Quality of Service Oriented Active Routers Design

Tomáš Rebok*, Petr Holub‡, and Eva Hladká*
*Faculty of Informatics and ‡Institute of Computer Science
Masaryk University
Botanická 68a, 602 00 Brno
E-mail: xrebok@fi.muni.cz, hopet@ics.muni.cz, eva@fi.muni.cz

Abstract— **The active network approach allows an individual user to inject customized programs into the active nodes in the network, usually called programmable/active routers, and thus process data in the network as it passes through. When a programmable router is used in a multi-user network environment, quality of service (QoS) for each passing stream needs to be ensured. QoS approaches in common networks enforce certain parameters (e.g., queuing strategy, priority) on the network flows. However, this is not sufficient in active routers where users' programs run on the routers and thus other parameters (e.g. processor time, amount of memory) have to be guaranteed as well. In this paper, we propose a QoS-enabled active router architecture that supports extended understanding of QoS. We also propose a virtual machine based router implementation for strict isolation of user processes.**

I. Introduction

Contemporary computer networks behave as a passive transport medium which delivers—or in case of best-effort service tries to deliver—data sent from the sender to the receiver. The whole transmission is done without any modification of the passing user data by the internal network elements[1]. These "dumb and fast" networks became mature product where only speed is ever increased and there is no ambition except for simple forwarding of the data. We believe that the future-generation networks may be extended beyond that paradigm and behave as an active transport medium, which processes passing data based on data owners or data users requests. Multimedia application processing (e.g., video transcoding) and security services (data encryption over distrusted links, etc.) are a few of possible services which could be provided. The principle called "Active Networks" or "Programmable Networks" is an attempt how to build such intelligent and flexible network using current "dumb and fast" networks as an overlay network.

We can consider a computer network as a system whose end nodes provide computations up to the application level, while inner elements (routers, switches, etc.) provide computations up to the network level, and all nodes are connected via passive links. While the elements may be programmable to some extent, the control is always in the hands of network administrators. The major difference in the active network is that the elements inside the network are directly programmable by users. Also, the nodes inside the active network can provide computations up to the application level. These inner elements are called *active nodes*, *active routers*, or *programmable routers* (all three with rather identical meaning). Users and applications have the possibility of running their own programs inside the network using these active nodes as processing elements.

[1]Not including firewalls, proxies, and similar elements, where an intervention is on the one hand usually limited and on the other not user controllable.

An application of software programmable routers in multi-user environment pose new challenges in the design of router operating systems and especially in the design of resource management system. Since more resources are shared among the users of the active router—router CPU cycles, state storage capacity, data storage together with traditional networking components like packet queues on network interfaces. To enable sharing of all these resources within the active node by its users in a secure and effective manner, much more complex Quality of Service (QoS) architecture needs to be deployed, including sophisticated resource accounting and resource scheduling algorithms that respects characteristics of individual resources.

The main goal of this paper is to propose a QoS-enabled active router (AR) architecture that supports complex QoS guaranties as described above. In order to achieve reasonable isolation among the users of the AR, the architecture is designed to facilitate implementation based on virtual machines (VM) approach [1]. The paper is organized as follows: Section II briefs previous work on a generic AR architecture, regardless of QoS, and Section III describes modified VM-based architecture suitable for QoS implementation. Proposed QoS implementation is analyzed in Section IV. Related work is summarized in Section V and concluding remarks and proposals for future work are in Section VI.

II. Generic AN architecture

When considering the architecture of the active networks, one possible classification criterion is the way active code is delivered to the active routers [2]:

- *Active nodes* – The code of an active program is injected into the active nodes separately from the data packets. The code can be implemented either as built-in functions or during the initial phase (the opening) of the data transfer. The advantage of this architecture is that the code is injected only once and thus its size is not limited and not critical. A disadvantage lies in the necessity to inject the code before data transmission which means larger startup latency and lower flexibility as it is hard to change the code during the actual data transmission.

