SOFTWARE PACKAGE FOR THE ACTIVE QUEUE MANAGEMENT MODULE MODEL VERIFICATION

Tatyana R. Velieva, Anna V. Korolkova, Migran N. Gevorkyan, Sergey A. Vasilyev
Department of Applied Probability and Informatics,
Peoples' Friendship University of Russia (RUDN University),
6 Miklukho-Maklaya St, Moscow, 117198, Russian Federation
Email: velieva_tr@rudn.university, korolkova_av@rudn.university,
gevorkyan_mn@rudn.university, vasilyev_sa@rudn.university

Ivan S. Zaryadov
Department of Applied Probability and Informatics,
Peoples' Friendship University of Russia
(RUDN University),
6 Miklukho-Maklaya St, Moscow,
117198, Russian Federation
and Institute of Informatics Problems,
FRC CSC RAS, IPI FRC CSC RAS,
44-2 Vavilova str., Moscow, 119333, Russia
Email: zaryadov_is@rudn.university

Dmitry S. Kulyabov
Department of Applied Probability and Informatics,
Peoples' Friendship University of Russia
(RUDN University),
6 Miklukho-Maklaya St, Moscow,
117198, Russian Federation
and Laboratory of Information Technologies
Joint Institute for Nuclear Research
Joliot-Curie 6, Dubna, Moscow region, 141980, Russia
Email: kulyabov_ds@rudn.university

KEYWORDS

Active queue management, simulation, NS2, Julia, self-oscillating

ABSTRACT

Self-oscillation modes in control systems of data transmission networks negatively affect the characteristics of these networks. To investigate the self-oscillation mode for systems with a control module, the analytical model of the active queue management module was developed. The problem of verification of the obtained theoretical results arises in the study. Previously, a software system was developed for software emulation of the router. However, its use has caused some difficulties. Alternatively, the simulation model of network with active queue management module was developed. The paper describes a software package for verifying theoretical calculations based on the NS-2 simulation system. For illustration, a numerical example is given.

INTRODUCTION

When modeling technical systems, there is often a question of verification of results. Either there is no access to data on the functioning of such systems, or the acquisition of data is associated with large resource and time costs. But some simulation experiments can be seen as a solution to this problem.

The problem of the occurrence of self-oscillatory regime in systems with control is considered (see Lautenschlaeger and Francini (2015)). In particular, the active queue management modules with RED-like algorithms were studied. Based on the theoretical model (see Misra et al. (1999); Kulyabov et al. (2018)) of the functioning of the RED module, the parameters of self-oscillating regimes were investigated. However, for completeness of the study, the verification of the results is necessary. We have developed the installation for a full-scale experiment on the basis of emulation of images of network equipment (see Velieva et al. (2015)). However, this approach required some extra resources, which were not at authors disposal. So, as a substitute, it was suggested to use the simulation model.

In this paper, the simulation model based on the NS-2 network protocol simulation tool is described. This approach proved to be more flexible in comparison with the full-scale experiment on the basis of emulation of network operating systems.

RED ADAPTIVE CONGESTION CONTROL MECHANISM

The RED algorithm (see Adams (2013); Kushwaha and Shwer (2013); Kushwaha and Gupta (2014)) uses a weighted queue length as a factor determining the probability of packets drop. As the average queue length grows, the probability of

Proceedings 32nd European Conference on Modelling and Simulation ©ECMS Lars Nolle, Alexandra Burger, Christoph Tholen, Jens Werner, Jens Wellhausen (Editors)
ISBN: 978-0-9932440-6-3/ ISBN: 978-0-9932440-7-0 (CD)

