

# Expanding the Ns-2 Emulation Environment with the Use of Flexible Mapping

Robert R. Chodorek<sup>1</sup>(✉) and Agnieszka Chodorek<sup>2</sup>

<sup>1</sup> Department of Telecommunications, The AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland

`chodorek@agh.edu.pl`

<sup>2</sup> Department of Information Technology, Kielce University of Technology, Al. Tysiaclecia Panstwa Polskiego 7, 25-314 Kielce, Poland

`a.chodorek@tu.kielce.pl`

**Abstract.** The Berkeley’s ns-2 simulator was, for a long time, one of the most popular open-source simulation tools. Although the new tool in the ns family, the ns-3, replaced it in the above ranking, the simplicity of the ns-2, with its flexibility and ability to operate at higher levels of abstraction caused the simulator to remain in use. This paper presents our enhancements to the mapping of incoming and outgoing traffic in the ns-2 simulator when it works in emulation mode. Our enhancements expand the build-in 1-to-1 MAC address mapping to 1-to-many address/port mapping, which allows the emulator to connect to more end-systems or subnetworks than the number of interfaces of the emulation server.

**Keywords:** ns-2 simulator · Performance evaluation · Emulation of computer networks · Elastic traffic · Streaming

## 1 Introduction

The name ns is an acronym for “network simulator” and refers to the family of open-source simulation tools, beginning with the discrete event LBNL<sup>1</sup> Network Simulator (later known as ns-1). Now the ns family consists of two versions, ns-2 [1] and ns-3 [2]. Both are discrete event-driven environments for the simulation of computer networks, and both are publicly available for research, education and development under the GNU license [3].

The ns-2 simulator is focused mainly on simulations of computer networks with relatively weak possibilities of network emulation. The simulator is a flexible tool for computer network analysis, able to provide large numbers of network types, topologies, technologies and protocols. It allows the user to carry out simulation experiments at different levels of abstraction. From low levels, demanding detailed description and exact simulation models (e.g. simulation

---

<sup>1</sup> LBNL stands for Lawrence Berkeley National Laboratory, the ns-1 place of development.

of TCP transmission with full protocol processing and functionality, or simulation of Wi-Fi LAN [4] with RTS/CTS frame exchange process) to high levels, demanding only a rough description of the problem and general network situation (ftp applications modelled as an infinitive source of data packets, long-distance links modelled as twin simplex links with defined throughput and propagation delay, etc.).

The ns-3 has a reputation of being more emulator than simulator. This programming tool achieves high functionality and high accuracy by using Linux mechanisms as accurate models of network functions. Full code and full processing of base network functions (e.g. protocols: TCP, UDP, IP and base routing) in the ns-3 simulator is mainly borrowed from the Linux operating system. Such an approach to network modelling allows the simulator to obtain a natural ability to network emulation at the level of protocol processing.

In the lower layers of the OSI/ISO Reference Model, the ns-3 simulator implements detailed models of many network solutions (including detailed and accurate propagation models for Wi-Fi [5], LTE [6], etc.). It also allows the running of user's code in applications and services. Both those advantages improves accuracy of simulations and supports network emulation. However, in event-driven environments, this accuracy and support is done at the cost of performance.

Generally, execution of certain operations of the ns-3 simulator requires more processor time than the execution of the corresponding operations of the ns-2. Simulations in the ns-3 are carried out using more precise models and the obtained results are more accurate, but the experiment lasts longer. In the context of real-time emulation, it means that on the same machine we'll be able to emulate a less complicated network environment than if the ns-2 is used. Moreover, for detail-oriented approaches to simulation based on very precise models, the testing of new ideas at early stages of deployment where higher levels of abstraction are usually needed is inhibited. Last but not least, there is no possibility of an easy transfer between these tools because of different programming user interfaces<sup>2</sup>) and lack of backward compatibility between ns-3 and ns-2 [7]. As a result, after initial enthusiasm with ns-3 which caused users to desert the ns-2, ns-2 has come back into favor. Nowadays, both simulation tools are under development and active maintenance. A new development version of the ns-2 tool (version 2.36) is planned for release in February 2016.

The aim of the paper is to present our proposition for extensions to the ns-2 simulator, which enable the ns-2 emulation platform to flexibly map incoming and outgoing traffic. The paper is organized as follows. Section 2 describes emulation possibilities of the ns-2. Section 3 presents proposed extensions. Section 4 summarizes the paper.

