A Policy Controlled IPv4/IPv6 Network Emulation Environment

Tomislav Grgic and Maja Matijasevic
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, HR-10000 Zagreb, Croatia
tomislav.grgic@fer.hr, maja.matijasevic@fer.hr

Abstract: **In QoS enabled IP-based networks, QoS signaling and policy control are used to control the access to network resources and their usage. The IETF proposed standard protocol for policy control is Common Open Policy Service (COPS) protocol, which has also been adopted in 3GPP IP Multimedia Subsystem (IMS) Release 5. This paper presents a prototype for policy-controlled IPv4/IPv6 network emulation environment, in which it is possible to specify the policy control and emulate, over a period of time, QoS parameters such as bandwidth, packet delay, jitter, and packet discard probability for media flows within an IP multimedia session. The policy control is handled by COPS, and IP channel emulation uses two existing network emulation tools, NIST Net and ChaNet, supporting IPv4 and IPv6 protocols, respectively. The scenario-based approach allows reproducible performance measurements and running various experiments by using the same network behavior. A graphical user interface has been developed to make the scenario specification more user-friendly. We demonstrate the functionality of the prototype emulation environment for IPv6 and analyze its performance.**

1. INTRODUCTION

The ability to provide quality of service (QoS) is of vital importance for supporting advanced multimedia applications in next generation networks based on IPv6 and IP Multimedia Subsystem (IMS) [1] . Since the actual networks or testbeds for such applications are, in general, expensive and not widely (if at all) available, the application developers, as well as network administrators, would greatly benefit from emulated environments in which application testing and performance measurements could be conducted in realistic network conditions and in a reproducible way. While the application designer is primarily concerned with the impact of network QoS on applications, it is up to the network administrator to set the policy rules to define the criteria for network resources access and usage. By setting the policies, the network administrator can control the priority level which a particular service, or kind of traffic (voice, video, data, etc.), would receive from the network. The Common Open Policy Server (COPS) protocol [2] is an IETF proposed standard for policy control, which has also been adopted for use in the 3GPP IMS Release 5.

This paper presents a prototype for policy-controlled IPv4/IPv6 emulation environment, in which it is possible to specify the policy control and emulate, over a period of time, QoS parameters such as bandwidth, packet delay, jitter, and packet discard probability for media flows within an IP multimedia session. A certain network behavior, called scenario, can be stored and reused later. For emulating the properties of the channel, existing network emulators for IPv4 and IPv6 networks have been applied. The scenario-based approach allows reproducible performance measurements and running various experiments by using the same network behavior. A graphical user interface (GUI) has been developed to make the scenario specification more user-friendly. We demonstrate the functionality of the prototype emulation environment and analyze its behavior when controlling bandwidth, delay, and jitter in several experiments.

The overall idea of the paper is illustrated in Figure 1. Network behavior in a QoS-enabled IP (IPv4 or IPv6) network is controlled by the network administrator through the QoS Control. A QoS scenario, defined by the administrator, determines how the network parameters, such as bandwidth, delay, and jitter, would change in time. Once the experiment starts, the QoS Control runs the scenario, and uses the previously defined rules to change the network parameters. The policy control is realized by using COPS. This approach separates the application data flow from the control data flow. The QoS-enabled IP network can be emulated by different network emulators. We designed our emulation environment so as to be able to use both IPv4 and IPv6 emulators, with just a “lightweight” adaptation handler.

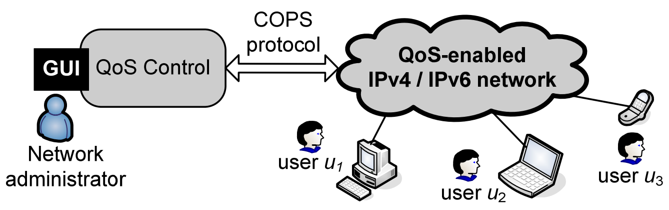


Figure 1 - Policy based QoS control over the network

This work is organized as follows. First, existing IPv4 and IPv6 network emulators are described. Next, we describe COPS and policy control, and continue with the description of our prototype emulation environment. We describe the model and its implementation in Java. Finally, we demonstrate the functionality by using a case study.

