An Empirical Study of NetEm Network Emulation Functionalities

Audrius Jurgelionis*, Jukka-Pekka Laulajainen†, Matti Hirvonen† and Alf Inge Wang*
* *Norwegian University of Science and Technology, Trondheim, Norway*
Email: audrius@idi.ntnu.no, alfw@idi.ntnu.no
† *VTT Technical Research Centre of Finland, Oulu, Finland*
Email: jukka-pekka.laulajainen@vtt.fi, matti.hirvonen@vtt.fi

***Abstract*—In this paper we have evaluated the main functionalities of NetEm, a popular Linux based network emulator, which we have used to stress test the performance of the Games@Large distributed gaming system. We have performed a number of tests on different NetEm functionalities in order to evaluate their practical performance conformity and validity versus the NetEm description and theoretical expectations. We have found that the NetEm behaviour conforms to expectations for the emulation of delay and packet loss without correlation. However, in the case of jitter emulation, the actual realized jitter is lower than the given input value. It is an important fact to be aware of when using NetEm for different application testing. This paper also provides a baseline methodology for network emulation tool validation.**

***Keywords*-network emulator; network measurement; application testing;**

I. Introduction

Network emulators are important tools for doing research and development related to network protocols and applications. With network emulation it is possible to perform tests of realistic network scenarios in a controlled manner, which is not possible by only using real network devices without emulation capabilities. One of the most popular network emulators in the research world is NetEm. This free open-source tool is widely used in different kinds of testbeds, but few publications analyze its performance. In this paper, we are presenting a study analyzing NetEm with test cases concerning delay, jitter, and packet loss emulation.

NetEm was used as a test tool in the Games@Large project [1] developing a platform for distributed gaming. Network impairment emulation, such as delay and jitter, was needed both in the project technical development as well as in the user-centred research. As an example, user-centred research identified that NetEm emulated jitter was causing problems for the user-perceived gaming experience. In the existing literature there was not enough information to really understand how to solve these problems in the technical implementation of the Games@Large system. Thus, it was decided to investigate if NetEm's emulated impairments are accurate and statistically correct. Work presented in this paper was motivated by the need of understanding the operation of the test tool, in order to be able to make reasonable decisions during the Games@Large development.

The paper is organized as follows. Section II briefly describes related work, Section III presents NetEm and the methods it is using to emulate network impairments, Section IV presents our testbed, Section V presents measurement methodology and the results of our evaluation, and finally Section VI concludes the paper.

II. Related Work

There are many network emulators that are being used in research, but the most popular are NetEm, Dummynet, and NIST Net [2]. These three solutions are of production quality and they are widely used by researchers. Dummynet is a standard part of FreeBSD and is implemented as part of the packet filtering mechanism. Dummynet does packet filtering on output. But it is completely self-contained and is not easily extended. NIST Net is a Linux kernel extension that provides complex delay, loss, and other emulation options. NIST Net is running in the Linux kernel 2.4 but is not available in the kernel 2.6. NetEm is included in the Linux kernel 2.6 and is somehow flexible in terms of modification possibilities. Thus it is currently widely used by researchers. NetEm usage and applications range from testbeds for Next Generation Networks (NGN) [3] evaluation to investigation of various TCP algorithms [4] and quality assessment of interactive voice applications [5]. However, few publications describe analyses of the performance of NetEm itself. [6] and [2] discuss the NetEm performance in detail, and [7] includes discussion only on particular properties.

Desired network condition emulation for Nist Net, another widely used network emulator, is tested and analysed in [8], mainly with regard to packet loss, delay and jitter. The report provides some methodology for the validation of network emulation tools. [9] presents the key elements of constructing an accurate network emulator.

Since NetEm's source code is constantly modified and there is no well documented description of its functionalities, it is not always obvious whether the NetEm emulation is correct, e.g. generated values are distributed according to the theoretical definition of the distribution functions. Possible design flaws in the NetEm software related to correlated packet loss emulation have already been discovered in [10]. Thus it is worthwhile evaluating the NetEm's functionalities.

This also shows the importance of having a common validation methodology and a well documented functionality as well as a validation result description that later could be accessible by users of the network emulation tool.

III. NetEm Parameter Overview

NetEm is an enhancement of the traffic control facilities of Linux that allows adding delay, packet loss and other scenarios. It has been largely improved in last few years and is available in kernel 2.6. It is very easy to deploy as it is implemented as a queuing discipline in Linux and can be activated by the Linux tc utility [6].