## 2 The Network Emulator Embedded in the Ns-2

A network emulator is a device, program or software-hardware environment that imitates network behavior to deceive real-world networks or real-world

---

<sup>2</sup> TCL-based user interface in the ns-2 and the Python-based interface in the ns-3.

end-systems connected to the emulator into behaving as if they are connected to a real (usually novel or very complex) network. The ns-2 simulation environment has several mechanisms to support network emulation. Moreover, although protocol agents included in the ns-2 simulation platform operate at rather higher levels of abstraction (for example, the TCP protocol is modelled as two separate parts, a sending one and a receiving one – `Agent/TCP` and `Agent/TCPSink`, respectively), some of them (for example, `Agent/TCP/FullTCP`) are a close enough match to their real-world equivalents to be used for emulation purposes.

## 2.1 Operating Modes

The assumption is that the cooperation of the ns-2 simulation platform with the real-world network environment allows one to build more or less complex network environments inside the simulation platform shared with the real-world networks or network equipment. Connecting real and virtual environments facilitates the implementation of new algorithms defining the work of network nodes (for instance, new queuing), as well as the implementation of new protocols or protocol mechanisms, and tests of selected elements of a system.

Depending on the interpretation of live data, introduced from a real network to the emulated one, designers of the ns-2 emulator have defined two operating modes of the designed platform [1]:

- opaque mode, where live data are treated as opaque data packets,
- protocol mode, where live data may be interpreted, processed and even generated by the simulator.

In the first, opaque operating mode, packets from real fragments of the network are captured and then introduced into the ns-2 simulator while the content of captured packets (headers, data segments) will not be processed. Thus, captured packets can be transmitted inside the ns-2 with a given delay. They can be buffered, dropped in network nodes (due to congestions, or according to established order or assumed error rate) and transmitted out of sequence. Then, if the captured packet is not dropped or damaged, such packet will leave the simulator and will be injected back into the real fragment of the network.

The opaque mode shows partial real-virtual interference, so both intermediate and end systems do not have to be fully implemented. The second, protocol operating mode, is a mode of full real-virtual interference. The ns-2 emulation platform (emulated network devices, especially endpoints) interferes with the content of captured packets and, also generates its own packets which interfere with captured ones. Interference with captured packets can be seen when headers and if necessary data segments are processed and modified. Such interference takes place mainly in intermediate nodes. Generation of new packets typically takes place in endpoints (e.g. the TCP endpoints send packets according to settings of the connected traffic generator). In practice, in the ns-2 emulation platform, implementation of the protocol operating mode is very limited. So limited that the manual reports that only the opaque mode was implemented. Existing functionality of the protocol mode can be the basis for further development.

In both operating modes, input and output modules<sup>3</sup> and the real-time scheduler are used to enable the ns-2 to work as an emulator and, as a result, to cooperate with real-world networks.

## 2.2 Real-Time Scheduler

The ns-2 is an event-driven simulator. Events are stored in the event calendar (the calendar queue), where they are queued in non-descending time order. The process of the extraction of events from the calendar, according to their chronology of execution, is served by a scheduler. During simulation, events in the ns-2 platform are executed in real-time, and order of execution is set according to the simulator's virtual time. It means that in reality, events are executed as soon as possible, and usually much faster than the virtual passage of time. Only in the case of simulations of very complex networks is it possible that simulated time will be longer than simulation time (time that has passed in the real world).

Cooperation of simulated networks with real-world networks demands that all events in the simulation platform must be executed in real-time. Because of this demand, in the ns-2 emulator the default system scheduler was replaced with a real-time scheduler able to execute events in real-time. If the testbed computer, on which the ns-2 emulator is running, is too inefficient (CPU "horsepower" is insufficient), there is a danger that delays in the execution of some events might happen. If such delay exceeds the threshold value, stored in the slop factor parameter, the ns-2 will produce a warning.

The default value of the slop factor is 10 ms, and it is only three or four times less than the video frame period (frame display time). In the case of the analysis of video streams, transmitted in real-time, such and even smaller delays will cause substantial falsification of research results. An answer to this problem were extensions to the ns-2 simulator, developed at the University of Magdeburg (Germany) [8,9]. Extensions included improvements to the real-time scheduler module, modification to modules enabling the emulator to cooperate with the real-world network (network objects and tap agents) and trace extensions.

