Paper

Comparison of Resource Control Systems in Multi-layer Virtual Networks

Bartłomiej Dabiński, Damian Petrecki, and Paweł Świątek

Institute of Computer Science, Wrocław University of Technology, Wrocław, Poland

Abstract—This paper describes the performance of various methods of QoS assurance for each connection in an environment composed of virtual networks and dedicate end-to-end connections inside them. The authors worked on the basis of research conducted with the use of the authorial network management system named Executed Management, which uses resources virtualization platforms VMware and Mininet for testing purposes. We briefly describe our system and techniques we used and some alternatives we tested and discarded because of their limitations. Functionality and performance of proposed solution to widespread implemented mechanisms as OpenFlow and MPLS are compared. Reasons for selecting well-known techniques to isolate networks and limit bandwidth on different levels of virtualization are considered. The purpose of this paper is to show out our studies and performance we achieved.

Keywords—Execution Management, MPLS, network virtualization performance, OpenFlow, parallel networks, virtual networks, virtual networks performance.

1. Introduction

The purpose of the project was to build a system that configures network in order to satisfy QoS requirements for large number of applications. The applications run in multiple virtual, isolated networks that share a single physical network. The system has to create connections inside these networks, with specified paths and guaranteed bandwidths, dynamically in response to applications' requests. It means that the system must be aware of network content. Only well proven and commonly implemented algorithms, protocols and techniques were to be used because the system should operate in the network built with generic equipment. This paper describes selected resources virtualization method and compares its performance to solutions providing similar capability.

Many techniques may have been applied in order for above mentioned goals to be achieved. First step of desribed work was to compare features of Multiprotocol Label Switching, IEEE 802.1ad (QinQ), IPv6-in-IPv6 tunneling and Provider Backbone Bridging. The authors decided to use VLANs to isolate virtual networks and to run dedicated virtual or physical machines for handling ISO OSI L3 networks inside these L2 networks. Then we decided to build the centralized system to manage network resources. The task of the system was to handle requests of applications and to create dedicated tunnels inside virtual networks in reply to the requests. Each connection between end nodes and each virtual network must have guaranteed bandwidth. A variety of traffic engineering methods were tested and the shaper of *tc* linux application was selected. The traffic of an end-to-end connection is limited in virtual interfaces (end-to-end tunnel entry points) by TBF classless queuing discipline. The traffic of virtual networks is limited in physical interfaces that send traffic into the network. u32 classifier filters packets belonging to a specific VLAN and the traffic is enqueued to the HTB classes to limit its bandwidth. Furthermore, the system ensures uninterrupted transmission without delay variation in case of tunnel path or bandwidth modification.

The laboratory network was built using VMware ESXi server (Debian hosts run in virtual machines), Mininet service [1], [2], Juniper EX4200 switch and two MikroTik RB800 devices. Mininet was designed as a network emulator for testing OpenFlow but we made some modification to make Mininet meet our needs. MikroTiks was used as a precise bandwidth and latency testing tools.

The authors experimented with the performance of the system both within the scope of delays in connection handling (creation, removal, modification) and within the scope of QoS ensuring, which means guaranteeing bandwidth requested by user's applications and preserving low packets flow latency in case of existing connections reconfiguration. In order to point advantages and disadvantages of proposed solution its functionality was compared to widespread implemented mechanisms like OpenFlow and Multiprotocol Layer Switching (MPLS).

In order to choose the best solution capability of MPLS, QinQ and Provider Backbone Bridging (PBB) along with suitable traffic shaping and policing techniques was analyzed, before we decided to modify the solution described in [3], [4] to meet our requirements. This solution uses VLANs and IPv6-in-IPv6 tunneling.

MPLS was dismissed due to the fact that most of its implementations in modern network equipment do not fully support IPv6. For deployment of MPLS and IPv6,

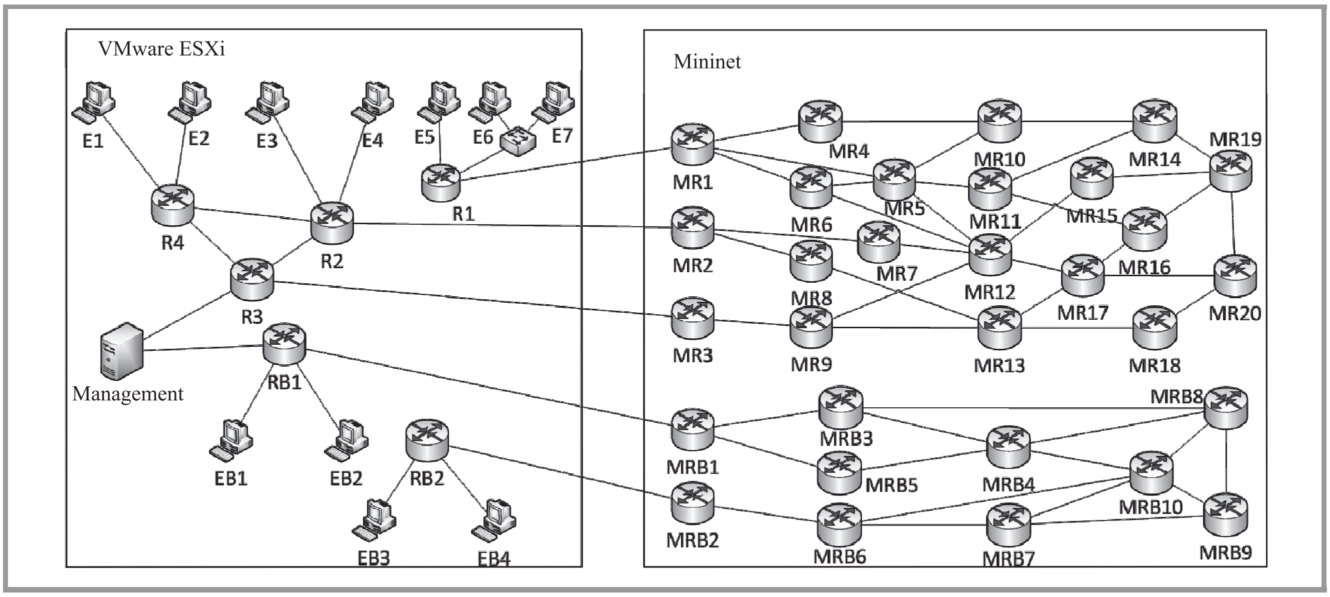


Fig. 1. Testing topology.