A. Packet delay, jitter and their distribution functions

NetEm delay and its variation (jitter) parameters are described by the average value (μ), standard deviation (σ), and correlation (ρ). By default, NetEm uses a uniform distribution ($\mu \pm \sigma$), but any distribution table can be used. The iproute2 includes tools to generate a normal distribution, Pareto distribution, Pareto normal distribution, and a sample based on experimental data [6]. According to [11], the actual tables (normal, pareto, paretonormal) are generated as part of the iproute2 compilation and placed in /usr/lib/tc directory. A correlation value can be used in order to emulate a more realistic delay behaviour.

B. Packet loss

Packet loss is implemented in NetEm by randomly dropping a percentage of the packets before they are queued. Loss is specified in the command interface as a percentage of packets to drop [6]. An optional correlation may also be added making the packet loss probability less random by adding dependency on previous packet being dropped or not. This makes it possible to emulate packet burst losses.

According to [10], the generation of loss with correlation is not working properly in the current version of NetEm. As a consequence, a patched version of NetEm implementing a new loss generation algorithm was released.

IV. Testbed

The measurements were conducted using a testbed (Fig 1) including three laptops. Two of them were identical (Core 2 Duo T9400, 2.53 Ghz, 4 GB, Windows XP Pro) and they were used as the endpoints of the traffic flows and the measurement. The third one (Pentium M 1400 MHz, 1 GB, CentOS 5.0, kernel 2.6.18, iproute2-ss061002) was used as a bridging network emulator between the two endpoints. 100BASE-TX connections were used between the laptops.

The network emulator was configured with timed shell scripts during the test cases. All the network impairments were strictly set only to affect the measured test traffic and not to influence the control traffic of the used measurement tools.

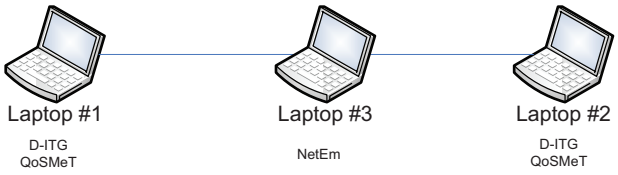

Figure 1. Measurement setup.

A. Test Traffic

Simple constant bit rate over the User Datagram Protocol (UDP) was used as test traffic. It was generated using open source Distributed Internet Traffic Generator (D-ITG) tool [12]. A packet size of 1000 bytes was used with a packet rate of 300 packets per second (inter-departure time of 3.33 ms), resulting in a traffic flow of about 300 kilobytes per second.

B. Measurement Tools

All the measurements were performed using QoSMeT tool [13]. The measurement endpoints were synchronized using Global Positioning System (GPS) to enable accurate one-way delay measurements. In addition, one-way delay jitter and packet loss were measured during the tests.

The one-way delay was defined in a same way as in RFC 2679 [14]:

$$D = T_r - T_s$$

where T_s is the time when the source sends the first bit of a packet, and T_r is the time when the destination receives the last bit of the packet.

We define one-way delay jitter to be the variation of one-way delay. This is measured as the absolute value of the difference between two consecutive one-way delay samples as follows:

$$J = |D_n - D_{n-1}|$$

where D_n is the one-way delay of the *n*-th packet and D_{n-1} is the one-way delay of the *(n-1)*-th packet.

Packet loss is simply calculated as a proportion of lost packets for a sliding window of one second in length.

In addition, standard deviation (σ) of delay was calculated from the one-way delay samples as:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(D_i - \bar{D})^2}$$

where *N* is the number of delay samples used in calculation, D_i is the ith one-way delay sample, and $\bar{D}$ the mean value of the one-way delay samples.

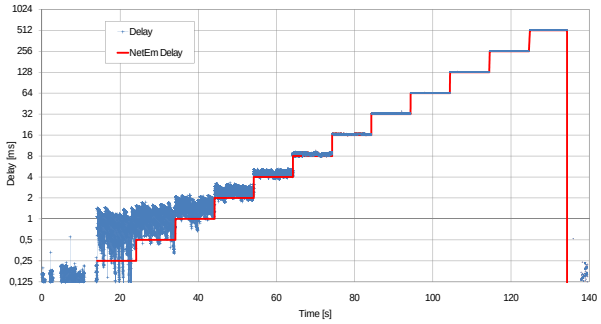

Figure 2. Measured and configured delay.