Improvements reported in [8,9] were intended for the emulation of mobile networks. In the case of the emulation of video transmission in a wired network, despite the usage of all above mentioned extensions, limitations to the emulated execution of real-time transmission still significantly falsified the test results. Therefore, further enhancements to the ns-2 emulator were needed. The first attempt at these enhancements was briefly mentioned in the paper [10]. In the second attempt, current extensions to the ns-2 simulator working in emulation mode were developed. Full extensions included network interface handling, address/port mapping, initial and final packet processing, improvements to the real-time scheduler and extensions for the emulator's protocol operating mode. The first two will be described in the next section.

---

<sup>3</sup> Input and output modules, respectively, captures traffic and (after processing) inject it into a real-world network.

### 3 Flexible Mapping of Incoming and Outgoing Traffic

When creating the ns-2 network emulator, it was assumed that it must cooperate not only with directly connected computers (which entails the usage of a homogeneous network, made with the use of one technology), but also with many computer systems connected via a complex, heterogeneous IP network. This assumption makes it possible to test the cooperation of applications executed on computer systems geographically located in different places and in different fragments of the Internet (for instance, in Kielce and Cracow). It also allows one to apply emulation only in chosen fragments of the end-to-end network path and only there, where we want to introduce test modifications. In other fragments of the end-to-end path, packet routing and processing will be carried out using existing, real-world network infrastructure, and test traffic will be subjected to a natural interaction with the real-world Internet traffic.

Practical implementation of the above assumption requires the development of new programming modules to serve the network interface, which allows the ns-2 emulator to flexibly capture packets received from the real network, and inject packets into the real-world network after processing. Improvements to modules capable of cooperation with the internal routing of the emulator also are needed. All modification and improvements are supposed to be able to serve all (unicast, broadcast and multicast) IPv4 addressing, and unicast and multicast IPv6 addressing (anycast addressing is omitted).

In order to achieve the assumptions, two programming modules were implemented: **Network/newIP** and **Agent/Tap/newIP**. They are derived classes of, respectively, **Network** and **Agent/Tap** superclasses. The **Network/newIP** module<sup>4</sup> realizes cooperation with real-world IP networks. The **Agent/Tap/newIP** module is associated with a given network node and performs the conversion of packets derived from the real-world network of packets used by the ns-2 emulator. The **Agent/Tap/newIP** module also performs the reverse conversion, where ns-2 packets are mapped as packets that will be injected into the real network. This module assures proper cooperation between the **Network/newIP** module and the given node of the emulated network.

The relationship between elements of the ns-2 emulator are illustrated with an example of a simple, five-node emulated network, shown in the Fig. 1. Nodes of the emulated network – R1, R4 and R5 – receive (or send) live data (to) the real-world network. This live traffic is transmitted through Tap agents (*a1*, *a2*, and *a3*), connected to emulated nodes R1, R4 and R5. The Tap agents are instances of class **Agent/Tap/newIP**. Agents which cooperate with real-world networks are seen by emulated nodes as typical agents of protocols or services (i.e. as other instances of the **Agent** base class).

The Tap agents also cooperate with instances of the **Network/newIP** class – *net1* and *net2*. As we can see from the Fig. 1, both 1-to-1 and 1-to-many

---

<sup>4</sup> The identifier of the module (**newIP**) denotes, that the **newIP** is a newly written module for cooperation with the IP protocol. This name was given, because **Network/IP** class already exists in the ns-2.

