

An Active Node Architecture with Resource Management

Yuhong Li¹, Lars Wolf

Institute of Telematics, University of Karlsruhe

76128 Karlsruhe, Germany

{li,wolf}@tm.uka.de

Abstract: Resource management in active nodes is important to protect the system resources offered by a node against abuse and to provide for application performance. Resource management mechanisms for active networks are needed including an according node architecture, approaches for the description of resource requirements, and adaptation methods to balance resource needs and availability among different applications. In this paper, we present a method for the description of resource requirements from applications, suggest a node architecture with a resource management system which also provides adaptation among different applications. Finally, we describe the implementation of the resource management system in the node architecture.

Keywords — Active networks, resource management

1. Introduction

In recent years, active and programmable networks attracted much attention because of their flexibility and application-oriented properties. In these systems, user-specific algorithms to be executed in network components can be developed and composed allowing for application-specific computations in network nodes and value-added services [TW96]. Hence, the needs of applications can be better fulfilled, applications can deploy new protocols very quickly and change their services dynamically for specific purposes.

In active networks (AN), mobile code can be injected by users into the network at run-time in the form of active packets. An execution environment is provided in the AN nodes for the execution of the injected codes. This way, users realize their more complicated tasks by acquiring additional “powers” from the networks. These powers are indeed resources in the networks. For AN nodes this means,

¹ Contact author, telephone: ++49-721-6088073, fax: ++49-721-388097

however, that more of their resources are open to the users: users are permitted to utilize the resources within the network nodes in ways not foreseen by the network providers. This poses a new challenge to the resource management in ANs.

Considering resource usage in ANs, it can be seen that more resources inside the networks are needed – with respect to the types of the resources and to their amount. Furthermore, more and more applications and services will be introduced and deployed in ANs. The result of this diversity of applications and services is that the consumption of different kinds of resources is not equally balanced: some applications prefer more communication resource, while others need more computation or memory resources or both of them. Therefore, the status of the different kind of resources in the network nodes may be greatly different at different time points. Hence, in some cases an application cannot be accepted by the network nodes only because a single resource is inadequate though there is a surplus of resources of other types.

The goal of our work is to approach the above problems by developing an active node architecture with explicit resource management. We introduce the resource-vector (RV) to describe the resource usage in ANs. Based on that, our system provides admission control and resource adaptation mechanisms inside an application or among different applications. The remainder of the paper is organized as follows: in section 2, we depict our method to express the resources consumed by active applications. Section 3 introduces our active node architecture with resource management. Following this in section 4, we discuss implementation issues of the architecture. We compare our work with other resource management and control methods in active and programmable networks in section 5, and finally conclude in section 6.

2. Description of Resource Consumption in Active Networks

As discussed above, the injection and execution of the application-specific code in active nodes have a great effect on the consumption of the network resources. In general, the following characteristics are distinct in ANs from traditional IP-based networks.

Multi-dimensional: just like the transmission bandwidth, both computation and storage resource are important and must not be neglected in ANs.

Complementarity: the different kinds of resources required by an application can sometimes be complementary to each other. For example, when there is not enough transmission bandwidth while sufficient computation resources in the network, the media data of a videoconference may be compressed more highly at a node and then transmitted further. Through this method, the ultimate performance of the applications can also be satisfied. Another example is the representation of certain data structures, like sparse matrices and hashtables, where well-know tradeoffs exist between storage space and element access time.

Higher-level sharing: In traditional IP-based networks, network resources are shared among links or connections. But in ANs, resources are consumed by different execution environments (EEs), i.e. resources in ANs are shared among EEs.

These unique characteristics, especially complementarity, result in that the resources required by an application in ANs cannot be easily expressed just using a simple list of different kinds of resources such as bandwidth, CPU cycles and memory size.

Considering the above characteristics of resource consumption in ANs, we use a resource-vector (RV) to describe the resource usage in an active node, e.g. the total resources in an active node, the resources required, reserved and consumed by applications. Our goal is to consider and approach the resource consumption in ANs by different applications in general and overall, taking the consumption of different types of resources into account.

We first review the resource organization in an active node, and then give the definition and description of the RV. Here we can give an overview about our resource description approach only. More details can be found in [LW01].

