

# Internet Architectures and Protocols (Netw\_II)

## TP #2: TCP and Buffer Management

E. W. Biersack

EURECOM

### 1 Introduction

This TP will illustrate the behaviour of TCP under some particular circumstances and also the impact of different buffer management policies.

#### 1.1 Definitions

**Definition 1 (Throughput)** We define the throughput  $T_F$  of a TCP flow  $F$  on link  $l$  during a time interval  $t$  as the total number  $B$  of bytes of flow  $F$  which passed through link  $l$  during the interval  $t$  divided by  $t$ :  $T_F = \frac{B}{t}$ .

**Definition 2 (Goodput)** We define the goodput of a flow as the bandwidth delivered to the receiver, excluding duplicate packets. Thus, the goodput of a flow  $F$  during a time interval  $t$  corresponds to the number of bytes of flow  $F$  the TCP stack forwards to the upper layers during the interval  $t$ .

### 2 TP

The version of TCP used is TCP-Tahoe including the following mechanisms: Slow Start (SS), Congestion Avoidance (CA), and Fast Retransmit. The size of the TCP data packets is 1000 bytes and the size of the TCP acknowledgment is 40 bytes. We assume a source (up to TCP) that always has a packet to send. Let  $W_{cwnd}$  be the sliding congestion window and  $W_R$  be the advertised receiver window. We always assume (unless explicitly specified)  $W_R$  larger than the highest  $W_{cwnd}$ . The delay specified in Figures 2, 4 is the One Way Propagation Delay (OWPD).

#### 2.1 Before the TP

##### 2.1.1 Before Starting the TP

- You will be working under Windows.
- Therefore, you first need to map the directory `\\datas\teaching`. For this you click on the “My Computer” icon and then under “tools” you select “map network drive”.
- When you are done, you go into the directory `courses\cours15_Erbi\TP2\Nam_files`.

- In this directory you will find the .nam files and the executable for nam.
- You click on nam-1.0a11a-win32.exe to launch the animator.

**WARNING:**

**Never** copy the files .nam into a local directory since they are very large.

### 2.1.2 How to Use Nam

Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data. Upon startup, nam will read the trace file, create topology, pop up a window, do layout if necessary, then pause at the time of the first packet in the trace file. Through its user interface, nam provides control over many aspects of animation.

nam does animation using the following building blocks: node, link, queue, packet, agent, monitor. They are defined below:

**node** A nodes represents a source/host/router, etc.

**link** Links are created between nodes to form a network topology.

**queue** Queue needs to be constructed in nam between two nodes. Nam queue is associated to a simplex link. In nam, queues are visualized as stacked packets. Packets are stacked along a line.

**packet** Packet is visualized as a block with an arrow. The direction of the arrow shows the flow direction of the packet. Queued packets are shown as little squares. A packet may be dropped from a queue or a link. Dropped packets are shown as rotating squares, and disappear at the end of the screen. Dropped packets are not visible during backward animation.

**agent** Agents are used to separate protocol states from nodes. They are always associated with nodes. An agent has a name, which is a unique identifier of the agent. It is shown as a square with its name inside, and a line link the square to its associated node.

In nam, a topology is specified by alternating node objects with edge objects. But to display the topology in a comprehensible way, a layout mechanism is needed. Currently nam provides two layout methods.

First, user may specify edges' orientations. During layout, nam will honor the given edge orientations. Generally, it will first choose a reference node, then place other nodes using edge orientation and edge length, which is determined by link delay. This works well for small and manually generated topologies.

Second, when we are dealing with randomly generated topologies, be it small or large, we may want to do layout automatically. An automatic graph layout algorithm is adapted and implemented. The basic idea of the algorithm is to model the graph as balls (nodes) connected by springs (edges). Balls will repulse each other, while springs pull them together. This system will (hopefully) converge after some iterations. In practice, after a small number of iterations (tens or hundreds), most graphs will converge to a visually comprehensible structure. There are 3 parameters to tune the automatic layout process:

**Ca** Attractive force constant, which controls springs's force between balls. Default value is 0.15

**Cr** Repulsive force constant, which controls the repulsive force between balls. Default value is 0.15

**Number of iterations** Self explained. Default value is 10.

For small topologies with tens of nodes, using the default parameters (perhaps with 20 to 30 more iterations) will suffice to produce a nice layout.

