Universität Stuttgart

INSTITUT FÜR KOMMUNIKATIONSNETZE UND RECHNERSYSTEME
Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn

Interner Bericht / Internal Report N° 47

Titel / Title	**ns-2/IKRSimLib comparison**
Verfasser / Author(s)	Fabio Querzola in cooperation with M. Scharf and C. Gauger

Datum / Date	June 17, 2004
Umfang / Size	20 Seiten / Pages
Schlüsselworte / Keywords	ns-2, IKRSimLib, TCP

Kurzfassung / Abstract

This paper studies the main differences, in terms of simulation results, between two network simulator tools (*ns-2* and *IKRSimLib*). We discuss our experience using these simulators for studying the TCP behavior in different network conditions: Varying bottleneck buffer size, bottleneck link bandwidth, number of TCP sources and packet loss probability. Simulating the same network model, we measured the performance in terms of throughput to understand the different TCP implementations in the two software libraries.

1 Introduction

Simulators are an important instrument to study networks behaviour and networks design, because they allow us to compare different alternatives and evaluate their performance under well-known conditions including a very large parameters range, which is not possible in the real-world.

In the context of computer networks we can affirm that a *simulator* has the purpose to capture the behaviour of a hypothetical system with the aim to help researchers on their work.

A simulator has the following properties [3]:

- It allows to create an approximated model of a real system, which is used to study system behaviour in certain conditions.
- The working conditions are well-known and well-defined, repeatability is necessary to understand events.
- In a discrete simulator variables change at discrete time between discrete values, computer networks are discrete by nature so *discrete simulators* are used to describe them.

Due to the fact that a lot of simulation tools exist and each one is developed independently, it is important to understand the differences among them. This makes it possible to compare the results and derive conclusions from different tools. As a matter of fact, simulating TCP is difficult because of the wide range of variables, different implementations and enviroments available [4]. Moreover, there are so many subtleties in TCP's mechanism that before starting a simulation (***network model setup phase***) it is very important to define what we want simulate, choosing right network elements, connect them and decide in which conditions they have to work.

For this reason we could be sure that if we would try to simulate the same model with two different tools, we will never obtain the same results.

In this paper we consider two simulation tools: *ns-2* (network simulator version 2) and *IKRSimLib* (simulation library), we give a short description of them, explain the network model used and present some simulation results.

2 Network simulator ver. 2

Ns-2 is a **discrete-event** simulator completely free and open source, developed in UC Berkeley, written in ***C++*** with an ***OTcl*** *interpreter* as a frontend.

The main part of this simulator is the *scheduler*, that is a particular object which schedules events during the simulation (discrete-event) and therefore manages the real time of the simulation [2]. To work with ns-2 we first have to install it and then run it with the shell command:

ns <script_name>.tcl

Writing an *OTcl script* is necessary to define a simulation scenario. By this script ns-2 receives all needed informations to start the simulation, such as nodes definition and interconnection, communications between applications through their transport agents (e.g. TCP) with related parameter, and definition of events that may occur during the simulation.

Objects used in the script belong to OTcl class hierarchy and have a one-by-one correspondence in C++ class hierarchy through the OTcl interpreter which translates the commands written in OTcl to their equivalence to the C++. Therefore the user can create a network scenario as if the simulator is written entirely in OTcl, and there is no need to know the C++ hierarchy, since the OTcl interpreter does this job for us, at least until when we want to create a new class.

It is possible to extend the simulator in two ways by inserting new classes:

- Using **C++ code**: This method is the most efficent, because the C++ code represents the core of the simulator and by it we are able to modify and access every class variable or function, or derive every kind of object in the C++ hierarchy.
- Using **OTcl code**: This method is less efficent than the previous one, because from OTcl level we are not able to access many class variables and functions defined at C++ level.

Moreover, through the Tcl script it is possible to extract results from the simulation:

- At any moment during the simulation we can *read* certain variables of an object and process them with an OTcl procedure to obtain the desired result or simply write them to standard output or to a file.
- Using particular *measurament object* called monitor and trace object:

  Monitor objects : Used to get particular statistics concerning packets traffic through an object (exploiting integrator class), or to get particular counters, such as packet arrivals, departures or drops.

  Trace objects : Record every event for each packet at a queue or at a link, such as an arrival, a departure, etc. . . , together with the time at which the event occurs, and write all into a file to be post-processed.

Ns-2 can be downloaded from the Internet [1] and the all-in-one version includes some useful tools, such as *nam* (network animator) and *Xgraph* which are software package used for the visualization of results.

3 Simulation library

Simulation library is a ***C++*** class library which is used for **event-driven** simulations in the communication networks field.

To setup a simulation we first have to define the simulation *model*. To do this we can use all allowed C++ programming ways, for example writing two C++ files (<*model*>.*cpp* and <*model*>.*h*) exploiting simulation library classes and functions. In this phase we mainly define network model layout, and we could also specify object parameter that do not change in all our simulations. Once created the model we have to write the "*main*" file which has the purpose to setup the simulation (managing events, input parameters and result output) using particular classes [5]. Written this three basic files we can compile them with a "makefile", to do this it is possible to use a template file. To control the simulation the "*main()*" function usually receives two files as input: <*model*>.*fmt*, used to define the format which the results will be printed, and <*model*>.*par* used to initialize the parameters of individual model components that could change from a simulation to another. Moreover, it is necessary to specify in the *main()* the output file (<*model*>.*log*) where results will be printed.

Now we can start the simulation running the executable file with its parameters files:

<*file_exe_name*> <*model*>.*par* <*model*>.*fmt* <*model*>.*log*

In the simulation library there are different classes of measurement objects that enable to extract results from the simulation. They are divided in two categories:

Filters : Used to observe the packets that are passing through a point or to modify an attribute of this packets.

Meters : Used to make statistical measurements, e.g. between to points in the network.

In order to reproduce stochastic processes in the simulation model, such as arrival process or service process, the simulation library offers a very large variety of pre-defined distribution functions to generate random variables.

4 The simulation model

The generic network model which has been simulated with both software tools is sketched in Fig.1 and represents a very simple network scenario. We want to capture the TCP behavior differences between the two simulators, varying some of the main network variables.

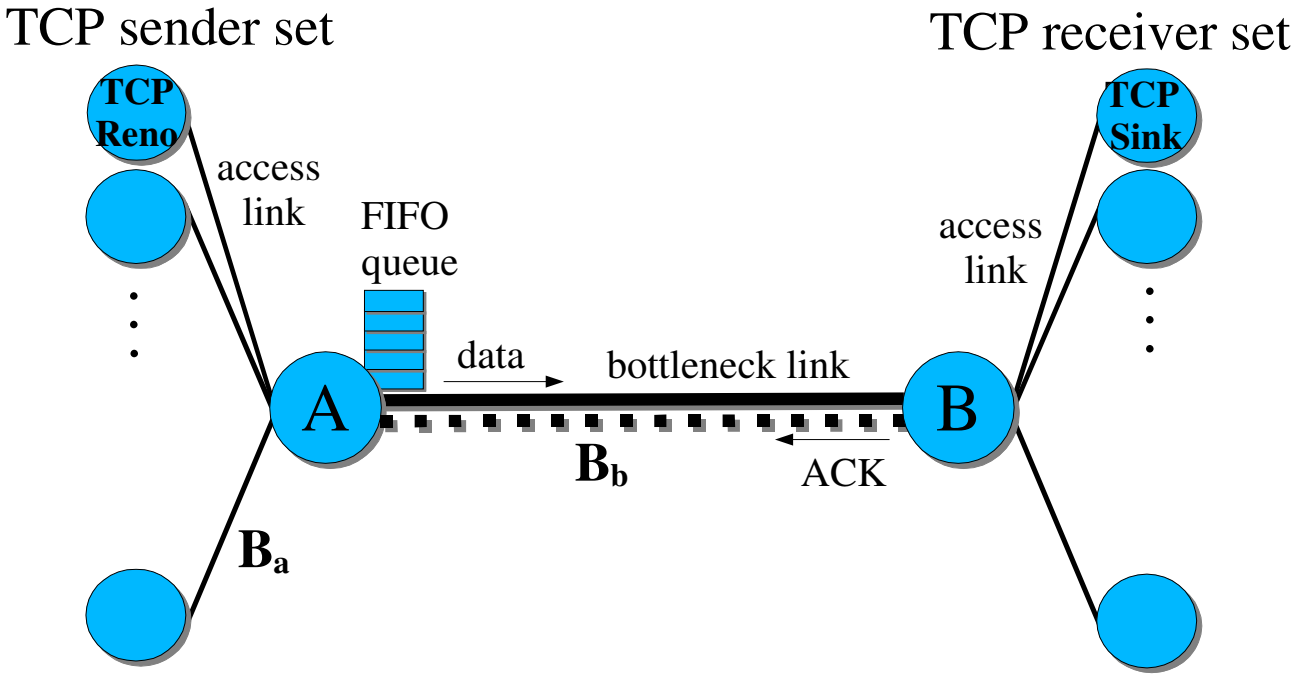

Figure 1: Network scenario

This model consists of:

- A *TCP sender set*: Composed of a variable number of TCP sources (**Reno** version), each one connected to the network through an ***access link*** with speed B_a in the uplink direction, and through an "ideal" link (infinite bandwidth and null propagation delay) in the reverse direction.
- A *network*: Composed of two ***simplex links***, one for the DATA path (*bottleneck link*) and one for the ACKnowledgements path, both with the same speed B_b and the same *propagation delay*. For the DATA path we assumed a Bernoullian *packet loss probability* with parameter p and a FIFO queue with dimension bs.