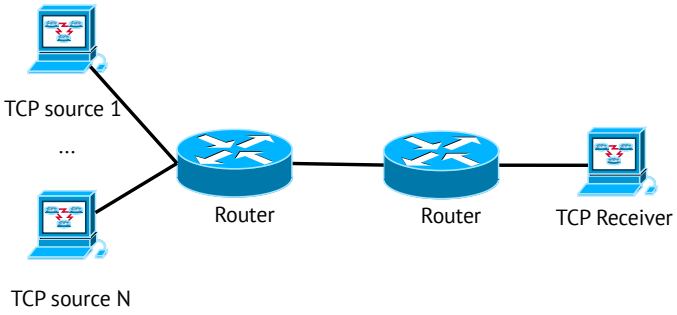

Fig. 1. Dumbbell topology

packets drop also increases. The algorithm uses two threshold values of the average queue length to control the drop function.

$$p(\hat{Q}) = \begin{cases} 0, & 0 < \hat{Q} \leqslant Q_{\min}, \\ \dfrac{\hat{Q} - Q_{\min}}{Q_{\max} - Q_{\min}} p_{\max}, & Q_{\min} < \hat{Q} \leqslant Q_{\max}, \\ 1, & \hat{Q} > Q_{\max}. \end{cases}$$

Here $p(\hat{Q})$ is the packets drop function (drop probability), $\hat{Q}$ is the exponentially-weighted moving average of the queue size average, $Q_{\min}$ and $Q_{\max}$ are the thresholds for the weighted average of the queue length, $p_{\max}$ is the the maximum level of the packets drop.

The RED algorithm is quite effective due to simplicity of its implementation in the network hardware, but it has a number of drawbacks. In particular, for some parameters values there is a steady oscillatory mode in the system, which negatively affects the quality of service (QoS) indicators (see Jenkins (2013); Ren et al. (2005); Lautenschlaeger and Francini (2015)). Unfortunately there are no clear selection criteria for RED parameters values, at which the system does not enter in self-oscillating mode.

SIMULATION MODEL

The full-scale experiment often involves certain difficulties. For example, the real equipment is not always available. Also the use of a virtual stand is associated with high demands on computer equipment (see Velieva et al. (2015)). In addition, since the simulation takes place in real time, the whole process is extremely long.

To save resources and time, simulation tools are usually used. The package ns2 (see Issariyakul and Hossain (2012); Altman and Jiménez (2012)) is a tool for network protocols simulating. This package was created as a reference modeling tool, so it is often used as an alternative to the full-scale experiment.

For an imitation experiment, we will use the so-called dumbbell topology (see Fig. 1). Additional TCP sessions are emulated by addition of extra sources.

The program for ns2 is written in TCL language (see Welch and Jones (2003); Nadkarni (2017)).

First, we need to create a simulator object. Let's set the experiment time.

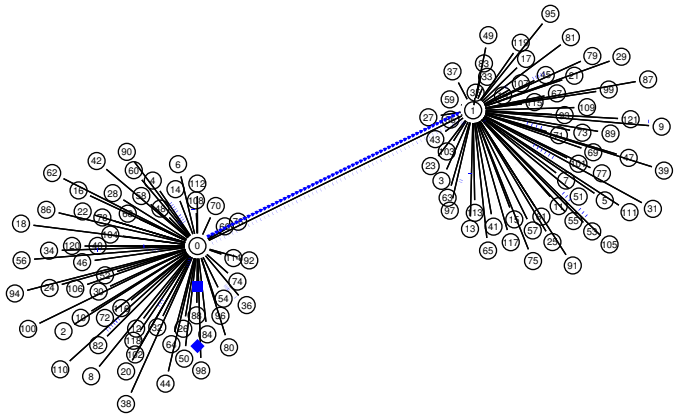

Fig. 2. Visualization of the simulation. Packets drop is shown

```
set ns [new Simulator]

set sduration 20
set simTime 20
```

We may write the data for the nam visualization tool (see Fig. 2). In the final version of the script, we will disable this feature to save resources.

```
# set nf [open out.nam w]
# $ns namtrace-all $nf
```

Let's set the number of TCP sessions (sources).

```
set numSrc 60
```

The current experiment is connected with the study of thresholds values influence on the occurrence of self-oscillation mode. Therefore, the threshold values are set as arguments.

```
if { $argc == 2 } {
  puts "[lindex $argv 0] [lindex $argv 1]"
  # thresh_ <=> Q_min
  # maxthresh_ <=> Q_max
  Queue/RED set thresh_ [lindex $argv 0]
  Queue/RED set maxthresh_ [lindex $argv 1]
}
```

In ns2, there are implementations of three varieties of the RED discipline: the original RED Floyd and Jacobson (1993) algorithm, the ARED algorithm, and the Gentle RED algorithm. The settings of the algorithms are controlled by parameters.