- *Active packets* – Each data packet contains the program code which is extracted on an active node and executed on the data part of this packet. This approach is flexible since individual data packets in one transmission can be processed by different programs. The node needs “only” to be able to extract the code and execute it. The disadvantage is that even the limited extent of the code tends to result in a large overhead for transmitted data.
- *Active packets and active nodes* – This combination of both previous architectures allows the use of more complex programs while remaining flexible enough. Usually a program is transferred before the actual data transmission occurs, but individual data packets contain some kind of parameters or specific program commands. This supports individualized packet processing without the limitations of the active packet approach. However, the substantial initial delay (latency) is not eliminated.

For our work we use a model of active node with loadable functionality published in [3]. The proposed active network architecture uses an “active node” approach to active networking and the concept of “sessions” similar to connections in connection-oriented networks or sessions in RSVP protocol.

The structure of an active node (router) plays a key role in this model. The router is a network element which is able to accept user-supplied programs and to execute them. The processing of user code consists of two separate but communicating processes. The first process controls the session establishment and management. It has the role of a control plane in active router processing and includes a process of loading user functions into the routers along the path between the source and the destination. The functions may be either pre-loaded (before or during the connection set up) or they may be loaded on demand during the data transmission (if a new requirement arises). Bookkeeping functions are also provided by the control process. The second process performs the data packet processing which includes executing the user code.

AN model described in [3] has never been fully implemented, but main ideas from this work were successfully used for a model and implementation of user empowered UDP packet reflectors to create virtual multicasting environment as an overlay on top of current unicast networks [4]. It also served as a basis for protocol research and development, e.g., “Active Node Authentication Protocol (ANAP)” [5] and “Active Router Transport Protocol (ARTP)” [6].

III. VM-ready AN architecture

Because of generic AR modular architecture, we have extended the generic AR architecture to support the complex QoS and also slightly modified the scheme in order to facilitate implementation based on virtual machines. This approach enables users not only to upload the active programs, which run inside some virtual machine, but they are allowed to upload the whole virtual machines with its operating system and let their passing data being processed by their own operating system running inside uploaded VM. VM approach ensures strict separation of different virtual machines and also allows efficient scheduling of resources to individual VMs, e.g., CPU, memory, and storage subsystem access.

The architecture of our VM-ready active router is shown in Figure 1. The bottom part is the VM-host layer where the core of the proposed VM-ready router is located. The core includes packet classifier, shared buffer pool, and packet scheduler modules. The modules relevant to resource management (resource management module and VM/AP scheduler module) are described in more detail in Section IV. Packet classifier module classifies all the incoming packets whether they belong to any active session running on the router and thus must be very efficient. It also extracts packets destined to the session management module and sends them directly to that module. The shared buffer pool module operates as the buffer space where all the incoming packets are stored before further processing and also all the outgoing packets before the packet scheduler module sends them onto the network.