the protocol stack has usually to include following elements: MPLS, IPv4, VPLS and then IPv6. Furthermore, MPLS does not solve the problem of two applications, when each of them needs an end-to-end connection with the same pair of hosts using different QoS requirements. On the other hand, MPLS was a strong candidate because it offer simplest, fully automated configuration of the end-to-end connection. It does not require our system connect to and configure each node on the path separately.

QinQ seemed to be easy in implementation due to widespread support of this standard but it has several limitations, which are crucial in the context of this work. The most important disadvantage is a necessity to block communication of the host with the entire network, while the host is connected to another host. It is an implication of the requirement that end-to-end connection must be isolated and we cannot require end hosts to implement QinQ. Then, it is impossible to fulfill these conditions and handle separate, concurrent end-to-end connections for multiple applications on a single host.

Provider Backbone Bridging might perform well in the core network but it is Layer 2 protocol, so it is difficult to distinguish between multiple applications on a single host. Furthermore, PBB is a novel, advanced standard and unsupported by most of devices available for our research. We tried to used PBB to our purposes but it has almost all MPLS disadvantages and adds some more because of lack of support in general equipment.

Chosen Virtualisation Method for Isolated Parallel Networks

For creating parallel networks VLANs was used. End nodes were connected to specified VLANs and thus they can communicate only within definite part of the network. That end nodes belong to a single virtual parallel network unless they have multiple network interfaces. End nodes in different networks may have a common physical gateway but traffic intended for specified virtual networks have to be forwarded to proper virtual gateways. The end nodes can also be connected to external, unmanaged networks without QoS warranties (such as GSM). In above mentioned case we assume that first managed by us router on the connection path is the connection gateway and that node receives traffic with proper VLAN tagging. All physical gateways forward traffic to proper logical gateway by VLAN tagging performed on switch to which end nodes are directly, physical connected. The use of physical router for serving the virtual network is possible in the case when it has routed VLAN interfaces or deal with only one virtual network. It is strongly discouraged except the situation when it is not possible to tag packets that come to the router from end nodes.

The virtualization of the level 1 of the core network looks similar. The packets that belong to a particular network are sent from a logical gateway and are tagged by a physical node that hosts the logical router. In the next physical node analogous actions are performed. Based on tags, received traffic is forwarded to appropriate virtual router, which serves a particular network. In this way networks isolation is assured and delegation of virtual network management is possible. Network managing for external entities such as clients that would buy one virtual network can be delegated and have full control of virtual routers in theirs network.

There is also possibility to create a section of the logical network (core network, edge routers and end nodes) on a single physical device by intentional configuring virtual logical connections between virtual machines, with the use of resources virtualizer. As you can see in Fig. 1, we use that to create testing purpose network with two physical hosts only.

When configuring the network, it is required to use a dynamic routing protocol (in our case it is OSPFv3) both between physical nodes and within virtual networks for logical nodes. It ensures full reachability of all the hosts and also communication for the management system and the nodes on the all virtualization levels.

The virtual networks of first level in the testing topology are built with the use of VMware ESXi virtualization platform, physical host with Mininet software, Juniper EX4200 switch and two MikroTik platforms RB800. ESXi served for virtualizing several access networks, end users' machines and virtual management server. Mininet was used in order to create two virtual backbone networks made of routers running Ubuntu operating system. The MikroTik devices were connected as physical end nodes for simulating users' computers and generating traffic for benchmark. The MikroTiks, through the Traffic Generator tool, which is built in RouterOS, collected statistics from transmitted/received data: packet loss rate and packet latency distribution. VLANs indicate which logical routers from access networks (ESXi) are allowed to communicate with specific logical routers in the core network (Mininet). In order to control Mininet host's incoming traffic, we had to modify Mininet scripts and create in this way properly defined virtual network with willful assignment of interfaces, addresses and connections. The Juniper EX4200 switch connects physically (Gigabit Ethernet) and logically (VLANs) the EX4200 machine and the host with Mininet. The testing topology consists of two parallel networks, whose only common point is a management node, which must be able to communicate with both virtual networks. The topology is presented in Fig. 1.

2. Network Virtualization

2.1. Chosen Method of End-to-end Connections Virtualization

End nodes operate in specific parallel network, so they can communicate with all other devices in this parallel network. At the time when they need to communicate, they do not start transmission directly but one node sends request to the Execution Management server. The management system analyzes QoS requirements for the new connection and available network resources and then it creates a proper IPv6-in-IPv6 tunnel. At the end of this process the application is informed that dedicated connection was established and it can start transmission. Virtual connections of second level are tunnels, which are dynamically created for each application use case. These tunnels are set on edge routers – the gateway routers for end nodes inside parallel networks. We do not impose any (non-standard for a generic IPv6 network node) requirements to end nodes owing to the approach in which traffic is directed to the tunnel by the mechanism implemented entirely in edge routers. This mechanism bases on packet filtering by following IPv6 and TCP headers fields: source address, destination address, source port, destination port. It is important that a computer, in case of having multiple active IPv6 addresses, uses for communication only the IPv6 address that was included earlier in the connection establishing request.