- A *TCP receiver set*: Composed of TCP receivers (**Sink Delayed ACK** version), each one connected to a different TCP sender agent and accessing to the network through an "ideal" link (infinite bandwidth and null propagation delay).

Connected to each TCP sender agent there is a so called *greedy source*, representing an application that delivers to the TCP agent always the amount of data that it can sent according to the TCP algorithm. So the TCP ingress buffer is always full.

Obviously this model is generic, because of every simulator implements each component in a different way.

5 Simulation results

Our simulations measured the *total* ***throughput***, i.e. the number of TCP segments arrived at the receiver set in a certain time.

To perform this throughput measurement, in ns-2 we used a *monitor* inserted on the bottleneck link queue which is able to count packets that cross the link. In Simulation Library we used a *meter* on TCP receiver set which counts the arrived packets.

The first diagram (Fig.2) shows the throughput measured in both simulators as a function of the packet loss probability p with B_a=2 Mbit/s, B_b=10Mbit/s, infinite buffers and one TCP source. The throughput is upper limited by the access bandwidth (250 Byte/ms correspond to 2 Mbit/s) and decreases with packet loss probability because of the TCP congestion control. The two curves almost completely overlap for small p values, and for p>0.01 they start to differ. This gap increases with p.

As we will see, *ns-2* throughput curves (ns) always stay below *IKRSimLib* throughput curves (SL).

The main behaviour remains the same if we change the access bit rate, as depicted in Fig.3. But in this case the maximum throughput is greater and limited by the TCP congestion control algorithm that does not allow a throughput saturation to the access bandwidth value (5 Mbit/s correspond to 625 Byte/ms). For p>0.01 the throughput behaviour does not depend on the access bandwidth value B_a, as well as the gap between IKRSimLib and ns-2 curves.

In the critical range of p values (greater than 0.01) the implementation of TCP algorithm behaves in different way in the two tools, and for p on the order of 10% we observed an error around 50%. So in this range we can consider the results obtained with two tools not comparable.

As a first reason of this gap between the two tools, we observed that *ns-2* implements the TCP algorithm measuring the window and the slow start threshold in ***packets*** (also called segments) whereas in *IKRSimLib* they are managed in ***Bytes***. This could be the main reason of the gap, because for high values of p the dynamics of the congestion window is emphasized and it is very probable that it works with not integer values. Slow start threshold remains at low values and the TCP works most of the time in congestion avoidance. This phenomenon has a worse consequence in *ns-2*, which counts the window in packets, than in *IKRSimLib*, which counts the window in Byte. As a matter of fact, *IKRSimLib* has a better granularity than *ns-2* measuring the amount of data that can be sent by the TCP agent at each round on the network. Regarding this we did not find any standard specifications, thus both tools implement TCP in a right way.

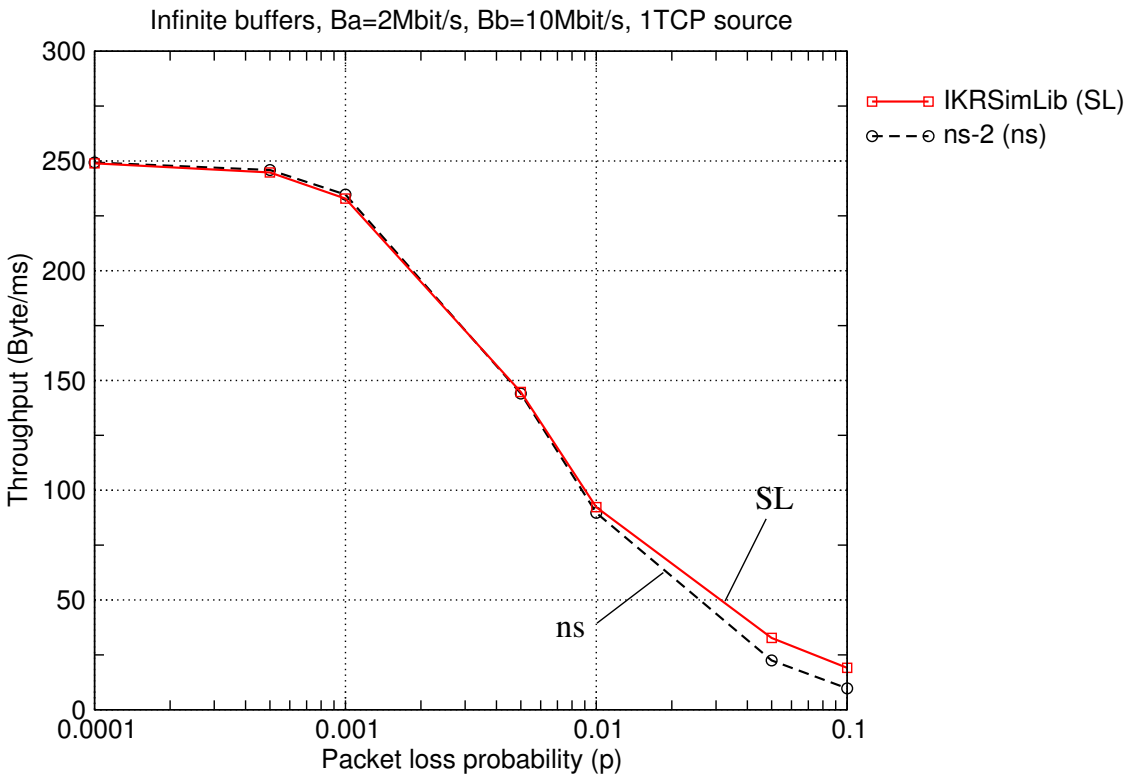

Figure 2: Throughput as a function of packet loss probability, with B_a=2 Mbit/s, B_b=10 Mbit/s, infinite buffers and one TCP source.

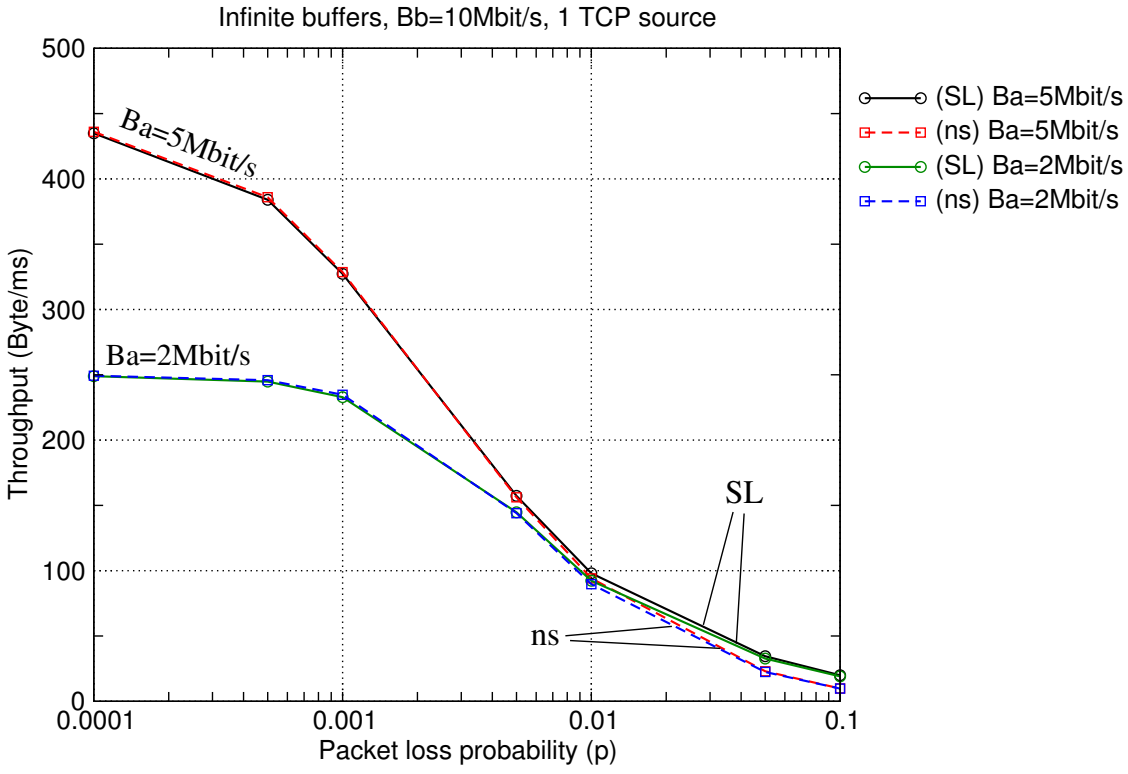

Figure 3: Throughput as a function of packet loss probability, varying B_a, with B_b=10 Mbit/s, infinite buffers and 1 TCP source.