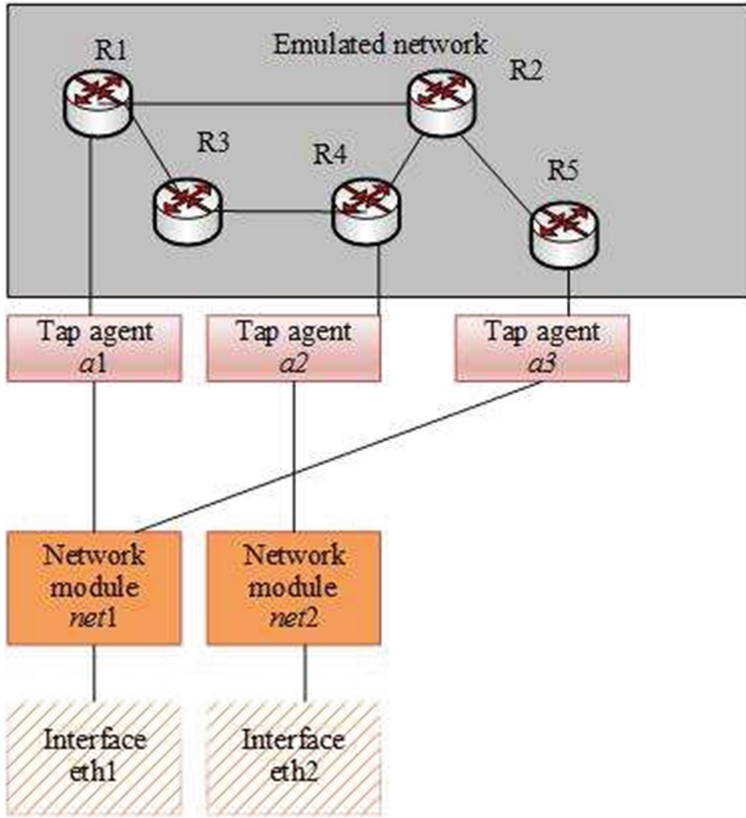


Fig. 1. Emulation system

cooperation is possible. In the picture, one **Network/newIP** module cooperates with one **Agent/Tap/newIP** agent (*a2* and *net2*), and one **Network/newIP** module (*net1*) cooperates with many (precisely, two: *a1* and *a3*) instances of the **Agent/Tap/newIP** class.

It's worth remarking that the original ns-2 mapping, available for emulation, permits only 1-to-1 mapping. In this typical solution, traffic introduced via a given interface from a real network to the emulated one is sending to a node of the emulated network using the typical ns-2 flat addressation (unlike hierarchical IP addressation), obtained by the agent. For example, IP address: 2, port: 3. If a packet is injected from the emulated network to the real one, it is injected via the output interface "as is" – it's just copied from the input buffer without any changes (including content of the TTL field).

In the case of the described extensions, the **Network/newIP** network module classifies packets according to rules defined by the user and sends the packet to a corresponding agent or agents. The module also retrieves parameters from the agent(s). If traffic is injected from the emulator to the real-world network, Tap

agents send packets to the **Network/newIP** network module, which, according to stored rules, makes the necessary changes in content to the packet, including address mapping. Address mapping refers to MAC addresses (in link layer's frames), IP addresses (in IP datagrams), and port numbers (in transport protocol's packets).

In the case of the mapping of IP addresses and/or port numbers, checksum processing is needed. Checksum processing is performed for transport protocols (TCP, UDP) and the IP protocol, version 4 (in IPv4 only headers are protected by checksum). Such operations are performed by **Network/newIP** modules.

Mapping of traffic introduced to the ns-2 emulator can be done on the basis of typical IP flow identifiers:

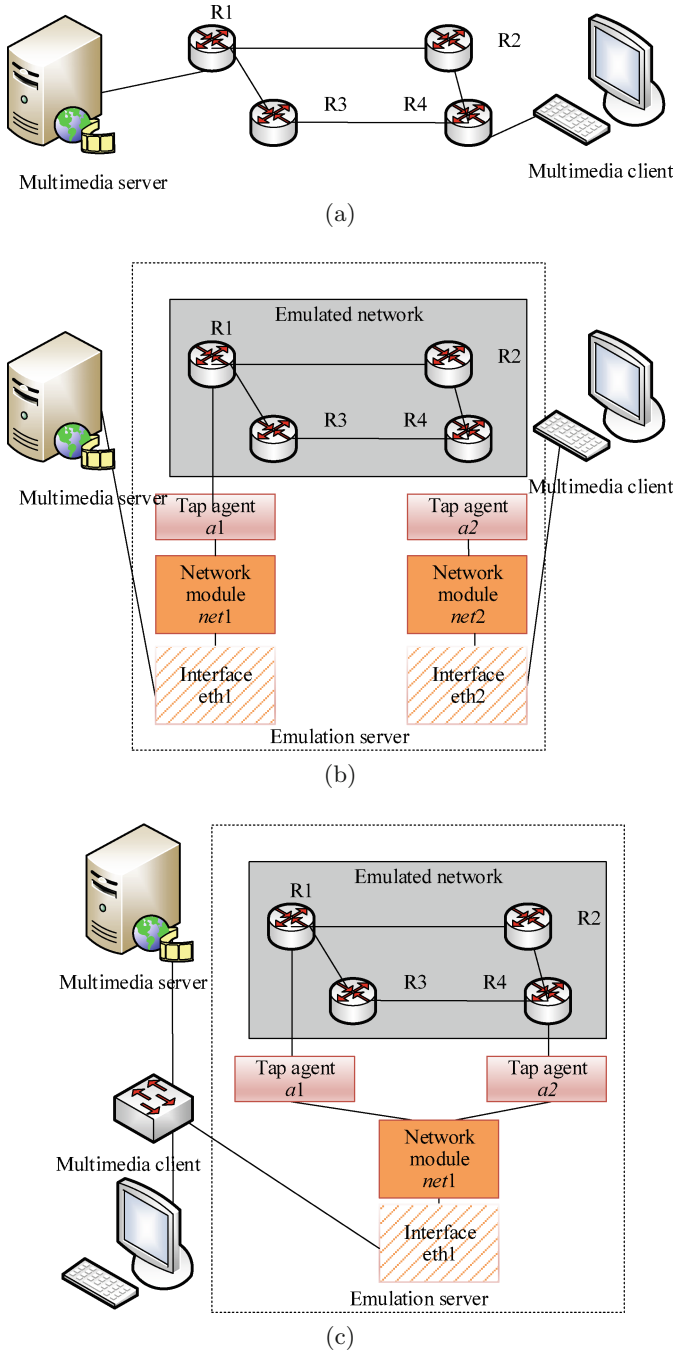
- quintuple (IPv4 and IPv6): source IP address, destination IP address, source port, destination port and type of transport protocol,
- triple (IPv6 only): flow label, source IP address, destination IP address.

