# Computer Networks: Network simulation

Jona Beysens (r0296825) & Arnout Devos (r0305883)
Teacher: Danny Hughes

May 1, 2014

# Exercise 1

## Question 1

Figure 1 shows the throughput at node 1 of the KotNet topology. In the beginning of the TCP transfer the slow start is observed. The rate at which the packets are received at node 1 converges to a constant value of around 500kbytes/s which equals the maximum download speed of 4Mbit/s. The little hills on top of the download rate can be explained by the inaccuracy of the perl visualization script.

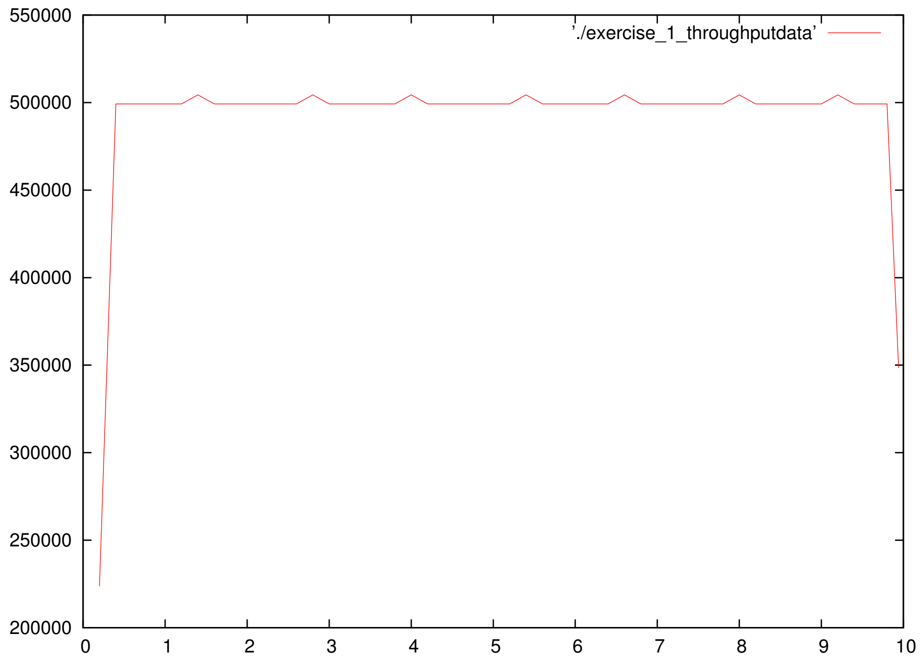

Figure 1: Throughput of FTP connection at node 1 of KotNet topology

## Question 2

In the beginning of the TCP transfer the slow start occurs. When the UDP transmission kicks in, a significant drop in TCP throughput is observed. This is caused by the upload bandwidth limitation of the cable modem: TCP sends an accumulated acknowledgements for received packets but the upload line is congested with UDP traffic as can be seen on the figure 2. Therefor, TCP throughput drops. Acknowledgements are lost due to the overloaded buffer. When the UDP traffic stops, the upload line isn't congested anymore and the slow start mechanism of TCP kicks in again. The throughput converges to the original value of 500kbytes/s.

