Virtual Routers: A Novel Approach for QoS Performance Evaluation

Florian Baumgartner† and Torsten Braun

Institute of Computer Science and Applied Mathematics
University of Berne
Neubrückstrasse 10, CH-3012 Berne, Switzerland
baumgart|braun@iam.unibe.ch

Abstract. Implementation and Evaluation of new Internet communication systems face some general problems. The implementation of Quality of Service concepts in kernel space is complex and time consuming, while the final setup of the evaluation networks lacks the desired size and flexibility. The usual alternative simulation cannot cope with the real world. This paper presents a concept to combine real components like hosts and routers with simulated nodes. This simplifies the setup of huge test scenarios and the implementation of new concepts, while keeping the evaluation results realistic. This paper presents the concept and the implementation of virtual routers by its application in a Differentiated Services Networks.

1 Introduction

A general problem in research on networking is the demand for setting up test beds of sufficient size and complexity to show the desired results or to prove a new concept. Alternatively network simulators like ns [ns] or OpNet [opn] can be used to prototype a device or a protocol in the special simulation environment and to run the desired tests. So the simulation normally precedes the setup of the test scenario in a laboratory. Nevertheless there are couple of drawbacks using this approach:

- The simulations can often only cover only a single aspect of the problem, ignoring side effects.
- Especially for application oriented research simulators lack complexity. So realistic traffic sources and sinks are missing as well as fully functional protocol implementations.
- Ad hoc implementations on real platforms are often extremely time consuming (Linux kernel hacking).

– The evaluation of new components depends on realistic traffic. Simulators normally allow to define an abstract traffic type or to simulate load based on a real network's traffic logs, but lack interactive real time evaluation.
– Most simulators are not able to combine real network components with simulated environments, a functionality, which can be very suitable during the development or debugging of programs.

Because of this, we propose an approach, to combine reality with simulations, using real devices, wherever needed and emulate the rest.

2 Softlink Device and Virtual Router

The basic idea of combining real hardware with emulated topology is shown on figure 1. The core mechanism is that of a Virtual Router (VR) emulating a real IP packet forwarder. Each VR as a couple of interfaces attached, which can be connected to other interfaces of VRs (the dashed lines) or via softlink device to the local host. Each host might run multiple VRs.

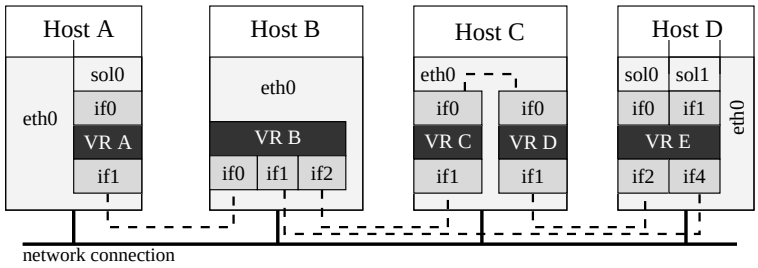

Fig. 1. The components of a VR

The network layer of the host system should not detect any differences between the real network and the emulated topology. So it is possible to define an emulated topology consisting over multiple VRs distributed of several machines. The communication between the real world and the emulated topology is achieved by the softlink devices. Such a softlink device acts as an interface between the operating systems network layer and a virtual router. For the OS kernel/user space it looks like a normal Ethernet device, transporting data to user space and vice versa. The only additional functionality is the truncating of the Ethernet headers, so raw IP is transferred to user space. For later versions this truncating should be omitted, requiring the processing of raw Ethernet frames in user space. Then the type of the encapsulated datagram could be analyzed to distinguish between different protocols. The softlink device has been implemented as a Linux Kernel module for kernels $\geq$ 2.2.12.