The top of the nam window is a menu bar. Three pulldown menus are on the left of the menu bar. The File menu, the View menu and the Analysis menu. We do not give details about these menus. Below the menu bar, there is a control bar containing 6 buttons, a label, and a small scrollbar (scale). They can be clicked in any order. We will explain them from left to right.

**Button 1** (◀◀) Rewind. When clicked, animation time will go back at the rate of 25 times the current screen update rate.

**Button 2** (◀) Backward play. When clicked, animation will be played backward in time.

**Button 3** (◻) Stop. When clicked, animation will pause.

**Button 4** (▶) Forward play. When clicked, animation will be played in time ascending order.

**Button 5** (▶▶) Fast Forward. When clicked, animation time will go forward at the rate of 25 times the current screen update rate.

**Button 6 (Chevron logo)** Quit.

**Time label** Show the current animation time (i.e., simulation time as in the trace file).

**Rate slider** Controls the screen update rate (animation granularity). The current rate is displayed in the label above the slider.

**WARNING:** All the rewind actions could produce display errors. Instead of doing rewind re-launch nam.

Below the first control bar, there is Main Display, which contains a tool bar and a main view pane with two panning scroll bars. The tool bar contains two zoom buttons. The button with an up arrow zooms in, the button with a down arrow zooms out. The two scroll bars are used to pan the main animation view. Clicking the left button on any of the objects in the main view pane will pop up a information window at the clicking point. For packet and agent objects, there is a 'monitor' button in the popup window. Clicking that button will bring out the monitor pane (if it is not there), and add a monitor to the object. For link object, there will be a Graph button. Click that button will bring out another popup window, where user can select drawing bandwidth utilization graph or link loss graph of one of the two simplex links of the duplex link clicked on. Currently only packet and agent may have monitor.

A packet monitor shows the size, id, and sent time. When the packet reaches its destination, the monitor will still be there, but saying the packet is invisible. A agent monitor shows the name of the agent, and if there are any variable traces associated with this agent, they will be shown there as well.

Below the Main Display, there is a Time Slider. It looks like a scaled rule, with a tag TIME which can be dragged along the rule. It is used to set the current animation time. As you drag the TIME tag, current animation time will be displayed in the time label in the control bar above. The left edge of the slider represents the earliest event time in the trace file and the right edge represents the last event time. Clicking left button on the rule (not the tag) has the same effect as Rewind or Fast Forward, depending on the clicking position.

The Automatic Layout Pane can be visible or hidden. If visible, it is below the time slider. It has three input boxes and one relay button. The labeled input boxes let user adjust two automatic layout constants, and the number of iterations during next layout. When user press ENTER in any of the input boxes, or click the relay button, that number of iterations will be performed.

The bottom component of the nam window is a Annotation Listbox, where annotations are displayed. An annotation is a (time, string) pair, which describes a event occurring at that time.

### 2.1.3 Screen shot

**Note:** A very useful feature is the visualize over the whole duration of the simulation how much bandwidth of a particular link is used. If it is the the link that has at one end a source, this allows you to visualize the activity of the source. For this you need to left click on the particular link, a pop-up menu comes, there you click on “graph” and then you choose “graph bandwidth” (pay attention to selecting the right direction, i.e the one in which the data packets flow)