2.1. System Resource Organization

As suggested in [AN01], functionally of an active node consists mainly of two parts: execution environments (EEs) and node operating system (NodeOS). The NodeOS multiplexes the node's

communication, memory and computational resources among the various packet flows that traverse the node, and provides basic functions that EEs can use on behalf of applications. Applications can however access the resources in the system only through EEs. In the same time, there may be multiple types of EEs in one active node, applications can specify in their packets to the networks, which type of EE they use[ABG+97].

When the first packet of an application arrives in the active node through the physical input port, a domain under the specified type of EE will be created for this application, and a certain amount of resources will be allocated to it if there are enough resources in the system. In this case the domain for an application can be considered as an instance of the specified EE type.

Therefore we use a hierarchical model to organize the system resources. The system resources are assigned to different types of EEs. Each type of EE possesses a certain amount of the system resources¹. The resources owned by each type of EE will be dynamically allocated to different instances of the EE based on RV introduced in the next section. The resources possessed by each EE instance can be simply described using RV.

2.2. RV Definitions

Resources allocated to instances

Let $r_{i,j}$ denote the portion of j^{th} type of resource used by i^{th} EE instance, then the total amount of resources occupied by instance I_i is : $\mathbf{r}_i = \{ r_{i,1}, r_{i,2}, \dots, r_{i,n} \}$. We call \mathbf{r}_i a **resource vector (RV)** of instance i . Apparently each component of \mathbf{r}_i represents one kind of resource required to guarantee the applications' performance. Here, n is the number of resource types that will be considered in the system, for example, if CPU, memory and network bandwidth resources are concerned in the nodes, n is equal to 3.

¹ The resource amount assigned to an EE can be based on statistics of the number of applications supported by each type of EE and their function.

RV Space

In order to acquire the guaranteed performance for an application, the resource consumption of the application in an AN is not a fixed value. The reason is that (i) the resource requirement specified by an application using a definite RV cannot always be satisfied in the “best-effort” networks and (ii) diverse resource combinations, each of which can be described using a RV, can sometimes meet the same performance requirements of the application. Hence the resource requirement from an application can be considered as a set of RVs. These RVs build a space. We call this space the RV space of an application. Each RV in the RV Space of an application can satisfy the performance requirement of the application.

In practical use, the RV space can be extended and constructs a restrictive condition for resource coordination, adaptation etc. For example, it can be used for adaptation in the phase of admission control if it is specified in the form of a possible adaptation RV space. For a certain application, there may be a lower limit to some type of resources. In other words, when one of these limits cannot be satisfied, no matter how many resources of other types are available in the system, the required performance of the application cannot be met. We use \mathbf{RV}_L ¹ to describe this lower limit. It records the minimum value of different kind of resources that an application needs. Similarly, for the sake of costs etc., there may be also an upper limit to resources that an application requires, we denote this by \mathbf{RV}_U . Therefore, a possible adaptation space can be stipulated with \mathbf{RV}_L , \mathbf{RV}_U , the worst efficacy and the highest cost that the application accepts; the resource adaptation can be done in this space.

RV Operations

In order to be able to use RV for resource management purposes, such as admission control and adaptation, we define some operations for RVs. Suppose $\mathbf{r}_1 = \{r_{1,1}, r_{1,2}, \dots, r_{1,n}\}$, $\mathbf{r}_2 = \{r_{2,1}, r_{2,2}, \dots, r_{2,n}\}$

¹ \mathbf{RV}_L and \mathbf{RV}_U itself may be not in the RV space, though they are used to describe the possible adaptation space.

}, we define operations such as $\mathbf{r}_1 + \mathbf{r}_2$, $\mathbf{r}_1 - \mathbf{r}_2$, $\mathbf{r}_1 / \mathbf{r}_2$, $\mathbf{r}_1 > \mathbf{r}_2$, $\mathbf{r}_1 < \mathbf{r}_2$, and $\mathbf{r}_1 = \mathbf{r}_2$ by applying the according operations on corresponding components of the RVs.¹

Clearly the importance of each type of resource is not same for the performance of an application. For example, a time-constrained application such as a video-conference prefers bandwidth over storage while a bulk transfer application probably prefers to store as much information as possible when links are congested. Hence, a weight factor is also defined together with RV in our system.