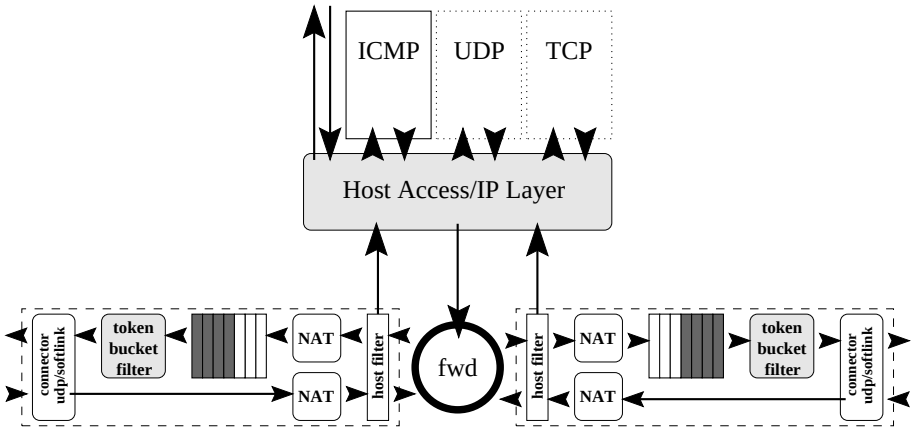

Fig. 2. The components of a VR

Virtual Routers (VRs) are used to realize the network topology to be emulated. Figure 2 shows the principal architecture of the program. Following the primary task the architecture is focusing on IP routing. The VR is completely implemented in plain C++, making the source extensible and easy to port.

The central forwarding mechanism (central circle) acts on standard routing rules, but was extended to allow routing decisions by source addresses, port numbers, protocol fields and TOS values.

As an interface to programs running on this virtual host, the VR will implement usual IP stacks with TCP, UDP and ICMP. Actually only a simple ICMP stack for debugging purposes is implemented, allowing a ping to the virtual host.

The main work regarding IP processing is assigned to the interface components underlying the routing mechanism. Figure 2 shows two of them. Each interface can be connected to a softlink device, acting as a transition point to the real network or to another VR-interface. For the connection to other VR interfaces we use UDP.

Received data is first processed by an IP address translation unit (NAT). After that step packets are delivered to the host filter. This unit is programmable and allows the processing of specific streams at higher layers. This simplifies the implementation of certain daemons, but it also a great facility to process streams during transmission (e.g. video- or audio data). As a default the host filter only checks for IP packets addressed to the virtual host and the Router Alert Option [Kat97].

Acting as sender, data is also transported through host filter and NAT to be put to the queueing system before transmitted by the softlink device or sent via UDP. A token bucket filter preceding the connector is used to limit the maximum bandwidth of the interface.

Because of it's flexibility the queueing system is the most complex part of the interface. It consists of a pool of components like queues, filters, shapers,

schedulers. The actual implementation offers the following components: a generic classifier, a Token Bucket Filter, a drop tail queue, a Random Early Detection queue (RED) [FJ93]), a Weighted Fair Queueing (WFQ) scheduler, a simple Round Robin (RR) scheduler and a Priority Round Robin (PRR) scheduler. Additional components are a RED queue with three drop precedences (TRIO), a special marker for differentiated services and a Priority Weighted Fair Queueing (PWFQ) scheduler for the implementation of Expedited [JNP98] and Assured Forwarding [HBWW98]. The configuration of the queuing system can be completely done at runtime via API or command line interface (CLI). The object oriented implementation of the queueing system and it's components makes it easy to add or modify single functionalities.

For the configuration of the VR a command line interface and an API has been implemented. The API allows programs, running on the virtual host, to alter interface setting, routing rules and so on. The command line interface is accessible via console or external[1] telnet.

3 Network Setup

3.1 A minimal Setup

In theory any network topology can be realized on only one host. However, before things get complicated we will demonstrate the most simple setup first, using only one VR, being connected to a host.

The host gets an additional interface `sol0` with the IP address 10.1.1.1 using the softlink device.

```
sol0      Link encap:Ethernet
          HWaddr 00:53:4F:46:54:4C  inet addr:10.1.1.1
          Bcast:10.1.1.255  Mask:255.255.255.0
          MTU:1500  Metric:1
          UP BROADCAST RUNNING NOARP MULTICAST
          [...]
```

We choose here non routed addressed of the type 10.x.x.x. As a next step we setup the VR. We configure an interface `if0` with the address `10.1.1.2` and connect it to the `sol0`, setup a minimum base queuing system consisting only of a single drop tail queue and set the according routing rules.

```
#
# Interface SETUP
#
interface if0 10.1.1.2 255.255.255.0
```

[1] This telnet connection is not provided by the VR, but by the host making it independent from any changes made to the VR. This simplifies setup of multiple machines crucially, because no changes to interfaces setup or queueing mechanisms can harm the TCP connection used for the configuration

```
interface if0 connect /dev/sol0
interface if0 sqc create droptail
interface if0 sqc chain 0 to 1
interface if0 sqc chain 1 to 0

# ROUTING TABLE
#
route add 10.1.1.0 255.255.255.0 if0
route add 0.0.0.0  0.0.0.0 if0
```

Now we can test the scenario by pinging to the ICMP stack of the VR. A `ping 10.1.1.2` results in:

```
PING 10.1.1.2 (10.1.1.2): 56 data bytes
64 bytes from 10.1.1.2: icmp_seq=0 ttl=187 0.5 ms
64 bytes from 10.1.1.2: icmp_seq=1 ttl=187 0.2 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=187 0.2 ms
```

Following these example it is easy to setup bigger topologies. To connect two VRs via UDP only the command `interface if0 connect /dev/sol0` must be modified.