- `bytes_`: turns on (`true`) or turns off (`false`) the mode *byte mode*, in which the size of the package affects the probability of their tagging to drop;
- `Queue-in-bytes_`: if the value of the parameter is set to `true`, then the average queue length will be measured in bits. Also, the `thresh_` and `maxthres_` will be measured by the calculated average packet size (`mean_pktsize_`). The default value is `false`;
- `thres_`: the minimum queue length threshold $q_{\min}$;

- `maxthres_`: the maximum queue length threshold $q_{\max}$;
- `mean_pktsize_`: an approximate estimation of the packet size in bytes. The default value is 500;
- `q_weight_`: the w_q weighting factor is used in calculating of the average queue length;
- `wate_`: this option allows to maintain the interval between drops packets if its valueis set as a `true`;
- `linterm_`: the inverse of the parameter $p_{\max}$. The default is 10;
- `setbit_`: takes the value `false`, if RED discards marked packets. If the value is set as `|true|`, a congestion bit is added to the marked packets;
- `drop-tail_`: when the value is true and the buffer is overflowed then the active queue management algorithm switches to the Drop Tail algorithm.

The default values for the parameters `q_weight_`, `maxthresh_` and `thres_` are 0.002, 15 and 5 respectively.

```
Queue/RED set q_weight_ 0.002
# Queue/RED set drop_tail_ true
Queue/RED set setbit_ false
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set gentle_ false
Queue/RED set mean_pktsize_ 1000
Queue/RED set cur_max_p_ 0.1
```

Two nodes that will play the role of routers are created.

```
set R1 [$ns node]
set R2 [$ns node]
```

Using a loop, we create the nodes that will simulate a TCP session.

```
# create the tcp/ftp src nodes
for {set i 1} {$i<=$numSrc} { incr i } {
    # Create node
    set n($i) [$ns node]
    # Create link
    $ns duplex-link $n($i) $R1 100Mb 20ms DropTail
    # Create TCP agent on node n($i)
    set tcp($i) [new Agent/TCP/Reno]

    $tcp($i) set window_ 32
    $tcp($i) set fid_ 2
    $tcp($i) set paketSize_ 1000
    $tcp($i) set class_ 1

    $ns attach-agent $n($i) $tcp($i)
    # FTP
    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp($i)
    $ftp($i) set type_ FTP

    # Create sink
    set s($i) [$ns node]
    # Create link
    $ns duplex-link $s($i) $R2 100Mb 20ms DropTail
    # Create sink agent on node s($i)
    set sink($i) [new Agent/TCPSink]
    $ns attach-agent $s($i) $sink($i)

    # Connect n($i) and s($i)
    $ns connect $tcp($i) $sink($i)

}
```

The queue is connected to the link between the routers. Since we are only interested in traffic in the forward direction, the discipline Drop Tail is set to the link in the opposite direction.

```
set flink [$ns simplex-link $R1 $R2 15Mb 35ms RED]
$ns simplex-link $R2 $R1 15Mb 35ms DropTail
$ns queue-limit $R1 $R2 300
```

One of the most important objects in ns2 is the queue monitor. It allows to gather information not only about the length of the queue, but also about arriving, departing and dropped packets.

```
set qmon [$ns monitor-queue $R1 $R2 [open qm.tr ↩
    w] 0.01]
[$ns link $R1 $R2] queue-sample-timeout
```

To monitor the parameters of the RED queue (for example, between the nodes $n2$ and $n3$), the following lines of code should be added:

```
set redq [[$ns link $n2 $n3] queue]
set traceq [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq
```

Here `curq_`is the current size of the queue, `ave_` is the average queue size. As a result, the output file consisting of three columns is obtained. The first column contains the flag `Q` (the current queue size) or `a` (average queue size). The other columns are the time and value of the observed parameter.

