle", "d_fifo_DATA_a_RWFD21 [idle
enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [idle enter execs]", state2_enter_exec)
{

}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [idle enter execs]", state2_enter_exec)

/** blocking after enter executives of unforced state. **/
FSM_EXIT (5,d_fifo_DATA_a_RWFD21)

/** state (idle) exit executives **/
FSM_STATE_EXIT_UNFORCED (2, "idle", "d_fifo_DATA_a_RWFD21 [idle exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [idle exit execs]", state2_exit_exec)
{

}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [idle exit execs]", state2_exit_exec)


/** state (idle) transition processing **/
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [idle trans conditions]",
state2_trans_conds)
FSM_INIT_COND (ARRIVAL)
FSM_TEST_COND (SVC_COMPLETION)
FSM_TEST_COND (GET_STAT1)
FSM_TEST_COND (GET_STAT2)
FSM_TEST_COND (END_OF_SIM)
FSM_TEST_COND (GET_STAT3)
FSM_TEST_COND (GET_QTARGET)
FSM_TEST_LOGIC ("idle")
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [idle trans conditions]",
state2_trans_conds)

FSM_TRANSIT_SWITCH
{
      FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;, "ARRIVAL", "", "idle", "arrival")
      FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;, "SVC_COMPLETION", "", "idle",
"svc_compl")
      FSM_CASE_TRANSIT (2, 5, state5_enter_exec, ;, "GET_STAT1", "", "idle",
"get_stat1")
      FSM_CASE_TRANSIT (3, 6, state6_enter_exec, ;, "GET_STAT2", "", "idle",
"get_stat2")
```

```
	FSM_CASE_TRANSIT (4, 7, state7_enter_exec, ;, "END_OF_SIM", "", "idle",
"end_sim")
	FSM_CASE_TRANSIT (5, 8, state8_enter_exec, ;, "GET_STAT3", "", "idle",
"get_stat3")
	FSM_CASE_TRANSIT (6, 9, state9_enter_exec, ;, "GET_QTARGET", "", "idle",
"Qtarget")
}
/*---------------------------------------------------------*/


/** state (svc_start) enter executives **/
FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "svc_start", "d_fifo_DATA_a_RWFD21
[svc_start enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [svc_start enter execs]",
state3_enter_exec)
{
	/* get a handle on packet at head of subqueue 0 */
	/* (this does not remove the packet)                    */
	pkptr = op_subq_pk_access (0, OPC_QPOS_HEAD);

	//op_pk_nfd_get( pkptr, "type", &pkt_type );

	/* determine the packets length (in bits)               */
	pkt_len = op_pk_total_size_get (pkptr);

	/* schedule an interrupt for this process               */
	/* at the time where service ends.                                */
	op_intrpt_schedule_self (op_sim_time () + pk_svc_time, 0);

	/* the server is now busy.                                        */
	server_busy = 1;

	}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [svc_start enter execs]",
state3_enter_exec)

/** state (svc_start) exit executives **/
FSM_STATE_EXIT_FORCED (3, "svc_start", "d_fifo_DATA_a_RWFD21 [svc_start exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [svc_start exit execs]", state3_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [svc_start exit execs]",
state3_exit_exec)


/** state (svc_start) transition processing **/
FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "svc_start", "idle")
/*---------------------------------------------------------*/



/** state (svc_compl) enter executives **/
FSM_STATE_ENTER_FORCED (4, state4_enter_exec, "svc_compl", "d_fifo_DATA_a_RWFD21
[svc_compl enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [svc_compl enter execs]",
state4_enter_exec)
{
	/* extract packet at head of queue; this       */
	/* is the packet just finishing service        */
	pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);
	++pkt_send;
```

```
q_size = op_subq_stat (0, OPC_QSTAT_PKSIZE);
if (q_size == 0.0) /* Collect time at which queue becomes empty for RED */
      {q_time = op_sim_time();}

/* forward the packet on stream 0, causing      */
/* an immediate interrupt at destination.       */
op_pk_send (pkptr, 0);