3.2 A TCP capable (Tunnelling) Setup

The minimal example has shown how a VR can be setup like a normal router connected to our host. But even if we can ping to the VR, it is not possible to open a TCP connection between our host and the VR, because of the missing TCP and UDP stacks. So we need two real hosts as source and sink and two VRs to transport the traffic over.

Figure 3 shows the principal setup of the two hosts. Source and sink are real machines, so we do not need to rely on a (more or less complete) simulative TCP implementation in the VRs, but can use wide spread, comparable TCP implementations of the host systems. When we open a TCP connection from Host A to 10.1.3.1, the traffic is routed over `sol0` to VR A, is then encapsulated in UDP packets and transported over some kind of UDP tunnel (the dashed line in figure 3) to Host B (to be more exactly to 130.92.70.7) and so to VR B which decapsulates the traffic and forwards it to `sol0` on Host B.

It is easy to see, that each host can serve as more than a source or a sink. Which each softlink device added to the host and the VR you gain a source respectively a sink. Of course the number of VRs is limited by the number of tunnels the physical connection between the hosts can manage. But as long as the configured bandwidth of the VRs' interfaces is small compared to the available bandwidth of the host there are no problems to be expected.

```
#
# Interface SETUP
#
interface if0 10.1.1.1 255.255.255.0
interface if0 connect /dev/sol0
```

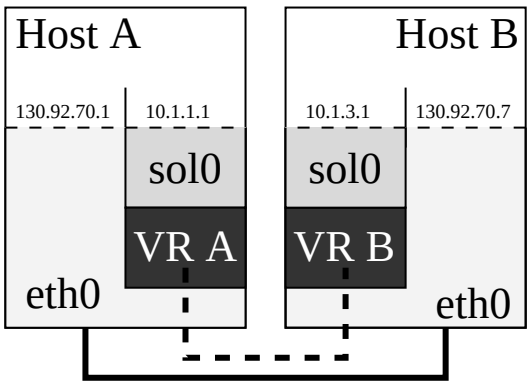

Fig. 3. A TCP capable Setup with two hosts and two VRs

```
interface if0 sqc create droptail
interface if0 sqc chain 0 to 1
interface if0 sqc chain 1 to 0
#
interface if1 10.1.2.1 255.255.255.0
interface if1 connect 130.92.70.7:8042 8041
interface if1 sqc create droptail
interface if1 sqc chain 0 to 1
interface if1 sqc chain 1 to 0
#
# ROUTING TABLE
#
route add 10.1.1.0 255.255.255.0   if0
route add 10.1.2.0 255.255.255.255 if1
route add 10.1.3.0 255.255.255.255 if1
route add 0.0.0.0  0.0.0.0 if0
```

The script shows the setup of VR A. VR B has to be configured in an analogous way. VR A has it's interface if0 connected to the softlink device on Host A and the interface if1 via UDP tunnel to the according interface on VR B.

3.3 Using Address Translation for Network Setup

On a long term it is not satisfying, that for the use of higher protocols at least two machines have to be used. Even if with Gigabit Ethernet the physically available bandwidth is no scarce resource this problem limits the usability of VRs. At the moment the VR has no TCP or UDP stack, requiring real hosts as end systems. So at for a minumum setup at least two systems are necessary. This was the reason why the already mentioned Network Address Translation unit was added to the interface structure (see Figure 2). In the following we will give a short example how to use NAT to route IP packets through a VR.