In the next graph (Fig.4) we kept all the network parameters fixed and varied only the number of TCP agents at the sender and receiver sets. The throughput curves have a saturation due to B_a for small p and they increase proportionally with number of TCP connections, up to a value of five TCP sources where the total throughput is limited by the bottleneck link bandwidth B_b (10 Mbit/s correspond to 1250 Byte/ms). The effect of p on the gap between the two tools is the same observed above, and it is emphasized due to aggregate traffic. The gap increases almost proportionally with the number of sources, and this means that this effect acts in the same way on each source.

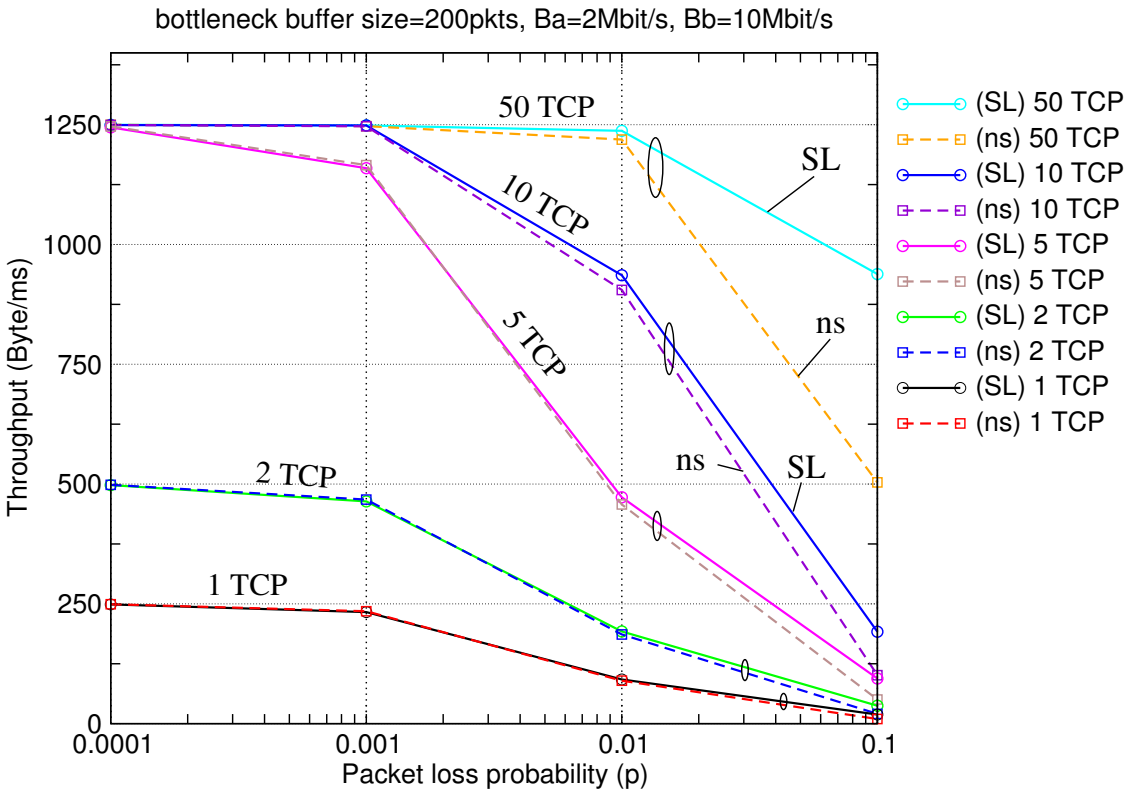

Figure 4: Throughput as a function of packet loss probability, varying number of TCP connections. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=200 packets.

Reducing the bottleneck buffer size, for example to five packets (Fig.5), we observe a further gap increase, especially for p<0.01. This is due to the fact that a small buffer size increases the total packet loss probability, in particular for a number of TCP sources greater than five, where we experienced a buffer overflow. In the cases of one, two and five sources the behaviour does not change because there is not a buffer overflow. Moreover, the further gap increase is observable only for values of p smaller than 0.01, because for greater values the TCP dynamics does not allow a buffer filling.

Again, in scenario of Fig.5 the total maximum throughput for fifty and ten sources has not a saturation at B_b, but it is lower, limited by the TCP congestion control. We can also observe that the best case for maximum throughput is with five sources, where it is possible to achieve a saturation at B_b for p=0.0001. We represented this phenomenon in Fig.6, where we studied the *throughput* behaviour together with the *goodput* as a function of the number of TCP sources for extreme cases of p and with a small bottleneck buffer size (bs=3 packets). This study was made only for IKRSimLib tool, because we already know the ns-2 behaviour.

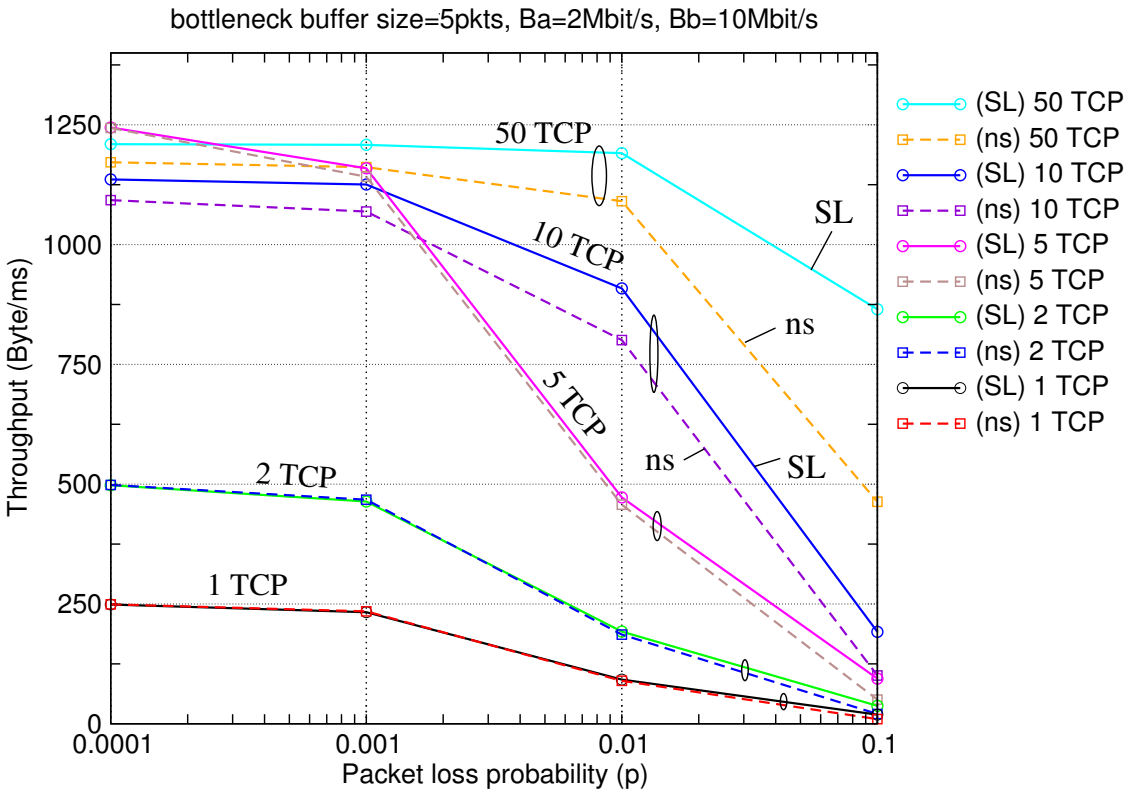

Figure 5: Throughput as a function of packet loss probability, varying number of TCP connections. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=5 packets.

Goodput is the amount of data delivered from the TCP agent to the application layer in a certain time. We compared throughput and goodput to understand whether there was a difference measuring one or the other. We concluded that there is no difference, they have the same behaviour unless for the fact that goodput is lower because it excludes the retransmitted packets.

For very small bottleneck buffer size, and for small p values, the maximum number of TCP sources that allow to achieve the maximum throughput (limited by B_b=1250 Byte/ms) is five. As a matter of fact, in this case we have five source that transmit at their maximum (single throughput limited by B_a=250 Byte/ms), and we also do not have a buffer overflow because the bottleneck link can serve all access link. Increasing the number of access links we exceed the bottleneck link capacity and create packet overflow (Fig.7).

As we mentioned above this overflow effect is emphasized for small packet loss rate on the bottleneck link (p), because of the TCP window dynamics. For p=0.1 each source is strictly limited by its TCP agent and the total throughput increase is less aggressive.

Considering again the case with p=0.0001 in Fig.6, the throughput behaviour for one, two and five TCP sources is clear, because it grows linearly with number of sources. If the number of sources is greater than five we have a buffer overflow and the congestion control has a strong impact on single source throughput. For every scenario with more than five sources the total throughput is limited by the TCP algorithm but it grows if the number of sources increases.

To explain better the impact of the bottleneck buffer, we also present the previous diagrams but with bs=200 packets (Fig.8, 9).

As we mentioned above a finite bottleneck buffer introduces a further packet loss probability due to overflow. Therefore there are two packet loss components not statistically independent, and the total packet loss measured is not simply the sum.