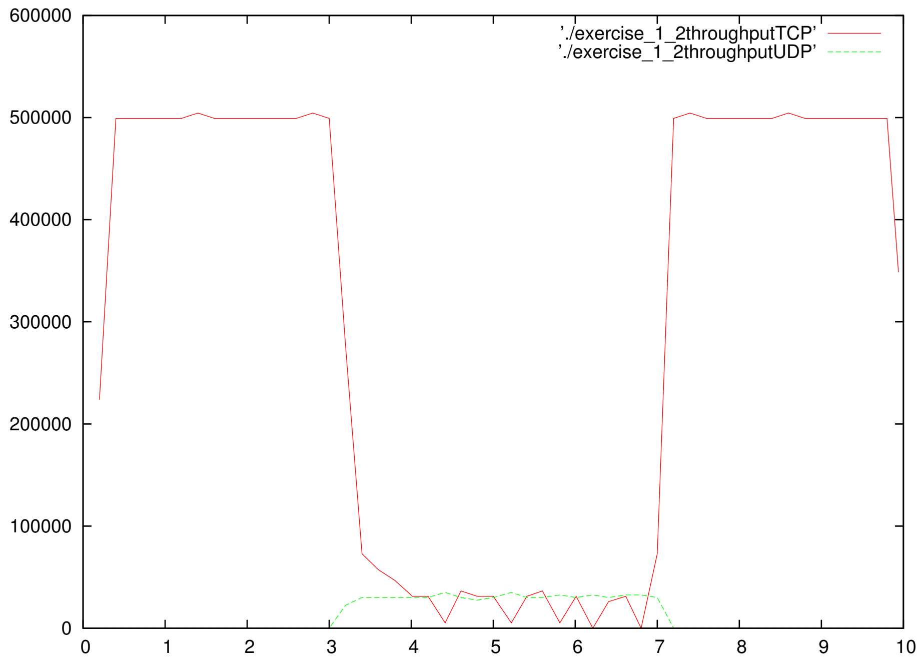

Figure 2: Throughput of FTP and CBR connection at node 1 of KotNet topology

## Question 3

The CBR connection won't take the needed upload bandwidth for the acknowledgements of the TCP connection. If the assigned bandwidth for TCP is large enough to transfer the acknowledgements without congesting the buffer, the TCP throughput will sustain its maximum value. The UDP throughput will adapt to the assigned bandwidth.

## Question 4

If we take the case where the 100 Mbit uplinks and downlinks are preserved, the traffic will be even more distorted. Because a bigger packet loss rate occurs at the buffers, even more TCP acknowledgements will be discarded which results in a lower TCP throughput. UDP throughput also drops to approximately 100kbits/s as the upload line is congested with UDP traffic.

## Question 5

The TCP throughput doesn't drop that much as the situation where the rate of sending UDP packets isn't limited to 30kbytes/s (see figure 3). Although TCP experiences some jitter as it needs a higher bandwidth to transmit all its acknowledgements immediately. A possible solution to this jitter would be to increase TCP packet size and by this reduce the acknowledgement rate which leads to a maximum use of the 4 Mbit bandwidth.

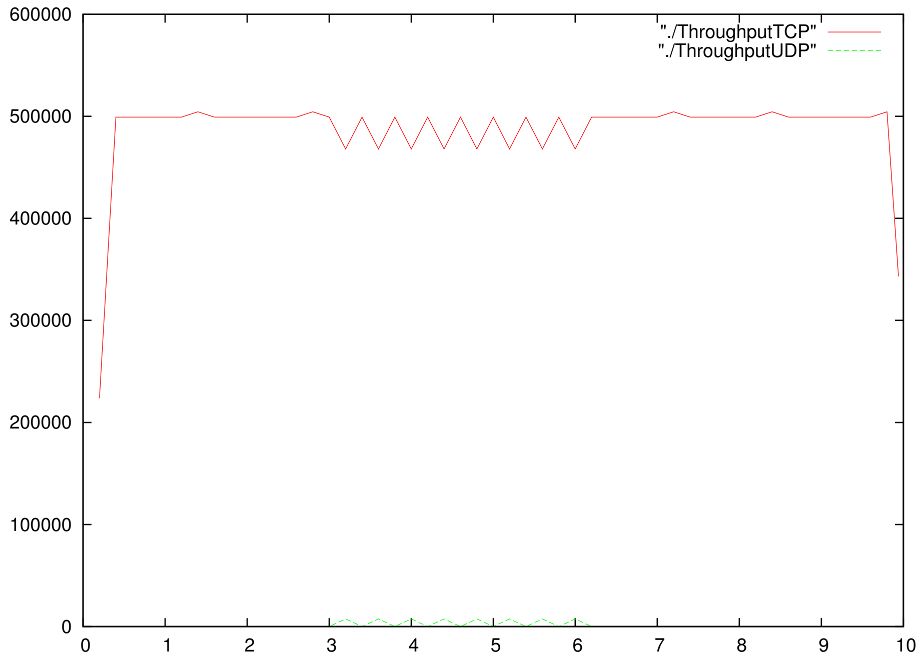

Figure 3: Throughput of FTP and CBR connection at node 1 of KotNet topology with limited upload bandwith of CBR connection

## Question 6

a) With 10 users, the amount of users and bandwidth scale with the same factor, so the performance will be the same. When there are only 5 users using the network, one can reach a better performance. Suppose that the UDP traffic also increases by the same factor. The TCP traffic will then have twice as much bandwidth available.

b) Yes, the performance will be more oscillating because the available bandwidth on the network varies randomly. Other users will suffer when one user decides to have a lot of UDP traffic because this UDP traffic obstructs acknowledgements from the TCP traffic which results in congestion. The buffers will be overloaded and time-outs will occur.

