

**CESNET Technical Report 26/2003**  
Experience with using simulations for  
congestion control research

**Sven Ubik, [ubik@cesnet.cz](mailto:ubik@cesnet.cz)**  
**Jan Klaban, [xklaban@quick.cz](mailto:xklaban@quick.cz)**

December 5, 2003

**Abstract**

As part of the CESNET End-to-end performance project [1] we investigate enhancement of congestion control mechanisms providing better performance on fast long-distance networks. Network simulations and emulations are techniques that enable us to compare different alternatives and evaluate their performance under varying and well-defined network conditions, which would not be possible on real networks with unpredictable traffic dynamics.

In this report we summarize our experience with using well known ns2 network simulator for studying congestion control mechanisms. We discuss difficulties that we encountered and describe our own enhancements to ns2, enabling to study more alternative and obtain more detailed information about their behaviour.

## 1 What is network simulation and how can it be useful?

Different researchers use somewhat different definitions of simulation depending on the way they conceive it and use it. In the context of computer networks, we can say that *simulation* imitates behaviour of a real world or hypothetical future system by a computer program for the purpose of study.

Properties of simulation include the following:

- Simulation creates a model of a system, which is used to explain system behavior and to see how the system performs under varying conditions to design the system with desirable performance characteristics
- Simulation usually represents the system on a smaller scale for easier study when compared to the full scale
- Simulation allows to study a system in well-defined and well-known conditions, repeatability is necessary in order to understand events

- In discrete simulations variables change at discrete time intervals between discrete values, in continuous simulations variables change anytime between any values, computer networks are by nature discrete and discrete simulations are used to represent them

Some people find network simulations (and to lesser extent also emulations) useful to conduct experiments on models of large-scale networks with many nodes and links, which could not be conducted on real large-scale networks. Study of routing protocol behaviour is an example of such experiments.

Other researchers use simulations to explore individual components of a computer networks, such as individual OSI-model layers. It includes study of properties of physical-layer transmission media and devices, such as optical amplifiers. Another example is study of behaviour of transport-layer congestion control mechanisms under varying conditions. The last example is subject of our report, contributed as part of the CESNET End-to-end performance project [1].

## 2 Ns2 network simulator

ns2 is a freely available discrete-event object-oriented network simulator, which provides a framework for building a network model, specifying data input, analysing data output and presenting results.

Source code is also available, enabling users to add new features to the simulator, such as support for new communication protocols, which can be tested by simulations.

ns2 has been enhanced and used by many researchers over a couple of years. This made it a standard trusted simulator. If you use ns2, everybody will take for granted that your simulations were working right. If you use another simulator, someone will probably ask why you did not use ns2 and whether your simulations were correct. ns2 trustworthiness may sometimes seem overrated. Installation is not as straightforward as "configure, make, make install" as we got from the GNU world and behaviour of the simulated TCP is not exactly the same as of the TCP implementations in real operating systems (which are however also not all exactly the same).

In real networks, four components compose end-to-end delay which a packet can experience. ns2 simulates all these delay components except for the processing delay:

- Processing delay - time needed to process a packet in network nodes
- Transmission delay - time needed to put the packet on network links
- Propagation delay - time needed for the energy representing a single bit to propagate along network links, bounded with the speed of light
- Queueing delay - time that the packet waits in queues in network nodes to be served

## 3 Installation and simulation scripts

ns2 is implemented in C++ and Tcl and should run on any Posix-like operating system (it was tested on FreeBSD, Linux, SunOS and Solaris) and on Microsoft Windows. ns2 uses several

other software packages (Tcl/Tk, xgraph and others), which can be installed either separately or together with ns2 from the 'ns-allinone' package. Some of these packages are mandatory, while others are optional, such as nam-l for animations of simulation runs.

Once ns2 is installed, a simulation task is specified by a simulation script written in Tcl. This script describes the network topology (nodes and their interconnection) communications protocols (e.g., TCP) and events (scheduling of data streams to be sent). You can also specify lengths of packet queues attached to links and the maximum size of TCP window. Writing simulation scripts is a complex task which requires understanding of ns2 object classes and Tcl programming. The definitive reference for writing simulation scripts is the ns manual [2].

## 4 TCP in ns2

There are two flavors of TCP in ns2. The first is a one-way TCP, which uses objects of different classes on the sender and receiver side. For the sender side, several classes are available for TCP Tahoe, Reno, Newreno, Vegas and Sack or Fack, supporting selective acknowledgements. For the receiver side, three classes are available for TCP receiver without delayed acknowledgements, with delayed acknowledgements and with selective acknowledgements. Subclasses can be derived from these supplied classes to implement modifications to standard TCP congestion control. The second flavor is a two-way TCP, which uses objects of the same class on both the sender side and the receiver side. The one-way TCP is used more frequently than the two-way TCP, which implements only Reno congestion control and is considered under development. Our observations about one-way TCP are described in more detail in [3].