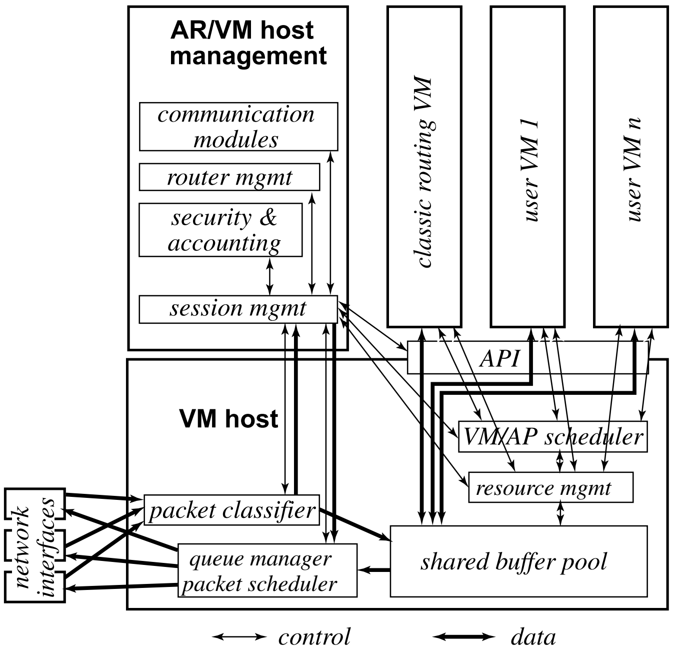

Fig. 1. VM-ready active router architecture

The VM-host management system is located in user space. Besides the other functions it has to manage the whole router functionality including uploading, starting and destroying of the virtual machines, security functions, session accounting and management. The virtual machines managed by the session management module could be either fixed, providing functionality given by system administrator, or user-loadable. The example of the fixed virtual machine could be a virtual machine providing classical routing as shown in Figure 1—it is an example of optional module, as the AR can run without the classical routing if only “active” traffic passes through the AR, e.g., if it works in a dedicated overlay network. Besides that, the one other fixed virtual machine could be started as an active program execution environment where the active programs uploaded by users are executed. This virtual machine serves especially for backward compatibility with original generic AR and this approach does not force users to upload the whole virtual machine in the case where active program uploading is sufficient.

The VM-ready AR architecture uses a connection-oriented approach similar to the one used in active router proposed in [3]. In terms of our active architecture, the connection is also called *"(active) session"*, but each active session consists of one or more active programs/virtual machines, one or more network flows and potential QoS requirements. The association of more VMs/active programs and network flows into one session is very useful especially when creating active programs working with more than one network streams (e.g., synchronization of two RTP streams when transmitting audio and video streams separately).

Besides the other information, the session initiation request encapsulated in an active packet contains minimal resource requirements for given active session and the active router decides, whether the requirements could be satisfied. If the request could be satisfied, the session is established and all the required resources are allocated and reserved to it. Otherwise the request is refused.

Once the session is established with the required resources, the data flow through the router could be briefly described in the following way: when a packet arrives to a network interface, the packet classifier module decides, whether the incoming active packet belongs to the given AR or not, based on information from the security and accounting module. If the packet is accepted, depending on resource allocations and actual scheduling algorithm, the classifier module forwards packet to the proper VM running on the AR or the new session establishment takes place. Depending on resource management, the active packet is processed in the VM and sent into the network through the shared buffer pool.

IV. QoS SUPPORT FOR VM-READY ACTIVE ROUTER

As obvious from VM-enabled AR architecture described above, there are the two main modules concerned with resource management: (1) resource management module and (2) VM/active program scheduler. Indirectly, the session management module also participates on this process.

Resource management module. This module implements the crucial resource management scheme with the following functionality:

- Possessing all the information about the resources in the AR.
- Providing necessary information to the session management module.
- Monitoring and adjusting the resources used by each active session and sending notifications to the active sessions through the session management module to inform them about the actual resource status of the AR (e.g., how many resources are available and can be used or how many resources are needed).

VM/active program scheduler module. This module schedules the execution of the applications and the transmission of the packets to the next node. It implements scheduling algorithms for different classes of resources to enforce the active sessions allocations of the AR resources—for the additional information on scheduling algorithms, see Section IV-B). Besides that, the accounting and resource limit checking functions are also the part of this module:

- It checks whether the active sessions are permitted to request given resources (e.g., when restricting the amount of given resources from allocating by specified users/sessions).
- It logs active sessions requests and replies from resource management system about allocating given resources (useful e.g., when active node resource utilization is paid).

A. Resource management system

Due to the structure of active sessions where each session consists of one or more virtual machines (simply active programs) and one or more network streams, the fine-grained hierarchical design of resource allocations is very desirable. I.e., when the session possesses allocated resources, it is possible to split these resources held by the session in a way the user of the given active session wants (Figure 2).