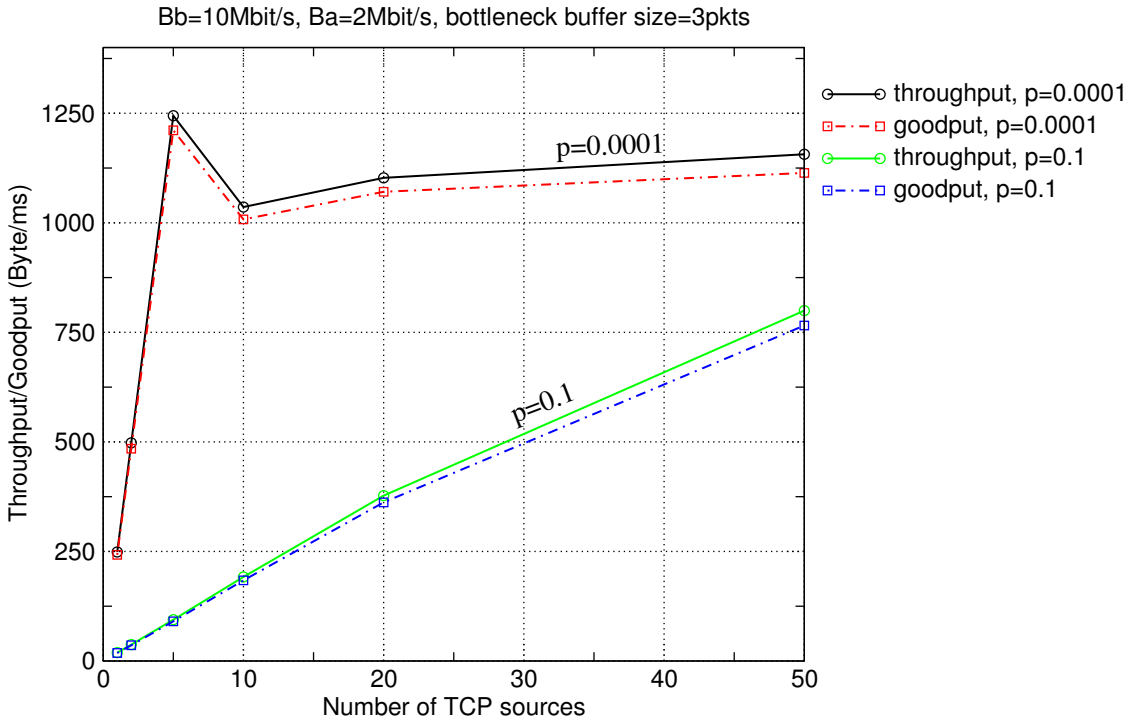

Figure 6: Throughput and goodput as a function of number of TCP sources, varying the packet loss probability. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=3 packets.

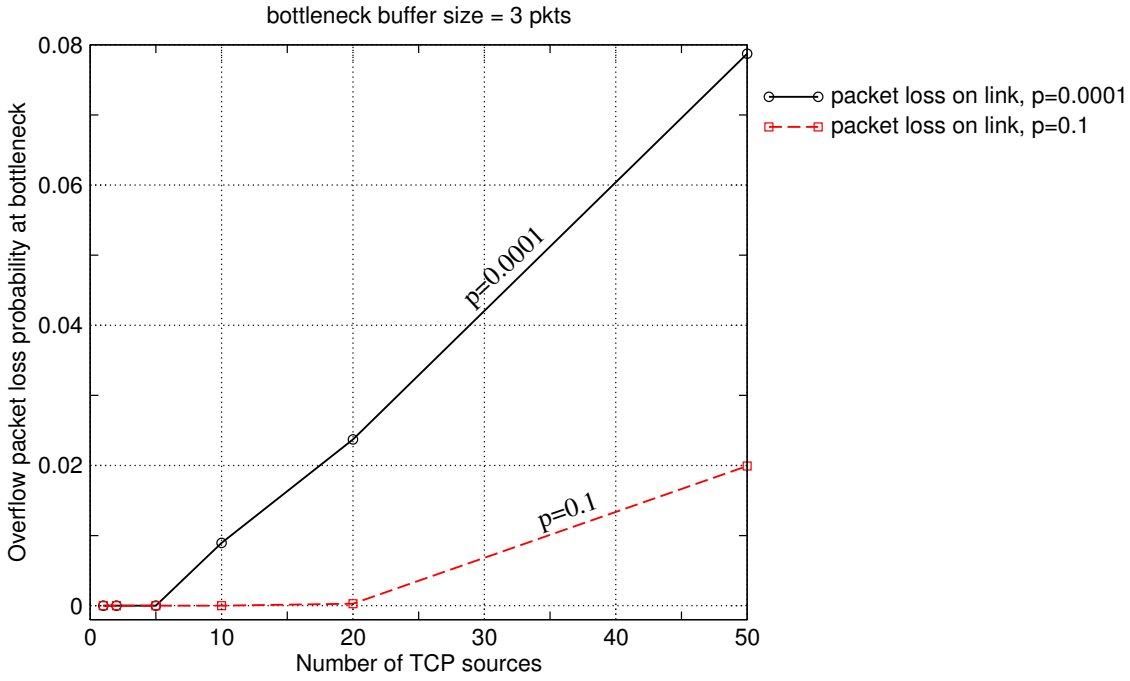

Figure 7: Overflow packet loss probability at bottleneck link buffer as a function of number of TCP sources, varying packet loss probability on the bottleneck link. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=3 packets.

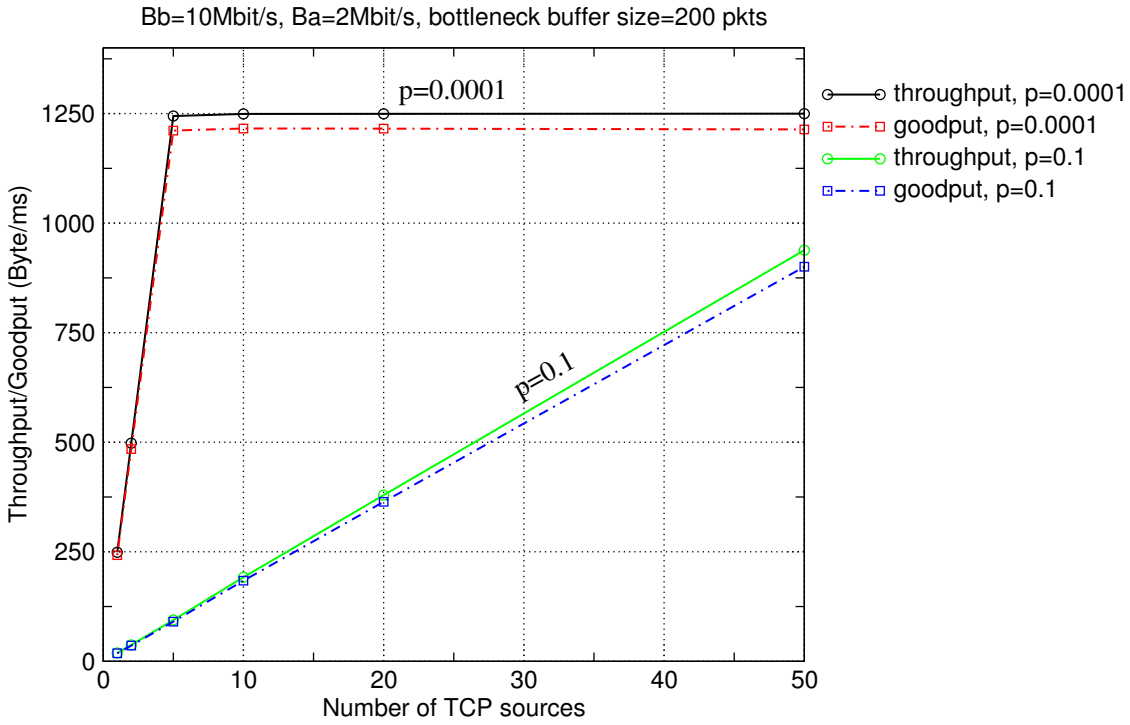

Figure 8: Throughput and goodput as a function of number of TCP sources, varying the packet loss probability. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=200 packets.