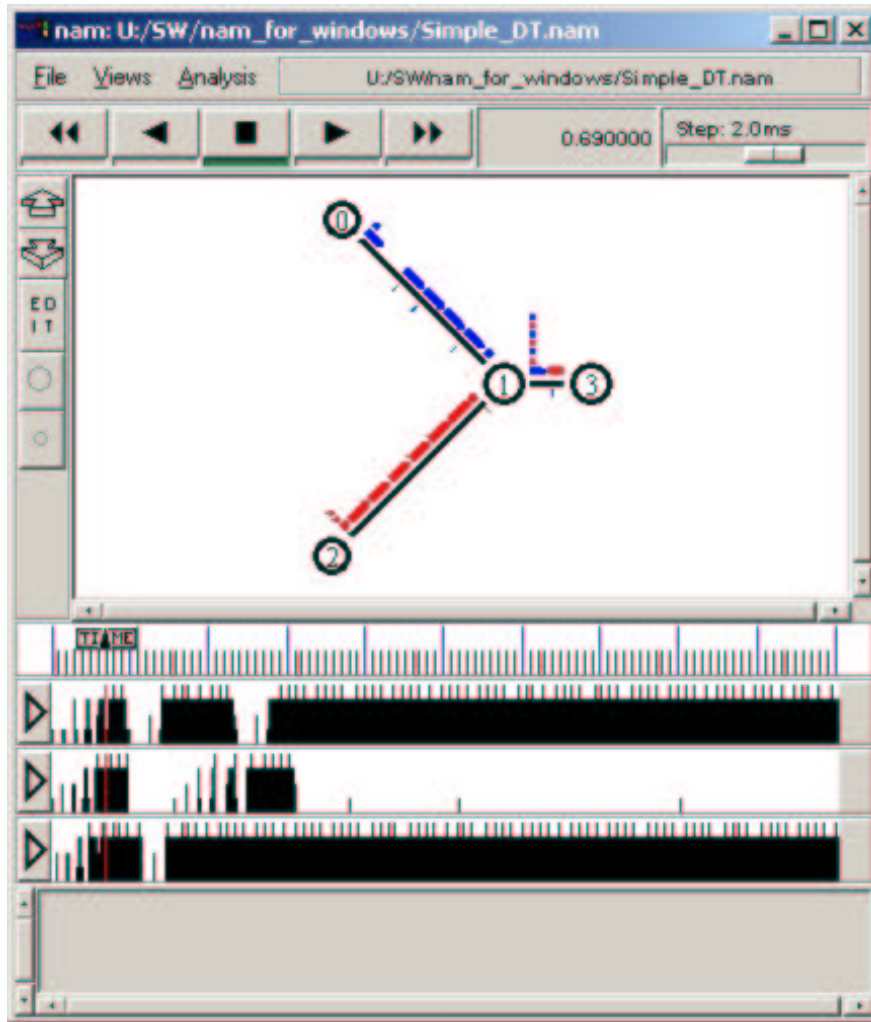


Figure 1: Screen shot of nam.

In figure 1 we see a screen shot of nam. The main window shows the network configuration with 2 sources at nodes 0 and 2 and the destination at node 3. We see the packets moving along the links and the buffer filling up at node 1. Below that window are 4 more windows. The first one shows the time line. The other three show the bandwidth used on the links 2-1, 0-1, and 1-3. You can see that the source at node 2 will be almost completely looked out after some time while source at node 0 keeps sending all the time.

### 3 Two-Way Connections.

#### Scenario

Here we consider **drop-tail buffer management and FIFO scheduling**.

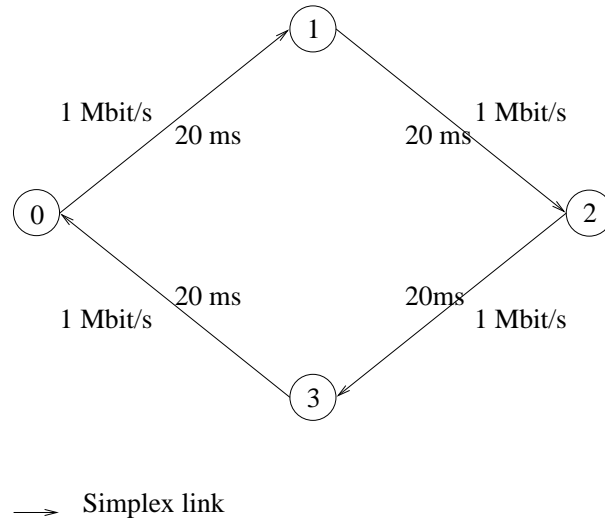


Figure 2: A simple topology with simplex links, symmetric links.

We consider the topology depicted in Figure 2. The buffer size for all the links is 100000 bytes (100 data packets). We put a TCP source (TCP1) at node 0 and the TCP receiver at node 2, we put another TCP source (TCP2) at node 2 and the TCP receiver at node 0. We start TCP1 at  $t = 0$ s and TCP2 at  $t = 5$ s and the simulation last for 50 simulated seconds. For all the experiments in this section we do not want TCP to experience losses as that can introduce noise in our observation. To avoid loss we control the advertised receiver window  $W_R$ . For each simulation we vary  $W_R$  for both TCP flows. The aim of this experiment is to understand how TCP behaves in case of two way connections. The main phenomenon pointed out here is *Ack aggregation*. We say there is Ack aggregation (or compression) when several Acks are bunched together. Ack aggregation appears when several Acks are queued together behind the same data packet.