/* Update the counters and get an estimate for xy */
for (ii=0; ii < 8; ii++)
{
      cy[ii]=cy[ii]+1;
                                                    }
      /* could possibly compress this into a for loop too... */
      if (cy[0]==8192) /* 8192 packets are served in a bit under a second */
      {
            cy[0]=0;
            hcx[0]=cx[0]; /* hold the current sample. */
            cx[0]=0;
      }
      if (cy[1]==4096) /* we can play with these numbers to get good samples */
      {
            cy[1]=0;
            hcx[1]=cx[1]; /* hold the current sample. */
            cx[1]=0;
      }
      if (cy[2]==2048)
      {
            cy[2]=0;
            hcx[2]=cx[2]; /* hold the current sample. */
            cx[2]=0;
      }
      if (cy[3]==1024)
      {
            cy[3]=0;
            hcx[3]=cx[3]; /* hold the current sample. */
            cx[3]=0;
      }
      if (cy[4]==512)
      {
            cy[4]=0;
            hcx[4]=cx[4]; /* hold the current sample. */
            cx[4]=0;
      }
      if (cy[5]==256)
      {
            cy[5]=0;
            hcx[5]=cx[5]; /* hold the current sample. */
            cx[5]=0;
      }
      if (cy[6]==128)
      {
            cy[6]=0;
            hcx[6]=cx[6]; /* hold the current sample. */
            cx[6]=0;
      }
      if (cy[7]==64)
      {
            cy[7]=0;
            hcx[7]=cx[7]; /* hold the current sample. */
            cx[7]=0;
      }
```

```
            /* Finally take an average to get x/y */
            xy =
a[0]*hcx[0]/8192+a[1]*hcx[1]/4096+a[2]*hcx[2]/2048+a[3]*hcx[3]/1024+a[4]*hcx[4]/512+a[5]
*hcx[5]/256+a[6]*hcx[6]/128+a[7]*hcx[7]/64;

            dqdt=xy-1;
            op_stat_local_write (XY, xy); /* Straight sample of xy */


            /* server is idle again.                                    */
            server_busy = 0;

      }

      FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [svc_compl enter execs]",
state4_enter_exec)

      /** state (svc_compl) exit executives **/
      FSM_STATE_EXIT_FORCED (4, "svc_compl", "d_fifo_DATA_a_RWFD21 [svc_compl exit
execs]")
      FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [svc_compl exit execs]",
state4_exit_exec)
      {
      }
      FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [svc_compl exit execs]",
state4_exit_exec)


      /** state (svc_compl) transition processing **/
      FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [svc_compl trans conditions]",
state4_trans_conds)
      FSM_INIT_COND (!QUEUE_EMPTY)
      FSM_DFLT_COND
      FSM_TEST_LOGIC ("svc_compl")
      FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [svc_compl trans conditions]",
state4_trans_conds)

      FSM_TRANSIT_SWITCH
      {
      FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;, "!QUEUE_EMPTY", "", "svc_compl",
"svc_start")
      FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "", "svc_compl", "idle")
      }
      /*---------------------------------------------------------*/



      /** state (get_stat1) enter executives **/
      FSM_STATE_ENTER_FORCED (5, state5_enter_exec, "get_stat1", "d_fifo_DATA_a_RWFD21
[get_stat1 enter execs]")
      FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat1 enter execs]",
state5_enter_exec)
      {
      /* Statistic collection each 100 msec */

op_stat_local_write( STAT1, pkt_send ); /* total number of packets sent by router */
op_stat_local_write( STAT1+1,(pkt_drop));/* total number of packets dropped by router */
op_stat_local_write( STAT1+2, pkt_total ); /* total number of packets entering router */
op_stat_local_write( STAT1+3, (pkt_send*pkt_len/delta1)/service_rate*100 ); /* link
utilization */
      if (pkt_total !=0)
      {
            packet_loss_rate = pkt_drop /pkt_total;
```