The advantages of using resource vector to describe the resources in ANs are summarized as follows:

- The RV reflects the fact that applications in ANs need multiple kinds of resources to fulfill their tasks.
- The RV space represents the characteristic in ANs that the resource requirement of an application for acquiring a definite performance is a space instead of a definite point.
- The resources used by active applications can be represented using RV, just like there is only one kind of resource needed in the classical network, in the resource reallocation or adaptation algorithm. This simplified greatly the description of the resource management algorithm.

3. Active Node Architecture with Explicit Resource Management

The architecture of our active node with an explicit resource management system is illustrated in Figure 1. This active node architecture consists also of the major three components as suggested in [AN99], i.e. NodeOS, EE and AA (Active Application). However, a major difference from [AN99] and other related projects in the field of active networking is that we introduce an explicit resource management system in the NodeOS to manage the communication, CPU and memory resources in a network node. Here we do not repeat the description of the EEs and NodeOS function, but we discuss the modules in the resource management system in the following.

¹ Note that for the division operation, the components of \mathbf{r}_2 cannot be zero, however, this is uncritical since all applications need at least some resources of the different types. Note also that we have partial orders only.

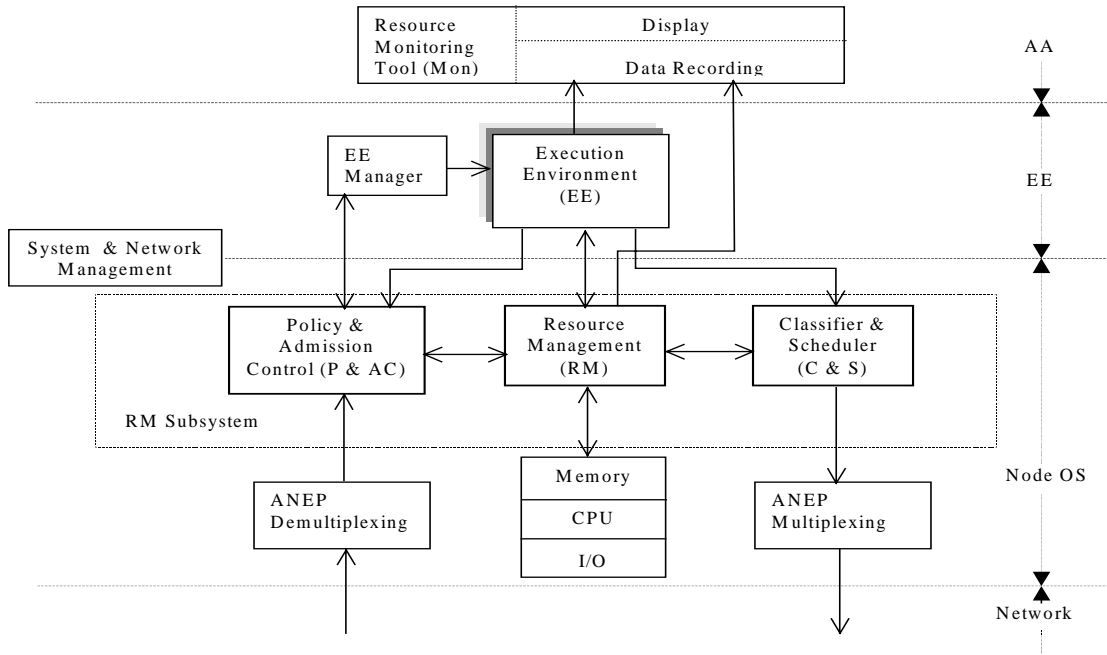


Figure 1: An Active Network Node Architecture with Resource Management

3.1. Resource Management System

As shown in Figure 1, the resource management system in the node architecture consists of mainly 3 modules:

Policy & Admission Control (P&AC): The policy control decides whether incoming packets from the user applications have the permission to be executed in this node. Admission control determines whether there are enough resources in the system when a new EE instance must be created according to the requirements of an application. During the admission control phase, the resource management scheme attempts to bind system resources to this application. A resource adaptation procedure can occur in this phase if there are not enough spare resources in the system.