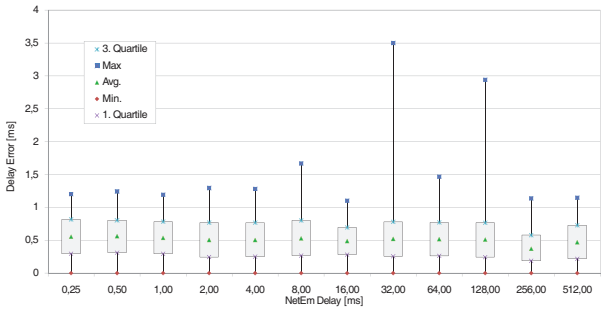

Figure 3. Measured delay error.

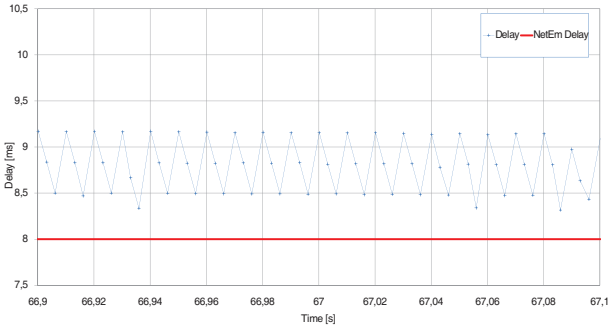

Figure 4. Sawtooth behaviour of the NetEm emulated delay.

V. Measurement Methodology and Results

Our study focused on adding delay, delay jitter and packet loss to network flows. To eliminate the effect of unwanted events such as high Central Processing Unit (CPU) load on any of the laptops used, all unnecessary services and applications running on the background were closed. Further, the results of each test were inspected after each test case and if any inaccuracies were noticed, the test case was executed again. The natural besaline delay without any NetEm added delay was measured to be on average 350 μs with a standard deviation below 50 μs. The average delay of 350 μs caused by the test arrangement was later subtracted from all the results to include only the effect of NetEm. When using NetEm in application testing, the effect of baseline delay should be compensated by setting the NetEm delay to a value that is calculated by subtracting the baseline delay from the targeted delay. Our test setup didn't introduce any packet losses so all losses in the tests were caused by NetEm.

The following subsections present the results.

A. Delay

We tested the NetEm's delay generation accuracy by setting a targetted delay value with the 'tc' command and measuring if the realized one-way delay was the same. The following delay values (ms) were used: 0, 0.25, 0.5, 1, 2, 4, 16, 32, 64, 128, 256, 512. Each of the delay levels was tested for 10 seconds to gather 3000 delay samples.

Figure 2 presents the configured and corresponding realized delay values. The difference between the set and realized values is shown in Figure 3. The average error of NetEm delay emulation in our measurements was around 500 μs in all of the cases. For emulation of low delay values, this level of error is unacceptable. Accuracy with average error lower than 1 percent of the target delay can be achieved only when emulating delays higher than 50 ms.

Figure 4 shows the delay values for individual packets in a sample of an experiment. The figure shows a sawtooth behaviour similar to the results of [2], confirming that our measurement arrangement is valid. The variation of delay between packets is caused by the limited timer interrupt speed (100 Hz) used in the Linux kernel of our network emulator laptop. This means that packets can be released by NetEm only 100 times a second, making some packets being sent later than they should be.

B. Jitter

Table I
NetEm parameter configuration for jitter tests

Delay, (ms)	Jitter, (ms)	Distribution
0	0	normal
256	1	pareto
	2	paretonormal
	4	
	8	
	16	
	32	
	64	
	128	
	256	
	512	

Table I presents the values for delay, jitter, distribution, and correlation settings used in the jitter tests. All the combinations of the presented values were tested during the measurements. Each of the delay levels was tested for 10 seconds to gather 3000 delay samples.

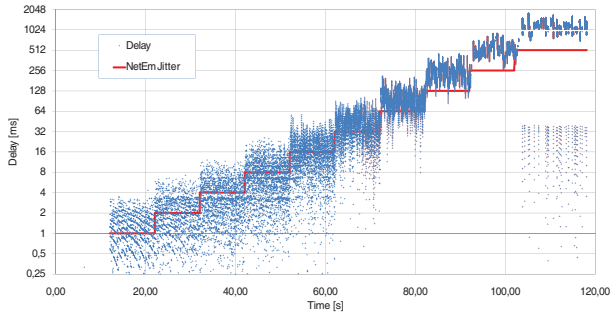

Figure 5. Measured delay for normal distribution and 0ms base delay.