Flow identification allows for the implementation more sophisticated traffic engineering. For instance, traffic introduced via interface `eth1` (Fig. 1) can be transmitted through the emulated network to the `eth2` interface according to assumed rules. For example, the TCP traffic is transmitted via routers R1, R3 and R4 and the UDP traffic is transmitted via routers R5, R2 and R4.

1-to-many mapping plays a crucial role in the emulator, because of hardware limitations. If we want to carry out emulation experiments in which we want to perform empirical analysis of interactions between different traffic sources (applications) executed on many servers, then we would connect  $N$  computers (application servers) to  $N$  interfaces with high performance emulation servers. However, in the case of laboratory tests, we usually have at our disposal many PC computers, each equipped with only one network card. Those computers are ideal to work as end systems and live traffic generators, but if we try to use one of them as an emulation server, in the case of 1-to-1 mapping we will only be able to connect one generator of live data to the ns-2 emulator. And we have no possibility of connecting separate, physical receivers of the transmitted data. In the case of tests of video transmission, where visual quality check (analysis of quality of experience) is important, the possibilities offered by 1-to-1 mapping are woefully inadequate.

Server computers are usually equipped with two, or four, different communication interfaces. However, further expansion to more interfaces can be difficult (for example, in the case of the smallest rack-mount servers, sized 1U). Additionally, a large number of interfaces can complicate the organization of experiments and time-consuming patching.

The proposed solution allows us to skip this problem. It assumes that the end system computers can create a group of machines connected through an efficient network to a physical switch, and the switch is, in turn, connected to the server interface. Such topology of the test network was used during experiments described in [11], where 1 server interface was able to serve a group of multicast receivers. A situation where a group of senders was connected to one interface of emulation servers also were investigated.



**Fig. 2.** Network topologies: (a) schematic diagram, (b) wiring diagram (emulation server equipped with two interfaces), (c) wiring diagram (emulation server equipped with one interface)



To validate the solution, performance tests were carried out. These tests were to transmit the TCP and UDP traffic between two endpoints (the endpoints were real computers outside an emulation environment). The proposed extension was verified by a long-term video traffic transmission in the presence of foreign traffic. It was checked to see whether the Ethernet frame loss reached zero and the emulated bandwidth was equal to the predetermined one.

On the basis of the results of a broad range of performance tests (conducted by the Authors during their experiments on SD and HD video transmission [11]) it can be stated that for many experiments 2 or 4 interfaces of an emulation server should be enough, if a 1-to-many mapping will be used. In the case of simple networks, with a small number of end-systems, 2 interfaces are enough. In the case of more complex systems, which need a partition, e.g. according to traffic sources, 4 interfaces should be used. It is possible that if the testbed network will be complex enough 4 is too small number of interfaces.

It should be remembered that during the experiments the critical infrastructure must be controlled. At a minimum, an Ethernet frame loss measurement at the switch must be performed. If the experiment is well prepared, Ethernet frame loss should be zero or, at the very least, negligible.