```
            op_stat_local_write( STAT1+4, packet_loss_rate);
        }
        /* used to collect scalar statistics at the end of simulation */
        if ( op_sim_time () >= 0.0 )
        {
            pkt_send_all = pkt_send_all + pkt_send;
            pkt_drop_all = pkt_drop_all + pkt_drop;
            pkt_total_all = pkt_total_all + pkt_total;
        }

            pkt_send = 0.0;
            pkt_drop = 0.0;
            pkt_total = 0.0;
            /* schedule self interrupt */
            op_intrpt_schedule_self( NOW + delta1, STAT1 );

}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat1 enter execs]",
state5_enter_exec)

/** state (get_stat1) exit executives **/
FSM_STATE_EXIT_FORCED (5, "get_stat1", "d_fifo_DATA_a_RWFD21 [get_stat1 exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat1 exit execs]", state5_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat1 exit execs]",
state5_exit_exec)


/** state (get_stat1) transition processing **/
FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "get_stat1", "idle")
/*---------------------------------------------------------*/



/** state (get_stat2) enter executives **/
FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "get_stat2", "d_fifo_DATA_a_RWFD21
[get_stat2 enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat2 enter execs]",
state6_enter_exec)
{
/* Statistic collection each 10 msec */

op_stat_local_write( STAT2, (double)ave_q ); /* RED average queue size */
op_stat_local_write( STAT2+1, (double)q_size ); /* instantaneous queue size */
op_stat_local_write (RED_PROBABILITY_DROPPING, prob_a); /* RED drop probability p_a */

op_stat_local_write( N_EFF, N_eff );

/* schedule self interrupt */
op_intrpt_schedule_self( NOW + delta2, STAT2 );

temp=dqdt*service_rate/pkt_len;

op_stat_local_write (DQDT, temp);

q_avg = (1-2.0/101)* q_avg + 2.0/101*op_subq_stat (0, OPC_QSTAT_PKSIZE);
op_stat_local_write(Q_AVG, q_avg);

}
```

```
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat2 enter execs]",
state6_enter_exec)

/** state (get_stat2) exit executives **/
FSM_STATE_EXIT_FORCED (6, "get_stat2", "d_fifo_DATA_a_RWFD21 [get_stat2 exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat2 exit execs]", state6_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat2 exit execs]",
state6_exit_exec)


/** state (get_stat2) transition processing **/
FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "get_stat2", "idle")
/*---------------------------------------------------------*/


/** state (end_sim) enter executives **/
FSM_STATE_ENTER_UNFORCED (7, state7_enter_exec, "end_sim", "d_fifo_DATA_a_RWFD21
[end_sim enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [end_sim enter execs]", state7_enter_exec)
{
	/* output statistics to shell window at the end of simulation */

printf("\n\n ********** Packet Loss Rate and Link Utilization Statistics **********");
printf("\nPacket Loss Rate %f", ((pkt_drop_all/pkt_total_all)*100) );
printf("\nLink Utilization %f", ((pkt_send_all/ ( (service_rate*(op_sim_time()))/pkt_len
)*100) ));

}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [end_sim enter execs]",
state7_enter_exec)

/** blocking after enter executives of unforced state. **/
FSM_EXIT (15,d_fifo_DATA_a_RWFD21)


/** state (end_sim) exit executives **/
FSM_STATE_EXIT_UNFORCED (7, "end_sim", "d_fifo_DATA_a_RWFD21 [end_sim exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [end_sim exit execs]", state7_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [end_sim exit execs]", state7_exit_exec)


/** state (end_sim) transition processing **/
FSM_TRANSIT_MISSING ("end_sim")
/*---------------------------------------------------------*/



/** state (get_stat3) enter executives **/
FSM_STATE_ENTER_FORCED (8, state8_enter_exec, "get_stat3", "d_fifo_DATA_a_RWFD21
[get_stat3 enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat3 enter execs]",
state8_enter_exec)
{
	if (op_sim_time() > 5)
	use_dyn_k = 1; /* switch to dynamic loss function */