In order to preserve integrity of the network, the tunnel is configured on virtual dedicated interfaces, which are subinterfaces of loopbacks on edge routers. The addressing schema within tunnels is calculated with the use of 48-bit subnet divided into 126-bit mask subnets. The external addresses of the virtual interfaces use reserved pool of routed addresses. It is worth to mention that authors used the potential of IPv6 addressing which offers enough addresses to assign separate addresses for each parallel network. The external addresses of the tunnels are used to route the packets of its tunnel through the fixed path. Owing to this approach we are able to control tunnel path easily, by applying static routing based on destination IPv6 only. It is significant advantage since we can use any generic router in the core network and we do not cause high CPU utilization.

2.2. Limiting of Network Resources

The first part of network resources managing was to limit the bandwidth of entire parallel network. Except assigning resources for particular networks, it was necessary to reserve some bandwidth for management traffic and OSPFv3 packets. The mechanisms built into networking hardware to limit bandwidth of an interface and a software tool that implemented HTB algorithm was used, in which appropriate classes based on VLAN tags were created. Then bandwidth limits with these classes was associated.

It is advised for an administrator of an isolated network to use traffic shaping or policing mechanisms for isolating the class of traffic for signaling purposes. In order to achieve this, the mechanism that classifies all the traffic that is not an IPv6-in-IPv6 tunnel (by checking corresponding IPv6 header field) and guarantees bandwidth for this class is proposed.

The remaining traffic belongs to IPv6-in-IPv6 tunnels. Due to the use of dedicated virtual network interfaces in edge routers bandwidth of given tunnel can belimited simply by one classless *tc qdisc* discipline (e.g., by recommended Token Bucket Filter – TBF) configured on this virtual tunnel interface. This approach does not require traffic classifying. Centralized management of dedicated connections, in the networks which resources is known, guarantees that all the connections have sufficient resources and thus QoS is ensured. If a new connection would exceed the capacity of the network, the system simply rejects the connection and informs requesting application about the reason.

The variant possibilities of tunnel bandwidth limiting are briefly described in Section 5.

3. Management System

The management system, which was used for testing, was built especially for such purposes. It is made up of the Execution Management module, a database, and a supporting QoS module. The database stores i.a. the state of the network, which is data concerning all the nodes and connections on all the levels of virtualization. This is used to find an appropriate path for a new connection.

The system receives requests via a web application (that was built for an administrator for network managing purposes) or directly from applications via XML messages. For communication with the management system a dedicated traffic class with guaranteed bandwidth for signaling is used. In order to make an application work independent on the network management system, it is possible to use intermediary layer. An example of such deployment is described in [4], [5]. After receiving a request, the system verifies it and sends a query for an eligible path to the module that finds a path satisfying QoS. Then the system receives a reply and prepares scripts, which are subsequently sent parallel to network devices in order to configure proper services.

The connection path may be modified in the case of increased QoS requirements or when given connection have to be released because of a resources rearrangement. Such situation occurs when a new connection must not be established unless the resources already occupied by another connection are released. This connection may be routed another path that also satisfy its QoS. This process is fully automated and is imperceptible for the user. The process has following steps: creation of a new tunnel with new addresses and a new path, redirection of the traffic from the old tunnel to the new tunnel, removal of the old tunnel. The connection may also be removed in response to a removal request from the application, which used this connection.

More about Execution Management and cooperating modules can be found in [6].

4. Performance Evaluation

4.1. Execution Management Performance

The performance of the management system is an important factor influencing QoE because it determines the time of connection establishment. Therefore, it was crucial to comply with ITU-T recommendations [7].

The tests were performed to determine the behaviour of the system for a large number of queries and under different load (from 1 up to 100 requests). As expected, the system was stable and retained full data integrity, even in the case of multiple simultaneous requests for the creation, modification, and removal connections. Each series of tests were performed using 100 requests for a tunnel creation, 100 requests for removal and 50 requests for modification. Test results are averaged over the repetition of each series (1, 10 or 100 simultaneous requests) 50 times. Requests for testing were randomly generated in network with 40 end nodes to test system behaviour in large network. All time values showed in Figs. 2–4 are measured in miliseconds.

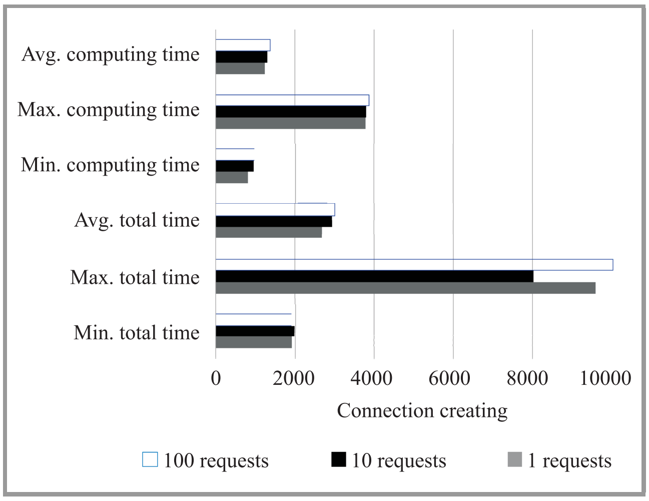


Fig. 2. Execution Management engine performance 1.

