Computer Networks & Network Simulation

Prof: Danny Hughes

Tom Stappaerts - Bachelor Informatica
Jeroen Sanders - Bachelor Informatica

April 23, 2014

1 Exercise 1

1.1 Question 1

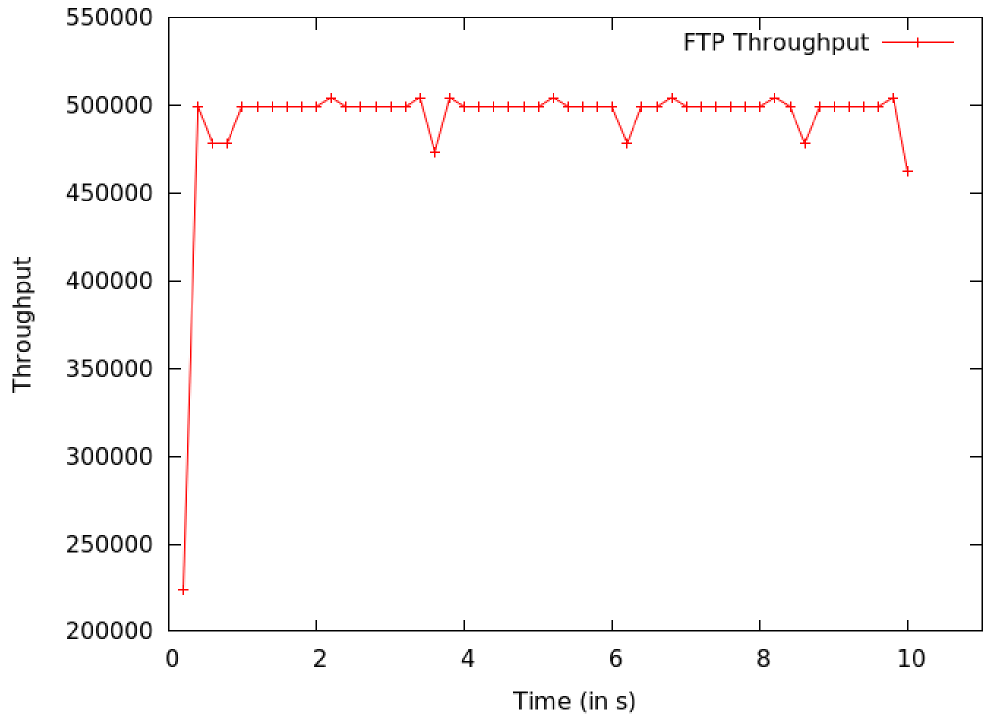

Figure 1: Throughput: bandwidth limitation and download connection

In figure 1 we plotted the throughput of the FTP application. We see that the connection stabilises on 4 Mbps. The download is limited by the strained connection between node 3 and 4.

1.2 Question 2

The FTP application uses all the available bandwidth. As soon as the CBR pplication starts it clogs up the network. It's packages are bigger than the tcp acknowledgement packages and it takes quite a while to upload them through the constrained upload link. Therefore we see a big drop in the graph, because the delivery of the FTP ACK-packets is delayed. In figure 1 we can see a sawtooth behaviour for the FTP throughput in the drop. Between the CBR packets, some ACK's of the FTP appliction get through the strained upload connection.

1.3 Question 3

A reserved amount of upload bandwidth for each application would prevent cannibalisation of the connection. A certain throughput would be guaranteed for both applications. Initially the FTP application would behave just like in question 1. When the CBR application starts, it's bandwidth will decrease because the CBR application has it's own guaranteed bandwidth.

1.4 Question 4

If the bandwidth is even more constrained, the saturation of the network is even higher as you can see in figure 3. It's worse because it takes longer for the ACK's to arrive. The throughput of the FTP application crashes to zero.