Figure 3 shows the behavior of a single TCP flow (one way TCP) when we increase the receiver window. The throughput increases with the receiver window  $W_R$  up to the maximum link bandwidth. This one-way TCP is the benchmark for studying the two-way TCP, TCP1 and TCP2. The traces of the simulations with two-way TCP flows are in the files of the format `out_twoway_sym_sameW_<math>W_{TCP1}>_<math>W_{TCP2}>.nam`. The traces of the simulations with one-way TCP flow are in the files of the format `out_twoway_sym_1TCP_<math>W_{TCP1}>.nam`. Visualize some traces for well chosen values of the receiver window (for this scenario choose  $W_{TCP1} = W_{TCP2}$ ).

#### Question 1

1. Identify Ack aggregation.
2. Explain the slight discrepancy between the curves TCP1 and TCP2.
3. Explain why for  $W > 5$  the two-way curves and the one-way curve diverge.

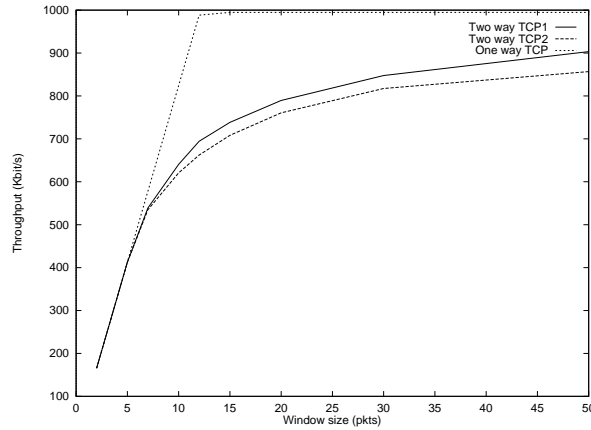


Figure 3: Throughput of two-way TCP when varying the receiver window for both TCP flows.

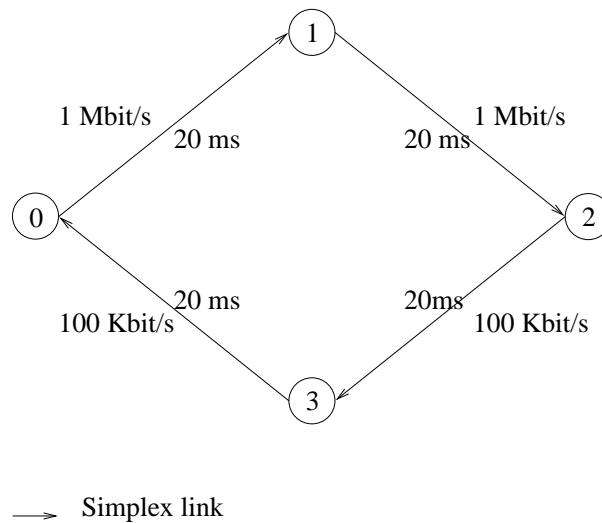


Figure 4: A simple topology with simplex links, asymmetric links.

4. Explain why Ack aggregation adversely impacts the TCP throughput.

Now we study the impact of Ack aggregation with an asymmetric links.

### Scenario

We consider the topology depicted in Figure 4). The buffer size for all the links is 100000 bytes (100 data packets). We put a TCP source (TCP1) at node 0 and the TCP receiver at node 2, we put another TCP source (TCP2) at node 2 and the TCP receiver at node 0. We start TCP1 at  $t = 0$ s and TCP2 at  $t = 5$ s and the simulation last for 50 simulated seconds. For each simulation we vary  $W_R$  for both TCP flows.

We define the efficiency for a TCP flow as the ratio of the mean throughput for this TCP flow and of the bottleneck bandwidth. Figure 5 show the efficiency for both TCP flows. The traces of the simulations are in the files of

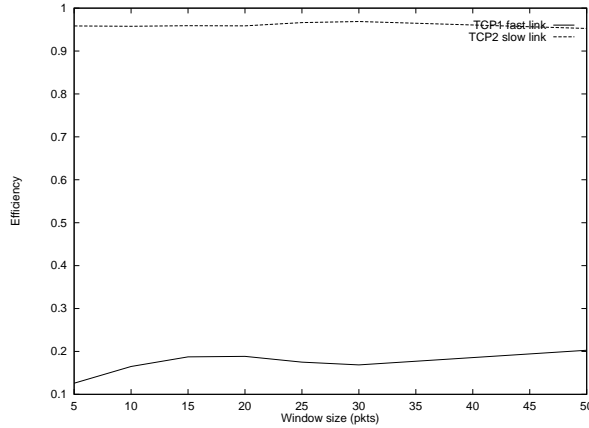


Figure 5: Throughput of two way TCP when varying the receiver window for both TCP flows in an asymmetric environment.