2. EXISTING NETWORK EMULATORS

Many network emulation tools have been developed to date. An overview of terms and generic structure of emulators, as well as a review of existing tools may be found in [3]. In this paper, we will focus on two tools used in this work. The IPv4 network emulation is based on well-known NISTNet network emulator. The NISTNet network emulator has been described as "a general-purpose tool for emulating performance dynamics in IP networks" [4]. It is implemented in Linux kernel, and it is able to emulate various performance scenarios, such as bandwidth limitation, tunable packet delay distributions, and congestion and background loss. It has an X Window System based graphical user interface. As of June 2007, this tool supports IPv4, but not IPv6. Thus, for IPv6 we used an IPv6 channel emulator ChaNet, developed in previous work [5] within our group. ChaNet is based on Linux Netfilter framework and Iptables, which enable control of packets which traverse through it. Packets are classified into flows based on their source and destination addresses and ports, and they are then forwarded into virtual channels, in which various network conditions (bandwidth, jitter, delay, and loss rate) are emulated. After passing through the channel, a packet is forwarded to the destination address. This tool is able to emulate several channels at the same time. Concatenation of channels enables routing the packets from one channel to another, creating a small virtual network. The user can set the static and dynamic packet classification rules. Static rules are written in a configuration file and applied when starting ChaNet. Dynamic rules consist of XML files which contain rules to be applied on the fly, which enables modifying of already created channels, as well as adding and deleting channels.

3. QOS POLICY CONTROL BASED ON COPS

As defined in [2], the COPS protocol is a client/server protocol, used to exchange policies between two entities, called the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP). In [6], a policy is defined as "the combination of rules and services where rules define the criteria for resource access and usage". The PDP is a logical entity that makes admission control and policy decisions in response to a request from a user who wants to access network resources. The PDP is typically a network server, on which the network administrator sets the policy rules. PEP is an entity which implements the policy by executing the resource reservation and policy control provided from the PDP. It is usually situated on a network router or layer-3 switch. PDP and PEP exchange policy information by using a COPS messages. In our model, the COPS framework and most of its rules were used to send control policies from the QoS Control and the QoS-enabled IP network.

4. PROPOSED MODEL

The requirements for the emulation environments were set as follows. First, the policy control over the emulated network must use the COPS protocol, which includes implementing all necessary entities, such as PDP and PEP. Network administrator must be able to create a scenario through a simple GUI, as well as to start it for each experiment this environment is set for.

Next, we wanted to be able to use any existing IPv4 and IPv6 network emulator tool, under some conditions: a tool must be able to create and dynamically modify the parameters of a single virtual channel. The emulator must have a means to control the parameters dynamically and from "outside", through an open interface. For example, this could be done by executing system commands, via an established control TCP connection, by exchanging XML messages, or in some other way. We use the NistNet emulator for the IPv4 network, and the ChaNet emulator for the IPv6 network.

Proposed model presents a unified way of creating and managing both IPv4 and IPv6 virtual channels. From the point of view of the network administrator, there is no difference in controlling different types of channels. The model, shown in Figure 2, consists of three components: Policy Decision Point (PDP), Policy Enforcement Point (PEP), and Channel Emulator.

4.1 Policy Decision Point

In our model, a PDP module is used by the network administrator to provide a complete control of network parameters such as delay, bandwidth, jitter, and loss rate. It is responsible for interacting with the administrator, making certain decisions, and creating policies which are passed to the PEP via COPS protocol. Our model of PDP module consists of four functional entities, which are described next.

User Interface is the module which enables the administrator to handle multiple virtual channels in different ways. For example, the administrator can specify a new channel and its behavior in time. The administrator can also store the channel behavior, or scenario, in the Scenario Repository, and retrieve it when necessary. After initially defining the channel properties, the channel may be created and the defined scenario may be initiated. The current channel configuration is displayed in the GUI, and the administrator can delete the channel at any time.

Scenario Repository module keeps all previously stored channels as well as their behaviors. Each channel is uniquely determined by its source and destination IP addresses and TCP/UDP ports. An example of a channel behavior is shown in Figure 3.

The parameters which can be controlled include bandwidth, delay, jitter, and loss rate. In Figure 3 we show