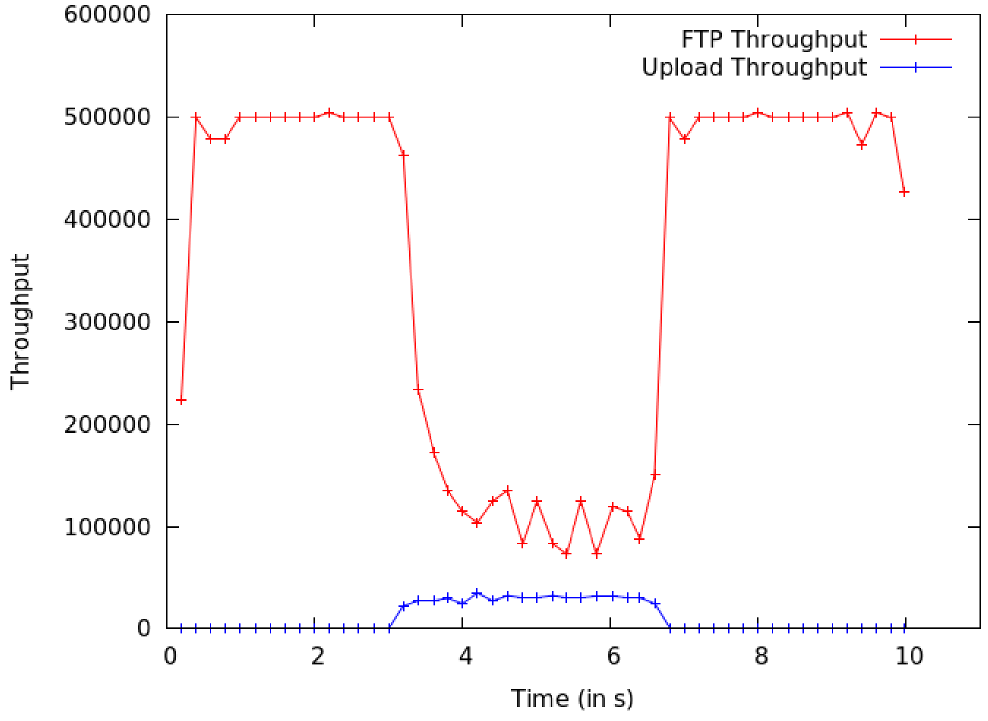

Figure 2: Throughput: bandwidth limitation, download and upload connection

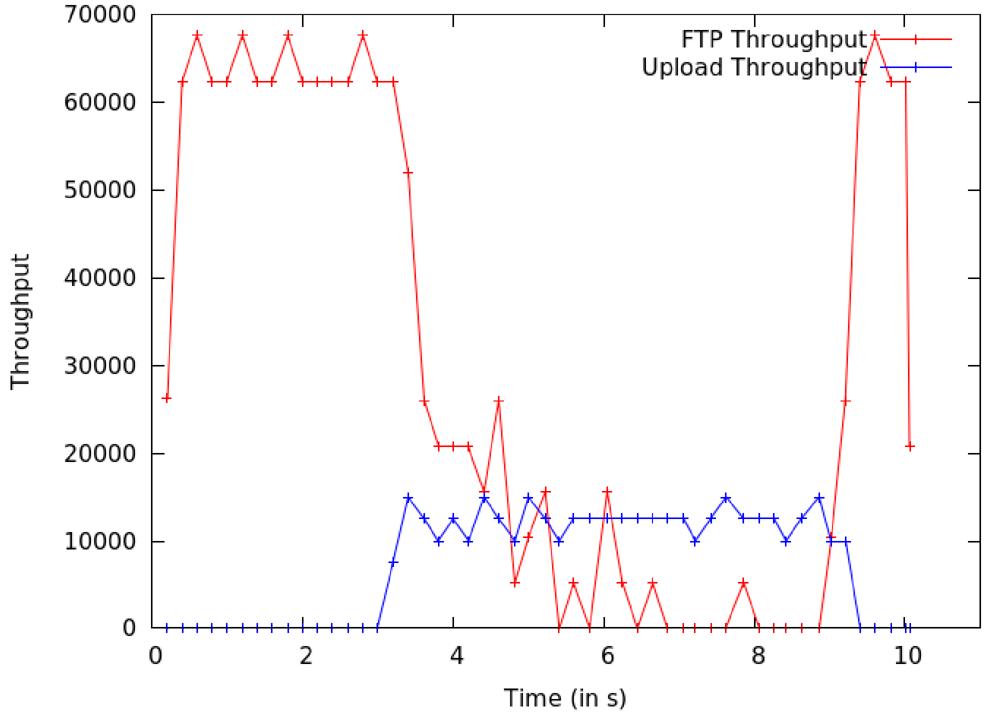

Figure 3: Throughput: bandwidth limitation, download and upload: even more restricted

1.5 Question 5

When the CBR upload connection is fixed to a rate of 30 kbps the FTP connection is not hindered by it as you can see in figure 4. To prevent hosts from interfering with each other, you could implement some sort of load balancing between hosts. In practice this would result in a (weighted) round-robin queue instead of a FIFO queue.

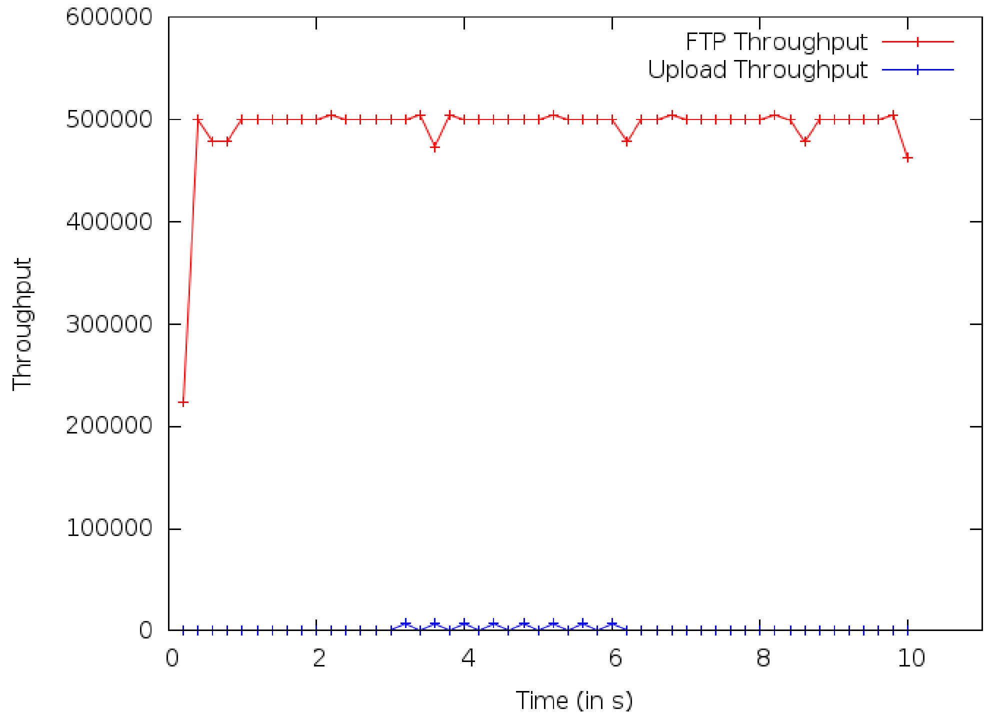

Figure 4: Throughput: upload rate at 30 000

1.6 Question 6

1.6.1 Question 6.1

Since the capacity has a linear increase relative to the load, we see a result similar to question 1 of this exercise. TCP will be responsible for equally distributing the bandwidth amongst the clients. Since big downloaders put more data into the network, they'll have more packets dropped. These big downloaders will then restrict their dataflow.
In the case where only 5 clients are connected with the modem we see a doubling in bandwidth used by each client. There are fewer connections competing for the same bandwidth.