	/* Statistic collection each 1 sec */
```

```
        /* schedule self interrupt */
        op_intrpt_schedule_self( NOW + delta3, STAT3 );

        /* Calculate new equilibrium loss rate */

        k_equ = (1.3098*N_eff)/(rtt_eff*service_rate/pkt_len);
        k_equ = k_equ*k_equ;
        op_stat_local_write( K_EQU, k_equ );

        r0 = rtt_eff;
        if (r0 < 0.001)
        r0 = 0.01;
        op_stat_local_write( R0, r0);

        if (q_avg < max_th)
        {
k_target = k_target - 2* k_target * ( max_th - q_avg)/((service_rate/pkt_len)*r0);
              if ( k_target > 0.02)
                    k_target = 0.02;
        }
        else
        {
        k_target = k_target - 2*k_target*(max_th-q_avg)/((service_rate/pkt_len)*r0);
        }
        q_avg_old = q_avg;
        op_stat_local_write (K_EQU, k_target);

}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat3 enter execs]",
state8_enter_exec)

/** state (get_stat3) exit executives **/
FSM_STATE_EXIT_FORCED (8, "get_stat3", "d_fifo_DATA_a_RWFD21 [get_stat3 exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [get_stat3 exit execs]", state8_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [get_stat3 exit execs]",
state8_exit_exec)

/** state (get_stat3) transition processing **/
FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "get_stat3", "idle")
/*---------------------------------------------------------*/


/** state (Qtarget) enter executives **/
FSM_STATE_ENTER_FORCED (9, state9_enter_exec, "Qtarget", "d_fifo_DATA_a_RWFD21 [Qtarget
enter execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [Qtarget enter execs]", state9_enter_exec)
{

/* Statistic collection each 1-10 sec */

/* schedule self interrupt */
op_intrpt_schedule_self( NOW + delta4, QTARGET );

op_stat_local_write ( K_AVG, k_avg );
op_stat_local_write ( Q_MIN, q_min );
op_stat_local_write ( Q_MAX, q_max );

op_stat_local_write( Q_TARGET, q_target );
op_stat_local_write ( DRED_DELTA+5, no_drop_threshold );
```

```
}

FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [Qtarget enter execs]",
state9_enter_exec)

/** state (Qtarget) exit executives **/
FSM_STATE_EXIT_FORCED (9, "Qtarget", "d_fifo_DATA_a_RWFD21 [Qtarget exit execs]")
FSM_PROFILE_SECTION_IN ("d_fifo_DATA_a_RWFD21 [Qtarget exit execs]", state9_exit_exec)
{
}
FSM_PROFILE_SECTION_OUT ("d_fifo_DATA_a_RWFD21 [Qtarget exit execs]", state9_exit_exec)


/** state (Qtarget) transition processing **/
FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "Qtarget", "idle")
/*---------------------------------------------------------*/

}
FSM_EXIT (0,d_fifo_DATA_a_RWFD21)
}
}

#if defined (__cplusplus)
       extern "C" {
#endif
       extern VosT_Fun_Status Vos_Catmem_Register (const char * , int ,
VosT_Void_Null_Proc, VosT_Address *);
       extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
       extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
       }
#endif

Compcode
d_fifo_DATA_a_RWFD21_init (void ** gen_state_pptr)
       {
       int _block_origin = 0;
       static VosT_Address obtype = OPC_NIL;

       FIN (d_fifo_DATA_a_RWFD21_init (gen_state_pptr))

       if (obtype == OPC_NIL)
       {
       /* Initialize memory management */
       if (Vos_Catmem_Register ("proc state vars (d_fifo_DATA_a_RWFD21)",
          sizeof (d_fifo_DATA_a_RWFD21_state), Vos_Vnop, &obtype) == VOSC_FAILURE)
       {
             FRET (OPC_COMPCODE_FAILURE)
       }
       }