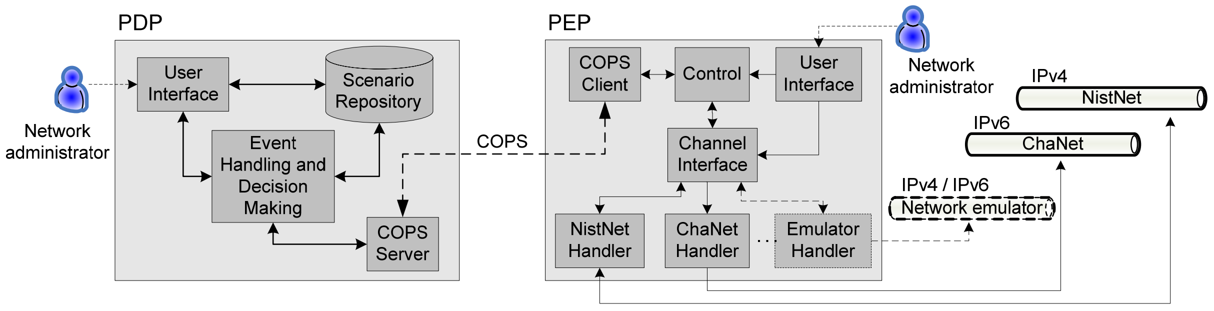


Figure 2 – Proposed model

the channel behavior as being divided into three intervals, each with its own network parameters. For example, time t1 is the time when the channel is created, with network parameters (delay) d1, (jitter) j1, and (loss rate) l1. When time t2 is reached, the channel will change the respective parameters to d2, j2, and l2.

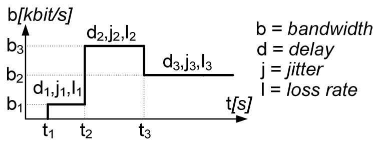


Figure 3 - Channel behavior

After specifying a new channel or uploading the existing scenario from the repository, the administrator can initiate the scenario by starting the Event Handling and Decision Making module. This is the main control unit, where certain decisions are made depending on the channel's behavior. After the decision is made, a trigger is sent to the COPS Server module containing updated parameters that should be sent to the PEP. There are three types of triggers: Create channel, Modify channel, and Delete channel trigger.

Create channel trigger is sent immediately after scenario execution is started, containing initial parameters which should be assigned to the channel. Event handling module then waits for a specified amount of time (t2–t1 in Figure 3), and then sends a Modify channel trigger with new parameters. This is repeated until the entire scenario is executed. The channel may be deleted by Delete channel trigger.

In order to send policy decisions from PDP to PEP, COPS protocol is used. The protocol is based on a client-server model: after receiving a request from the PEP, the PDP sends a proper answer by provisioning policies to be enforced on the PEP. The request contains client specific information like supported client type and context. In order to establish communication between entities in our model, COPS server and COPS client need to be run on the PDP and the PEP, respectively. COPS server receives network parameters from the Decision Making module, encapsulates them in COPS Decision message, and sends the message to the PEP. On the PEP side, policy information is extracted from the message and forwarded to the other PEP entities.

4.2 Policy Enforcement Point

In our model, PEP accepts new policies via COPS protocol, adapts them for the channel emulator in use, and forwards them to the emulator. Its main characteristic is extensibility, because it can be adjusted to work with any emulator which satisfies the requirements listed earlier in this paper. PEP consists of several modules.

User Interface module enables choosing which emulator to use when starting the PDP, selecting which PDP to connect to, and controlling the exchanged COPS messages.

COPS Client handles communication with the PDP and forwards incoming policies to the Control unit, which analyzes the policy, saves it and forwards to the Channel Interface module.

Channel Interface module realizes the idea of extensibility. It separates the content of the policy from the enforcement of the policy itself. The model does not "see" any specific emulator, but this interface. The interface contains three methods, one of each for adding, modifying and removing a channel. New types of emulators can easily be introduced by creating a handler which would implement this interface.

Depending on which emulators are used in our model, handlers for each of them have to be made. It is necessary because instructions for enforcing the same policy are emulator-specific. In order to be compatible with the model, each Emulator Handler must implement the Channel Interface module. The current implementation contains Emulator Handlers for two types of emulators: NistNet, an IPv4 emulator, and ChaNet, an IPv6 emulator.

4.3 Channel Emulator

NistNet and ChaNet emulators used in the model emulate IPv4 and IPv6 network, respectively. No modifications or other adjustments need to be made on them. It is only required that the selected emulator is up and running before starting the PEP module.

5. MODEL IMPLEMENTATION

The proposed model was implemented in Java programming language. The portability of Java enables testing of emulators designed for different operating systems. Handlers for two different emulators were implemented, as described in previous section. Graphical user interfaces were made for PDP and PEP to make policy handling more user-friendly. COPS client implements PR (Policy-Provisioning) [7] client-type, since it is the most suitable for controlling all the network parameters we need. Laboratory testbed needed to run the software is shown in Figure 4.