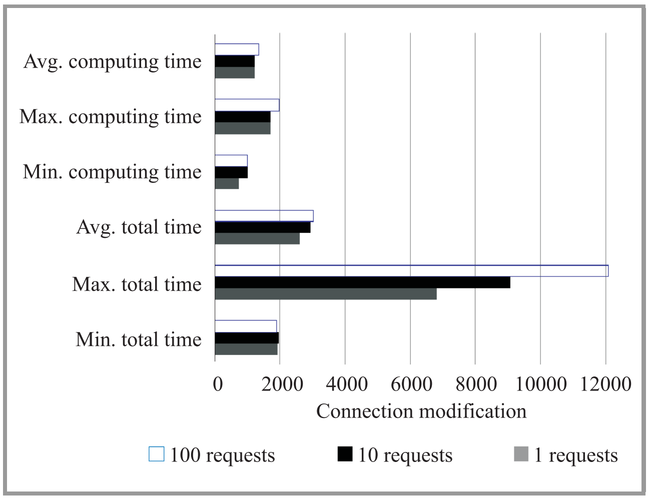


Fig. 3. Execution Management engine performance 2.

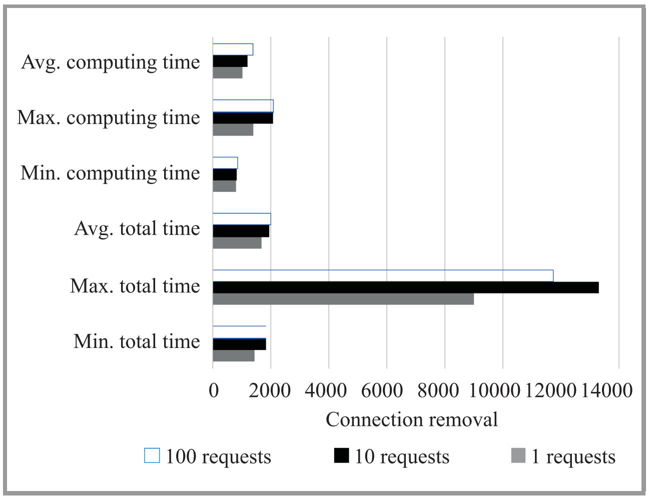


Fig. 4. Execution Management engine performance 3.

Approximately 80% of the processing time in the case of creation or modification, and up to 92% of the removal time is consumed by database operations. Albeit these numbers are significant, the result is much better when comparing to the previously used external database implementation. It probably can be further improved by creating our own library for handling database queries created by EM.

The total execution time depends primarily on the status of the network rather than on the performance of the virtualization tools. In some cases the times were very high due to temporary high load of Mininet host network operator or VMware Server. In such situations, it dramatically lengthened the waiting for SSH connection to the nodes.

Tests exclude situation when two requests require a connection via SSH to the same node. In this case, the waiting time for connection can be extended up almost double. This is due to the sequential SSH connections handling by a device. If there is a need for multiple connections to the same device in a single request (e.g., to configure two static routes or route and tunnel), all commands to be sent are combined in one. However, there is not a mechanism responsible for this in the case of multiple requests that have to be configured on the same node because it would starve scripts from earlier demands by continually appended subsequent commands.

There is not necessary to run the tunnel removals in parallel because the latency caused by removal process is not significant. Then each network is removed sequentially.

4.2. Performance of Network Mechanisms

This part of the chapter concerns the performance of implemented virtualized network environment. For testing MikroTik Traffic Generator tool was used. This is an advanced tool built into RouterOS that allows to evaluate performance of DUT (Device Under Test) or SUT (System Under Test). The tool can generate and send RAW packets over specific ports. It also collects latency and jitter values, Tx/Rx rates and counts lost packets.

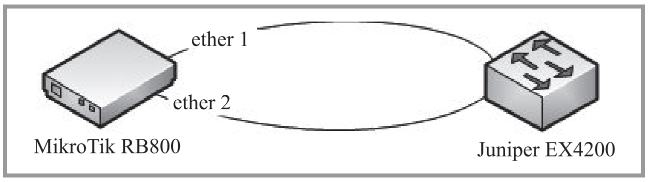


Fig. 5. Topology for testing MikroTik RB800 and Juniper EX4200 switch.

The goal of the first experiment was to examine the throughput of MikroTik RB800. We built the scenario as shown in Fig. 5. One stream of packet to examine unidirectional maximal throughput in half duplex (when network works stable) and two parallel streams of packets to examine bidirectional maximal throughput (full duplex) were used. The real maximal transmission rates were higher but connection was unstable. There were high packets losses and high latency. We arbitrary assumed that acceptable packet loss is 0.1%. Transmissions that have more packet loss were not considered. Results are presented in Table 1. The values are rounded down to nearest 5 Mbit/s. Each test lasted for 90 seconds.

Table 1
Results of performance tests of MT RB800 and Juniper EX4200

Direction	Packet size [bytes]	Throughput [Mbit/s]	Latency
ether1→ether2	1500	980	Min: 109 μs Avg: 6.5 ms Max: 11.5 ms
ether1→ether2 ether2→ether1	1500	760	Min: 32 μs Avg: 255 μs Max: 1.35 ms
ether1→ether2	100	240	Min: 23 μs Avg: 62 μs Max: 773 μs
ether1→ether2 ether2→ether1	100	100	Min: 21 μs avg: 64 μs Max: 883 μs

The subsequent test concerned the performance of the laboratory virtual networks. The network with two MikroTik RB800 platforms connected was used as shown in Fig. 6. The logical topology is presented in Fig. 7. Two RB800 devices are used since single RB800 could be a bottleneck in some cases. The results are presented in Table 2.