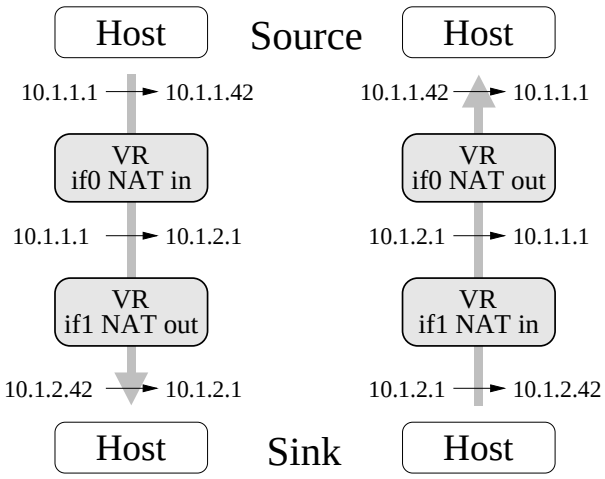

Fig. 4. A Setup using Network Address Translation features

Figure 4 shows the IP address translations occurring during forwarding through the router. Each address pair represents the source and destination IP addresses of the ICMP ping packet at each state in the VR. The interface comes with two NAT filters, one for incoming packets and the destination address and one for outgoing packets and the source address.

It should be mentioned, that the source host and the sink host are the same machine, but on two different softlink interfaces. (10.1.1.1 and 10.1.2.1). The source connects to a dummy address 10.1.1.42, being routed over the VR, which converts during transmission the packet's destination address to 10.1.2.1 and the source address to 10.1.2.42. Any response sent to 10.1.2.42 will be converted in an analogous way.

```
#
# Interface SETUP
#
interface if0 10.1.1.2 255.255.255.0
interface if0 connect /dev/sol0
interface if1 10.1.2.2 255.255.255.0
interface if1 connect /dev/sol1
#
# MAP IP ADDRESSES
#
interface if0 rmq add 10.1.1.42 255.255.255.255 10.1.2.1
interface if1 smq add 10.1.1.1  255.255.255.255 10.1.2.42
#
interface if1 rmq add 10.1.2.42 255.255.255.255 10.1.1.1
interface if0 smq add 10.1.2.1  255.255.255.255 10.1.1.42
#
[... Setup of Queueing Systems ...]
#
```

```
# ROUTING TABLE
#
route add 10.1.1.0 255.255.255.0 if0
route add 10.1.2.0 255.255.255.0 if1
route add 0.0.0.0  0.0.0.0 if0
```

Now we will apply a simple queueing system to the interfaces. Figure 5 shows the setup for bandwidth reservation. The classifier C forwards packets according to their header data to different queues (Q_1 and Q_2) with the branch over Q_1 being limited by a token bucket filter. As scheduler a standard Round Robin is used limiting all non TCP traffic to a maximum of 2 Mbps.

Figure 6 shows a simple evaluation of this queueing system with an aggressive UDP and a TCP flow. The TCP datagrams are forwarded over Q_2, UDP over Q_1. The token bucket filter T is set to a bandwidth of 2 Mbps. The VR interfaces are limited to 4 Mbps. The UDP source sends in intervals, to visualize the reaction of TCP and the queueing system to massive UDP bursts.

One can clearly observe, how the TCP bandwidth decreases from 4 to 2 Mbps, when UDP uses the available bandwidth of 2 Mbps. The short peak of the TCP flow below the guaranteed bandwidth is caused by TCP congestion control. The graph was obtained by a setup using one VR with two interfaces and network address translation (NAT) as shown on figure 4, so the TCP source and the TCP sink where hosted on the same machine.

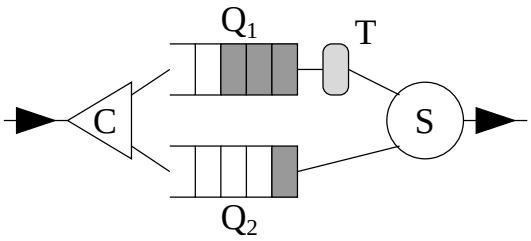

Fig. 5. A minimum queueing system setup for bandwidth allocation

This scenario is comparable with a real setup of three machines, one acting as sink, one as intermediate router and one as source.

4 Differentiated Services Setup with Virtual Routers

In this section we show how a typical Differentiated Service evaluation scenario can be setup using Virtual Routers. Of coarse any mixture of real and virtual components is possible, so VRs might play a role during developing real implementations for debuging purposes, without occupying a whole pool of unix workstations and routers.

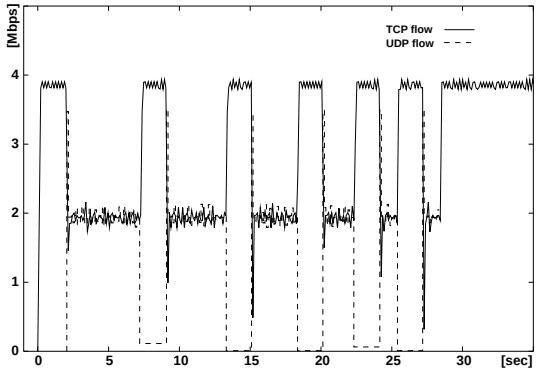

Fig. 6. Bandwidth allocation to specific a TCP flow

4.1 Setup of a DiffServ Queueing System

Figure 7 shows a possible configuration of an VRs queueing system for Differentiated Services.