# Exercise 2

## Question 1

From figure 4 we can see that when the web traffic bursts start at 5.0s, the long lasting FTP application does not immediately experience a drop in throughput. This is due to the slow start mechanism of the TCP connection which only reaches the maximum throughput at around 6.0s. The burst of HTTP requests will cause congestion. This results in reduced throughput and retransmission of dropped packets. We can see this on the graph where the burst traffic lasts until 8.0s instead of 7.0s. The bursts of HTTP traffic beginning at 10.0s and 15.0s can be analyzed analogously.

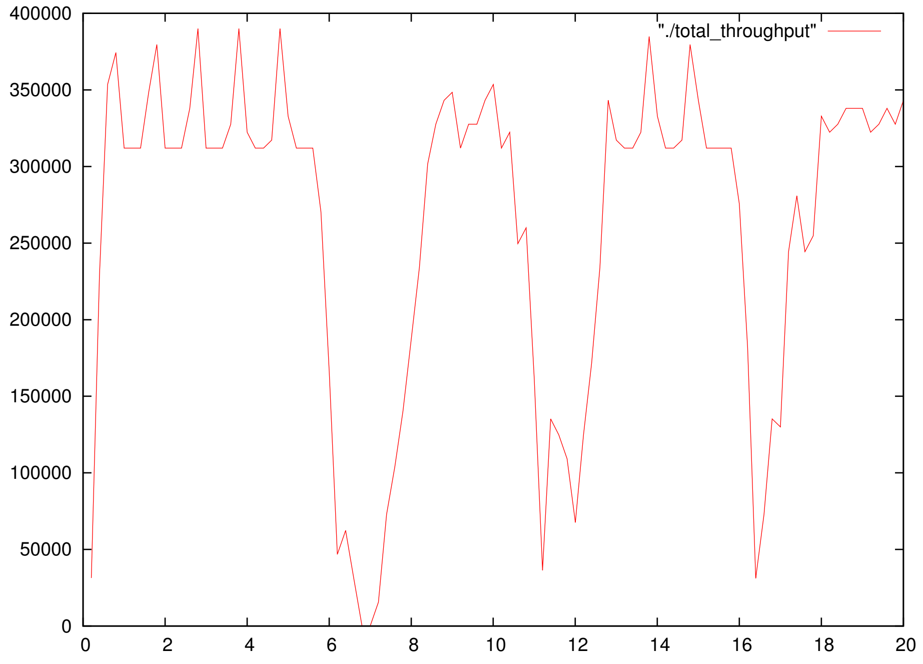

Figure 4: Throughput of FTP connection at node 2 of topology of exercise 2

## Question 2

The different phases of the TCP algorithm are the slow start phase and the congestion avoidance phase in which the window size grows exponentially and linearly respectively. We can see this on figure 5: the algorithm begins in the slow start phase. The window size grows exponentially. When the window size equals the slow start threshold, a nearly linearly increasing windows size is observed. The TCP algorith is now in the congestion avoidance phase. When congestion occurs and is detected, the window size will drop and the algorithm will start over again. A new threshold will be calculated when congestion occurs. The value of this new threshold is equal to half the value of the current slow start threshold.