The tasks of the P&AC module are:

- Receive an application request from the ANEP Demultiplexing module which contains the amount of resource requirements in the form of RV and the possible resource adaptation space.
- Ask the EE Manager module if the incoming packet belongs to an existing EE instance.
- Ask the RM module whether the presented amount of the resources can be met.

- Notify the EE Manager module that a new instance must be created with the given resources after receiving a positive response from the RM module.
- Discard the packet in case of a negative response from the RM module.

Only the resource requirements from the users are taken into account by this module for the moment. Whether the applications have permission to run in this node is for further study.

Classifier & Scheduler (C&S): This module schedules the execution of the applications and the transmission of packets to the next node. The resources that an application can acquire are mapped into the hybrid thread/event and channel model as suggested in [AN01] by our implementation of the node.

- Each thread pool is given a scheduling period and time slice according to the Rate Monotonic policy [Liu73], its threads are only eligible to run when it receives CPU time. Additionally, each thread pool is pre-allocate a definite number of ideal threads. These ideal threads are dispatched to process events such as timeout and exceptions.
- A channel gets a transmission scheduling period and slice of transmission time following the outgoing rate guarantees that an instance gets from the system finally.

Clearly this module needs information from both different instances and the system.

Resource Management (RM): The resource management scheme is implemented in this module. The main functions of the RM are to:

- possess all the information about the resources in this network node;
- provide necessary information to the P& AC and the C&S module;
- monitor and adjust the resources used by each EE instances and send notifications to the EE instances to tell them the resource status of the system, e.g. how many resources are available and can be used or how many are needed;
- hold different resource management algorithms, e.g. the resource management scheme discussed below is stored in this module. The general resource management schemes can be set by the system and network management EE or be received from application EEs at runtime.

3.2. Further Components

Beside the resource management system, we also introduced an active application, namely a resource monitor tool, in the node architecture. The goal is to observe the status of the resources in the network node, e.g. to monitor the resource usage by the different EE instances as well as the unused resources which are still available in the node. For the moment, we implemented a graphical monitoring tool as an application running in the active nodes. It consists of mainly two blocks, one each for Data Recording and Display. The Data Recording block gathers the resource state information from the RM module and the EE instances. The Display block correspondingly provides a graphical presentation of the recorded resource state in the system. Using this tool, the resource consumption of different applications and the system resource state can be dynamically and graphically illustrated. Moreover, with the help of the Data Recording block, remote monitoring and resource state collection of the network node can be performed. This application needs to exchange some information with EE instances and the NodeOS. In order to reduce the complexity of the active node, in the current implementation this graphical monitor application uses a direct communication with the NodeOS.

Additionally due to the introduction of admission control and the hierarchical organization of the resources, we also provide a simple manager for each type of EE. Its main tasks are to create a new EE instance and to decide if a newly incoming packet belongs to an existing EE instance.

The System and Network Management EE module is used to configure the system, e.g. to set ANTS2.0 as EE, Janos as NodeOS etc. Besides this, the resource management scheme used in the RM module is also set by the system and network management EE module. This allows for a dynamic configuration and even exchange of the RM module during runtime. However, the System and Network Management EE itself is created statically at system start time.

3.3. Resource Management Scheme

The resource management scheme contained in the RM module of the resource management system is very important. It affects not only the admission control in the P&AC module, but also the allocation of the resources possessed by each type of EEs which can then be assigned to the instances. It directs also

the C&S to transmit packets to the next hop in the whole network. Moreover, it also takes care of the adaptation of resources assigned to the different EE types. In the following we briefly present the currently used resource adaptation scheme(see [LW01] for further details).

As mentioned above, the resource adaptation scheme is configured by the System and Network Management EE. This occurs in two cases:

- In the phase of admission control. When the RV from a new arriving application cannot be satisfied directly, i.e. if one or several dimensions of the RV cannot be met because of the lack of the corresponding resources in the system.
- During the “pre-lifetime” of an application. I.e. some resources have been reserved for applications, but still have not been used. In this case, the existing RVs may be changed.