1.6.2 Question 6.2

When the users perform their activities at random times, we expect varying performance. This is because not all the nodes will be active at the same time. In the worst case we get the performance like in the non-random case with 10 users.
If some users always send data when the DropTail queue is full, their packets will always be dropped. A different queue design (like round-robin) could aleviate this situation.

2 Exercise 2

2.1 Question 1

At 5s the throughput of the main FTP application drops, because of the burst connections to the webserver. In figure 5 and 6 we see that the throughput does not drop immediately at 5s. This is because the different connections to the webserver do not start immediately at 5s but at randomly from 5s to 7s.

2.2 Question 2

The different phases of the TCP algorithm are annotated on the graph in figure 7. At first we see the slow start. After we reach the slow start threshold we reach the additive increase phase. When the

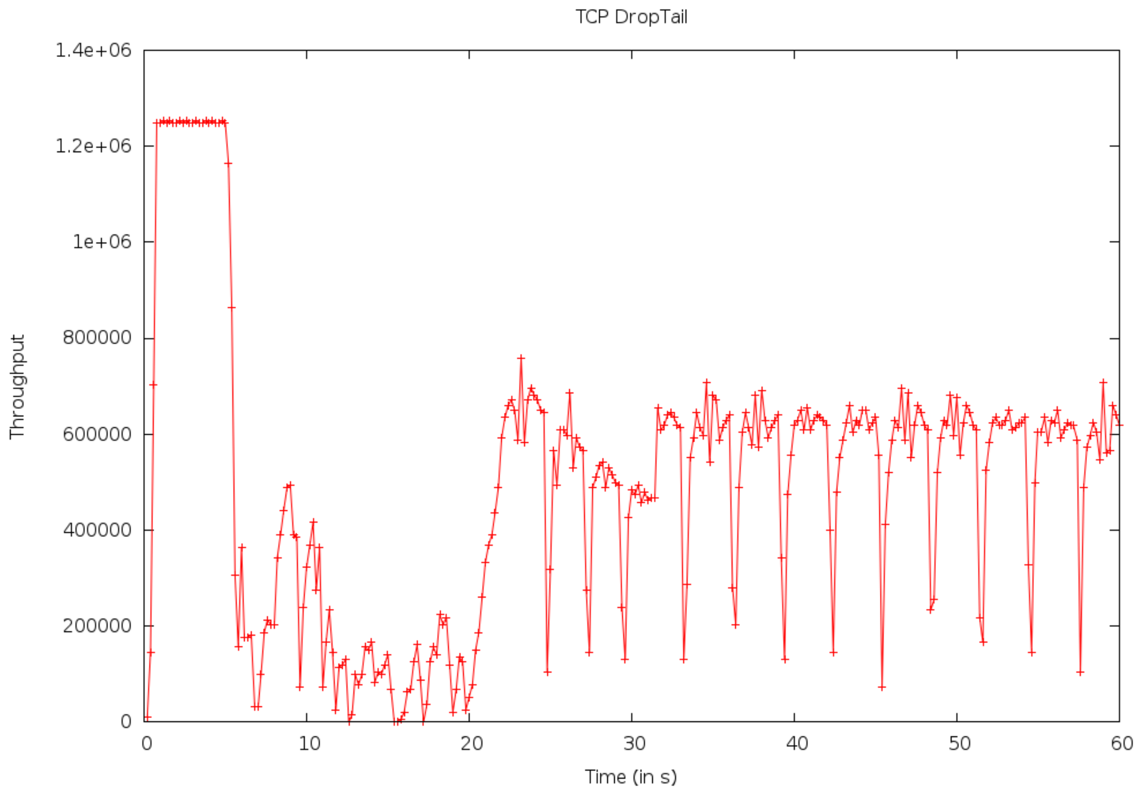

Figure 5: Throughput main FTP connection

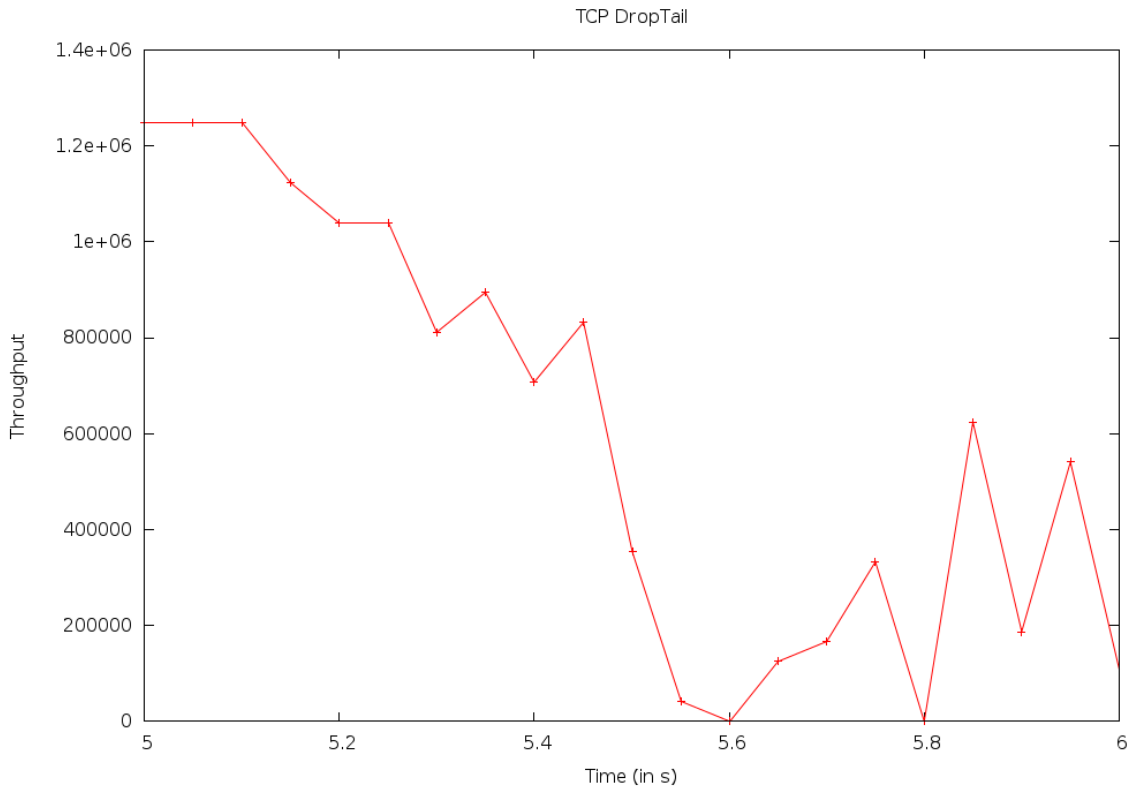

Figure 6: Throughput main FTP connection (zoomed in)

connection times out/ to many packets are lost, the congestion window is set to 1 and the slow start threshold is set at half of the reached window size.