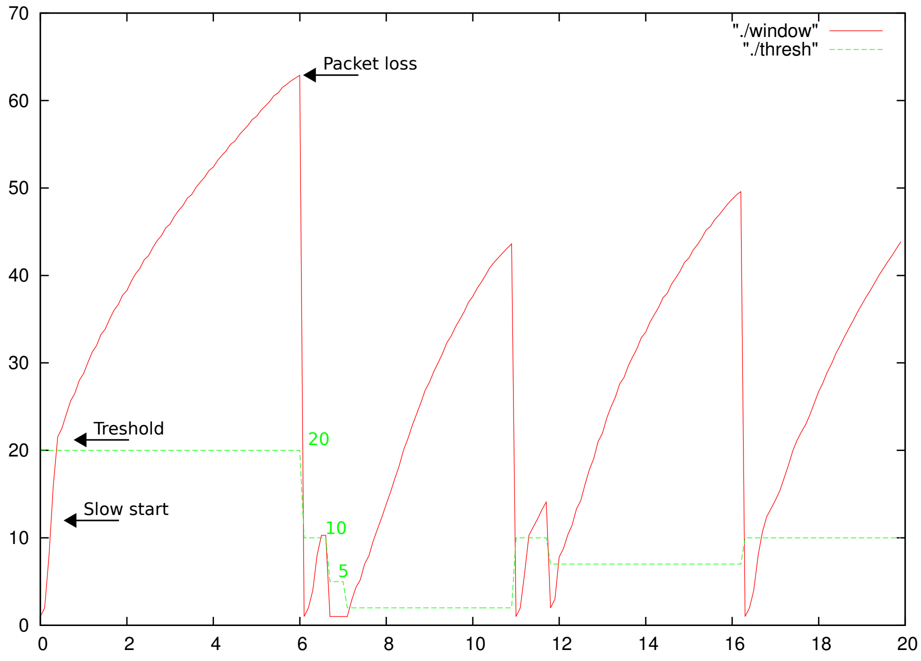

Figure 5: Congestion window size and slow start threshold of main FTP application with TCP Tahoe

## Question 3

The *additive increase/multiple decrease (AIMD)* algorithm is used in TCP to make a feedback loop based on congestion signals. As congestion is a bad thing to happen because it also influences other clients on the network, a rapid decrease in traffic is needed to minimize the effect of the congestion. This is done by multiple decrease whereas when the network is not congested one can slowly increase the traffic with additive increase. It is shown by **Chiu et al**[1]

[1]Chiu, Dah-Ming; Raj Jain (1989)."Analysis of increase and decrease algorithms for congestion avoidance in computer networks". Computer Networks and ISDN systems 17:1-14.

that this asymmetry in buildup and break down speeds results in convergence to an optimal point.

A few reasonable values for the congestion window are shown on figure 5. The first interval where the TCP congestion avoidance algorithm is active is [0.45  5.45] as can be seen on the graph.

## Question 4

The difference between Tahoe and Reno lies in the fact that when congestion occurs, Tahoe always reduces the window size to 1 whereas Reno will perform fast recovery. This fast recovery will ensure that after 3 received duplicate acknowledgements the packet has not been transmitted succesfully. The congestion window size will be halved instead of lowered to 1 as with Tahoe. When more than one packet is lost and fast retransmission does not recover adequately, Reno acts the same way as Tahoe: the congestion window size will drop to 1 and slow start occurs. This happens most of the time as can be seen on figure 6.

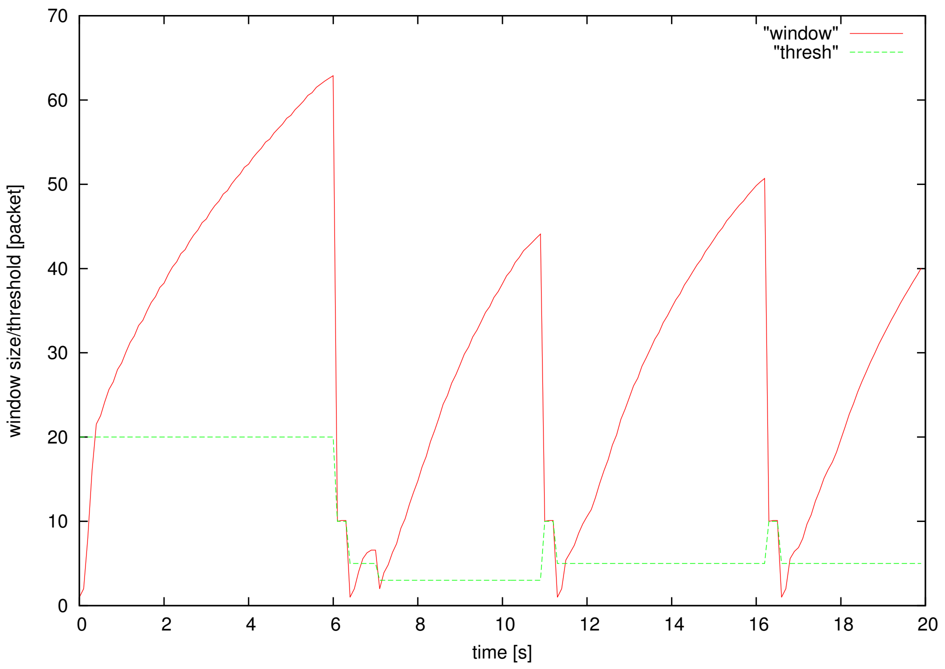

Figure 6: Congestion window size and slow start threshold of main FTP application with TCP Reno