When the resource adaptation occurs, the following steps are performed:

- Adaptation within an EE instance, i.e. change the length of some dimensions of the RV, is done first. In order to guarantee the QoS of the applications, the adjustment of the RV should be made in the possible RV adaptation space. This means that if one kind of the resources is not sufficient, say network bandwidth, CPU resource can be used as a complement to it. For instance, a more network bandwidth efficient compression method can be applied for the transfer of video data. In result the total delay observed by the video application may be the same as before.

Currently, we use a “lowest price” method to adjust RV, namely if there are multiple possibilities to adjust RV (e.g. both CPU and memory can complement bandwidth), we first choose a method to ensure that the cost of the total resource consumption by the application is lowest. For this purpose we use the price to represent the resource status in the system. The fewer one type of resource in percentage, the higher the price. Therefore the total cost of an application is a function of the resource price and the amount of the resources that an instance consumes. Another strategy for the adjustment of RV could be, for example, to make the decrease of an application’s efficacy lowest.

- Adaptation among EE instances that belong to the same type of EE is performed as second choice if it is impossible to adjust the dimension of a RV in the possible adaptation space specified by the

specific application. Then an adjustment among RVs of other applications resp. EE instances must be done. The result must be that all the affected RVs still reside in their own RV space.

- If also the adjustment among EE instances belonging to the same type of EE cannot succeed, adjustments can occur among RVs that belong to different EE types.

4. Implementation

4.1. NodeOS and EE

Though the particular NodeOS and EEs are not the focus of our discussion, they are the basis of the resource management system. To date several NodeOS and EEs have been defined and implemented, including Janos, PAN, ANTS, PLAN and CANE etc. Among these we selected Janos and ANTS2.0 as the basis of our NodeOS and EE respectively.

We use the Janos [JAN] developed in the university of Utah as the basis of our NodeOS in the network node architecture. The most important reason is that Janos [PGH+01][THL01] provides already some basic resource control functions through limiting the different kinds of resources used by active applications. Moreover, Janos provides also most of the functions specified by the AN Node Working Group in the NodeOS Interface Specification [AN01], and fulfills part of our tasks of implementing a general active node architecture with resource management. Our resource management system, as shown in Figure 1, is added into the Janos.

ANTS2.0 [ANT] from the University of Washington and the University of Utah has been selected as the basis of our EE¹. ANTS2.0 is also based on Janos; uses it to access the network, create threads etc. Further, ANTS2.0 absorbed the intelligent per-protocol and per-application resource controls from the Janos project.

However, we applied some modifications to Janos and ANTS2.0 to implement the resource management function in the active node.

¹ Our current node architecture implementation provides for one type of EE only. Hence, for the moment, adaptations are done within one EE instance and among EE instances that belong to the same type of EE only.

- The whole resource management system, including the RM, P&CA and C&S modules, are integrated into the Janos. RV is the basic unit for resource description in our node architecture.
- Some interfaces are added to Janos and ANTS2.0 in order to communicate with the resource management system.

For example we have defined a class ResAdaptation with the methods of adaptReqReceived(), sendAdaptResp() etc. in ANTS2.0 as a corresponding response to the RM module in NodeOS.

- In Janos, resources are specified in a very precise, hardware-dependent way, for example, physical memory limits are specified in terms of memory pages. Since we introduced RV as our resource description method, we had to add a method to map between the RV in our resource management system and the precise values that Janos needs.

For this purpose, we add the classes of CPUSpec and NetBandwidthSpec etc. that are not supported currently in JanosVM and define the classes of RVtoMemSpec, RVtoCPUSpec as well as RvtoNetBandwidthSpec etc.

In addition, the EE manager is integrated into ANTS2.0, whose functions are mainly receiving requests from the P&AC module and deciding if new flows should be created in the ANTS2.0 and Janos.