For resource allocation and scheduling purposes, a *session element* (or just an element for short) denotes active stream ("active" network flow), virtual machine, or active program. When providing hierarchical resource management, all the schedulers must know about required resources of each element for a given active session. As said before, when creating a new active session it must request overall amount of resources wanted. If all the resources requested are available, the active session is established and all the resources are allocated. Since the active session holds an unique identifier of allocated resources, the assigned identifier must be provided when the session wants to work with a specific shared resource. This identifier is also used when the active session wants the resource management module to redistribute the allocated resources to its element(s). In this case, the (master) identifier is extended with element sub-identifier. When an element of the active session wants to use a shared resource, the master identifier or the extended identifier has to be provided depending on whether the element wants to utilize the overall amount of given resources available to the session or just the amount of resources previously redistributed inside the active session.

The resource management module thus provides mainly the following functions:

- *Create/Delete* – *Create* allows creating a resource allocation with given requirements and returns a key, a unique identifier of given allocation. *Delete* takes the key as an input parameter and removes the corresponding allocation. The allocation's resource share is then returned to the system.
- *Bind/Unbind* – *Bind* allows an active session to specify the resource requirements of elements of the active ses-

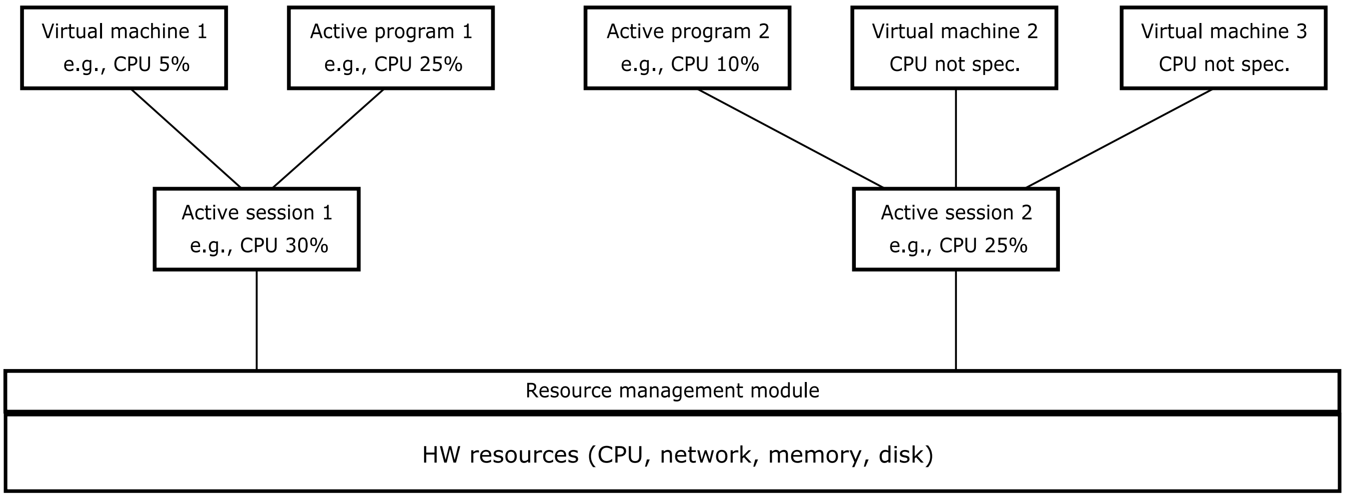

Fig. 2. Hierarchical resource assignment.

sion. Then, the session (master) key is extended to be able to unambiguously determine the active session and its element, and returned. *Unbind* deletes such binding inside the session.

- *Modify* – allows user to reconfigure a resource allocation with a given key.
- *Info* – provides the information about allocated and free resources in the system.

