NS2: Contents

- NS2 – Introduction to NS2 simulator
- Some NS2 examples
  - RED example
  - Enhanced RED example
- NS2 project work instructions

Example 1

- Task
  - simulate a queue operating under RED control
    - using elementary topology
    - traffic is a superposition of greedy TCP sources
  - measure instantaneous queue length
  - file: redtcp.tcl

Running the tcl-scripts (1)

- Ns2 is not installed in the machines of B215
- Ns2 can be found on the Linux machines in Maarintalo (maintained by Computing Center)
  - rooms Maari A and Maari C
- Take a remote connection to one of the Linux machines in Maari A/C
  - E.g., listing of the machines in Maari A can be found from

    http://www.hut.fi/atk/luokat/Maari-A.html
- Save the example ns2/tcl files from course homepage in your directory
  - Example 1: redtcp.tcl
  - Example 2: redtcpmain.tcl and redtcpsub.tcl

Running the tcl-scripts (2)

- In order to be able to use ns2, you first have to do the following
  - Type in your shell

    ```
    source /p/edu/s-38.180/usens2.csh
    ```

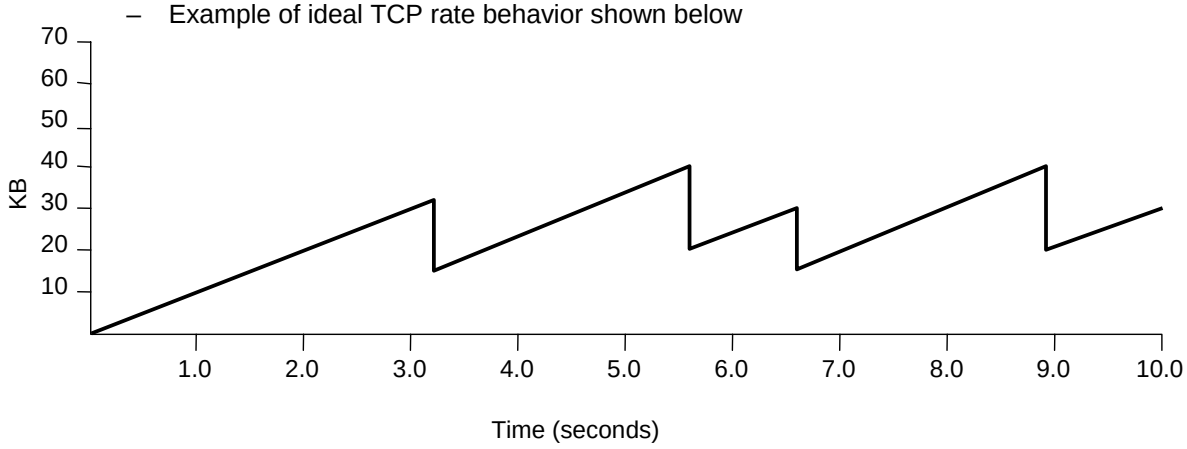
  - This file contains the required settings for environment variables
  - Give this command each time you start an ns2 session in a shell
- After that you can use ns2 simply by writing in your shell

  ```
  ns my_script.tcl
  ```

- Top directory where ns2 source files are is:
  - /p/edu/s-38.180/src/ns-2.1b9a_standard/

TCP

- Provides reliable file transfer over Internet
- Includes functionality for congestion control
  - contains many sophisticated algorithms for realizing congestion control
  - Basic idea: increase rate slowly, but decrease quickly when facing congestion
  - Congestion detected from packet losses (i.e., TCP only reacts to losses)
  - Example of ideal TCP rate behavior shown below



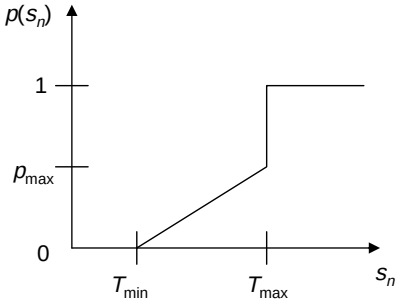
RED

- RED (Random Early Detection/Drop)
  - Active Queue Management (AQM) method proposed by S. Floyd
  - designed to cooperate with TCP-friendly congestion control
  - tries to prevent buffer overflows by discarding packets prior to the buffer becoming full
    - TCP friendly rate control reacts to packet losses and (some) sources slow down their sending rates ⇒ serious congestion is avoided
    - packet dropping probability depends on load
- RED algorithm (approximately)
  - for each arriving packet, compute exponentially averaged queue length (≈ load), s_n