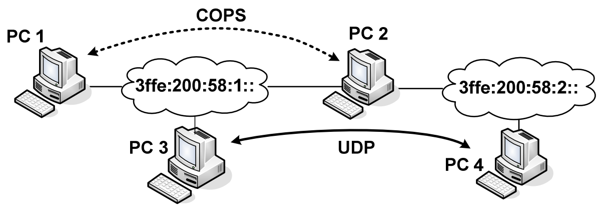


Figure 4 - A laboratory testbed

Figure 4 shows the experimental IPv6 network, which consists of two sub-networks connected with a router (PC2), on which a virtual channel is emulated. Network administrator controls the behavior of the virtual channel by using PDP graphical user interface on PC1. PC3 and PC4 are the host computers on which the application under study is placed. For the use case we made in this work, we used two applications for performance measurements, ping6 and iperf. Standard Linux ping6 tool measures Round Trip Time between two end nodes in the IPv6 network. It is used for measuring delay and jitter while executing the experiments. Iperf [8] tool was used for measuring bandwidth between the end nodes. Table 1 shows computer configurations in the testbed as well as the entities installed on each of them.

Table 1 - PC configurations

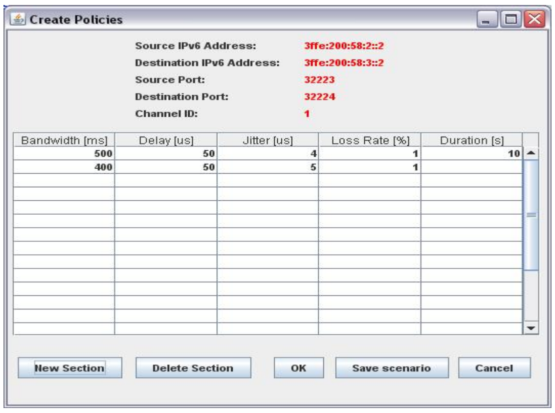
	CPU [MHz]	RAM [MB]	OS	Installed entities
PC 1	1600	512	Win XP	**PDP**
PC 2	600	512	Mandrake 10.1	**PEP, ChaNet, NistNet**
PC 3	400	64	Mandrake 10.0	**Iperf client, Ping6**
PC 4	2600	512	Mandrake 10.1	**Iperf server, Ping6**

6. CASE STUDY

In order to demonstrate the use of the emulation environment, we present a case study involving policy-based control of bandwidth, delay, and jitter. A network testbed was established in a laboratory environment as shown in Figure 4. ChaNet was running on a PC-based router PC2 between two sub-networks. It intercepted packets from PC4, sent them through the virtual channel, and forwarded to PC3.

For each measurement a different channel behavior was created and stored in the Scenario repository. Figure 5 presents a PDP graphical user interface used for creating a new scenario. At the top of the window channel characteristics are shown. Each line of the table presents a scenario interval with its specific network parameters. In this example, the channel has initial bandwidth of 500 kbit/s, which is decreased after 10 seconds to 400 kbit/s. Jitter is also increased for 1 µs.

For each channel a new thread is created on the PDP, which starts to run once the administrator initiates the scenario. PEP controls ChaNet by sending XML messages from the ChaNet handler, which contains information extracted from COPS messages.



Source IPv6 Address: 3ffe:200:58:2::2
Destination IPv6 Address: 3ffe:200:58:3::2
Source Port: 32223
Destination Port: 32224
Channel ID: 1

Bandwidth [ms]	Delay [us]	Jitter [us]	Loss Rate [%]	Duration [s]
500	50	4	1	10
400	50	5	1	

Figure 5 - Channel behavior interface

6.1 Bandwidth measurement

We used the Iperf tool for bandwidth measurement. Iperf server generated the UDP traffic with specific bitrate for the duration of 10 seconds, and Iperf client measured the received traffic. The channel through which the packet flow was routed was specified by using ChaNet. We defined a scenario in which the channel bandwidth was increased from 300 kbit/s to 500 kbit/s in steps of 50 kbit/s every 20 seconds. After reaching 500 kbit/s, bandwidth was not increased any more. Delay, jitter, and loss rate were set to zero during the scenario. At the same time, UDP flow bitrate generated by Iperf server was manually increased every 20 seconds from 300 kbit/s to 850 kbit/s in steps of 50 kbit/s and duration of 10 seconds, and measured on Iperf client. The result is shown in Figure 6.

It may be noted that the amount of the received traffic is the same as the amount of the sent traffic as long as the channel bandwidth is below the threshold of 500 kbit/s. When the sent bitrate reaches 500 kbit/s, the channel begins to discard all superfluous packets.