The size of the windowWe will be under control.

```
# Obtaining TCP CWND Window information

# plotWindow(tcpSource file k): Write CWND of k ↩
    tcpSources in file
# The output format is as follows:
# TIME Win_flow1 Win_flow2 Win_flow3 ... ↩
    Win_flowN
proc plotWindow {tcpSource file k} {
  global ns numSrc

  set time 0.03
  set now [$ns now]
  set cwnd [$tcpSource set cwnd_]
```

```
  if {$k == 1} {
    puts -nonewline $file "$now \t $cwnd \t"
  } else {
    if {$k < $numSrc } {
      puts -nonewline $file "$cwnd \t"
    }
  }

  if { $k == $numSrc } {
    puts -nonewline $file "$cwnd \n"
  }
  if { $k == $numSrc } {
    puts -nonewline $file "$cwnd \n"
  }
  $ns at [expr $now+$time] "plotWindow    ↩
      $tcpSource $file $k"
}

# Start plotWindow() for all tcp sources
# Output to stdout
for {set j 1} {$j<=$numSrc} { incr j } {
    $ns at 0.1 "plotWindow $tcp($j) stdout $j"
}
```

The process of simulation is started:

```
proc finish {} {
    exit 0
}

for {set i 1} {$i<=$numSrc} {incr i} {
    $ns at 0.0 "$ftp($i) start"
    $ns at $simTime "$ftp($i) stop"
    # $ns at $simTime "calc_throughput $tcpsrc($j)    ↩
      $j $simTime"
}

$ns at $simTime "finish"
$ns run
```

If the file with the model is named as `red.tcl`. Then, in order to run the simulation, the `ns red.tcl` command should be executed.

PROCESSING OF THE SIMULATION RESULTS

After the simulation experiment a large amount of raw data is obtained and it is necessary to process this data.

By using the output data the parameters of self-oscillations may be obtained. Here are the fragments of the program in the Julia language (see Joshi and Lakhanpal (2017)), in which the spectral portrait of the self-oscillatory mode is constructed on the basis of the Fast Fourier Transform algorithm (see Rao et al. (2010)).

The file being processed is passed as an argument.

```
#!/usr/bin/env julia
if length(ARGS) > 0
  try
    global const window_size = readcsv(ARGS[1])[:, 2]
    global const count = readcsv(ARGS[1])[:, 1]
  catch err
    if isa(err, SystemError)
      error("File $(ARGS[1]) not found")
    else
      throw(err)
    end
  end
end
```

Actually, the spectral analysis is carried out on the basis of a Fast Fourier Transform algorithm.

```
delta = count[2] .- count[1]
Fd = 1.0 ./ delta
fftCount = length(window_size)
X = fft(window_size[1:512])
amplitude_spectrum = 2 .* abs.(X) / 512.0
amplitude_spectrum[1] = amplitude_spectrum[1] ./ 2.0
frequency = collect(0:Fd./512.0:Fd./2 - 1.0./512.0)
```

As a result the amplitude and frequency of the first harmonic of self-oscillations are obtained.

```
max_amp=findmax(amplitude_spectrum,1)
println(max_amp[1][1],",",frequency[max_amp[2]][1])
```

In addition, we may derive the point values of the spectrum in order to build a graph on it later.

```
for (f, A) in zip(frequency, amplitude_spectrum)
  println(f, ",", A)
end
```