       *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
       if (*gen_state_pptr == OPC_NIL)
             {
             FRET (OPC_COMPCODE_FAILURE)
             }
       else
             {
             /* Initialize FSM handling */
             ((d_fifo_DATA_a_RWFD21_state *)(*gen_state_pptr))->current_block = 0;

             FRET (OPC_COMPCODE_SUCCESS)
             }
```

```
        }

void
d_fifo_DATA_a_RWFD21_diag (void)
        {
        /* No Diagnostic Block */
        }

void
d_fifo_DATA_a_RWFD21_terminate (void)
        {
        int _block_origin = __LINE__;

        FIN (d_fifo_DATA_a_RWFD21_terminate (void))

        Vos_Catmem_Dealloc (pr_state_ptr);

        FOUT
        }


/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in d_fifo_DATA_a_RWFD21_svar function. */
#undef server_busy
#undef alpha
#undef drop_probability
#undef no_drop_threshold
#undef threshold
#undef error
#undef service_rate
#undef own_id
#undef pkt_total
#undef pkt_total_all
#undef pkt_drop
#undef pkt_drop_all
#undef pkt_send
#undef pkt_send_all
#undef buffer_overflow_pattern
#undef early_loss_pattern
#undef max_th_overflow_pattern
#undef pkt_len
#undef discarding_mechanism
#undef q_size
#undef ave_q
#undef w_q
#undef q_time
#undef min_th
#undef max_th
#undef max_p
#undef R
#undef prob_b
#undef prob_a
#undef uniform_dist
#undef s
#undef m
#undef count
#undef red_ave_q
#undef packet_loss_rate
#undef physical_memory_limit
#undef delta1
#undef delta2
#undef beta_max_p
```

```
#undef ave_max_p
#undef q_size_last
#undef filtered_drop_probability
#undef drop_probability_filter_gain
#undef filtered_pd
#undef filtered_actual_loss_rate
#undef loss_rate_technique
#undef filtered_packet_loss_rate
#undef actual_loss_rate_filter_gain
#undef filtered_prob_a
#undef prob_a_filter_gain
#undef last_statistic_sent_time
#undef stat_wire_delta
#undef dropping_technique
#undef filtered_q_size
#undef use_prob_b
#undef time_21
#undef ipkt_total
#undef rttlist
#undef rtt_eff
#undef N_eff
#undef k_equ
#undef r0
#undef use_dyn_k
#undef Nlist
#undef Nlistcnt
#undef t_last
#undef uniform_int_dist
#undef droplist
#undef q_target
#undef delta3
#undef delta4
#undef k_avg
#undef q_min
#undef q_max
#undef q_avg
#undef rtt_meas
#undef k_target
#undef q_rec
#undef dqdt
#undef cx
#undef cy
#undef hcx
#undef xy
#undef a
#undef q_avg_old
#undef min_p
#undef tserv_start
#undef serv_cnt
#undef rho
#undef rho_target
#undef zero_buffer
#undef inter_drop_time
#undef last_drop
#undef data_get3
#undef pk_type

void
d_fifo_DATA_a_RWFD21_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	d_fifo_DATA_a_RWFD21_state		*prs_ptr;

	FIN (d_fifo_DATA_a_RWFD21_svar (gen_ptr, var_name, var_p_ptr))
```

```
if (var_name == OPC_NIL)
        {
        *var_p_ptr = (char *)OPC_NIL;
        FOUT
        }
prs_ptr = (d_fifo_DATA_a_RWFD21_state *)gen_ptr;

if (strcmp ("server_busy" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->server_busy);
        FOUT
        }
if (strcmp ("alpha" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->alpha);
        FOUT
        }
if (strcmp ("drop_probability" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->drop_probability);
        FOUT
        }
if (strcmp ("no_drop_threshold" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->no_drop_threshold);
        FOUT
        }
if (strcmp ("threshold" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->threshold);
        FOUT
        }
if (strcmp ("error" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->error);
        FOUT
        }
if (strcmp ("service_rate" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->service_rate);
        FOUT
        }
if (strcmp ("own_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->own_id);
        FOUT
        }
if (strcmp ("pkt_total" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->pkt_total);
        FOUT
        }
if (strcmp ("pkt_total_all" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->pkt_total_all);
        FOUT
        }
if (strcmp ("pkt_drop" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->pkt_drop);
        FOUT
        }
```