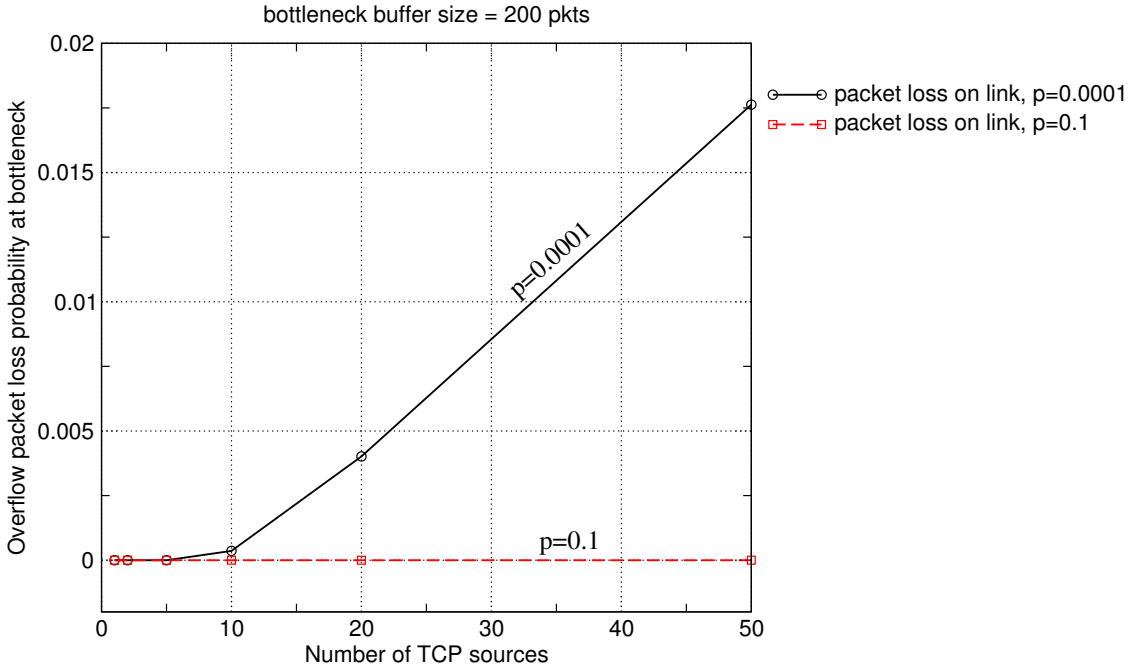

Figure 9: Overflow packet loss probability at bottleneck link buffer as a function of number of TCP sources, varying packet loss probability on the bottleneck link. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=200 packets.

For the scenario with bs=200 packets and for p=0.0001, we can consider p as the total packet loss probability up to ten TCP sources, because for a greater number of sources (Fig.9) the component due to the overflow is not negligible, and the real packet loss rate is greater than p. With p=0.1, p represents in all cases the real packet loss probability.

Fig.10 presents the same results from Fig.4 in a different view. It shows how the throughput gap between ns-2 and IKRSimLib increases with the number of TCP connections for different values of packet loss probability p, fixed B_a=2 Mbit/s, B_b=10 Mbit/s and bottleneck buffer size to 200 packets. The total throughput is upper bounded by the bottleneck link bandwidth B_b, and the curves with higher loss probability stay always below.

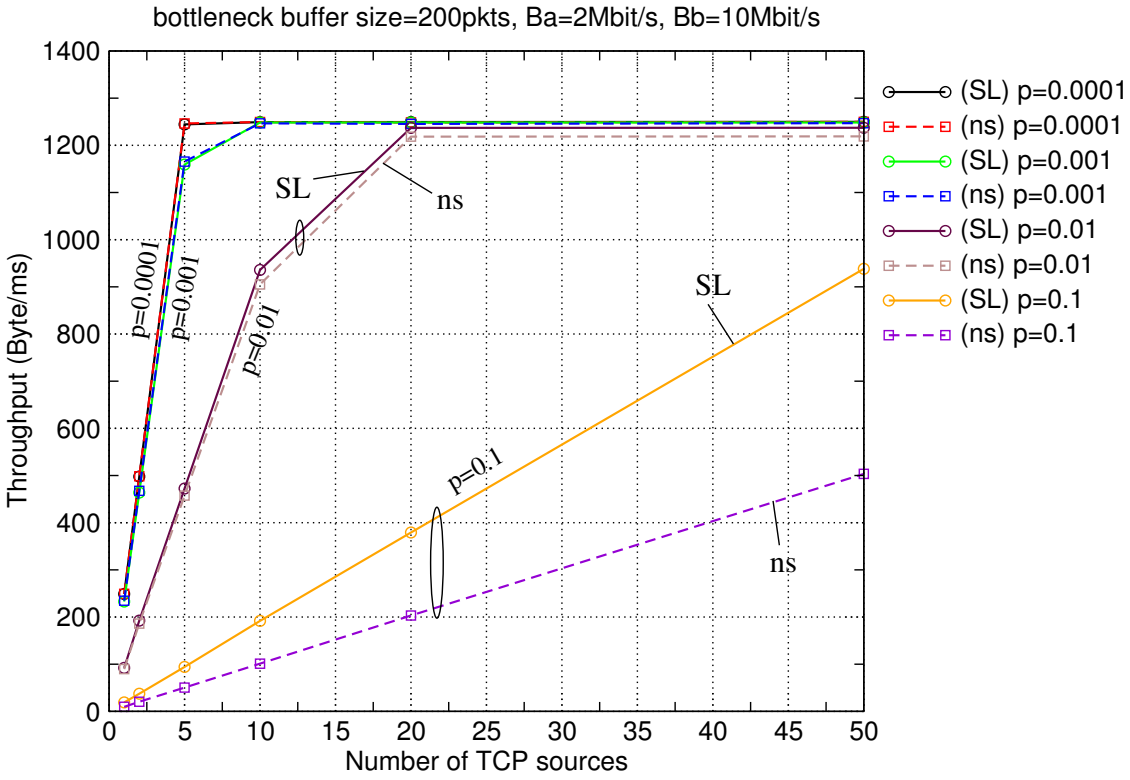

Figure 10: Throughput as a function of TCP connections number, varying packet loss probability. With B_a=2 Mbit/s, B_b=10 Mbit/s, bs=200 packets.

The next four graphs (Fig.11, 12, 13, 14) represent throughput as function of bottleneck buffer size, varying the number of TCP sources, for different packet loss probability.

In Fig.11 we can observe that for reasonable buffer size values (greater than 10 packets) and for p=0.0001, *ns-2* and *IKRSimLib* have a good fit. The maximum throughput is achieved with five TCP sources. For smaller buffer size the throughput decreases (ten and fifty TCP sources curves) and the gap increases, because of larger total packet loss probability values.

Again we can observe that the gap between *ns-2* (ns) and *IKRSimLib* (SL) increases with packet loss probability p, and ns-2 throughput is always lower than *IKRSimLib* throughput.

For increasing values of p, whereas the gap increases, we can see a general decrease of the throughput upper bound, limited by the TCP congestion control.

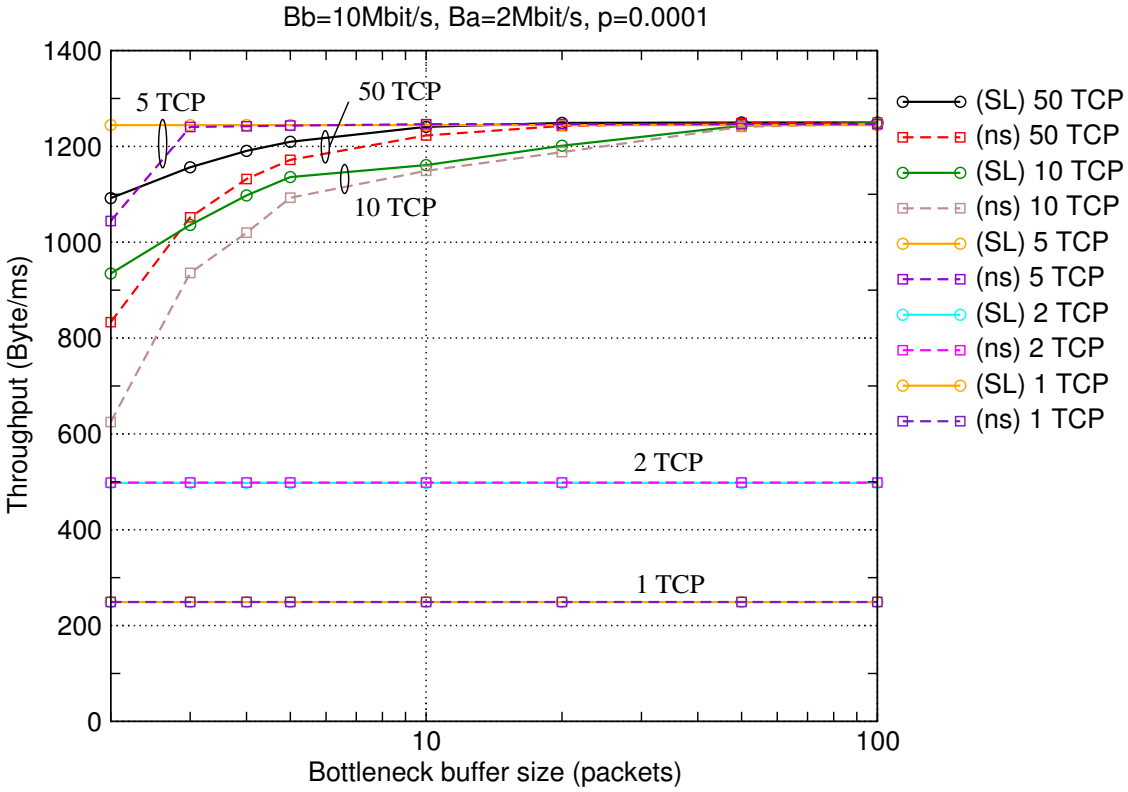

Figure 11: Throughput as a function of bottleneck buffer size, varying TCP sources number. With B_a=2 Mbit/s, B_b=10 Mbit/s, p=0.0001.