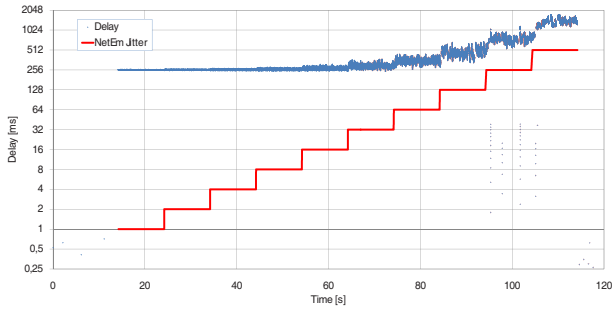

Figure 6. Measured delay for normal distribution and 256ms base delay.

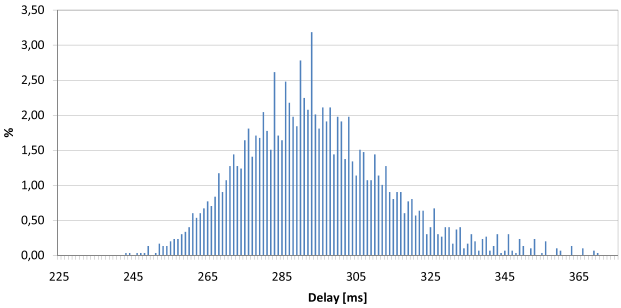

Figure 7. Normal distribution of measured delay, jitter 32 ms.

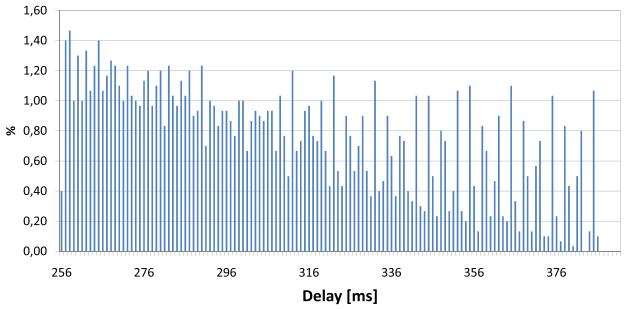

Figure 8. Pareto distribution of measured delay, jitter 32 ms.

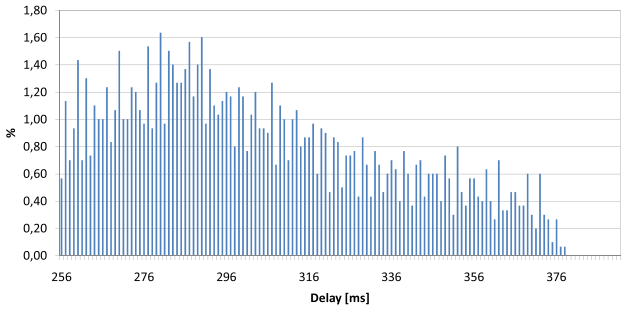

Figure 9. Paretonormal distribution of measured delay, jitter 32 ms.

Figure 5 presents the realized delay values along with the configured jitter value when setting the base delay to 0 ms. Figure 6 presents the similar results for the base delay of 256 ms. In both cases, the distribution is normal and the correlation is 0. Similar figures were obtained for pareto and paretonormal distribution.

The results show that jitter can be added without any base delay configured for NetEm. Even in this case, the average delay values mostly vary above the set jitter value. However, with some jitter configurations, NetEm tends to introduce some noticeably small delay values (e.g. Fig 5, NetEm jitter below 32 ms or 512 ms). For other jitter configurations this behaviour is not present (e.g. Fig 5, NetEm jitter 256 ms).

With a base delay, NetEm intuitively should use delay values varying around the set base delay. This is not the case, and the jitter configuration affects also by increasing the mean level of delay.

The effect of distribution looks reasonable. Pareto and paretonormal distributions stand up with sharper lower and upper delay limit, while normal distribution has clearly most of the samples around the mean value as common to normal distribution [15]. Figures 7, 8, 9 show the distribution of delay values when using NetEm for generating 32 ms jitter on top of 256 ms base delay for normal, pareto, and paretonormal distributions correspondingly.