## 5 Example of simulation with ns2

One of the network topologies frequently used in simulations is shown in Fig. 1. Hosts connected to router R1 send data to hosts connected to router R2. The sum of data rates produced by source hosts is usually bigger than throughput of the link between router R1 and router R2, making it a bottleneck link. This link has also a specified packet loss rate and one-way delay. Links between hosts and routers are lossless and have fixed one-way delay and throughput.

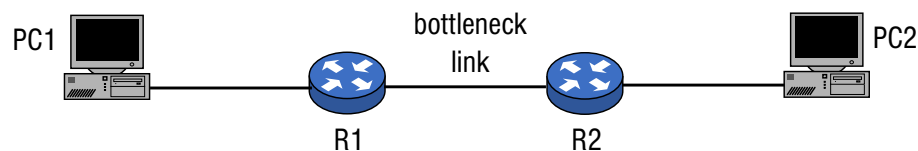


Figure 1: Simulation topology

We will use the following example simulation script corresponding to Fig. 1 to illustrate different tasks to be specified:

```
# 1. Create an object of ns2 simulator
set ns [new Simulator]
```

```

$ns color 0 Red
$ns color 1 Blue

proc finish {} {
    exit 0
}

# 2. Create objects for network nodes, links and queues attached to links
# and specify their parameters, thus creating the network topology
set pc1 [$ns node]
set pc2 [$ns node]
set r1 [$ns node]
set r2 [$ns node]
set em [new ErrorModel]

# Set link characteristics
$ns duplex-link $pc1 $r1 90Mb 20ms DropTail
$ns duplex-link $r1 $r2 50M 100ms DropTail
$ns duplex-link $r2 $pc2 90Mb 20ms DropTail

$ns queue-limit $pc1 $r1 6000000
$ns queue-limit $r1 $r2 300000

$ns duplex-link-op $pc1 $r1 orient right
$ns duplex-link-op $r1 $r2 orient right
$ns duplex-link-op $r2 $pc2 orient right

$em unit pkt
$em ranvar [new RandomVariable/Uniform]
$em set rate_ 0.0001
set streams 5
set segsize 1500

for {set i 0} {$i < $streams} {incr i} {
# 3. Create objects for TCP sender and TCP receiver and specify maximum
# window size
set tcpz($i) [new Agent/TCP/Reno]
set tcpc($i) [new Agent/TCPSink]
$ns attach-agent $pc1 $tcpz($i)
$ns attach-agent $pc2 $tcpc($i)
$tcpz($i) set fid_ 0
$tcpc($i) set fid_ 1
$ns connect $tcpz($i) $tcpc($i)
$tcpc($i) listen
$tcpz($i) set window_ 500

```

```

    $tcpz($i) set segsize_ $segsize
# $tcpz($i) set maxburst_ 100000
# $tcpz($i) set tcpTick_ 0.000004
# 4. Create objects for sending application and receiving application
#    and attach them to objects for TCP sender and TCP receiver, respectively
    set snd($i) [new Application/FTP]
    set rcv($i) [new Application/TCPCNT]
    $snd($i) attach-agent $tcpz($i)
    $rcv($i) attach-agent $tcpz($i)
}

set null [new Agent/Null]
$em drop-target $null
$ns lossmodel $em $r1 $r2

# 5. Schedule events, such as start and end times of data streams and
#    when the simulation should stop
for {set i 0} {$i < $streams} {incr i} {
    $ns at 0 "$snd($i) start"
}
$ns at 0 "$rcv(0) settimer 0.1"
$ns at 0 "$tcpz(0) settimer 0.1"

for {set i 0} {$i < $streams} {incr i} {
    $ns at $TIME "$snd($i) stop"
}

$ns at $TIME "$rcv(0) stop"
$ns at $TIME "finish"

# 6. Start simulation
$ns run

```

We need to do the following steps (numbers refer to corresponding parts of the example simulation script above):

1. Create an object of the ns2 simulator
2. Create objects for network nodes, links and queues attached to links and specify their parameters, thus creating the network topology
3. Create objects for TCP sender and TCP receiver and specify maximum window size
4. Create objects for sending application and receiving application and attach them to objects for TCP sender and TCP receiver, respectively

5. Schedule events, such as start and end times of data streams and when the simulation should stop
6. Start simulation

## 6 Synchronous log of TCP connection dynamics

The TCP sender can produce a log of connection variables such as `rtt`, `cwnd` or `ssthresh`. When enabled, a line is added to this log whenever some of the logged variables changes. That is why we call it a synchronous log. Later in the description of enhancements to ns2 that we implemented we will describe an asynchronous log.