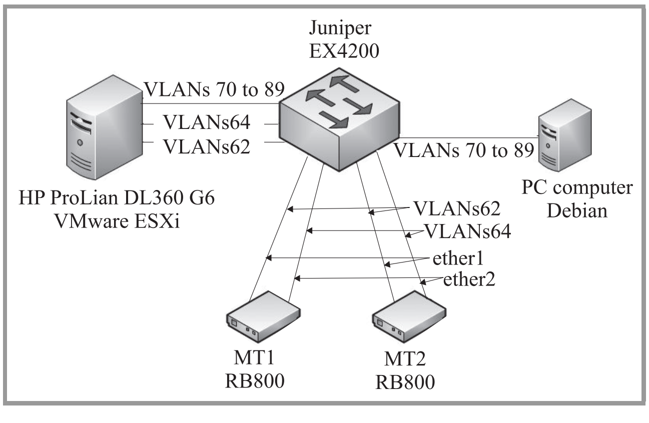


Fig. 6. Physical topology for tests.

The results show that the performance of examined virtual network is comparable to the maximal throughput of physical devices when transmitted packets are large (1500 bytes, which equals MTU size). The significant difference in the case of two-way traffic is caused by the fact that the traffic that flows in one direction passes the physical link between ESXi and Debian twice. Thus the real throughput in this link is doubled.

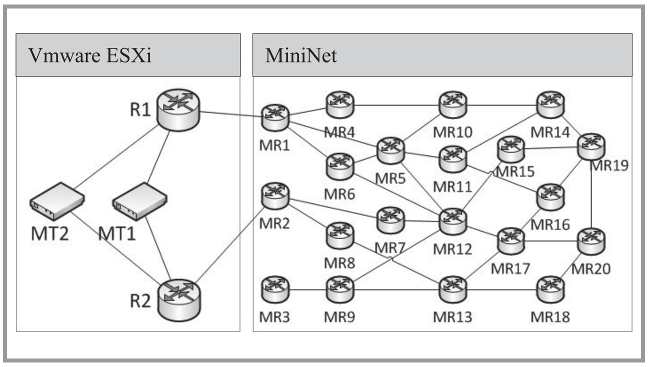


Fig. 7. Logical topology for virtual environment tests.

Table 2
Results of virtual networks performance tests

Paths	Packet size [bytes]	Through-put [Mbit/s]	Latency
R1→MR1→MR5→ MR12→MR7→ MR2→R2	1500	840	Min: 471 μs Avg: 1.9 ms Max: 15.2 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→MR12→ MR5→MR1→R1	1500	420	Min: 397 μs Avg: 2.2 ms Max: 13.5 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2	100	40	Min: 107 μs Avg: 253 μs Max: 8.14 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→MR12→ MR5→MR1→R1	100	40	Min: 110 μs Avg: 354 μs Max: 10.4 ms

In the case of small packets (100 bytes) the difference is much bigger. It is caused by the fact, that this testing scenario involves much higher packets per second rates and each packet has to be served by each virtual router (7 times) so the CPU performance of the virtualizers is the bottleneck.

The second parts of tests concern the IPv6-in-IPv6 tunnel and bandwidth limiting mechanism for this tunnel. The logical topology of this scenario is presented in Fig. 8. The bold lines indicate the tunnel path. The routers R1 and R2 are tunnel entry/exit points. Table 3 presents the test results. The maximal packets used for these tests were smaller comparing to the packets in previous tests because the MTU of a tunnel is decreased by the additional IPv6 header (40 bytes). Nevertheless, the presented throughput does not involve these additional 40 bytes of data.

With default configuration of IPv6-in-IPv6 tunneling in Debian 6, the displayed MTU of a tunnel interface is 1460 but the real MTU is 1452 bytes. These missing 8 bytes was reserved for an encapsulation limit extension header, which was confusing since this extension header was not transmitted. We explicitly disabled encaplimit in order to transmit full 1460 bytes packets.

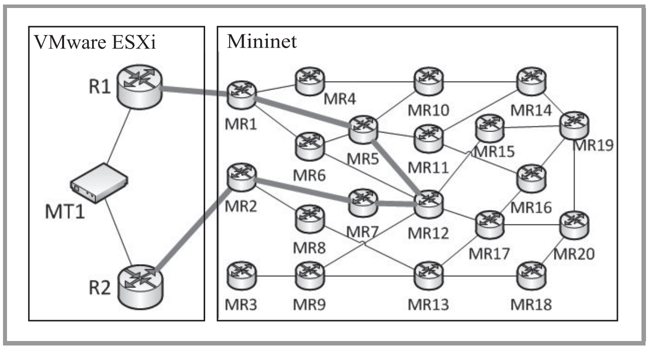


Fig. 8. Logical topology for tunnel testing

The *Transmission rate* column of the Table 3 means the fixed rate at which the MikroTik was sending data. *Tunnel bandwidth* means the value of tunnel bandwidth limit. *Throughput* is the rate at which packets were coming back to the MikroTik (after passing the tunnel).

The results presented in Table 3 are comparable to the results presented in Table 2. It means that implemented by us mechanisms does not introduce significant overhead to packet processing. The drop of the throughput is most noticeable in case of small packets and it is about 12.5%.

5. Bandwidth Limiting Methods

5.1. Limiting Bandwidth of Virtual Network

VLAN interfaces were chosen for implementation of parallel networks, which determined the methods of bandwidth limiting that might be used. It was important to limit bandwidth of virtual networks from outside of these networks. This implicates that virtual networks administrators do not need to take any action to limit bandwidth of their network. Because the limits are on different level of virtualization, they are invisible for the administrators and they cannot be exceed.

Ensuring bandwidths for VLANs was implemented by setting bandwidth limits for all the VLAN interfaces on the node. Specific implementation is equipment dependent but almost each carrier-class switch or router is capable to perform this task and such implementation is fairly easy.