Normal distribution looks more or less like it should. Pareto distribution exhibits high peaks at the higher end of the distribution and paretonormal (Fig. 9) looks like a combination of normal and pareto distribution in Figure 7 and 8. It should be noted that the mean value μ for all distributions is not at the base delay of 256 ms as it should but around 296 ms. This property of NetEm jitter generation is not documented, but it is important to know when using NetEm.

Figure 10 presents the combined results of all the jitter levels for 0 correlation and 0 ms base delay using normal distribution. The figure includes both the standard deviation and the mean jitter (as the absolute value of difference between two concecutive delay samples) of the measured delay calculated as a sliding average of 300 samples together with the configured jitter values. Jitter is configured as standard deviation of delay for NetEm. The induced delay should be $\mu \pm \sigma$, but the results show different. In all the

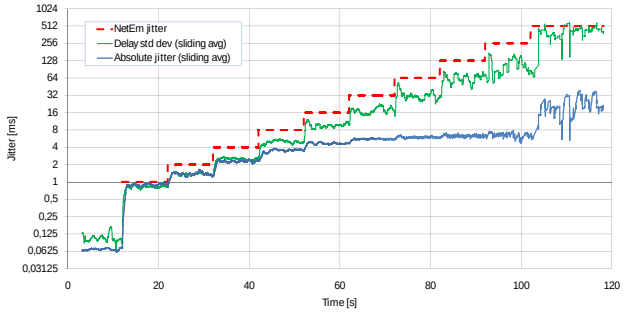

Figure 10. Standard deviation and average jitter of measured delay, base delay 0 ms, normal distribution.

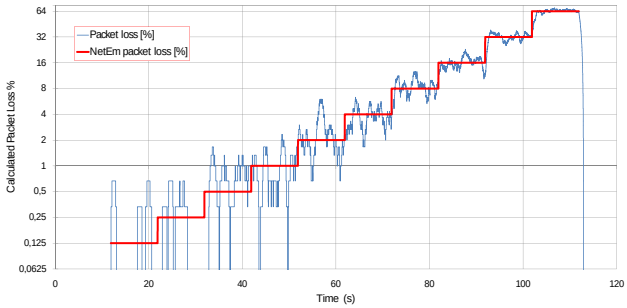

Figure 11. Measured and configured packet loss.

cases the realized standard deviation is significantly less than configured. The actual realized jitter is much lower, indicating a significant correlation between the generated delay values even though 0 correlation was used. These findings are very important to be aware of when using the NetEm's jitter generation functionalities for research and development purposes.

C. Packet Loss

The following values (%) were used in the packet loss tests: 0.125, 0.25, 0.5, 1, 2, 4, 16, 32, 64. Each of the packet loss levels were tested for 10 seconds to gather 3000 samples. We did not include cases with different packet loss correlation values since this has been identified as a weak point of NetEm already in [10].

Figure 11 presents the combined results for the packet loss tests and the corresponding error values are presented in Figure 12. The results show that the average realized packet loss stays within the limit of 99 percent accuracy with packet loss values up to 4 percent. With higher values the accuracy gets weaker.

VI. Discussion and Conclusions

The goal of our work was to investigate the NetEm network emulator concerning aspects that were not covered well in the previous work. The main results of the study show that NetEm is able to generate constant delay higher

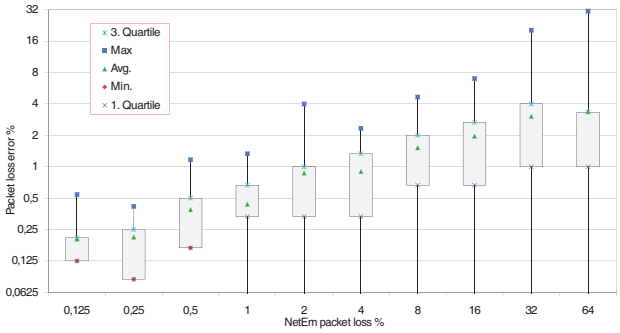

Figure 12. Measured packet loss error (percent units).

than 50 ms and uncorrelated packet loss fairly accurately. However, the outcome of additional jitter added on top of the constant base delay is not so obvious. The delay values are not varying around the given base delay level, but in most of the cases on levels above it. On the other hand, the resulting measurable standard deviation is lower than the given value and the realized jitter is much lower than we intuitively could expect.