## TCL code of exercise 2

```
#Create simulator
set ns [new Simulator]

$ns color 1 Red
$ns color 2 Blue
$ns color 3 Green
#trace file
set tf [open exercise_2.tr w]
$ns trace-all $tf

#nam tracefile
set nf [open exercise_2.nam w]
$ns namtrace-all $nf

proc finish {} {
        #finalize trace files
        global ns nf tf
        $ns flush-trace
        close $tf
        close $nf

        exec nam exercise_2.nam &
        exit 0
}

# 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
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
$ns duplex-link $n0 $n4 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

#set queue limit for link between nodes n0 & n1
$ns queue-limit $n0 $n1 20

#setup 1 ftp over tcp connection between nodes n3 & n2
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n3 $tcp0
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $n2 $sink
$ns connect $tcp0 $sink
$tcp0 set fid_ 4
$tcp0 set window_ 80
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#setup 120 ftp over tcp connections between nodes n5 & n4
for {set i 1} {$i < 121} {incr i} {
 set tcp($i) [new Agent/TCP]
$ns attach-agent $n5 $tcp($i)
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp($i) $sink
if { $i < 121 } {
$tcp($i) set fid_ 1
}
if { $i < 81 } {
$tcp($i) set fid_ 2
}
if { $i < 41 } {
$tcp($i) set fid_ 3
}
set ftp($i) [new Application/FTP]
$ftp($i) attach-agent $tcp($i)
}

#setup delays with their respective distributions
set delay_init [new RandomVariable/Exponential]
$delay_init set avg_ 0.05

set RVSize [new RandomVariable/Pareto]
$RVSize set avg_ 150000
$RVSize set shape_ 1.5
#setup start and end times of long lasting ftp connection between nodes n3 & n2
$ns at 0.0 "$ftp0 start"
$ns at 20.0 "$ftp0 stop"
#setup start and end times of bursts of traffic over ftp connection between noc
set startT(0)  5.0
for {set i 1} {$i < 41} {incr i} {
set Size($i) [expr [$RVSize value]]
set startT($i) [expr $startT([expr $i - 1]) + [$delay_init value]]
$ns at $startT($i) "$ftp($i) send $Size($i)"
$ns at 7.0 "$ftp($i) stop"
}

set startT(40)  10.0
for {set i 41} {$i < 81} {incr i} {
set Size($i) [expr [$RVSize value]]
set startT($i) [expr $startT([expr $i - 1]) + [$delay_init value]]
$ns at $startT($i) "$ftp($i) send $Size($i)"
$ns at 12.0 "$ftp($i) stop"
}

set startT(80)  15.0
for {set i 81} {$i < 121} {incr i} {
set Size($i) [expr [$RVSize value]]
set startT($i) [expr $startT([expr $i - 1]) + [$delay_init value]]
$ns at $startT($i) "$ftp($i) send $Size($i)"
$ns at 17.0 "$ftp($i) stop"
}

proc plotWindow {tcpSource1 outWindow} {
   global ns
   set now [$ns now]
   set cwnd [$tcpSource1 set cwnd_]

   puts  $outWindow "$now $cwnd"
   #Recursive call
   $ns at [expr $now+0.1] "plotWindow $tcpSource1  $outWindow"
}

proc plotThresh {tcpSource2 outThresh} {
   global ns
   set now [$ns now]
   set ssthresh [$tcpSource2 set ssthresh_]

   puts  $outThresh  "$now $ssthresh"
   #Recursive call
   $ns at [expr $now+0.1] "plotThresh $tcpSource2  $outThresh"
}

set outWindow [open  "window"  w]
$ns  at  0.0  "plotWindow $tcp0  $outWindow"

set outThresh [open  "thresh"  w]
$ns  at  0.0  "plotThresh $tcp0  $outThresh"

#finishing statements
$ns at 20 "finish"
$ns run
```