5.2. Limiting Bandwidth of End-to-end Connection

Recall that all the functions concerning QoS are performed on edge routers, which are gateways for end users. It is true

Table 3
Results of tunnel performance tests

Paths	Packet size [bytes]	Transmission rate [Mbit/s]	Tunnel bandwidth [Mbit/s]	Throughput [Mbit/s]	Latency
R1→MR1→MR5→ MR12→ MR7→ MR2→R2	1460	950	790	825	Min: 1 s Avg: 1 s Max: 1.06 s
R1→MR1→MR5→ MR12→MR7→ MR2→R2	1460	790	790	790	Min: 494 μs Avg: 1.2 ms Max: 9.4 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→ MR12→ MR5→MR1→R1	1460	950	410	412	Min: 1 s Avg: 1 s Max: 1.03 s
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→ MR12→MR5→MR1→R1	1460	410	410	410	Min: 353 μs Avg: 4.5 ms Max: 19.4 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2	100	400	35	35	Min: 1.01 s Avg: 1.01 s Max: 1.01 s
R1→MR1→MR5→ MR12→MR7→ MR2→R2	100	35	35	35	Min: 137 μs Avg: 514 μs Max: 7.6 ms
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→ MR12→ MR5→MR1→R1	100	200	35	35	Min: 1 s Avg: 1 s Max: 1.02 s
R1→MR1→MR5→ MR12→MR7→ MR2→R2 R2→MR2→ MR7→ MR12→ MR5→MR1→R1	100	35	35	35	Min: 149 μs Avg: 624 μs Max: 36.2 ms

in our case but the architecture of our system does not disenable the use of separate, hardware traffic shapers, which is the case in many professional applications. Guarantee of the bandwidth for end-to-end connections in described implementation is achieved by limiting the bandwidth for the IPv6-in-IPv6 tunnel interfaces. It is a mechanism similar to the above mentioned mechanism for limiting the bandwidth for VLANs: in both cases we limit the bandwidth of a virtual interface. Nonetheless, the actual implementation may vary significantly because handling VLANs is usually performed by switches and IPv6 tunneling by routers since most switches (even with L3 support) are not capable of IPv6-in-IPv6 tunneling.

We focused on implementation end-to-end connection bandwidth limit in Debian 6 OS. A *tc* tool for performing traffic control was chosen because this is a very efficient and common tool, almost each traffic control application in linux bases on *tc*. Indeed, *tc* is very powerful and has functions that meet the requirements.

We used Token Bucket Filter (TBF) classless queuing disciplines. It suits our requirements best because we do not need hierarchical structure offered by classful disciplines (we have only one class in one interface, without involving priorities). TBF is less CPU algorithm, which is important advantage for us because the tests showed that in some cases the performance of virtualizer CPU is bottleneck, because the host CPU is engaged in not only virtual CPUs virtualization but in virtual network adapters too. These advantages make TBF the recommended qdisc for limiting the bandwidth of the entire interface.

Here is an example configuration for the tunnel bandwidth limiting in Debian 6 for the tunnel interface named *tunnel*. The *rate* parameter is our bandwidth limit.

```
tc qdisc add dev tunnel root tbf
rate 2 Mbit latency 1000 ms burst 15000
```

Please note how *tc* calculates units:

1 Mbit = 1000 Kbit = 1000000 bps

It is commonly confused that specified rate of 1 Mbit equals $1024 \cdot 1024$ bytes.

The *latency* parameter limits the buffer of the algorithm. It means that packets that would wait longer than 1000 ms are simply dropped. Instead of *latency*, it is possible to use *limit* parameter, which means the size of the buffer in bytes. These two parameters are mutually exclusive.

The *burst* parameter is the size of the bucket, in bytes. We increased this value from default 5000 to 15000 in order to let algorithm handle high traffic rates (above 100 Mbit/s). On the other hand, if buffer is too large, the algorithm's precision recede considerable. We chose the value 15000 by experiments.

The only problem concerning traffic control that we were not able to solve is the lack of preciseness when the rates are very high (especially higher than 500 Mbit/s). It is the result of the fact that traffic control is performed by CPU that operates with non-zero time slots and it follows with finite resolution of traffic control mechanisms. The lack of precision is up to 5% of declared rate while the value is 800–1000 Mbit/s. In real applications it does not seem to have serious implications because such large bandwidth rates are rarely reserved for a single end-to-end connection. Yet even in this situation, the QoS is still ensured if 5% of mechanism inaccuracy is calculated in bandwidth allocation plan. If one needs perfect precision with high transmission rates, it is advisable to use a high-class hardware traffic shaper.

6. Performance Comparison and Conclusions

6.1. Comparison of the Performance of the System with the Performance of Alternative Solutions

The authors did not conduct experiments but used the results delineated in [8] in the case of OpenFlow and in [9] in the case of MPLS. Regarding to benchmarking OpenFlow, two factors are to be considered. The former is the performance of a controller and the latter is the bandwidth of switches.

Table 4 presents the performance of different controllers running in one-thread mode on a highly efficient machine with 16-cores processor AMD, 40 GB RAM memory and Debian Wheezy operating system, which hosted simultaneously 16 switches with 100 unique MAC addresses each [8]. The controllers were limited to one-thread operation because the Mirage controller does not support operations on many cores simultaneously.

Table 4
Performance of OpenFlow controllers

Controller	Average throughput [Mbit/s]	Average latency [ms]
NOX fast	122.6	27.4
NOX	13.6	26.9
Maestro	13.9	9.8
Mirage UNIX	68.1	21.1
Mirage XEN	86.5	20.5

As shown in Table 4, the performance of different controllers may vary widely, up to 10 times. It means that the system in the worst case may be not able to handle a new flow that is directed to the controller, hence it will not ensure QoS. Execution Management does not have this issue because it informs an application whether the connection may be established. The application may start to transmit only when a dedicated tunnel with guaranteed bandwidth is active. Due to this proceeding, an end user waits a little bit longer to start for example VoIP conversation, but after the connection is established he has ensured sufficient bandwidth for a VoIP transmission with required quality.