the format `out_twoway_asym_<math>W_{R_{TCP1}}>_<math>W_{R_{TCP2}}>.nam`. Visualize some traces for well chosen values of the receiver window (for this scenario choose  $W_{R_{TCP1}} = W_{R_{TCP2}}$ ).

## Question 2

1. Explain why the TCP on the fast link experiences such a bad efficiency compared to the TCP on the slow link.

## 4 Background: Random Early Detection (RED)

Random Early Detection, or RED, is an active queue management algorithm for routers introduced in [?]. In contrast to traditional queue management algorithms, which drop packets only when the buffer is full, the RED algorithm drops arriving packets probabilistically. The probability of drop increases as the estimated average queue size grows. Note that RED responds to a time-averaged queue length, not an instantaneous one. Thus, if the queue has been mostly empty in the "recent past", RED won't tend to drop packets (unless the queue overflows, of course!). On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped.

The RED algorithm itself consists of two main parts: estimation of the average queue size and the drop function. Consider a router with a buffer size of  $K$  packets. With the RED buffer management scheme, incoming packets are dropped with a probability that is an increasing function  $d$  of the average queue size  $\hat{k}$ . The average queue size is estimated using an exponential weighted moving average:

$$\hat{k} \leftarrow (1 - w)\hat{k} + wk, \quad (1)$$

where  $w$  is a fixed (small) parameter and  $k$  is the instantaneous queue size. A typical drop function  $d$  is defined by three parameters  $min_{th}$ ,  $max_{th}$ , and  $max_p$ , as follows:

$$\begin{aligned} d(\hat{k}) &= 0 & \text{if} & \hat{k} < min_{th}, \\ d(\hat{k}) &= 1 & \text{if} & \hat{k} > max_{th}, \end{aligned}$$

$$d(\hat{k}) = \frac{\hat{k} - \text{min}_{th}}{\text{max}_{th} - \text{min}_{th}} \text{max}_p \quad \text{otherwise.}$$

See Figure 6.

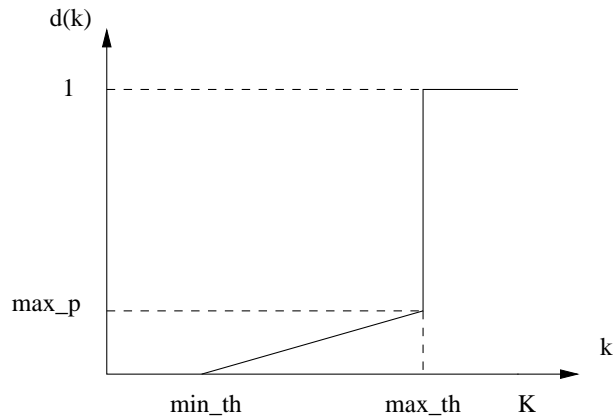


Figure 6: Drop function of RED

In Gentle RED, the packet-dropping probability varies from  $\text{max}_p$  to 1 as the average queue size varies from  $\text{max}_{th}$  to twice  $\text{max}_{th}$ .

## 4.1 Drop Tail

The traditional technique for managing router queue lengths is to set a maximum length (in terms of packets) for each queue, accept packets for the queue until the maximum length is reached, then reject (drop) subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted. This technique is known as "Drop Tail", since the packet that arrived most recently (i.e., the one on the tail of the queue) is dropped when the queue is full. This method has served the Internet well for years, but it has some important drawbacks.

### 4.1.1 Scenario

We consider the simple topology depicted in Figure 7. The queue size at node 1 is 14 packets. We put a TCP source at node 0 and another one at node 2, the TCP receivers are at node 3. We start the TCP sources at  $t = 0$ s and the simulation lasts for 10 simulated seconds. The buffer Management scheme used is Drop Tail. The trace of the simulation is in the file `Simple_DT.nam`. Visualize the trace with `nam`. The queue size at node 1 in function of time is shown in the following figure 8.