B. Schedulers

Scheduling algorithms are the most important part of the whole resource management system in our active router because they affect both overall performance and keep all required resources in desired limits. Since resource characteristics vary, scheduling algorithms must be designed in a resource specific manner. For example, CPU context switching is more expensive compared to switching between flows in network scheduling [7]. Therefore, efficiency of CPU scheduling improves if active programs can receive a minimum CPU quantum before being preempted. Disk scheduling, unlike both CPU and network, must consider request locations to limit seek time and rotational latency overheads. Memory schedulers, in order to match actual memory use, must estimate the current working set of active programs. All the schedulers must therefore examine relevant resource states (e.g. disk state, whether it is spinning or parked) in addition to QoS specifications.

The resource requirements are often related to the others. For example, when requesting high network bandwidth while having only a small amount of CPU time, it is not possible to reach required bandwidth, because there is insufficient CPU time to send all the packets. Thus the scheduling algorithm's design must be sophisticated enough to take such interdependencies into the account.

For sake of conciseness, we do not delve into detailed description of scheduling algorithms here, but we describe the most important demands on each of the active router scheduler focusing on CPU, network, memory and disk schedulers.

Because the quality of service assurance in active routers is closely related to multimedia applications, the requirements on the scheduling algorithms in our active router are very similar to the requirements in multimedia operating systems [7].

1) CPU scheduler: The CPU scheduling algorithms are the best-developed scheduling algorithms in current information technology. Unfortunately, majority of proposed algorithms are QoS-unaware and thus very huge research in this area should be made.

Thanks to the hierarchical resource management system in our active router architecture the hierarchical CPU scheduling algorithm is desirable. The operating system thus partitions the CPU bandwidth among more active sessions, and each active session, in turn, partitions its allocations among its VMs or active programs.

The other desirable features of CPU scheduling algorithms are as follows:

- *Admission criteria* – the admission of a new active session should not infringe the QoS guarantees given to currently established and running active sessions. If so, necessary steps need to be taken like re-negotiation or rejecting the new active session.
- *Real-time guarantees* – the design of the CPU scheduling algorithms must satisfy real-time constraints in terms of ensuring guaranteed scheduling for each active program within their jitter bounds, if any.
- *Fairness criteria* – it should be possible to schedule all the types of active programs that are competing for the CPU—if there is non-reserved CPU time, the lower priority non-guaranteed applications should not be completely starved out of CPU by higher priority tasks corresponding to guaranteed services.
- *Maintenance and policing criteria* – policing criteria requires to ensure that the deadline violating tasks do not

infringe the QoS guarantees of other tasks competing for CPU resources. Mechanisms like software watchdog that suspends an active program on deadline violations, are means of ensuring service guarantees. The maintenance criteria imply setting up re-negotiations or dropping further requests in case of CPU overload condition.

- *Throughput criteria* – the scheduling policy should be able to schedule as many active sessions as possible.

2) Network scheduler: Current network scheduling algorithms are well-developed and only their adaptation to active networks is necessary. The typical objective network scheduler parameters are bandwidth, latency and jitter, and the common criteria on network schedulers are following:

- *Admission criteria* – the scheduler must ensure that the requested bandwidth plus the currently allocated bandwidth does not exceed a threshold of the total available bandwidth.
- *Real-time guarantees* – it must also ensure that the network interface scheduling delays are bounded and the enough buffer provisioning is done. The scheduling algorithm must consider that the mechanisms like retransmissions may not be suitable for applications requiring hard delay bounds.
- *Fairness criteria* – all types of applications should get a fair share of network bandwidth.
- *Maintenance and Policing criteria* – Policy criteria should ensure that the application do not take up more than the network bandwidth that has been guaranteed by QoS negotiation during active session setup.