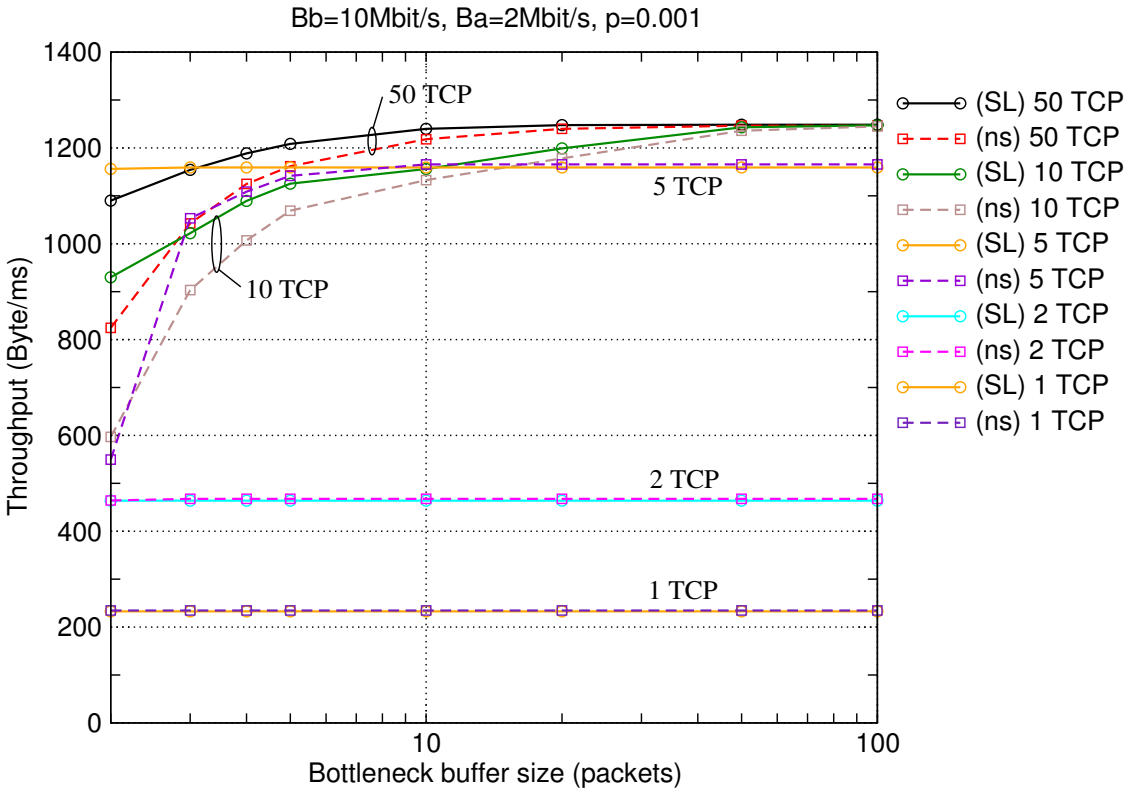

Figure 12: Throughput as a function of bottleneck buffer size, varying TCP sources number. With B_a=2 Mbit/s, B_b=10 Mbit/s, p=0.001.

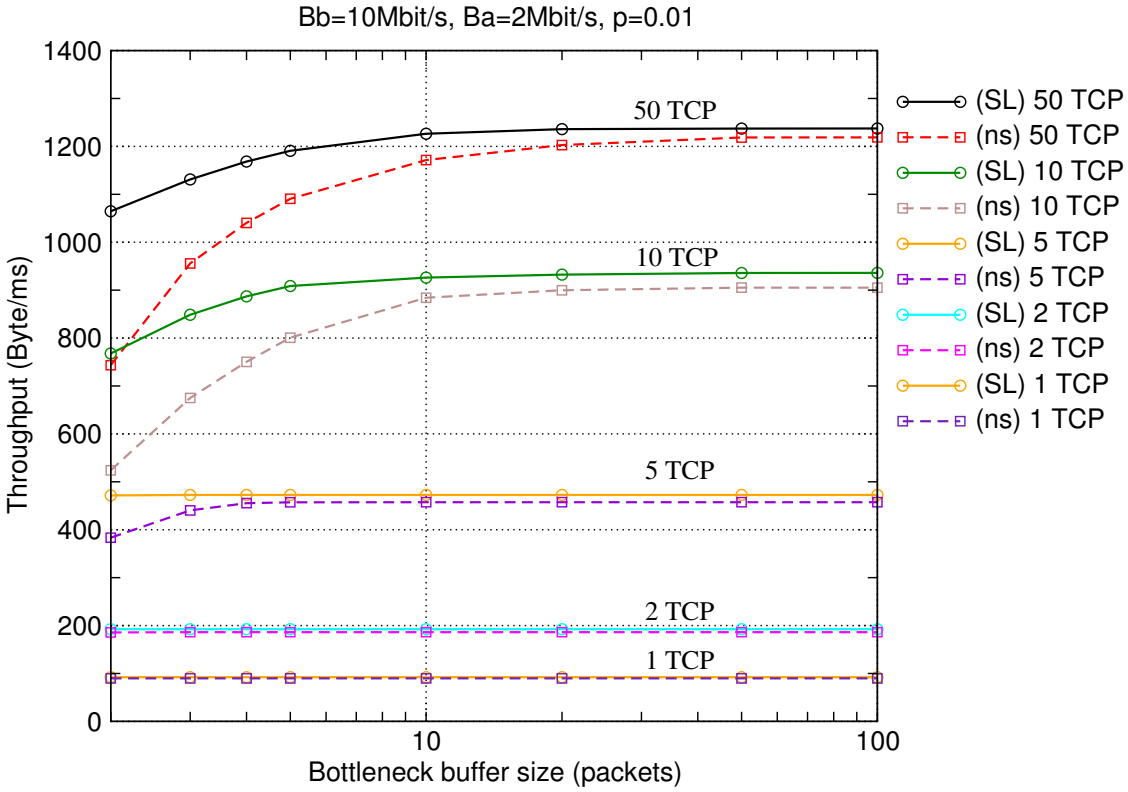

Figure 13: Throughput as a function of bottleneck buffer size, varying TCP sources number. With B_a=2 Mbit/s, B_b=10 Mbit/s, p=0.01.

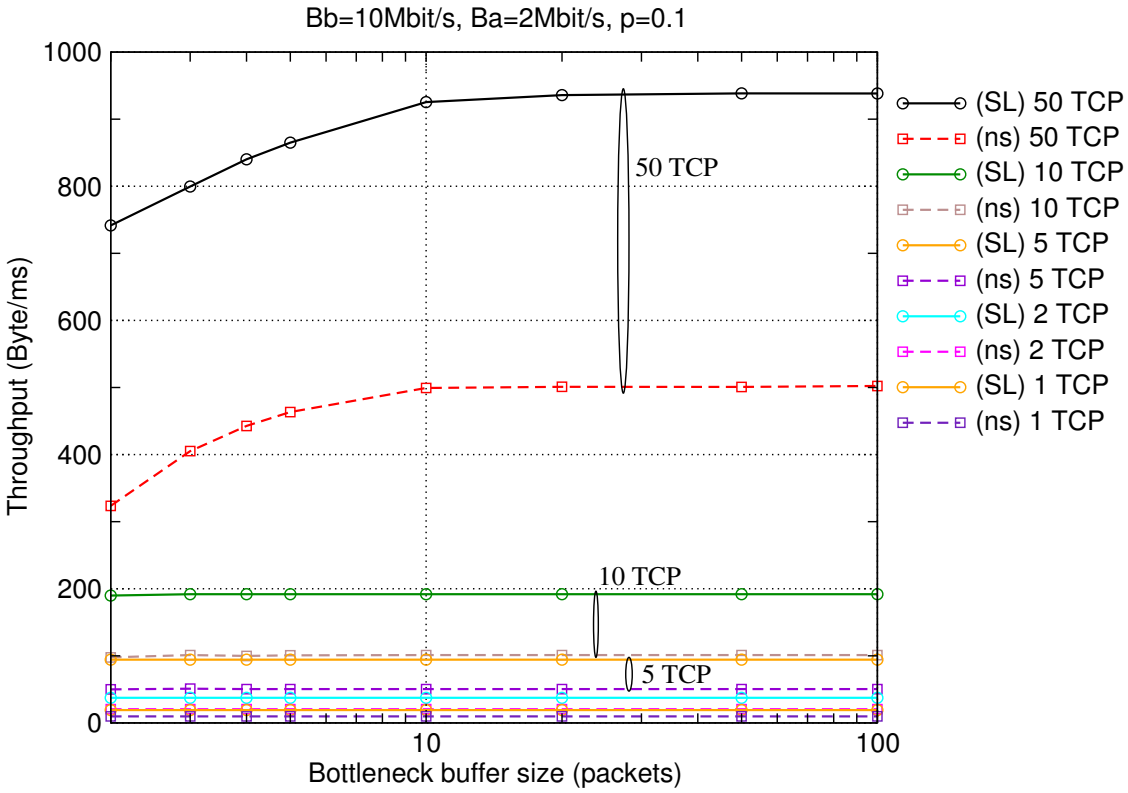

Figure 14: Throughput as a function of bottleneck buffer size, varying TCP sources number. With B_a=2 Mbit/s, B_b=10 Mbit/s, p=0.1.