This inadequacy could be caused by the delay calculation algorithm which returns values with the correct statistical distribution, but causes the observed mean and standard deviation values to drift off. The source code analysis [16] revealed that the NetEm delay calculation does not include correction factors, e.g. used in Nist Net [17], to give the right apparent μ and σ values. The correction is implemented by an additional component, however, there is no detailed explanation on how this component was derived.

The outcomes of our work, revealing how NetEm behaves when emulating typical network impairments such as delay or jitter, should be useful for other application testing. Moreover, the method used for the verification of NetEm functionalities presents a baseline validation methodology for other network emulators.

Acknowledgment

This work was developed in the IST Games@Large project [1], which is an Integrated Project under contract no IST038453 and is partially funded by the European Commission, and also carried out during the tenure of an ERCIM fellowship.

References

[1] Y. Tzruya, A. Shani, F. Bellotti, A. Jurgelionis, *Games@Large - a new platform for ubiquitous gaming*, In: Broadband Europe 2006, Geneva, Switzerland (2006)

[2] L. Nussbaum, O. Richard. *A Comparative Study of Network Link Emulators*, Proceedings of the 12th Communications and Networking Simulation Symposium (CNS'09), March 22 - 27, 2009, San Diego, USA

[3] J. Fabini, P. Reichl, C. Egger, M. Happenhofer, M. Hirschbichler, L. Wallentin, *Generic Access Network Emulation for NGN Testbeds*, Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks and communities, TRIDENTCOM 2008, March 17-20, 2008, Innsbruck, Austria

[4] T. Yamamoto, *Estimation of the advanced TCP/IP algorithms for long distance collaboration*, International Journal of Fusion Engineering and Design, Volume 83, Issues 2-3, April 2008, Pages 516-519

[5] A. P. Couto da Silva, M. Varela, E. de Souza e Silva, R. M.M. Leão, G. Rubino, *Quality assessment of interactive voice applications*, Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 52, Issue 6, April 2008, Pages 1179-1192

[6] S. Hemminger, *Network Emulation with NetEm*, Proceedings of the 6th Australia's National Linux Conference (LCA2005), April 2005, Canberra, Australia

[7] H-W. Jin, S. Narravula, K. Vaidyanathan, D. K. Panda, *NemC: A Network Emulator for Cluster-of-Clusters*, Proceedings of the 15th International Conference on Computer Communications and Networks, ICCCN 2006, 9-11 October, 2006, Arlington, USA

[8] T. Hoßfeld, D. Hock, P. Tran-Gia, K. Tutschku, M. Fiedler, *Testing the IQX Hypothesis for Exponential Interdependency between QoS and QoE of Voice Codecs iLBC and G.711*, University of Würzburg, Institute of Computer Science, Research Report Series, Report No. 442, March 2008

[9] Zhihao Guo, B. Malakooti, K. Bhasin, A. Holtz, *Design of Accurate and Efficient Network Emulation Systems with Application to Inter-planetary Networks*, Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, ICNSC '06, 2006, pp. 895-900

[10] S. Salsano, F. Ludovici, A. Ordine, Technical Report, *Definition of a general and intuitive loss model for packet networks and its implementation in the NetEm module in the Linux kernel*, University of Rome, July 2009

[11] The Linux Foundation, *NetEm Network Emulator*, http://www.linuxfoundation.org/en/Net:Netem, 2011

[12] A. Botta, A. Dainotti, A. Pescape, *Multi-protocol and multi-platform traffic generation and measurement*, INFOCOM 2007 DEMO Session, May 2007, Anchorage, Alaska, USA

[13] J. Prokkola, M. Hanski, M. Jurvansuu, M. Immonen, *Measuring WCDMA and HSDPA Delay Characteristics with QoSMeT*, Proceedings of IEEE International Conference on Communications ICC 2007, June 2007, Glasgow, Scotland, UK.

[14] G. Almes, S. Kalidindi, M. Zekauskas, *RFC-2679 A One way Delay Metric for IPPM*, 9/1999, Internet RFC 2679.

[15] M. H. DeGroot, *Optimal Statistical Decisions*, Wiley Classics Library. 2004. (Originally published (1970) by McGraw-Hill.)

[16] Linux Cross Reference, *Network emulator*, http://lxr.free-electrons.com/source/net/sched/sch_netem.c, 2011

[17] M. Carson and D. Santay, *NIST Net: a Linux-based network emulation tool*, SIGCOMM Comput. Commun. Rev. 33, 3 (July 2003), 111-126