The second, important dimension of the performance is the throughput of devices. In the case of OpenFlow use, it is required to covey to the controller instructions for dealing with a particular connection. It requires the middleware that would communicate with applications and the controller. In such case, the throughput is not constrained by switches. Due to the hardware handling of traffic on low implementation level, the performance of OpenFlow switches does not diverge from the performance of a generic hardware switch. Another issue is limiting resources. The newest OpenFlow version 1.1.0 enables the use of shapers that are built in switches, which means that the mechanism is capable of limiting bandwidth according to requirements even if there are large number of separate traffic flows.

Execution Management uses classles queuing disciplines of TBF type to limit bandwidth of tunnels. It works well for low bandwidth rates. When large bandwidth is configured (more than 500 Mbit/s) it is possible that mechanism would accept the transmission with higher rate than configured. The order of magnitude of the difference is several percent and it depends on configured bandwidth limit. The general rule is that the higher limit is set the bigger is inaccuracy, although slight anomalies may occur while experimenting with narrow range of bandwidth values. We increased the *burst* parameter of TBF discipline (using *tc qdisc* tool) to 10 times MTU size, which cause that the real maximal bandwidth is always equal or higher than configured value. It means that application always has its requested bandwidth and sometimes it may have slightly larger bandwidth than requested. This difference (5% of configured value) is included in QoS arrangement of all the available bandwidth of the network.

In the case of applying OpenFlow, the accuracy of bandwidth limiting depends on the accuracy of traffic control mechanisms implemented in particular device that was chosen by the controller, providing that on the path there is a device that implements traffic control mechanisms.

The second mechanism chosen for comparison is MPLS. In this case there is also lacking a module that allows application to communicate with a network management system in order to establish requested connections automatically. Furthermore, MPLS does not have measures to distinguish between different applications running on a single host. On the other hand, the performance tests shows advantage of MPLS over other solutions. Due to the hardware implementation of packets switching in network nodes, the performance of MPLS based network is the same as the performance of routed network (with hardware routers).

For traffic management LDP and RSVP protocols might be used. In case of applying both protocols for path and bandwidth managing it is possible to achieve our goals (limiting bandwidth of end-to-end connections, determining the connection's path) but it may cause significant packet loss. According to research [9], the use of above mentioned protocols causes the packet loss of 0.03% to 0.17%. For comparison – our solution does not influence the packet loss rate. Only in the extreme situation, while switching large traffic (more than 100 Mbit/s) from one tunnel to another, the loss of up to 16 packets may occur. Such packet loss does not occur with each iteration (with most iterations no packet is lost). Even in this extreme situation, the packets loss in a second of tunnel path changing is no more than 0.02%.

6.2. Conclusion

For the performance tests of virtual networks with resources guarantees for dedicated connections or for entire isolated virtual networks, we used the Execution Management system for end-to-end connections establishment. We focused on existing solutions and techniques implemented in common network equipment. Owing to this fact, our solution can be deployed in almost any network, for example as point of reference in benchmarking. Except for testing cases, Execution Management is also suitable for business purposes. It allows selling multiple particular, limited network resources (access to virtual networks) of a single physical infrastructure.

The performance of our system is slightly lower comparing to low level hardware-based mechanisms like MPLS and OpenFlow. The advantages of our system are the simple architecture and capability of traffic engineering (in terms of path and bandwidth) in a heterogeneous environment made of generic equipment. The time needed for new connections establishing is not excessively high and rises only slightly in the case of handling large number of requests simultaneously. Moreover, the larger network is served, the higher is probability that multiple request of new connection will be handled faster, providing constant number of parallel requests.

It is worth to mention that Execution Management has several significant functions that are unavailable in competitive solutions, such as discrimination multiple applications on a single host, creation many independent end-to-end connections between a pair of hosts and full support for IPv6. Furthermore, the system cooperates with comfortable, graphical web client, which may be used by the administrator or shared with virtual networks' administrators.

The presented system does not provide excellent performance in the case of large number of connections with high QoS requirements. Nevertheless, the system despite its disadvantages meets the assumed requirements and offers functions that other systems are lacking. On this ground, the system is suitable for reference testing and for design, implementation and management of laboratory testbeds for prototype QoS-aware applications. Execution Management was for example utilized for initial evaluation of applcations designed in the Future Internet Engineering project such as: eDiab [10], SmartFit [11], Online Lab [12] and others, e.g., [13], [14]. In the future work the authors will compare the performance our solution to the performance of the IIP System [15]–[18].

Acknowledgements

The research presented in this paper has been partially supported by the European Union within the European Social Fund and the European Regional Development Fund program no. POIG.01.01.02-00-045/09.

References

[1] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks", in *Proc. 9th Ninth ACM Worksh. Hot Topics in Netw. HotNets-IX*, Monterey, CA, USA, 2010.

[2] C. Guo *et al.*, "SecondNet: a data center network virtualization architecture with bandwidth guarantees", in *Proc. 6th ACM Int. Conf. Co-NEXT 2010*, Philadelphia, PA, USA, 2010.

[3] K. Chudzik, J. Kwiatkowski, and K. Nowak, "Virtual networks with the IPv6 addressing in the Xen virtualization environment", in *Computer Networks*, A. Kwiecień, P. Gaj, snf P. Stera, Eds. Berlin, Heidelberg: Springer, 2012, pp. 161–170.