4.2. Resource Management Subsystem

According to the resource management scheme introduced in section 3.3, the resource management subsystem must be able to know the status of the resources in the system at any time. For this purpose, query methods are used to acquire system information in the RM module. Additionally, as another important issue of the resource management subsystem, accounting for the resources consumed by each instance must be provided. With the help of Janos, information about the bandwidth and memory used by an instance can be gathered relatively easily. However, the CPU time used by an instance is difficult to calculate. At the moment, we account the per-thread CPU time, similar to [CE98], but the threads are limited to one domain.

The system resource information as well as the information about the resources that an instance has consumed is periodically sent to the Monitor module. Correspondingly, the P&AC and EE manager module etc. need the system status information from the RM module. Figure 2 illustrates a scenario of the information exchange in a normal case inside the active node. ↻ means that an adaptation has occurred.

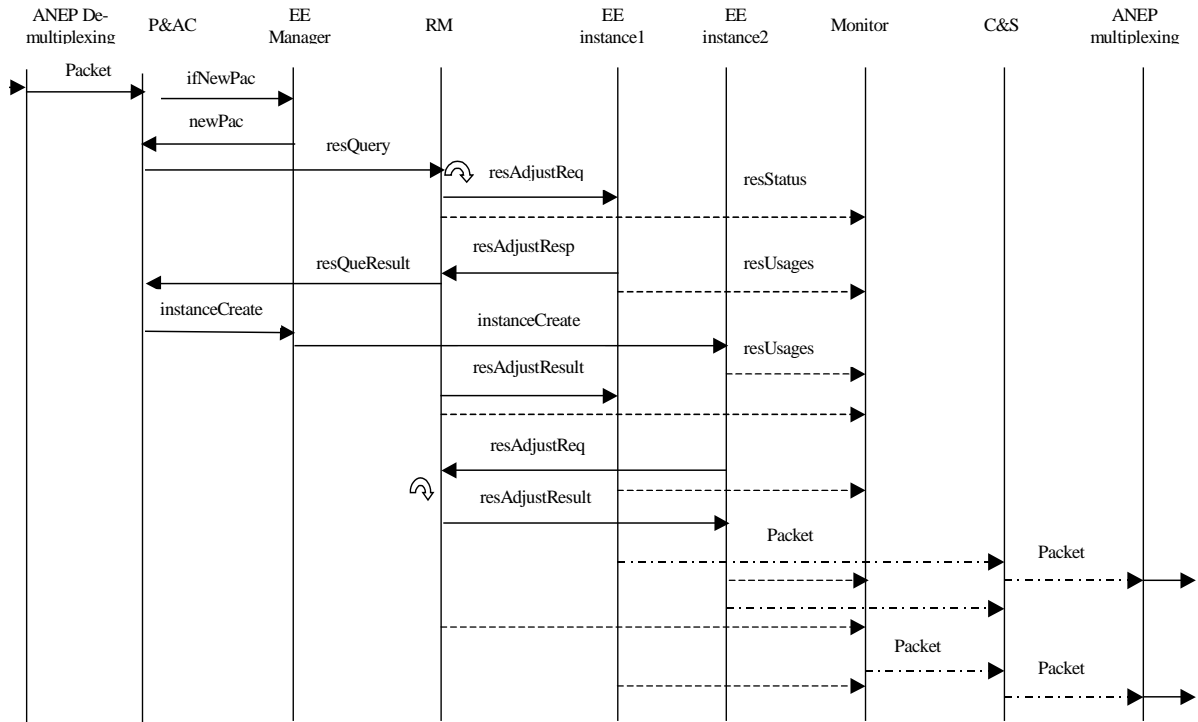


Figure 2: Information Exchange inside the Network Node (Example)

To test our resource management scheme, we developed an application running on top of ANTS2.0. The application sends packets. Their number and length is inverse proportional to the CPU requirement that it specifies in the resource request to the system. Hence, its bandwidth requirement is inversely proportional to the CPU requirement and the total resource requirement of the application can be satisfied easily after the adaptation of different kinds of resources. We use a background program to control the CPU load of the system. Through the monitor tool we can see whether the application is accepted by the node system and what happens, i.e. the changes of the amount of CPU and bandwidth resource, when the application is accepted.

5. Related Work

In the following we contrast our work with those related to resource management in ANs. Other work, e.g., addressing more current IP-based networks such as [BGO+98][DGL+97][JLD+95] is not used for comparison due to the specific characteristics of resource consumption in ANs.