Resuming the performance measured we can conclude that considering p as the total packet loss probability, in the worst case (p=0.1) ns-2 differs from IKRSimLib of about 49%, for p=0.05 of about 30%, for p=0.01 of about 3%, whereas for p<0.01 the two tools fit very well.

We observed a strange behaviour for *IKRSimLib* in the particular case of five TCP sources: With very small buffer size (bs=2 packets), the throughput does not decrease and there is not buffer overflow. This means that the five sources never transmit packets to the bottleneck link all at the same time, but this is not a real behaviour. About ns-2 we experienced a throughput decrease for bs=2 packets, and also analyzing the queue implementation we realized that in ns-2 setting a buffer size of n packets means to have a queue dimension of $n - 1$ packets. This fact is mentioned in the ns-2 manual as a note and the explanation is that according to the queueing theory a queue with a dimension of one packet does not work. Thus the minimum dimension for a queue is two packets. In IKRSimLib we experienced that a queue with one packet dimension works anyway.

As last graph we present the measurement of simulation duration for a particular case (Fig.15), varying the number of TCP sources. We simulated both models for the same simulation time, i.e. 1500 seconds, and on the same machine with a 1,5GHz processor and 2GByte memory.

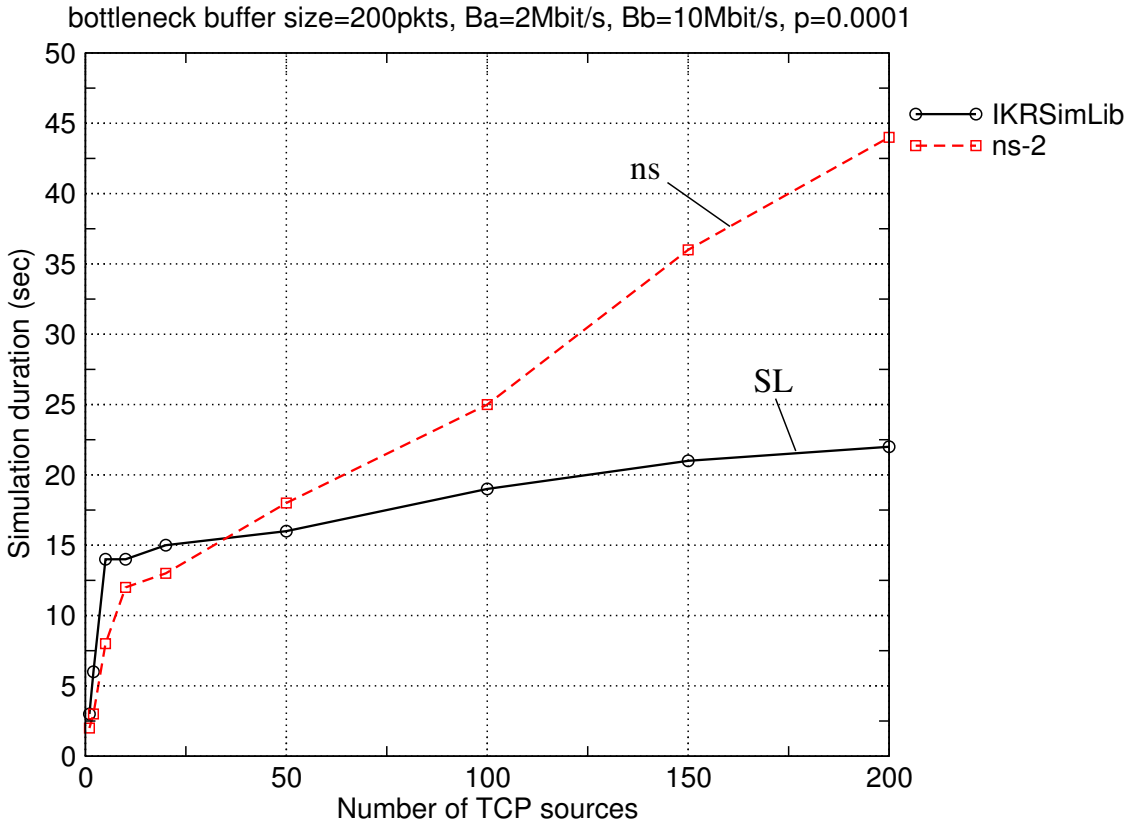

Figure 15: Simulation time comparison as a function of TCP number sources, with B_a=2 Mbit/s, B_b=10 Mbit/s, bs=200 pkts and p=0.0001.

For values up to five sources the duration of the simulation grows steeply and *ns-2* is always twice faster than *IKRSimLib*. It seems that *IKRSimLib* has a saturation simulating more sources while *ns-2* has a almost linear increase. Therefore, *IKRSimLib* becames much faster than *ns-2* if we simulate a large number of sources. The difference between *ns-2* and *IKRSimLib* seems to be rather large and moreover we have to consider that in *IKRSimLib* we used meters for every connection to measure a large set of single statistics, instead in *ns-2* we used only a monitor on the bottleneck queue to measure total throughput. In *IKRSimLib* we compiled the code with the optimized version and in *ns-2* we did

not use any particular strategies to save simulation time[1] because one of them did not work[2] and the others required to learn particular programming techniques.

6 Experience running simulations

Since the used simulators are implemented in different ways, we encountered some difficulties putting both in the same conditions. We already discussed about some different parameters in TCP implementation, but more than the parameters, the largest difference is that *ns-2* implements TCP algorithm measuring some variables in "packets" (e.g. congestion window, receiver advertised window, slow start threshold, buffer size, ...), whereas in *IKRSimLib* all data variables are expressed in "bytes" and time variables in "ms". Therefore to give the same parameter to both simulators we converted all variables from one unit to another.

Moreover, in *ns-2* every link is composed of a queue, a bit-rate and a propagation delay. Thus if the first bit of the first packet in the queue enters the link at the instant $t = 0sec$, the last bit leaves the queue after a transmission time equal to *packet_size[bit]/bit_rate[bit/s]*. The scheduler assumes the packet arrival event when the last bit arrives at the other side of the link. Therefore from the departure event to the arrival event there is a time equal to *transmission_time+propagation_delay*. In *IKRSimLib* we can consider or not the transmission time (or service time) by using particular queue classes. So, in the *IKRSimLib* simulation model, to include the additive delay present in each link of *ns-2* and not present in *IKRSimLib* non-queued links, we added on the reverse path a delay equal to the sum of packet service times on each link crossed in *ns-2* model. Probably this delay is negligible, because it regards only the ACKnowledgements that have a dimension of 40 Bytes.

The big disadvantage using *ns-2* is that it needs a deep knowledge of two programming languages: OTcl and C++, whereas using *IKRSimLib* we need only a C++ knowledge.

In my opinion the best feature of *IKRSimLib* torwards *ns-2* is the large distribution probability library, which together with *meter* classes allows to make every kind of statistical measurements simply connecting the meter between the measurement points. Making statistical measurements in *ns-2* I encountered many difficults due to some bugs also reported on ns-mailinglist. Moreover, particular statistical measurements in *ns-2* need to post-process a trace file that has a typical dimension of hundreds of MBytes.

The great advantage using *ns-2* regards the very large community that use this simulation tool, due to this you can compare results with many other people and find many solutions on the Web.

7 Conclusions

In this paper we compared TCP implementation in two network simulator tools (*ns-2* and *IKRSimLib*). According to simulation results we realized that the two tools behave in the same way for values of the total packet loss probability lower than p=0.01, whereas for p values greater than 0.01 there is a rather large difference between the two TCP behaviours that increases with packet loss probability. In particular the *ns-2* throughput is always smaller than the *IKRSimLib* throughput, with errors included in a range from 3% (for p=0.01) to 49% (for p=0.1). We tried to explain this with the different implementation of TCP algorithm, because of *ns-2* measures congestion window in packets, whereas *IKRSimLib* in Bytes.