[4] B. Dabiński, D. Petrecki, and P. Świątek, "Performance of mechanisms of resources limiting in networks with multi-layered wirtualization", in *Proc. 17th Polish Teletraf. Symp. 2012*, Zakopane, Poland, 2012, pp. 91–98.

[5] D. Gawor, P. Klukowski, D. Petrecki, and P. Świątek, "Prototyp systemu zdalnego monitoringu parametrów przyżyciowych człowieka w sieci IPV6 z gwarancją QoS", in *Proc. ICT Young 2012*, Gdańsk, Poland, 2012, pp. 409–414 (in Polish).

[6] D. Petrecki, B. Dabiński, and P. Świątek, "Prototype of self-managing content aware network system focused on QoS assurance", in Information systems architecture and technology [electronic doc.]: networks design and analysis, A. Grzech *et al.*, Eds. Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 2012, pp. 101–110.

[7] ITU-T Rec. E.271 (10/1976).

[8] C. Rotsos *et al.*, "Cost, Performance & Flexibility in OpenFlow: Pick Three", 23.10.2012 [Online]. Available: http://www.cs.nott.ac.uk/rmm/papers/pdf/iccsdn12-mirageof.pdf

[9] G. Liu and X. Lin, "MPLS performance evaluation in backbone network", in *Proc. IEEE Int. Conf. Commun. ICC 2002*, New York, USA, 2002, vol. 2, pp. 1179–1183.

[10] J. Świątek, K. Brzostowski, and J. M. Tomczak, "Computer aided physician interview for remote control system of diabetes therapy", in *Adv. Analysis and Decision-Making for Complex and Uncertain Systems*, Baden-Baden, Germany, 2011, vol. 1, pp. 8–13.

[11] K. Brzostowski, J. Drapała, A. Grzech, and P. Świątek, "Adaptive decision support system for automatic physical effort plan generation – data-driven approach", *Cybernet. and Syst.*, vol. 44, no. 2–3, pp. 204–221, 2013.

[12] P. Świątek, K. Juszczyszyn, K. Brzostowski, J. Drapała, and A. Grzech, "Supporting Content, Context and User Awareness in Future Internet Applications", in *The Future Internet*, Lecture Notes in Computer Science 7281, pp. 154–165. Springer, 2012.

[13] P. Świątek, P. Stelmach, A. Prusiewicz, and K. Juszczyszyn, "Service composition in knowledge-based SOA systems", *New Generation Comput.*, vol. 30, no. 2, pp. 165–188, 2012.

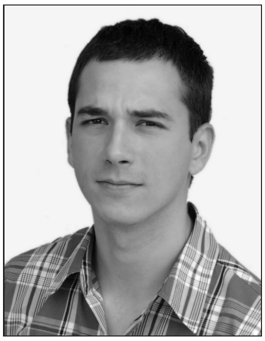
[14] A. Grzech, P. Świątek, and P. Rygielski, "Dynamic resources allocation for delivery of personalized services", in *Software Services for e-World*, W. Cellary and E. Estevez, Eds. IFIP Advances in Information and Communication Technology 341, pp. 17–28. Springer, 2010.

[15] H. Tarasiuk *et al.*, "Architecture and mechanisms of Parallel Internet IPv6 QoS", *Przegląd Telekom. + Wiadomości Telekom.*, vol. 84, no. 8/9, pp. 944–954 2011 (in Polish).

[16] A. Bęben *et. al.*, "Architektura sieci świadomych treści w systemie IIP" ("Architecture of content aware networks in the IIP system"), Przegląd Telekom. + Wiadomości Telekom., vol. 84, pp. 955–963, 2011 (in Polish).

[17] W. Burakowski *et al.*, "Provision of End-to-End QoS in Heterogeneous Multi-Domain Networks", *Ann. of Telecommunications*", Springer, vol. 63, pp. 559–577, 2008.

[18] H. Tarasiuk *et al.*, "Performance evaluation of signaling in the IP QoS system", *J. Telecommun. Inform. Technol.*, no. 3, pp. 12–20, 2011.



Paweł Świątek received his M.Sc. and Ph.D. degrees in Computer Science from Wrocław University of Technology, Poland, in 2005 and 2009, respectively. From 2009 he is with Institute of Computer Science, Wrocław University of Technology, where from 2010 he works as an Assistant Professor. His main scientific interests are focused on services optimization and personalization, optimization of service-based systems, resources allocation, QoS delivery in heterogeneous networks, mobility management in wireless networks and application of service science for e-health.

E-mail: pawel.swiatek@pwr.edu.pl
Institute of Computer Science
Wrocław University of Technology
Wybrzeże Wyspiańskiego st 27
50-370 Wrocław, Poland

Damian Petrecki received his M.Sc. degree in Computer Science from Wrocław University of Technology, Poland, in 2013. From 2010 to 2013 he was working at the Institute of Computer Science inter alia within Future Internet Engineering project. His scientific interests were focused on multilayered network virtualization as a way to achieve QoS guaranties in multi-agent or centralized network topology as well as on modern car stabilization using decision-making system and environment awareness.

E-mail: damian.petrecki@gmail.com
Institute of Computer Science
Wrocław University of Technology
Wybrzeże Wyspiańskiego st 27
50-370 Wrocław, Poland

Bartłomiej Dabiński received the M.Sc. in Information and Communication Technologies from the Wrocław University of Technology in 2013. He has worked as a networking engineer for ISP companies since 2006. During 2011–2012, he worked at the Institute of Computer Science, Wrocław University of Technology. His professional and research interests includes performance analyzing of WISP class equipment, multi-layered virtualization of network resources and QoS mechanisms.

E-mail: bartlomiej@dabinski.pl
Institute of Computer Science
Wrocław University of Technology
Wybrzeże Wyspiańskiego st 27
50-370 Wrocław, Poland