Resource management in ANs first attracted attention for the sake of security and some measures have been taken against the abuse of system resources. ANTS[WGT98][Wet99] uses TTL to limit the times that a packet can access active nodes; however, there are no limitations for the amount of resources an active packet can use at one time. Janos is an operating system for AN nodes designed to prevent separate active applications from interfacing with one another and to provide administrators with strong controls over active applications' resource usage[THL01]. Janos' primary focus is strong resource management and control of untrusted active applications. However, resource management in the Janos system means primarily resource control. Similar to Janos, the starting point of the resource controlled active networks environment (RCANE) [Men99] is to withstand many classes of denial of service (DoS) attack. RCANE supports an AN programming model over the Nemesis Operating System, providing robust control and accounting of system resources, including CPU and I/O scheduling, and garbage collection overhead.

Compared with these approaches, we provide an explicit resource management subsystem in our node architecture. We keep and use the resource control functions offered by Janos and ANTS2.0. Additionally, our work emphasizes the adaptation among the different types of resources needed by an active application, which have not been considered till now by other works. Besides this, we introduce a new and more general method for the description of the resource consumption.

The core router operating system support (CROSS) [YC01] is a project that concerned with the development of an OS that will be used in the software programmable routers. With regard to the resource management, CROSS targets diverse router resource types, including CPU cycles, network bandwidth, state-store capacity and disk bandwidth, and support multiple virtual machines exporting

different router APIs, i.e. different type of EEs. They use different scheduling techniques for different kinds of resources. However, they do not concern the relationship between different kinds of resource.

The research group in NIST concentrates on predicting and controlling resource usage in a heterogeneous AN [GMC01a][GMC01b]. Their starting point is that at different network nodes (with different levels of CPU speed, memory capacity, operating system, etc.) the resource requirements (especially the CPU requirement) for mobile code is different. Hence a model is needed to predict the CPU resource requirement in the intermediate nodes according to that in the source node. They use AVNMP (Active Virtual Network Management Prediction) as an overlay network over which a simulation model runs ahead of real time to predict resource demand among network nodes. Compared with this work, our work concentrates on one active node without studying the problem of heterogeneous active nodes at the moment. However, we address the problem of resource adaptation in our architecture.

Within the work on Spawning Networks [CKV+99] from the programmable networks community, the resource management system Virtuosity in the Genesis Kernel [CVV99] [VCV01] addresses the function of admission control, resource allocation, monitoring and policing etc. Virtuosity implements a measurement-based virtual network admission control test during the spawning phase of a child network. In the case that the parent has insufficient resources to accommodate the new child network then it needs to renegotiate its needs with its own parent at the next level down its virtual network hierarchy tree, called dynamic provisioning. Compared with Virtuosity, our scheme can be also regarded as hierarchically and dynamic to some degree since the resource adaptation can be done in the level of application resp. EE instance, EE type and the whole active node. However, Virtuosity addresses mainly link bandwidth and there is no negotiation among different type of resources, but these functionality is offered by our system.

6. Conclusions

In this paper we discussed an AN node architecture with an explicit resource management system. We use RVs to describe the resource requirements from applications and the resource capabilities in the

system. The possible resource adaptation space of a RV and the resource adaptation scheme in the RM module make it possible and easy to realize the complementarity between different kinds of resources in case of resource scarcity in the AN node. Based on the resource control functions provided by Janos NodeOS and ANTS2.0, together with the admission control and resource adaptation functions in the resource management system, our node architecture provides for better performance guarantees and more flexibility to the applications.