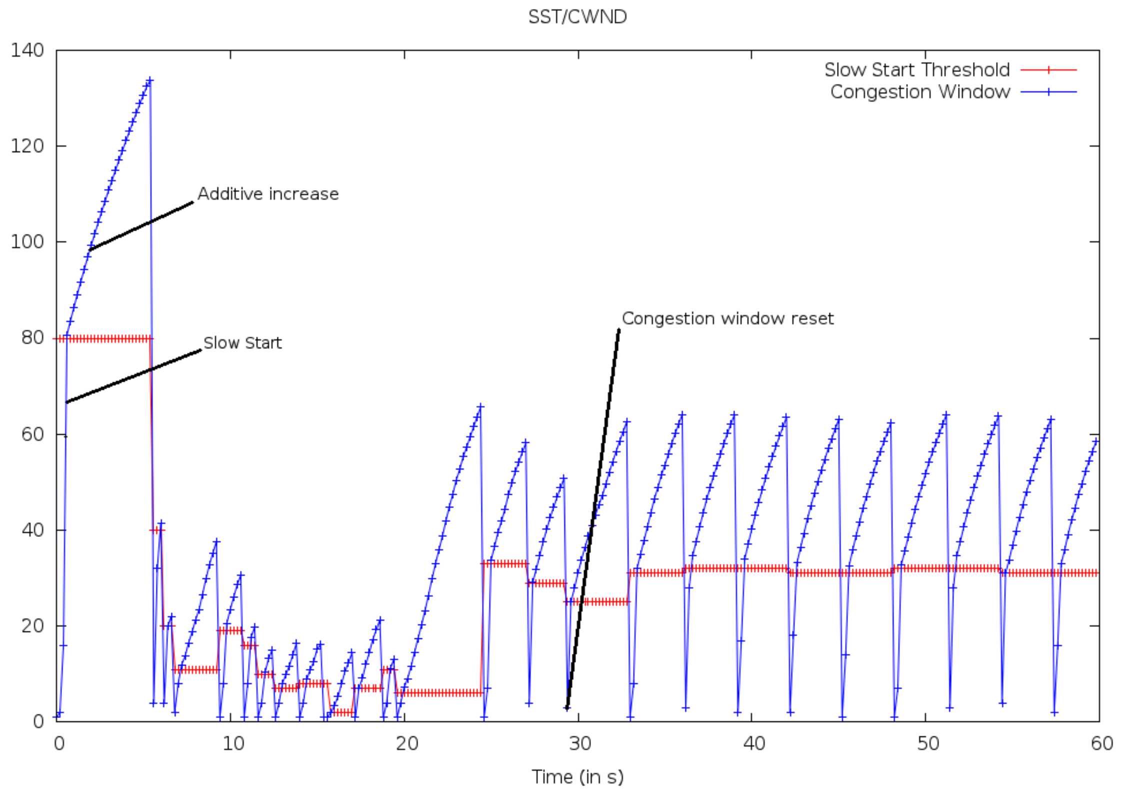

Figure 7: Congestion Window and Slow Start Threshold.

2.3 Question 3

In the TCP-implementation congestion avoidance is always active. In the first “sawtooth” we first see slow start. After the congestion window has been reached addiditve increase has been started. This is all part of the congestion avoidance. After three duplicate ACKS have been received or a timeout has occured, TCP-Tahoe sets the slow start threshold to half the current congestion window, resets the congestion window to 1 and restarts with slow start.

2.4 Question 4

The TCP-Reno implementation uses fast-recovery. This means that instead of resetting the congestion window to one, the congestion window is halved and the slow start threshold is set to the same value. This only happens when there are only three duplicate ACK‘s received. If a timeout occurs, as we can see in figure 8 after 5 seconds, the congestion window is reset to 1 and slow start is re-initiated.