As in the current experiment the dependence of self-oscillations on the threshold values of the RED algorithm is investigated, we will generate files with different threshold values. This data will be used in carrying out the simulation.

```
#!/usr/bin/env python3
import itertools
import numpy as np

DIR = 'parameters'
Q_START = 10
Q_STOP = 90
Q_DELTA = 1

Q = np.arange(Q_START, Q_STOP+1, Q_DELTA)
gen = (p for p in itertools.product(Q, Q) if p[0]    ↩
    < p[1])
for i, p in enumerate(gen):
    with open("./{0}/{1:04d}".format(DIR, i),    ↩
      mode='w', encoding='utf-8') as f:
        f.write("{0[0]} {0[1]}".format(p))
```

In order to collect all the elements together, we will use the *Snakemake* assembly system [https://snakemake.readthedocs.io]. This system is ideologically similar to the Make assembly

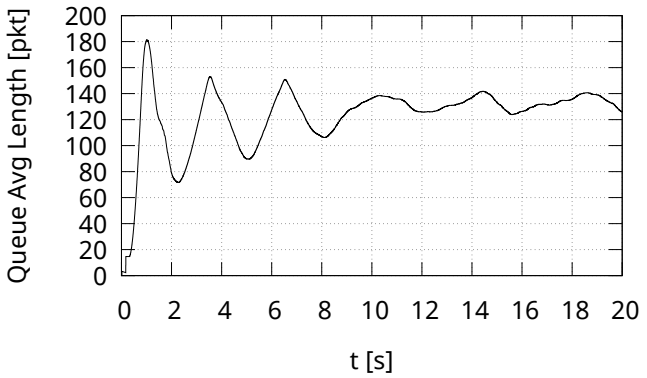

Fig. 3. Average queue length at link capacity $C = 5$ Mbps

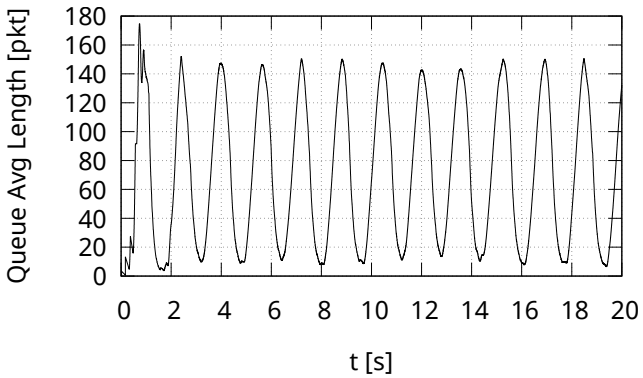

Fig. 4. Average queue length at link capacity $C = 20$ Mbps

system. However, it is not aimed at assembling software, but for reproducible and scalable data analyses. The syntax of *Snakemake* language is similar to the *Python* language.

```
rule all:
  input:
    ['spectrum/{0}'.format(f) for f in
      os.listdir("./parameters")]
  output:
    touch(".status")

rule fft:
  input:
    "parameters/{file}"
  output:
    "4fft/{file}"
  shell:
    "ns red.tcl `cat {input}` > {output}"

rule spectrum:
  input:
    rules.fft.output
  output:
    "spectrum/{file}"
  shell:
    "julia spectrum.jl {input} > {output}"
```

The resulting set of programs can be parallelized according to the SPMD ideology (single program, multiple data) (see Darema (2001)).

SIMULATION EXPERIMENT

As an illustration, we give a concrete example. Let's set the following parameters of the RED algorithm: the number of sessions $N = 60$, round-trip time $T_p = 0.075$ s, thresholds $Q_{\min} = 75$ packages and $Q_{\max} = 150$ packets, drop probability $p = 0.1$, parameter $w_q = 0.002$.

In the study examines the impact of the parameters on the character of the self-oscillation mode. Let us investigate the dependence of self-oscillation on the link capacity C.