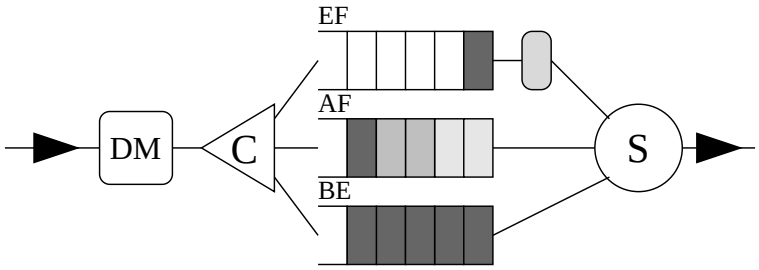

Fig. 7. A Differentiated Services Queueing System with an Best Effort, an Expedited Forwarding and a Queue for one Assured Forwarding class

The first component BM is a generic DS marker, measuring flows and reclassifying them according to their Service Level Agreement. C is the classifier forwarding the packets to the according queues. The queues for Expedidted Forwarding (EF) and for Best Effort Traffic (BE) are standard drop tail queues. Assured Forwarding (AF) traffic is sent to a TRIO queue (Three state RED with in and out [FJ93]), dropping packets according to their ToS field values. The scheduler S is a Priority Weighted Fair Queueing (PWFQ) scheduler. This scheduler has been specially designed at our institute for the implementation of Differentiated Services and allows to favour some queues as priority queueing and to allocate a share of the available reosources to the others as WFQ does. So EF packets get the absolute priority while the other queues get a share of the

bandwidth according to their weight. To prevent EF traffic to block all other flows, EF is limited by a token bucket filter.

4.2 Setup of the Evaluation Topology

In this section we describe a setup for a Differentiated Services evaluation and it's port to an VR architecture. As a basis for the network layout we use the topology of the SWITCH network in Switzerland. Figure 8 shows the existing network and the representation with Virtual Routers. Each access network is realised by one softlink device. Of course a more complex setup of the access network could also be implemented. The nodes of the SWITCH network are emulated on three hosts, each running several instances of Virtual Routers.

5 Summary and Outlook

The idea of Virtual Routers and softlink devices presented here has proven useful for the quick developement and evaluation of new traffic conditioning equipment and for emulating bigger testbeds for the debugging of 'real' programs.

The results available so far correspond measurements on real hardware. Of course Virtual Routers are not capable to emulate exactly a real network behaviour and will never be. Especially physical effects like link delays are hard to emulate. Fortunately these effects are small in fast full switched local area networks available today.

Future extensions will focus on two objectives: managability and portability. So work on a graphical user interface for the setup and control of multiple Virtual Routers distributed over multiple hosts has just started and even direct interfaces to topology generators like Tiers [CDZ96] are planned. The goal is to allow the setup of evaluation scenarios with dozens of routers distributed over a pool of workstations in a time you usually need to configure one device.

Because of better portability between Virtual Routers and real machines an port of Berkeley Sockets to the VR platform is planed, allowing to run programs under a VR environment using the same techniques to access the network as on real unix hosts. These enfolds also the implementation of an Active Networking environments and the according protocols like the Active Network Encapsulation Protocol (ANEP) [ABG+97], providing a platform for the evaluation of Active Networking and traffic control cooperation.

6 Acknowledgement

The work described in this paper is part of the work done in the project 'Quality of Service support for the Internet Based on Intelligent Network Elements' funded by the Swiss National Science Foundation (project no 2100-055789.98/1). Parts of the implementation platform were funded by the SNF R Equip project no. 2160-53299.98/1.