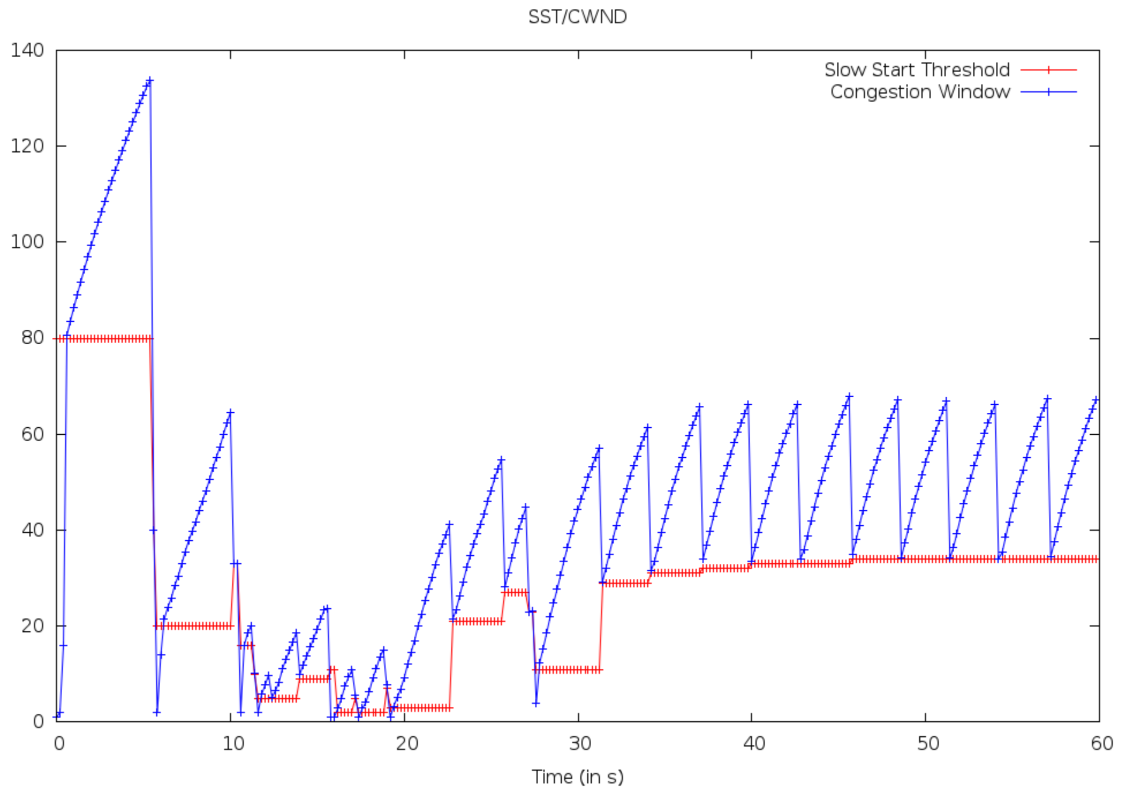

Figure 8: Congestion Window and Slow Start Threshold TCP-Reno

3 Code: Excercise 1

```
#Create simulator
set ns [new Simulator]
#Trace files: nam and tr
set tf [open /tmp/example.out.tr w]
$ns trace-all $tf
set nf [open /tmp/example.out.nam w]
$ns namtrace-all $nf
#Finish procedure: flushes all simulator data to file
proc finish {} {
    #finalise trace files
    global ns nf tf
    $ns flush-trace
    close $tf
    close $nf

    #call nam visualiser
    exec nam /tmp/example.out.nam &
    exit 0
}

#Define Colors
$ns color 1 Blue
$ns color 2 Red

#Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

#Create links between nodes
```

```
$ns duplex-link $n1 $n2 10Mb 0.2ms DropTail
$ns duplex-link $n0 $n2 10Mb 0.2ms DropTail
$ns duplex-link $n2 $n3 10Mb 0.2ms DropTail
$ns simplex-link $n3 $n4 265Kb 0.2ms DropTail
$ns simplex-link $n4 $n3 4Mb 0.2ms DropTail
$ns duplex-link $n4 $n5 100Mb 0.3ms DropTail
$ns duplex-link $n5 $n6 100Mb 0.3ms DropTail
$ns duplex-link $n5 $n7 100Mb 0.3ms DropTail

#Node orientation
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link-op $n5 $n6 orient right-up
$ns duplex-link-op $n5 $n7 orient right-down
$ns duplex-link-op $n4 $n5 orient right

############
#Downloader#
############
# TCP Connection
set tcp6 [new Agent/TCP]
$ns attach-agent $n6 $tcp6
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp6 $sink1
$tcp6 set fid_ 1
$tcp6 set window_ 80
# Ftp Application
set ftp61 [new Application/FTP]
$ftp61 attach-agent $tcp6

##########
#Uploader#
##########
# UDP Connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set null7 [new Agent/Null]
$ns attach-agent $n7 $null7
$ns connect $udp0 $null7
$udp0 set fid_ 2
# CBR Application
set cbr07 [new Application/Traffic/CBR]
$cbr07 attach-agent $udp0
$cbr07 set packetSize_ 1500
$cbr07 set random_ false

#schedule events here (example)
$ns at 0.1 "$ftp61 start"
$ns at 3.0 "$cbr07 start"
$ns at 6.0 "$cbr07 stop"
$ns at 9.9 "$ftp61 stop"
$ns at 19.71 "finish"
#finally execute the simulator
$ns run
```