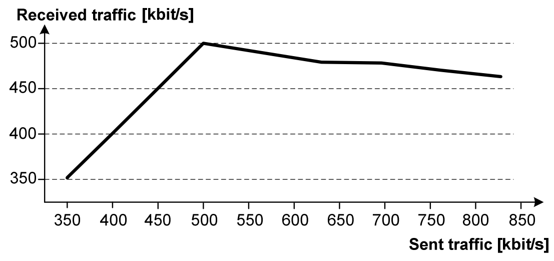


Figure 6 - Bandwidth measurement

6.2 Delay measurement

For delay measurement we started ping6 tool on the server, with the channel's behavior defined as follows: Bandwidth was set to fixed value of 100 kbit/s, jitter and loss rate were set to zero, and delay was increased every 20 seconds from 0 ms to 0.5 ms, 1 ms, 2 ms and 5 ms. The results are shown in Figure 7. It is easy to see how the round trip increases as the channel characteristics change. For a given delay value, the delay variation is very small since jitter value was set to zero.

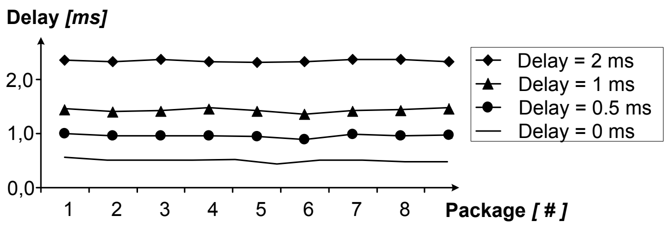


Figure 7 - Delay measurement

6.3 Jitter measurement

The measurement in this case followed the same procedure as in the previous measurement, with a new channel behavior. Bandwidth was set to 100 kbit/s, loss rate was set to to 0%, and delay was set to a fixed value of 1 ms. The jitter was increased every 20 seconds from 0 ms to 0.4 ms and 0.7 ms. The results are shown in Figure 8. We can see how jitter affects delay of packets. By increasing jitter, RTT scattering increases as well.

In the described measurements only one QoS parameter was varied, while other parameters were fixed. To create realistic scenarios, all parameters can be combined in a single scenario, with the QoS characteristics as needed.

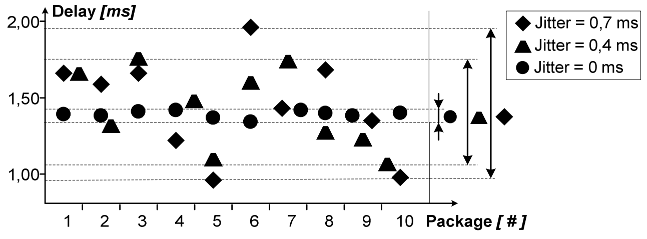


Figure 8 - Jitter measurement

7. CONCLUSIONS AND FUTURE WORK

We presented a model and a prototype implementation of a network emulation environment for policy-controlled QoS scenarios. The implementation uses the existing ChaNet and NistNet emulation tools. The case study demonstrated the functionality of the emulation environment by showing how the virtual channels can be configured and used for performance measurements. The performance of the emulation environment in terms of accuracy for a given scenario was satisfactory.

Future work will explore behavior of various audio and video multimedia applications in a laboratory environment, while controlling the network parameters.

ACKNOWLEDGEMENT

This work was carried out within the research project 036-0362027-1639 "Content Delivery and Mobility of Users and Services in New Generation Networks", supported by the Ministry of Science, Education and Sports of the Republic of Croatia.

REFERENCES

[1] G. Camarillo and G.-M. Miguel-Angel: "The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds", John Wiley & Sons, 2004.

[2] D. Durham, J. Boyle, R. Cohen, S. Herzog., R. Rajan, A. Sastry: "The COPS (Common Open Policy Service) Protocol", IETF RFC 2748, Jan. 2000.

[3] D. Herrscher, K. Rothermel: "A Dynamic Network Scenario Emulation Tool", Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002), pp. 262–267, Miami, FL, USA, October 2002.

[4] NIST Net, the Network emulation tool, http://www-x.antd.nist.gov/nistnet/

[5] H. Komerički, V. Levačić: "ChaNet – IPv6 channel emulation tool", Proceedings of the 13th IEEE Mediterranean Electrotechnical Conference MELECON 2006, pp.709-712, Benalmadena, Malaga, Spain, May 2006.

[6] R. Yavatkar, D. Pendarakis, R. Guerin: "A Framework for Policy-based Admission Control", IETF RFC 2753, Jan. 2000.

[7] K. Chan, J. Seligson, D. Durham, S. Gai, K. Mccloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith: "COPS Usage for Policy Provisioning (COPS-PR)", IETF RFC 3084, Mar. 2001.

[8] Iperf, the TCP/UDP Bandwidth Measurement Tool, http://dast.nlanr.net/Projects/Iperf