- Note 1: throughout the whole TP, TCP sources always have data to send.
- Note 2: In this simulation the TCP window size is limited to 32 in order to single out the desired phenomenon.

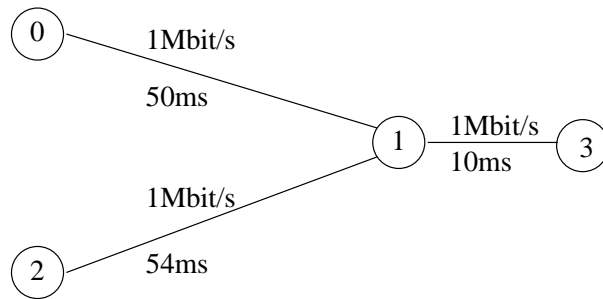


Figure 7: A simple topology for studying Drop Tail.

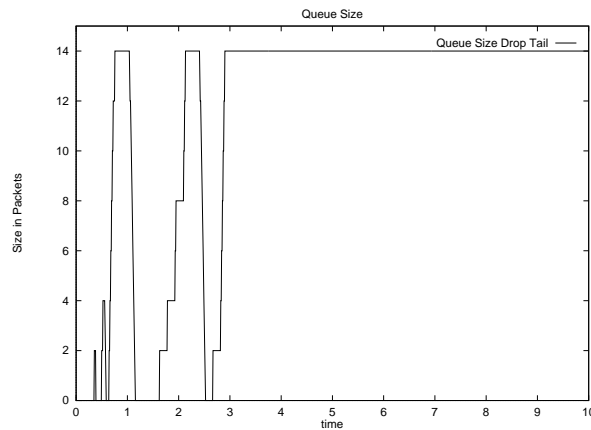


Figure 8: Drop Tail: Queue Size in function of time. Delay of 54 ms

## Questions

1. What is the effect of the Drop Tail discipline on the end-to-end delay, queue size, and the ability to absorb bursts? Do you think Drop Tail can cause multiple consecutive packet drops?
2. The Global Synchronization phenomenon is observed when multiple TCP hosts reduce their transmission rates in response to packet drops causing a sustained period of lowered link utilization, reducing overall throughput. Do you think Drop Tail can cause this phenomenon?
3. Observe what happens starting 2.5s. Which connection gets more bandwidth, Why?
4. From the above, identify two undesirable behaviors of Drop Tail.

## 4.2 RED

In the current Internet, dropped packets serve as a critical mechanism of congestion notification to end nodes. The solution to the full-queues problem is for routers to drop packets before a queue becomes full, so that end nodes can respond to congestion before buffers overflow. Such a pro-active approach is called "active queue management". By dropping packets before buffers overflow, active queue management allows routers to control when and how many packets to drop. Random Early Detection (RED), see section 4, is an active queue management scheme.

### 4.2.1 Scenario

We consider the same topology as in the previous section, see Figure 9. This time the queue management scheme used at node 1 is RED with the following parameters:

Queue Size  $K = 15$ ,  $min_{th} = 3$ ,  $max_{th} = 7$ ,  $w = 0.008$ , and  $max_p = 0.1$ .

The trace of the simulation is in the file `Simple_RED.nam`. Visualize the trace with `nam`.

A plot of the actual and time-averaged queue length (which is used by RED to determine the drop probability) at node 1 as function of time is in figure 10.

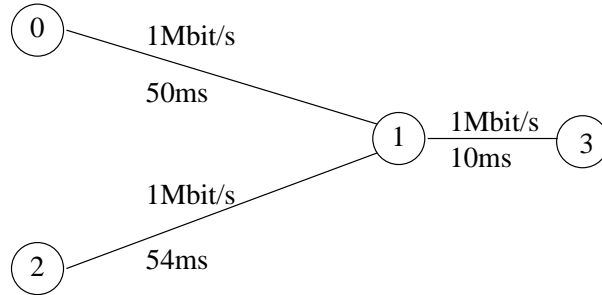


Figure 9: A simple topology for studying RED.

### Questions

1. Notice how some packets are dropped even when the queue is not full. What does it signal to the source?
2. Around  $t = 3s$  RED drops a burst of several consecutive packets (which imply that RED can suffer from TCP global synchronization!). Explain why. Which other variation of RED removes this phenomenon? How does it work?
3. Observe the evolution of the instantaneous and average queue length. What would happen if RED was based on the instantaneous length of the input buffer rather than the average length?