4 Code: Excercise 2

```
#Create simulator
set ns [new Simulator]
#Trace files + log files
set tf [open /tmp/example.out.tr w]
$ns trace-all $tf
set nf [open /tmp/example.out.nam w]
$ns namtrace-all $nf
set lf [open /tmp/example.out.log w]
set sf [open /tmp/example.out.fs w]

#Finish procedure: flushes all simulator data to file
proc finish {} {
    #finalise trace files
    global ns nf tf lf sf
    $ns flush-trace
    close $tf
    close $nf
    close $lf
    close $sf

    #call nam visualiser
    exec nam /tmp/example.out.nam &
    exit 0
}

#Define Colors
$ns color 1 Blue
$ns color 7 Red

#Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]


#Create links between nodes
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
$ns duplex-link $n4 $n0 10Mb 10ms DropTail
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n1 $n5 10Mb 10ms DropTail

$ns queue-limit $n0 $n1 20

#Node orientation
$ns duplex-link-op $n2 $n0 orient right-down
$ns duplex-link-op $n4 $n0 orient right-up

$ns duplex-link-op $n0 $n1 orient right

$ns duplex-link-op $n1 $n3 orient right-up
$ns duplex-link-op $n1 $n5 orient right-down

######Long lasting ftp connection:#######
# TCP Connection
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $n3 $tcp3
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp3 $sink2
$tcp3 set fid_ 1
$tcp3 set window_ 80
# Ftp Application
set ftp32 [new Application/FTP]
$ftp32 attach-agent $tcp3

#Array with the times of departure
set time(0) 5
set time(1) 10
set time(2) 15
#Number of requests per burst
set nrRequests 40

#TCP Sources, destinations, connections
foreach {key value} [array get time] {
    for {set i 1} {$i <= $nrRequests} {incr i} {
        set tcpsrc($key,$i) [new Agent/TCP]
        set tcp_sink($key,$i) [new Agent/TCPSink]
        $tcpsrc($key,$i) set window_ 80
        $ns attach-agent $n5 $tcpsrc($key,$i)
        $ns attach-agent $n4 $tcp_sink($key,$i)
        $ns connect $tcpsrc($key,$i) $tcp_sink($key,$i)
        set ftp($key,$i) [$tcpsrc($key,$i) attach-source FTP]
    }
}

#Create  random generators:
```

```
set rng1 [new RNG]
$rng1 next-substream
$rng1 next-substream
set rng2 [new RNG]
$rng2 next-substream
#Random inter-send times of tcp transfer
set RV [new RandomVariable/Exponential]
$RV set avg_ 0.05
$RV use-rng $rng1
#Random size of files to transmit
set RVsize [new RandomVariable/Pareto]
$RVsize set avg_ 150000
$RVsize set shape_ 1.5
$RVsize use-rng $rng2

#schedule events here (example)
#setting up number of request
foreach {key value} [array get time] {
    set t $value
    for {set i 1} {$i<= $nrRequests} {incr i} {
        #Set the beginning time of next transfer.
        set t [expr $t + [$RV value]]
        set size [expr [$RVsize value]]
        $ns at $t "$ftp($key,$i) send $size"
        puts $sf "$t $size"
        }
}

proc writelog {} {
   global ns lf tcp3
   set rate 0.2
   set now [$ns now]
   set cwnd [$tcp3 set cwnd_]
   set sst [$tcp3 set ssthresh_]
   puts $lf "$now $cwnd $sst"
   $ns at [expr $now+$rate]  "writelog"
}

$ns at 0.0 "writelog"
$ns at 0.1 "$ftp32 start"
$ns at 60 "finish"

#finally execute the simulator
$ns run
```