[1] Explained in *ns-2* official web site at the link "Tips and statistical data for running large simulations in ns"

[2] Probably a *ns-2* bug about the OTcl function "remove-all-packet-headers"

A OTcl script used in ns-2 simulations

```
set ns [new Simulator]


set SIM_TIME 1500.0; # in sec

set TCP_AGENT %%NOS

set Bf 10;   # Mbit/s
set Ba 2;        # Mbit/s

set p  %%pLoss
set Buff_Size  %%bs

set delay 100;       # in ms
set h_delay [expr $delay/2]

set ran_del [new RandomVariable/Uniform]
$ran_del set min_ 0.0
$ran_del set max_ 0.100


#---------------------TRACE FILES----------------------------
#set nam_f [open out.nam w]
#$ns namtrace-all $nam_f

#set trace_f [open out.tr w]
#$ns trace-all $trace_f

#set eventfile [open eventfile.tr w]
#$ns eventtrace-all $eventfile
#------------------------------------------------------------------------
#Trace set show_tcphdr_ 1
#------------------------------------------------------------------------
Agent/TCP set packetSize_  1500;  # Byte (1460 MSS + 40 TCP/IP_header)
Agent/TCP set maxcwnd_  32;   # in SEGMENTS
Agent/TCP set window_  43;   # receiver advertised window   (fixed)
Agent/TCP set windowInit_ 2;   # initial/reset value of cwnd_  (default)

Agent/TCP set rfc2988_ false
Agent/TCP set minrto_ 0
Agent/TCP set maxrto_ 10

Agent/TCP set tcpTick_ 0.200

Agent/TCP set rtxcur_init_ 1
Agent/TCP set rttvar_init_ 0
Agent/TCP set srtt_init_ 1

Agent/TCP set ssthresh_ 32;  #initial value

Agent/TCP set syn_ false
#-------------------------------------------------------
#-------------------------------------------------------
#Agent/TCPSink set packetSize_ 40; #default
Agent/TCPSink/DelAck set interval_ 0.200
#-------------------------------------------------------
#--------------------------
Queue set limit_ 900000000000000000000000
#Queue/DropTail set queue_in_bytes_ false
#--------------------------

set entry_node [$ns node]
set exit_node [$ns node]

set fLink_DATA [$ns simplex-link $entry_node $exit_node [set Bf]Mb [set h_delay]ms DropTail]
set fLink_ACK [$ns simplex-link $exit_node $entry_node [set Bf]Mb [set h_delay]ms DropTail]

$ns queue-limit $entry_node $exit_node $Buff_Size

set loss_module [new ErrorModel]
$loss_module set rate_ $p
```

```
$loss_module unit pkt
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]

$ns link-lossmodel $loss_module $entry_node $exit_node


for    {set cont1 0}  {$cont1 < $TCP_AGENT}  {incr cont1}  {

set tcp($cont1) [new Agent/TCP/Reno]

set app($cont1) [new Application/FTP]
$app($cont1) attach-agent $tcp($cont1)

$ns  at   [$ran_del   value]   "$app($cont1) start"
$ns   at    $SIM_TIME    "$app($cont1) stop"

set TCP_node($cont1) [$ns node]

$ns attach-agent $TCP_node($cont1) $tcp($cont1)
$ns    at    $SIM_TIME "$ns detach-agent $TCP_node($cont1) $tcp($cont1)"

set Sender_Access_Link($cont1)
             [$ns duplex-link $entry_node $TCP_node($cont1) [set Ba]Mb 0ms DropTail]


set sink($cont1) [new Agent/TCPSink/DelAck]

set SINK_node($cont1) [$ns node]

$ns attach-agent $SINK_node($cont1) $sink($cont1)
$ns at $SIM_TIME "$ns detach-agent $SINK_node($cont1) $sink($cont1)"

set Receiver_Access_Link($cont1)
              [$ns duplex-link $SINK_node($cont1) $exit_node  1e25Mb 0ms DropTail]

$ns connect $tcp($cont1) $sink($cont1)

}


set queue_fLink [$ns monitor-queue $entry_node $exit_node [$ns get-ns-traceall]]

set integ [$queue_fLink get-pkts-integrator]

proc print {} {
global ns  queue_fLink   SIM_TIME p TCP_AGENT Ba Bf delay h_delay Buff_Size integ

set IntegraleQ [$integ set sum_]
set mean_queue_length [expr $IntegraleQ/$SIM_TIME]

set p1_arr [$queue_fLink set parrivals_]
set p1_dep [$queue_fLink set pdepartures_]
set p1_drops [$queue_fLink set pdrops_]

set b1_arr [$queue_fLink set barrivals_]
set b1_dep [$queue_fLink set bdepartures_]
set b1_drops [$queue_fLink set bdrops_]


set pkts_of [expr $p1_arr - $p1_dep]
set byte_of [expr $b1_arr - $b1_dep]

set pkts_err [expr $p1_drops - $pkts_of]
set byte_err [expr $b1_drops - $byte_of]

set throughput_B_ms [expr [expr $b1_dep - $byte_err]/[expr $SIM_TIME*1000]]


puts "                                  "
puts "                                  "
puts " Number of segments arrived at the Entry node = $p1_arr "
puts " Number of segments leaved the Entry node = $p1_dep "
puts "                                  "
```

```
puts " Number of total segments lost = $p1_drops "
puts " Number of segment lost by queue overflow = $pkts_of "
puts " Number of segments lost on fLink = $pkts_err "
puts "                                  "
puts " Number of Byte arrived at the Entry node  = $b1_arr Byte"
puts " Number of Byte leaved the Entry node = $b1_dep Byte"
puts "                                  "
puts "                                  "
puts "                                  "
puts " Number of Sources = $TCP_AGENT "
puts " Single access bandwidth = $Ba Mbit/s"
puts " Bottleneck bandwidth = $Bf Mbit/s"
puts " Bottleneck Buffer Size = $Buff_Size pkts"
puts " Network delay = $h_delay + $h_delay = $delay ms"
puts " Segment loss probability (p) = $p"
puts "                                  "
puts "                                  "
puts " Mean Bottleneck Queue Length = $mean_queue_length (in packets)"
puts "                                  "
puts " Throughput = $throughput_B_ms Byte/ms"
puts "                                  "
}


$ns at $SIM_TIME "print"
$ns at $SIM_TIME "finish"

proc finish {} {
puts [exec date]



exit 0
}

puts "running ns..."
puts [exec date]

$ns run
```

B Parameter file used in IKRSimLib simulations

```
{

//   SimulationControl  {
//      Batches = 10;
//      BatchCalls = 10000;
//      TransientCalls = 10000;
//   }

     TransientTime = 1;
     BatchTime = 100000;     // 1500 sec simulation
     NoOfBatches = 15;


   SenderSet {
      NoOfSenders = %%NOS;

      GreedyGenerator {
//    BlockSize = 4096;
      OffsetDist { Uniform { LowerBound = 0; UpperBound = 100; }}
      }
      TCPSender {
         Version = "Reno";
         MSS = 1460;
```

```
         TCPIPHeaderLength = 40;
         InitialCWnd = 2920;             // in Byte (2 seg)
         InitialSSThresh = 46720;     // in Byte (32 seg)
         InitialRTT = 1000;               // in msec (1 sec)
         MaxCWnd = 46720;            // in Byte (32 seg)
         DupACKThreshold = 3;
         TimerGranularity = 200;      // in msec  (0.200 sec)
         SWSATimerValue = 5000;
         MaxRTO = 10000;              //in msec  (10 sec)
         MaxBackoff = 64;                // ... in ns setted to 64   (original 256)
         Nagle = false;
         GoBackN = true;
         Karn = true;
         FlightSizeRecovery = false;
         RenoConservation = true;
         ImmediateBackoffReset = false;
//       Debug;
      }
      NetworkAccess {
 BufferSize = -1;
 ServiceRate = 250;   // 250 Byte/ms -> (2 Mbit/s); 625 Byte/ms -> (5Mbit/s);
      }
   }


   ReceiverSet {
      NoOfReceivers = %%NOS;

      TCPReceiver {
         MSS = 1460;
         TCPIPHeaderLength = 40;
         BufferSize = 65636;
         DelayedACK = true;
         ACKDelayTime = 200;
         SendPeriodicACKs = false;
         PeriodicACKInterval = 1000;
         ACKSchedulingDelay = 0;
         MinUserBlockSize = 1;
         MaxUserBlockSize = 65636;
         UserBlockProcessingDelay = 0;
//       Debug;
      }
//       NetworkAccess
//       {
//       ServiceRate = 1.25e4;   // 100 Mbit/s
//       }

//     PropagationDelay = 100;                          // [in ms]
   }



//   Link {
//      BufferSize = -1;           // original 66 packets (-> 100000:1500)
//     ServiceRate = 1250;         // -> 10 Mbit/s
//   }


     SimplexDelayLink_DATA {
               QueueServer {
                       BufferSize = %%bs;
                       ServiceRate = 1250;            //  10Mbit/s
                       }
                       Delay = 50;
     }

     SimplexDelayLink_ACK {
               QueueServer {
                       BufferSize = -1;
                       ServiceRate = 1250;
                       }
                       Delay = 50+40/1250+40/250;
     }
```

```
   Dropper {
          DropProb = %%pLoss;
   }

   PacketDelayMeter {
      ArraySize = 100;
      LowerBound = 0;
      UpperBound = 100;
   }
}
```

References

[1] *The ns Manual*, http://www.isi.edu/nsnam/ns/

[2] Chadi Barakat, *Network Simulator "ns"*, INRIA Sophia Antipolis, France; PLANETE research group.

[3] Sven Ubik, Jan Klaban, *Experience with using simulation for congestion control research*, CESNET Technical Report 26/2003.

[4] Mark Allman, Aaron Falk, *On the Effective Evaluation of TCP*, ACM Computer Communication Review, October 1999.

[5] Stefan Bodamer, Klaus Dolzer, Christoph Gauger, *IND Simulation Library 2.3 - User Guide - Part I: Introduction*, University of Stuttgart.