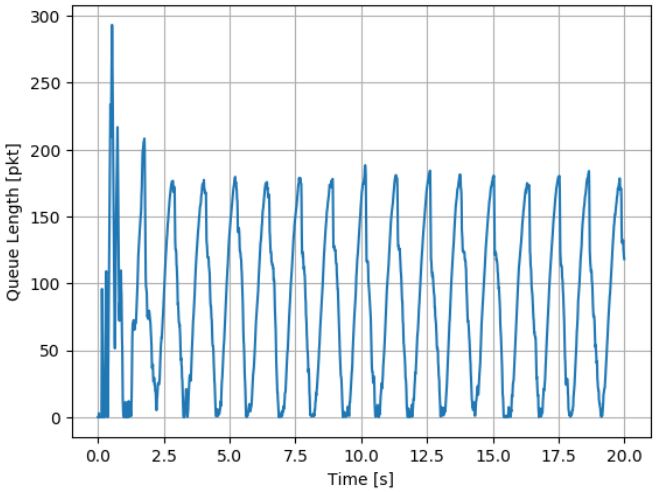

Fig. 5. Instantaneous queue length at link capacity $C = 20$ Mbps

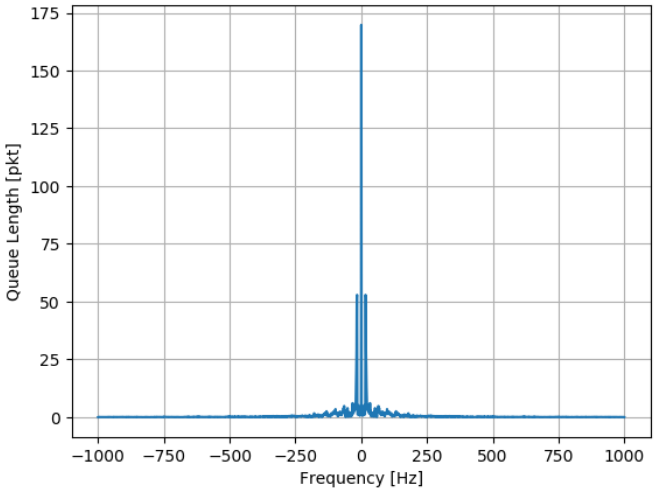

Fig. 6. Spectrum of self-oscillations of instantaneous queue length at link capacity $C = 20$ Mbps

The Fig. 3 and Fig. 4 show the behavior of the average queue length for link capacity $C = 5$ Mbps and $C = 20$ Mbps. In the second case clearly shows the presence of the self-oscillation mode. Theoretically obtained characteristic of this mode: oscillation frequency $\nu = 0.6$ Hz, oscillation amplitude $A = 150$ packets. In the spectral study of the results of the simulation, we obtained the following characteristics: the frequency of self-oscillations $\nu = 0.5$ Hz, the amplitude of the oscillations $A = 169$ packets (see Fig. 5 and Fig. 6). As can be seen, the theoretical and experimental values are very close. Thus, our program complex can serve the purposes of verification of theoretical studies of the self-oscillatory regime in control systems.

CONCLUSION

The authors have developed the set of programs for simulation experiment in order to investigate the self-oscillation mode of control systems and to verify theoretical results. It is assumed that this experiment will confirm the analytical model of the active queue management module with the RED-like algorithm proposed by the authors.

ACKNOWLEDGMENT

The publication has been prepared with the support of the "RUDN University Program 5-100" and funded by Russian Foundation for Basic Research (RFBR) according to the research project No 16-07-00556. The computations were carried out on the Felix computational cluster (RUDN University, Moscow, Russia) and on the HybriLIT heterogeneous cluster (Multifunctional center for data storage, processing, and analysis at the Joint Institute for Nuclear Research, Dubna, Russia).

REFERENCES

Adams, R. (2013), Active Queue Management: A Survey, *IEEE Communications Surveys & Tutorials* 15(3), 1425–1476.

Altman, E. and Jiménez, T. (2012), NS Simulator for Beginners, *Synthesis Lectures on Communication Networks* 5(1), 1–184.

Darema, F. (2001), The SPMD Model: Past, Present and Future, pp. 1–1.

Floyd, S. and Jacobson, V. (1993), Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking* 1(4), 397–413.