A simple emulation experiment, where the testbed network was partly realized in reality and partly emulated, is shown in Fig. 2. The network consists of one sender (multimedia server), one receiver (multimedia client) and four routers (R1 ... R4). A schematic diagram of the testbed network is depicted in Fig. 2a. Traffic, generated by the sender, enters the interface of router R1. Then, via router R2 or R3 (according to internal routing rules), it is directed to router R4, which sends the traffic to the receiver.

Figure 2b shows a diagram of cable connections in the case of an emulation server equipped with two network interfaces. The R1 and the R4 are boundary routers of the emulated network. Because the number of available server interfaces is greater than or equal to the number of connected real-world end systems, senders and receivers can be connected to separate physical interfaces.

Packets are captured and mapped to virtual ns-2 data structures in the network modules (instances of **Network/newIP** class). One network module is associated with one network interface of the emulation server. ns-2 packets are transferred to Tap agents (instances of **Agent/Tap/newIP** class). Tap agents are associated with data streams or flows and are attached to nodes of the emulated network. The 1-to-1 relation between network module and Tap agents is created.

The emulation server, presented in Fig. 2c, has only one network interface. As is shown in the wiring diagram, an auxiliary equipment (switch) was introduced to the real-world network to enable sharing of the interface. Because of the coexistence of input and output data streams in the same server's interface, the 1-to-many relation between network module and Tap agents is created.

## 4 Conclusions

In the paper our enhancements to the network simulator ns-2, working as an emulator, were presented. Improvements refer to the mapping of incoming and

outgoing live packets onto internal ns-2 packets. Improvements were designed to exceed a hardware limitation that restricts the number of end-systems connected to the emulator to the number of network interfaces of the emulation server.

The idea of the enhancement is that a high-speed switch, connected to the interface via a high-speed network, serves as a hardware expander, and the proposed flexible 1-to-many mapping (instead of the build-in 1-to-1 one) based on both MAC and IP addresses, as well as port numbers, serves as a multiplexer/demultiplexer of live traffic.

The proposed solution was successfully tested for HD video transmission.

**Acknowledgment.** The work was supported by the contract 11.11.230.018.

## References

1. Fall, K., Varadhan, K.: The ns Manual (2014). <http://ftp.isi.edu/nsnam/dist/release/rc1/doc/>
2. Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) *Modeling and Tools for Network Simulation*, pp. 15–34. Springer, Berlin (2010)
3. Henderson, T.R., et al.: Network simulations with the ns-3 simulator. SIGCOMM demonstration (2008)
4. Bhaskar, D., Mallick, B.: Performance Evaluation Of MAC Protocol For IEEE 802.11, 802.11Ext. WLAN And IEEE 802.15. 4 WPAN Using NS-2. *International Journal of Computer Applications* 119.16 (2015)
5. Pei, G., Henderson, T.: Validation of ns-3 802.11b PHY model (2009). <http://www.nsnam.org/~pei/80211b.pdf>
6. Piro, G., Baldo, N., Miozzo, M.: An LTE module for the NS-3 network simulator. In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2011)
7. Rene, S., et al.: Vespa: Emulating infotainment applications in vehicular networks. *IEEE Pervasive Comput.* **13**(3), 58–66 (2014)
8. Mahrenholz, D., Svilen, I.: Real-time network emulation with NS-2. In: *Proceedings of the 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications*. Budapest Hungary (2004)
9. Mahrenholz, D., Svilen, I.: Adjusting the ns-2 Emulation Mode to a Live Network. In: *Proceedings of KiVS'05*. Kaiserslautern, Germany (2005)
10. Chodorek, R., Chodorek, A.: An analysis of QoS provisioning for high definition video distribution in heterogeneous network. In: *Proceedings of the 14th International Symposium on Consumer Electronics (ISCE 2010)*, Braunschweig, Germany (2010)
11. Chodorek, R.R., Chodorek, A.: Providing QoS for high definition video transmission using IP Traffic Flow Description option. In: *Proceedings of the IEEE Conference on Human System Interaction*, pp. 102–107, Warsaw, Poland (2015)

Computer Networks

23rd International Conference, CN 2016, Brunów,  
Poland, June 14-17, 2016, Proceedings

Gaj, P.; Kwiecień, A.; Stera, P. (Eds.)

2016, XVI, 436 p. 174 illus., Softcover

ISBN: 978-3-319-39206-6