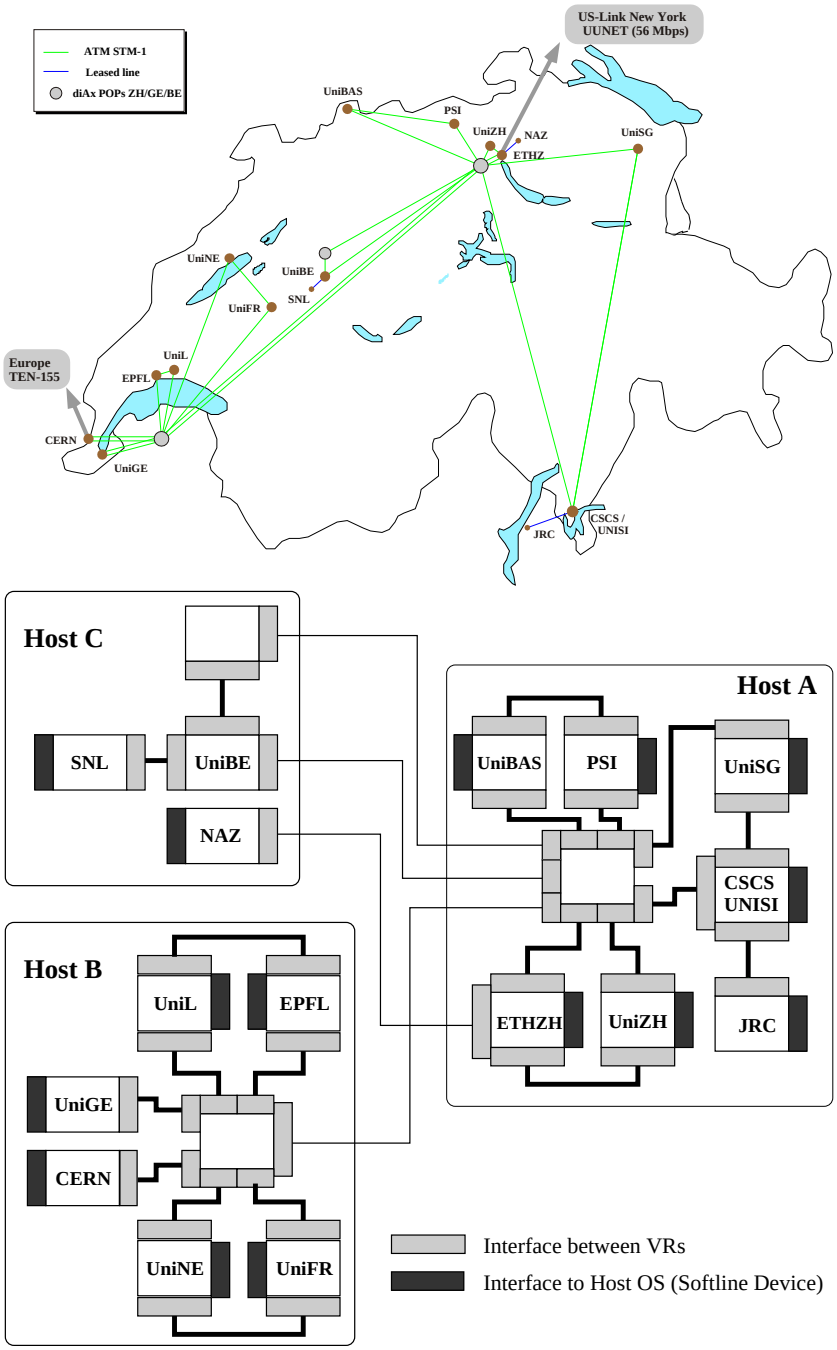

Fig. 8. The SWITCH network (from http://www.switch.ch/lan/national.html) and it's representation with Virtual Routers

References

[ABG+97] D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden, and David Wetherall. Active network encapsulation protocol. RFC draft, July 1997.

[CDZ96] K. Calvert, M.B. Doar, and E.W. Zegura. Modeling internet topology. *IEEE Global Telecommunications Conference/GLOBECOM'96, London*, November 1996.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[HBWW98] Juha Heinanen, Fred Baker, Walter Weiss, and John Wroclawski. Assured forwarding phb group. Internet Draft `draft-ietf-diffserv-af-02.txt`, October 1998. work in progress.

[ISI81] University of Southern California Information Sciences Institute. Internet protocol. RFC 791, September 1981.

[JNP98] Van Jacobson, K. Nichols, and K. Poduri. An expedited forwarding phb. Internet Draft `draft-ietf-diffserv-af-02.txt`, October 1998. work in progress.

[Kat97] D. Katz. Ip router alert option. RFC 2113, February 1997.

[ns] Ucb/lbnl/vint network simulator - ns (version 2). URL: http://www-mash.CS.Berkeley.EDU/ns/.

[opn] Opnet modeler. URL: http://www.mil3.com.