3) Memory scheduler: The memory scheduling algorithms must manage the whole memory subsystem using virtualization mechanism and it must guarantee the required amount of free memory to active sessions. Because of the virtualization mechanism the appropriate allocation of free page frames and redistributing released frames to other sessions are the main jobs of the memory scheduler. The following are the requirements on the memory schedulers in order to support QoS:

- *Admission criteria* – new active session can be admitted if and only if its memory buffer requirements plus the current buffer allocations of other sessions do not exceed the threshold of the total available memory.
- *Real-time guarantees* – during the run of given active session some time-critical applications need the memory access time to be minimal. With virtual memory, it is important to have paging mechanisms that have an acceptable upper bound on access latency.
- *Fairness criteria* – the memory scheduler must ensure the minimal availability of memory buffers for all the active sessions and their elements.
- *Maintenance and Policing criteria* – maintenance criteria require setting up re-negotiations or dropping further requests in case of a buffer shortage. Policy criteria may require that the offending active session should be notified for the re-negotiations or in the extreme case terminated.

4) Disk (I/O) scheduler: While a secondary data storage is not a traditional router resource, it is very important for the active router with QoS support. The disk scheduler may support either the transfer bandwidth of given disk or the amount of free disk space only or both. The requirements on suitable disk scheduling algorithms could be summarized into the following criteria:

- *Admission criteria* – the effective disk transfer bandwidth is reduced due to seeking and latency overheads, which are a function of the disk scheduling algorithm and the disk request size. The admission criterion ensures that the sum of the data rates of all the active sessions, including the new one, do not exceed the effective disk transfer bandwidth.
- *Real-time guarantees* – the scheduler must be able to schedule the disk accesses for all admitted streams so as to meet their data rate guarantees and the response time for all the streams must be acceptable.
- *Fairness criteria* – provisioning may be done to ensure that all types of active sessions get a fair share of disk transfer bandwidth.
- *Maintenance criteria* – the scheduling algorithm has to monitor the data rates being provided to real-time streams with respect to the guarantees provided before; the QoS re-negotiations must be provided in cases of shortfalls.
- *Resource reservation* – except the reservations of disk bandwidth each active session requires at a minimum a buffer for the consuming virtual machine (resp. active program) and a buffer for the producing virtual machine (active program). Thus, this amount of memory needs to be reserved for each admitted stream.

V. Related Work

In this section we brief the results of our work in the context of related projects in the resource management and QoS assurance in active networks research. We also notice relevant projects whose ideas may be implemented in our AR architecture.

An active node architecture with resource management [8] is an attempt how to introduce the architecture with explicit resource management system, which also provides an adaptation among different applications. The node operating system is based on the Janos project [9] developed at the university of Utah. The Janos project is in comparison with our work oriented to the execution of untrusted Java byte-code only and thus has limited flexibility. As a part of this project the method for the description of resource requirements from applications using the resource vectors and resource vectors space were introduced. We will explore the resource vector method in more detail and assess its possible application to our router architecture.

The CROSS project [10] is another attempt of introducing the resource management system in software-programmable router operating systems. This project was proposed by David K. Y. Yau and Xiangjing Chen in 2001 and it uses virtual machines as the active programs providing thus fixed router functionality only, but higher security and efficiency, because the CROSS system communicates directly with the hardware layer.

Friendly virtual machines [11] devises techniques that enable multiple virtual machines to share underlying resources on the same host both fairly and effectively. Instead of deploying complex resource management techniques in the hosting infrastructure, an alternative approach of self-adaptation in the virtual machines themselves was introduced based on feedback about resource usage and availability. Thus, the virtual machines that adjust their demands for system resources, so they are both efficiently and fairly allocated to competing virtual machines, are very important idea for the resource management subsystem in our router, where the principle of Friendly virtual machines could be used for the "competing" virtual machines with no explicit resource requirements.

QoS specification languages for distributed multimedia applications could be used for the description and negotiation of QoS requirements of active sessions in our architecture. Such languages were studied by Jingwen Jin and Klara Nahrstedt in [12]. They studied lots of languages including script languages and XML-based markup languages. The languages for the hierarchical resource requirements description probably suitable for our router architecture were also studied.