The synchronous log is enabled by the following commands in the simulation script:

```
set tcptrace [open tcptrace w]

set tcpz [new Agent/TCP/Reno]
$ns add-agent-trace tcpz trname $tcptrace
$ns monitor-agent-trace $tcpz
$tcpz tracevar rtt\_
$tcpz tracevar cwnd\_

proc finish {} {
    global ns tcptrace
    $ns flush-trace
    close $tcptrace
    exit 0
}
```

Each line of the synchronous log looks as follows. The important fields are the first one, which is the simulation time when a connection variable changed and the last two, which are variable name and its new value:

```
0.64017 0 0 1 0 rtt_ 0.640
0.64017 0 0 1 0 cwnd_ 46.000
. . .
```

`cwnd` is shown in MSS packets. Internally however it is maintained in double variable and it is incremented by  $1/cwnd$  after receiving each acknowledgement in the congestion avoidance phase. This is different from the Linux TCP implementation, which maintains `cwnd` in integer variable and increases it by the whole MSS packets.

In fact, `cwnd` can be adjusted by ns2 using different algorithms, which can be selected by `wnd_option_` instance variable of the TCP sender object, such as of the `Agent/TCP/Reno` class. There are eight possible values, default value is 1. Other values are used for experimental purposes.

## 7 Memory requirements

The volume of memory that ns2 requires for a simulation depends on the number of packets that are in the simulated network and the number of packet headers maintained for each packet. In fast long-distance networks, which are often subject of current research in congestion control, the number of packets in the network can be at the order of tens or hundreds of thousands and the volume of memory required can grow to several gigabytes. We can decrease memory requirements by first removing all packet headers and then adding only required headers. For example, the following commands can be added at the beginning of a simulation script:

```
remove-all-packet-headers
add-packet-header TCP IP
```

## 8 Scripts for batch processing

For evaluation of congestion control mechanisms under various network conditions, we often need to run a set of simulations of certain network topology where network characteristics of the bottleneck line are varying. These include the link bandwidth, packet loss rate and one-way delay. We also want to experiment with different MSS, number of parallel streams, as well as different test duration and time granularity for computing resulting characteristics, such as achieved throughput.

We added logging of TCP connection characteristics and created a set of scripts to simplify use of ns2 for simulation of common experimental scenarios with various link characteristics and protocol parameters. Description of these scripts is described in [3].

## 9 Throughput measurement

We added a new class `Application/TCPCNT` to compute throughput at the application level and modified class `Agent/TCPSink` to compute throughput at the TCP level. Description of these enhancements can be found in [3].

## 10 Reaction to increase or decrease of available bandwidth

In order to study reaction of a congestion control mechanism on increase or decrease of available bandwidth, we created a sender side application class `Application/TCPFTP`, which generates periodic bursts of packets. To start the application use the following commands in the simulation script:

```
set snd [Application/TCPFTP]
$snd set interval_ n
$snd set burstsize_ m
$snd start
```

where  $n$  is the period in seconds and  $m$  is the number of MSS packets to be sent in each period. The application must be attached to the TCP sender. See example simulation scripts. To stop the application, use the following command in the simulation script:

```
$snd stop
```

## 11 Changing AIMD speed

The speed of slow start and congestion avoidance in original ns2 are fixed, the latter is based on AIMD(1, 0.5). For experiments with recent proposals of Fast TCP we need to change AIMD parameters. We modified certain ns2 classes to allow to specify speed of both slow start and congestion avoidance. Description of these enhancements can be found in [3].

## 12 Asynchronous log of connection dynamics

We already described that ns2 can produce synchronous log of connection dynamics. Sometimes we do not need a detailed log of every change of specified connection variables. An asynchronous log which samples values of connection variables periodically at a specified rate can be sufficient and easier to read. We modified certain ns2 classes to produce an asynchronous log of connection dynamics. Description of this enhancement can be found in [3].

## 13 Difficulties we ran into

We encountered several symptoms of unexpected behaviour and ran into some problems when using ns2 including the following:

- TCP sometimes stops sending data for a few seconds
- Slow start occurs in TCP Reno after router queue overflow
- Throughput diagram shows fluctuations in fine-timescale
- Non-numeric artefacts appear in simulation log

We document these effects and describe explanations for some of them in [3].

## 14 Conclusion

We presented our experience with using computer network simulator ns2 for research in congestion control mechanisms. We found some differences in ns2 TCP implementation when compared to TCP specified in RFC documents. We described our enhancements for detailed monitoring of TCP dynamic behaviour, extensions for modified AIMD mechanism and batch processing as well as difficulties we ran into.

## References

- [1] *End-to-end performance project*, CESNET, <http://www.cesnet.cz/english/project/qosip>.
- [2] *The ns Manual*, <http://www.isi.edu/nsnam/ns/ns-documentation.html>.
- [3] Sven Ubik, Jan Klaban. *Experience with simulations for congestion control research - Progress report*, End-to-end performance project, CESNET, 2003.