```
if (strcmp ("pkt_drop_all" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->pkt_drop_all);
      FOUT
      }
if (strcmp ("pkt_send" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->pkt_send);
      FOUT
      }
if (strcmp ("pkt_send_all" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->pkt_send_all);
      FOUT
      }
if (strcmp ("buffer_overflow_pattern" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->buffer_overflow_pattern);
      FOUT
      }
if (strcmp ("early_loss_pattern" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->early_loss_pattern);
      FOUT
      }
if (strcmp ("max_th_overflow_pattern" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->max_th_overflow_pattern);
      FOUT
      }
if (strcmp ("pkt_len" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->pkt_len);
      FOUT
      }
if (strcmp ("discarding_mechanism" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->discarding_mechanism);
      FOUT
      }
if (strcmp ("q_size" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->q_size);
      FOUT
      }
if (strcmp ("ave_q" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->ave_q);
      FOUT
      }
if (strcmp ("w_q" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->w_q);
      FOUT
      }
if (strcmp ("q_time" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->q_time);
      FOUT
      }
if (strcmp ("min_th" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->min_th);
```

```
        FOUT
        }
if (strcmp ("max_th" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->max_th);
        FOUT
        }
if (strcmp ("max_p" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->max_p);
        FOUT
        }
if (strcmp ("R" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->R);
        FOUT
        }
if (strcmp ("prob_b" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->prob_b);
        FOUT
        }
if (strcmp ("prob_a" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->prob_a);
        FOUT
        }
if (strcmp ("uniform_dist" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->uniform_dist);
        FOUT
        }
if (strcmp ("s" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->s);
        FOUT
        }
if (strcmp ("m" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->m);
        FOUT
        }
if (strcmp ("count" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->count);
        FOUT
        }
if (strcmp ("red_ave_q" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->red_ave_q);
        FOUT
        }
if (strcmp ("packet_loss_rate" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->packet_loss_rate);
        FOUT
        }
if (strcmp ("physical_memory_limit" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->physical_memory_limit);
        FOUT
        }
if (strcmp ("delta1" , var_name) == 0)
```

```
        {
        *var_p_ptr = (char *) (&prs_ptr->delta1);
        FOUT
        }
if (strcmp ("delta2" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->delta2);
        FOUT
        }
if (strcmp ("beta_max_p" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->beta_max_p);
        FOUT
        }
if (strcmp ("ave_max_p" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->ave_max_p);
        FOUT
        }
if (strcmp ("q_size_last" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->q_size_last);
        FOUT
        }
if (strcmp ("filtered_drop_probability" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_drop_probability);
        FOUT
        }
if (strcmp ("drop_probability_filter_gain" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->drop_probability_filter_gain);
        FOUT
        }
if (strcmp ("filtered_pd" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_pd);
        FOUT
        }
if (strcmp ("filtered_actual_loss_rate" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_actual_loss_rate);
        FOUT
        }
if (strcmp ("loss_rate_technique" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->loss_rate_technique);
        FOUT
        }
if (strcmp ("filtered_packet_loss_rate" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_packet_loss_rate);
        FOUT
        }
if (strcmp ("actual_loss_rate_filter_gain" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->actual_loss_rate_filter_gain);
        FOUT
        }
if (strcmp ("filtered_prob_a" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_prob_a);
        FOUT
```