$$s_n = (1-\beta)s_{n-1} + \beta q_n$$

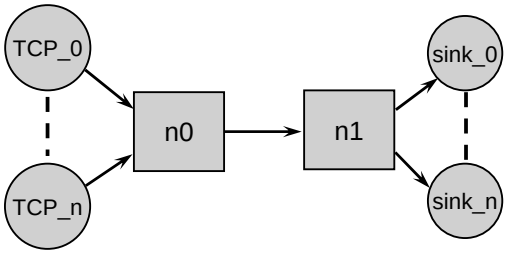
  - drop packet with probability

$$p_n = \begin{cases} 0, & s_n < T_{\min} \\ \dfrac{p_{\max}(s_n - T_{\min})}{T_{\max} - T_{\min}}, & T_{\min} \le s_n \le T_{\max} \\ 1, & s_n > T_{\max} \end{cases}$$



Simulator objects for example 1

- Traffic sources
  - greedy TCP Reno sources (constantly sends traffic)
  - need to create a TCP connection and attach an FTP agent to the TCP source
  - parameters
    - nof of sources
    - maximum window size
    - segment size
- Bottleneck link
  - finite buffer with RED queue
  - parameters
    - queue size
    - RED parameters
- Required traffic objects (topology)
  - 2 nodes
  - 1 link (with RED queue)
  - N TCP sources (source/sink)



Tracing for example 1

- Aim:
  - trace the variable that represents instantaneous queue length
- See red.h (in /ns-allinone-2.1b9a/ns-2.1b9a/)
  - variables with type TracedInt (or TracedDouble, etc.) are variables defined in the C++ class that are also visible at the OTcl level
  - to find out what traced variables are defined for RED queue (run in above directory)

```
fgrep Traced red.h
```

  - variable “TracedInt cur_” represents instantaneous queue length (as seen by an arriving packet)
- Creating a trace object
  - create file for output and create a trace object ($redobj represents a RED queue object)

```
set outfile [open data.txt w]
$redobj trace cur_
$redobj attach $outfile
```

  - tracing started after warm-up time

Plotting the queue length process

- One can plot the realization of the queue length process
  - can experiment with RED parameters to examine stability, e.g., play with the averaging parameter `q_weight_` and `linterm_`
  - example shows scenario where number of flows changes over time
- The output file contains rows with following entries:

```
Q 20.0041 11
Q 20.0151 9
Q 20.0225 10 ...
```

- Remove extra 'Q' from beginning of each line by:
  - from command line: `awk '{print $2, $3}' qlen.dat > qq1.dat`
  - from ns2-tcl script: `exec awk {{print $2, $3}} qlen.dat > qq1.dat`
- Plot the data in qq1.dat using xgraph
  - from command line: `xgraph qq1.dat`
  - from n2-tcl script: `exec xgraph qq1.dat`

NS2: Contents

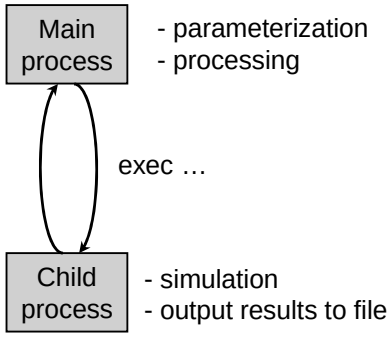
- NS2 – Introduction to NS2 simulator
- Some NS2 examples
  - RED example
  - Enhanced RED example
- NS2 project work instructions

Example 2

- Task
  - measure the steady state mean of the instantaneous queue length as a function of offered load
- We need to …
  - be able to run consecutive independent simulations, and
  - compute steady state mean from the measured data
  - files: redtcpmain.tcl and redtcpsub.tcl

Running independent simulations

- To run independent simulations, we must either be able to …
  - re-initialize all simulator objects (simulation clock, event scheduler, all traffic objects, etc.), or somehow re-execute simulation scripts
  - re-initializing in ns2 is difficult ⇒ we need to make repeated executions of ns2 scripts
- In Unix, the operating system executes each Tcl/ns2 script as a process
- For repeated simulations we need …
  - a main program that controls and parametrizes simulations, and a sub-program that executes each simulation run
  - in unix terminology, we need a main process that spawns a child process for execution of the actual simulation runs
  - in Tcl (and most script languages), the command for this is ”exec …”




```
exec ns myscript.tcl command_line_args
```

Measuring time averages

- Post-processing
  - read data from output file and process it
  - in our case, output consists of tuples <Q, time, queue_len>
- Reading from file
  - open file for reading and read a line from the file

```
set $outfile [open data.txt r]
gets $outfile tmp
```

- To compute time average of the data, use Integrator class
  - creating and adding points

```
set integ [new Integrator]
$integ newpoint $time $value
```

  - variable `sum_` contains cumulative sum