VI. Conclusions And Future Work

In this paper, we have proposed a virtual machine oriented active router architecture and studied the resource requirements and QoS implementation. The typical scenario of hierarchical resource management system has been explored and the requirements on the schedulers for such QoS-enabled active router, concerning the CPU scheduler, network scheduler, memory scheduler, and disk scheduler were also discussed.

The main feature of our router architecture is that it provides a predictable and assured access for active sessions to system resources. These resources can be subsequently redistributed to multiple virtual machines, giving flexible choice and the ability for services to seamlessly evolve.

Concerning the future challenges, the proposed router architecture will be implemented based on Xen virtual machine monitor [13]. Further we want to explore extending current architecture into a distributed environment to be able to deal with high-speed networks. In this case, all the resource schedulers, the whole router, and session management systems must be modified. For the efficiency purposes, another interesting topic for our future work is the implementation of some parts of the router architecture (especially the packet classifier module) in hardware, e.g., based on FPGA-based programmable hardware cards [14].

Acknowledgments

This project has been supported by a research intent "Optical Network of National Research and Its New Applications" (MŠM 6383917201), "Parallel and Distributed Systems" (MŠM 0021622419), and "Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems" (No. 102/05/H050).

References

[1] J. E. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. Elsevier Inc., 2005.

[2] K. Psounis, "Active networks: Applications, security, safety and architectures," *IEEE Communication Surveys*, 1999.

[3] E. Hladká and Z. Salvet, "An active network architecture: Distributed computer or transport medium," in *Networking – ICN 2001: First International Conference Colmar, France, July 9-13, 2001, Proceedings, Part II*, ser. Lecture Notes in Computer Science, P. Lorenz, Ed., vol. 2094. Heidelberg: Springer-Verlag, Jan. 2001, pp. 612–619.

[4] E. Hladká, P. Holub, and J. Denemark, "An active network architecture: Distributed computer or transport medium," in *3rd International Conference on Networking (ICN'04)*, Gosier, Guadeloupe, Mar. 2004, pp. 338–343.

[5] J. Denemark, "Autentizace v aktivních sítích (authentication in active networks)," Master's thesis, Faculty of Informatics, Masaryk University in Brno, Apr. 2003, czech only.

[6] T. Rebok, "Active router communication layer," CESNET, Tech. Rep. 11/2004, 2004. [Online]. Available: http://www.cesnet.cz/doc/techzpravy/2004/artp-protocol/

[7] B. Ghose, V. Jain, and V. Gopal, "Characterizing qos-awareness in multimedia operating systems," 1999, http://computing.breinestorm.net/qos+cpu+scheduling+criteria+admission/%.

[8] Y. Li and L. Wolf, "An active network node system with adaptive resource management," in *International Conference on Telecommunications*, June 2002.

[9] P. Tullmann, M. Hibler, and J. Lepreau, "Janos: A java-oriented os for active network nodes," 2001. [Online]. Available: citeseer.ist.psu.edu/652534.html

[10] D. K. Y. Yau and X. Chen, "Resource management in software programmable router operating systems," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, Mar. 2001. [Online]. Available: http://citeseer.ist.psu.edu/324757.html

[11] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly virtual machines - leveraging a feedback-control model for application adaptation." [Online]. Available: citeseer.ist.psu.edu/article/zhang04friendly.html

[12] J. Jin and K. Nahrstedt, "Qos specification languages for distributed multimedia applications: A survey and taxonomy," *IEEE MultiMedia*, vol. 11, no. 3, pp. 74–87, 2004.

[13] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. B. m, and R. Neugebauer, "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, Oct. 2003.

[14] J. Novotný, O. Fučík, and D. Antoš, "Project of IPv6 Router with FPGA Hardware Accelerator," in *Field-Programmable Logic and Applications, 13th International Conference FPL 2003*, P. Y. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds., vol. 2778. Springer Verlag, September 2003, pp. 964–967.