References

- [ABG+97] D.Alexander, B.Braden, C.A.Gunter, A.W.Jackson, A.D.Keromytis, G.J.Minden and D.Wetherall, “Active Network Encapsulation Protocol (ANEP)”, July 1997.
- [AN01] AN Node OS Working Group. Node OS Interface Specification. January 10,2001.
- [AN99] Active Network Working Group. Architectural framework for active networks, version 1.0. Available from <http://www.darpa.mil/ito/research/anets>, July 1999.
- [ANT] <http://www.cs.washington.edu/research/networking/ants/>
- [BGO+98] J.Bruno, E.Gabber, B.Ozden and A.Silberschatz, “ The Eclipse Operating System: Providing QoS via Reservation Domains”, Proceedings of the USENIX Annual Technical Conference, June 1998.
- [CE98] G. Czajkowski and T. von Eicken, “JRes: A Resource Accounting Interface for Java”. Proc. of the 1998 ACM OOPSLA Conference, Vancouver, BC, October 1998.
- [CKV+99] A.T.Campbell, M.E.Kounavis, D.Villela, J.Vicente, H.De Meer, K.Miki, and K. S. Kalaichelvan “Spawning Networks”, IEEE Network, Vol. 13 No. 4 pg. 16-30, July/August 1999.
- [CVV99] A.T.Campbell, J.Vicente, and D.A.Villela, “Virtuosity: Performing Virtual Network Resource Management”, 7th IEEE/IFIP International Workshop on Quality of Service (IWQOS'99) , pg. 65-76, London, June 1999.
- [DGL+97] T.Dewitt, T.Gross, B.Lowekamp, N.Miller, P.Steenkiste and J.Subhlok, “ReMos:A Resource Monitoring System for Network Aware Applications”, Tech. Rep. CMU-CS-97-194, Dec. 1997.

- [GMC01a] Virginie Galtier, Kevin L. Mills, Yannick Carlinet, Stephen Bush, and Amit Kulkarni, “Predicting and Controlling Resource Usage in a Heterogeneous Active Network”, WOAMS 2001, San Francisco, August 6, 2001
- [GMC01b] V. Galtier, K. Mills, and Y. Carlinet , S.Bush and A:Kulkarni, “Predicting resource demand in heterogeneous active networks”, to appear in MilCom 2001. <http://w3.antd.nist.gov/active-nets/>.
- [HMA+99] Michael Hicks, Jonathan Moore, D.Scott Alexander, Carl Gunter and Scott Nettles, “PLANet: An Active Internetwork”. Proceedings of IEEE Infocom’99, March 1999.
- [JAN] <http://www.cs.utah.edu/flux/janos>
- [JLD+95] M.B.Jones, P.J.Leach, R.P.Draves, and J.S.Barrera, “Modular Real-Time Resource Management in the Rialto Operating System”, Proceedings of the 5th Workshop on Hot Topics in Operating Systems, May 1995.
- [Liu73] C.Liu and J.Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment”. Journal of the ACM, 20(1) 46-61, Feb. 1973.
- [LW01] Yuhong Li, Lars Wolf, “Resource Description and Adaptive Resource Management in Active Networks”, submitted for publication.
- [Men99] Paul Menage, “RCANE: A resource controlled framework for active network services”, presented at the IWAN, Berlin, Germany, 1999.
- [PGH+01] Larry Peterson, Yitzchak Gottlieb, Mike Hibler, Patrick Tullmann, Jay Lepreau , Stephen Schwab, Hrishikesh Dandekar, Andrew Purtell, John Hartman. An OS Interface for Active Routers. To appear in IEEE Journal on Selected Areas of Communication. March 2001.
- [THL01] Patrick Tullmann, Mike Hibler, and Jay Lepreau. Janos: A Java-oriented OS for Active Network Nodes. IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, March 2001.
- [TW96] D.Tennenhouse and D.Wetherall, “Towards an Active Network Architecture”, Proc. Multimedia Computing and Networking, San Jose, CA, 1996.

- [VCV01] D. Villela, A.T.Campbell, and J. Vicente, "Virtuosity: Programmable Resource Management for Spawning Networks", Computer Networks, Special Issues on Active Networks, 2001.
- [Wet99] David Wetherall. Active Network vision and reality: lessons from a capsule-based system. Inproceedings of the 17th Symp. on Operating System Principles, December 1999.
- [WGT98] David Wetherall, John Guttag and David Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In IEEE OPENARCH98, San Francisco, CA, April 1998.
- [YC01] David K.Y.Yau and X.Chen, "Resource Management in Software-Programmable Router Operating Systems", IEEE Journal on selected Areas in communication, vol.19, No.3, March 2001.