```
        }
if (strcmp ("prob_a_filter_gain" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->prob_a_filter_gain);
        FOUT
        }
if (strcmp ("last_statistic_sent_time" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->last_statistic_sent_time);
        FOUT
        }
if (strcmp ("stat_wire_delta" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->stat_wire_delta);
        FOUT
        }
if (strcmp ("dropping_technique" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->dropping_technique);
        FOUT
        }
if (strcmp ("filtered_q_size" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->filtered_q_size);
        FOUT
        }
if (strcmp ("use_prob_b" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->use_prob_b);
        FOUT
        }
if (strcmp ("time_21" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->time_21);
        FOUT
        }
if (strcmp ("ipkt_total" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->ipkt_total);
        FOUT
        }
if (strcmp ("rttlist" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->rttlist);
        FOUT
        }
if (strcmp ("rtt_eff" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rtt_eff);
        FOUT
        }
if (strcmp ("N_eff" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->N_eff);
        FOUT
        }
if (strcmp ("k_equ" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->k_equ);
        FOUT
        }
if (strcmp ("r0" , var_name) == 0)
        {
```

```
        *var_p_ptr = (char *) (&prs_ptr->r0);
        FOUT
        }
if (strcmp ("use_dyn_k" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->use_dyn_k);
        FOUT
        }
if (strcmp ("Nlist" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->Nlist);
        FOUT
        }
if (strcmp ("Nlistcnt" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->Nlistcnt);
        FOUT
        }
if (strcmp ("t_last" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->t_last);
        FOUT
        }
if (strcmp ("uniform_int_dist" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->uniform_int_dist);
        FOUT
        }
if (strcmp ("droplist" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->droplist);
        FOUT
        }
if (strcmp ("q_target" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->q_target);
        FOUT
        }
if (strcmp ("delta3" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->delta3);
        FOUT
        }
if (strcmp ("delta4" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->delta4);
        FOUT
        }
if (strcmp ("k_avg" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->k_avg);
        FOUT
        }
if (strcmp ("q_min" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->q_min);
        FOUT
        }
if (strcmp ("q_max" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->q_max);
        FOUT
        }
```

```
if (strcmp ("q_avg" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->q_avg);
      FOUT
      }
if (strcmp ("rtt_meas" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->rtt_meas);
      FOUT
      }
if (strcmp ("k_target" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->k_target);
      FOUT
      }
if (strcmp ("q_rec" , var_name) == 0)
      {
      *var_p_ptr = (char *) (prs_ptr->q_rec);
      FOUT
      }
if (strcmp ("dqdt" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->dqdt);
      FOUT
      }
if (strcmp ("cx" , var_name) == 0)
      {
      *var_p_ptr = (char *) (prs_ptr->cx);
      FOUT
      }
if (strcmp ("cy" , var_name) == 0)
      {
      *var_p_ptr = (char *) (prs_ptr->cy);
      FOUT
      }
if (strcmp ("hcx" , var_name) == 0)
      {
      *var_p_ptr = (char *) (prs_ptr->hcx);
      FOUT
      }
if (strcmp ("xy" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->xy);
      FOUT
      }
if (strcmp ("a" , var_name) == 0)
      {
      *var_p_ptr = (char *) (prs_ptr->a);
      FOUT
      }
if (strcmp ("q_avg_old" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->q_avg_old);
      FOUT
      }
if (strcmp ("min_p" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->min_p);
      FOUT
      }
if (strcmp ("tserv_start" , var_name) == 0)
      {
      *var_p_ptr = (char *) (&prs_ptr->tserv_start);
```

```
        FOUT
        }
if (strcmp ("serv_cnt" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->serv_cnt);
        FOUT
        }
if (strcmp ("rho" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rho);
        FOUT
        }
if (strcmp ("rho_target" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rho_target);
        FOUT
        }
if (strcmp ("zero_buffer" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->zero_buffer);
        FOUT
        }
if (strcmp ("inter_drop_time" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->inter_drop_time);
        FOUT
        }
if (strcmp ("last_drop" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->last_drop);
        FOUT
        }
if (strcmp ("data_get3" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->data_get3);
        FOUT
        }
if (strcmp ("pk_type" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->pk_type);
        FOUT
        }
*var_p_ptr = (char *)OPC_NIL;

FOUT
}
```