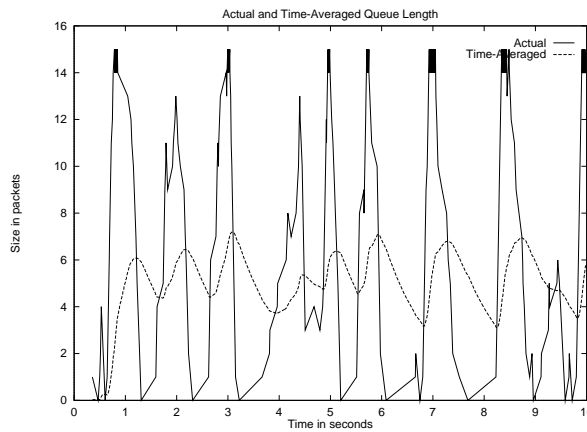


Figure 10: RED: Actual and Time-averaged Queue Size in function of time.

### 4.3 Isolation from high rate unresponsive flows

We consider the topology depicted in Figure 11 with fifteen TCP sources and one UDP/CBR source of 10Mb/s (packet size 1000 bytes). The delay of the links (0,16) ... (15,16) vary from 20ms for link (0,16) to 1s for link (15,16). We use alternatively the following combination of buffer management schemes and scheduling algorithms for the bottleneck link (16,17):

1. FIFO scheduling with a buffer of 124 packets and Drop Tail buffer management.
2. FQ scheduling with 124 packet distinct buffers for each flow (each flow has its own buffer of 124 packet with Drop Tail).
3. FQ with a shared buffer of 124 packets and Longest Queue Drop (LQD) for buffer management. under LQD, when the buffer is full, the tail packet of the flow with the biggest number of packets in the queue is dropped. In case there are several flows with the same biggest number of packets in the queue, one of them is chosen randomly.
4. FIFO with 124 shared buffer and RED for the buffer management: Queue Size  $K = 124$ ,  $min_{th} = 24$ ,  $max_{th} = 100$ ,  $w = 0.002$ , and  $max_p = 0.1$ .

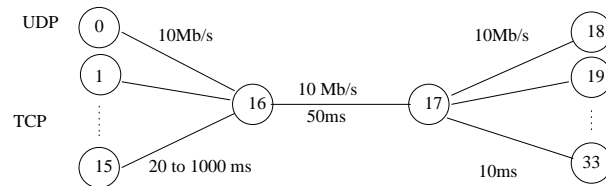


Figure 11: A topology for studying flow isolation.

### Questions

The table 1 resumes the results of the simulations.

1. Comment on scheme 1.
2. FQ with distinct buffers (scheme 2) performs the best. However FQ with LQD (scheme 3) gives a good flow isolation against unresponsive flow (i.e. CBR). How do you explain this?
3. CBR gets more bandwidth with RED (scheme 4) than Drop Tail (scheme 1). Why?

<b>Parameter</b>	<b>FIFO</b>	<b>FQ</b>	<b>FQ+LQD</b>	<b>RED</b>
tcp1 goodput (Kb/s)	71.749	643.5	896.44	101.66
tcp2 goodput (Kb/s)	431.733	636.4	821.28	85.53
tcp3 goodput (Kb/s)	51.626	633.3	702.31	65.36
tcp4 goodput (Kb/s)	81.450	629.8	697.23	60.79
tcp5 goodput (Kb/s)	51.429	628	605.66	45.91
tcp6 goodput (Kb/s)	38.4	625.4	524.8	36.88
tcp7 goodput (Kb/s)	36.634	620.6	466.87	25.4
tcp8 goodput (Kb/s)	51.493	618.5	513.79	23.63
tcp9 goodput (Kb/s)	30.416	617	516.6	23.49
tcp10 goodput (Kb/s)	35.69	612.5	444.37	19.14
tcp11 goodput (Kb/s)	38.837	610.7	490.16	17.41
tcp12 goodput (Kb/s)	47.621	608.4	422.41	16.43
tcp13 goodput (Kb/s)	37.546	602.7	400.27	15.41
tcp14 goodput (Kb/s)	7.008	596.7	435.38	13.21
tcp15 goodput (Kb/s)	284.896	586.5	414.97	12.17
udp goodput (Mb/s)	8.6	0.681	1.6	9.4

Table 1: Results for studying flow isolation.