Issariyakul, T. and Hossain, E. (2012), *Introduction to Network Simulator NS2*, Springer US, Boston, MA.

Jenkins, A. (2013), Self-Oscillation, *Physics Reports* 525(2), 167–222.

Joshi, A. and Lakhanpal, R. (2017), *Learning Julia*, Packt Publishing.

Kulyabov, D. S., Korolkova, A. V., Velieva, T. R., Eferina, E. G. and Sevastianov, L. A. (2018), The Methodology of Studying of Active Traffic Management Module Self-oscillation Regime, W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak and J. Kacprzyk, eds, DepCoS-RELCOMEX 2017. Advances in Intelligent Systems and Computing, Vol. 582 of *Advances in Intelligent Systems and Computing*, Springer International Publishing, Cham, pp. 215–224.

Kushwaha, V. and Gupta, R. (2014), Congestion Control for High-Speed Wired Network: A Systematic Literature Review, *Journal of Network and Computer Applications* 45, 62–78.

Kushwaha, V. and Shwer, R. (2013), A Review of Router based Congestion Control Algorithms, *International Journal of Computer Network and Information Security* 6(1), 1–10.

Lautenschlaeger, W. and Francini, A. (2015), Global Synchronization Protection for Bandwidth Sharing TCP Flows in High-Speed Links, Proc. 16-th International Conference on High Performance Switching and Routing, IEEE HPSR 2015, Budapest, Hungary.

Misra, V., Gong, W.-B. and Towsley, D. (1999), Stochastic Differential Equation Modeling and Analysis of TCP-Windowsize Behavior, *Proceedings of PERFORMANCE* 99.

Nadkarni, A. P. (2017), *The Tcl Programming Language: A Comprehensive Guide*, CreateSpace Independent Publishing Platform.

Rao, K. R., Kim, D. N. and Hwang, J. J. (2010), *Fast Fourier Transform - Algorithms and Applications*, Signals and Communication Technology, Springer.

Ren, F., Lin, C. and Wei, B. (2005), A Nonlinear Control Theoretic Analysis to TCP-RED System, *Computer Networks* 49(4), 580–592.

Velieva, T. R., Korolkova, A. V. and Kulyabov, D. S. (2015), Designing Installations for Verification of the Model of Active Queue Management Discipline RED in the GNS3, 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), IEEE Computer Society, pp. 570–577.

Welch, B. and Jones, K. (2003), *Practical Programming in Tcl and Tk*, 4th edn, Prentice Hall.

AUTHOR BIOGRAPHIES

TATYANA R. VELIEVA postgaduate student in Peoples' Friendship University of Russia. Her current research activity focuses on mathematical modeling. Her email address is velieva_tr@rudn.university.

ANNA V. KOROLKOVA received her Ph.D. in Mathematics in 2010. Since then, she has worked as associate professor in RUDN University (Peoples' Friendship University of Russia). Her current research activity focuses on mathematical modeling. Her email address is korolkova_av@rudn.university.

MIGRAN N. GEVORKYAN received his Ph.D. in Mathematics in 2013. Since then, he has worked as associate professor in RUDN University (Peoples' Friendship University of Russia). His current research activity focuses on mathematical modeling. His email address is gevorkyan_mn@rudn.university.

IVAN S. ZARYADOV received his Ph.D. in Mathematics in 2010. Since then, he has worked as associate professor in

RUDN University (Peoples' Friendship University of Russia). His current research activity focuses on probability theory. His email address is zaryadov_is@rudn.university.

DMITRY S. KULYABOV received his Ph.D. in Physics in 2000. Since then, he has worked as associate professor in RUDN University (Peoples' Friendship University of Russia). His current research activity focuses on mathematical modeling. His email address is kulyabov_ds@rudn.university.

SERGEY A. VASILYEV received his Ph.D. in Physics in 2003. Since then, he has worked as associate professor in RUDN University (Peoples' Friendship University of Russia). His current research activity focuses on mathematical modeling. His email address is vasilyev_sa@rudn.university.