

PREDICTING RESOURCE DEMAND IN HETEROGENEOUS ACTIVE NETWORKS

V. Galtier, K. Mills, and Y. Carlinet
National Institute of Standards and Technology

S. Bush and A. Kulkarni
General Electric Corporate R&D

ABSTRACT

Recent research, such as the Active Virtual Network Management Prediction (AVNMP) system, aims to use simulation models running ahead of real time to predict resource demand among network nodes. If accurate, such predictions can be used to allocate network capacity and to estimate quality of service. Future deployment of active-network technology promises to complicate prediction algorithms because each “active” message can convey its own processing logic, which introduces variable demand for processor (CPU) cycles. This paper describes a means to augment AVNMP, which predicts message load among active-network nodes, with adaptive models that can predict the CPU time required for each “active” message at any active-network node. Typical CPU models cannot adapt to heterogeneity among nodes. This paper shows improvement in AVNMP performance when adaptive CPU models replace more traditional non-adaptive CPU models. Incorporating adaptive CPU models can enable AVNMP to predict active-network resource usage farther into the future, and lowers prediction overhead.

INTRODUCTION

Growing availability of processing power and bandwidth in communication networks encourages innovative approaches to network management. One specific innovative idea envisions injecting simulation models into network nodes, and then running those models in parallel with the operational network, but ahead in time, in order to predict traffic and resource use. If the models predict accurately, then network management systems can better allocate capacity in anticipation of varying demands and network operators can better estimate the quality of service (QoS) that customers can expect. This paper describes one approach, the Active Virtual Network Management Prediction (AVNMP) system [1], which aims to predict network traffic. AVNMP uses active-network technology [2] to inject simulation models into network nodes, and to run those models concurrently with corresponding applications. AVNMP then compares estimated performance against measured

performance, and maintains predictions from the simulation within specified error bounds, when compared against measurements from the application.

AVNMP can estimate resource requirements for each node in a conventional communication network, such as the Internet. In conventional networks, accurate estimates for message quantity and size directly imply nodal resource requirements for bandwidth, memory, and processor (CPU) cycles. This holds because packets receive the same fundamental processing within each node. Unfortunately, the future deployment of active network technology promises to negate the simple, fixed relationship between packets and resource use. This will occur because each packet in an active network can carry code, or a reference to code, which must be loaded on demand and applied to the packet. This implies that in active networks the processing of individual packets can differ, demanding varying quantities of node resources, particularly CPU cycles. We set out to investigate how AVNMP might be used to predict resource consumption in an active network. This paper reports our initial findings.

The paper is organized into seven sections. First, we provide a brief tutorial on active networks. Second, we describe how AVNMP uses active-network technology to predict traffic load in a conventional network. Third, we discuss how we augmented AVNMP to predict CPU usage in heterogeneous active networks. Here heterogeneity implies that the active network comprises a wide range of node types with various hardware capabilities and software configurations. This creates additional complexity because the demand for CPU cycles varies not only by packet type but also by node type. Fourth, we outline an experiment where we used AVNMP to predict resource consumption by an active audio application. Our results suggest that adaptive CPU models [3], which accommodate variations in node capabilities, can improve the accuracy of AVNMP predictions, and reduce prediction overhead. Fifth, we suggest some additional applications for AVNMP, and similar prediction systems. Sixth, we identify some future research suggested by our work. Finally, we present our conclusions from the current experiment.

ACTIVE NETWORKS

Active-network technology augments traditional networking with the possibility that individual packets carry executable code, or references to executable code. Conventional packets are forwarded on the fast path of a router, while active packets are delivered to a higher-level execution environment that can identify and run code associated with the packet. Networking applications built with active packets are referred to as active applications. Figure 1 illustrates the architecture of an active-network node [4].

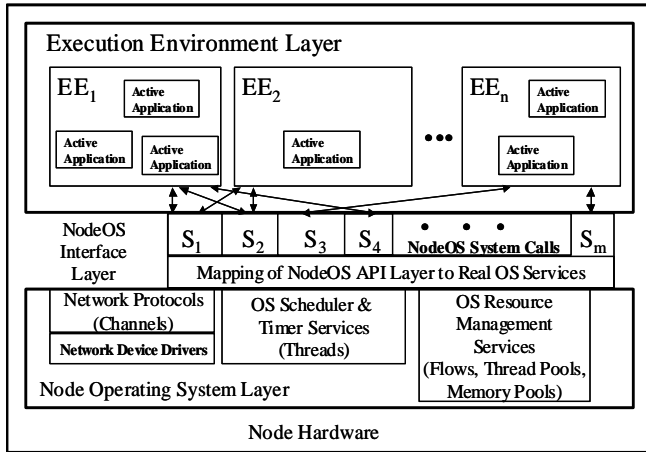


Figure 1. Architecture of an Active-Network Node

Underlying each active-network node is a node operating system, which transforms the hardware into a software abstraction to provide execution environments with controlled access to resources such as CPU cycles, memory, input and output channels, and timers. In order to allow many possible operating systems to provide services to many possible execution environments, the active-network node architecture includes a standard specification of system calls (the Node OS Interface Layer in Figure 1) [5]. Execution environments, similar to virtual machines, can be loaded onto an active node using ANETD [6], a daemon that implements a load-and-go protocol for execution environments. Each execution environment accepts active packets that can initiate the execution of packet-specific code. Each related code base and flow of active packets is known as an active application. In our experiment, we used the Magician execution environment [7], implementing an active audio application. AVNMP is also implemented as an active application.

PREDICTING TRAFFIC LOAD

To predict traffic load in a network, AVNMP constructs a shadow topology that overlays the operational network and then runs a simulation in the shadow topology.

Figure 2 illustrates the relationship between the operational network and the shadow, prediction-overlay network. Using Magician, AVNMP deploys driving processes (DP) at each source node and logical processes (LP) at each intermediate and destination node in the topology of the operational network. DPs and LPs are deployed as active applications within an active virtual-overlay network (space dimension in Figure 2). Each DP contains a model that simulates message sources, generating virtual messages that flow along links in the virtual-overlay network, which share physical links between nodes but remain logically isolated from operational traffic. As virtual messages arrive, the LP updates variables in the node's management information base (MIB) [8]. Each LP updates the future state of relevant MIB variables, providing the MIB with predicted state to complement the current and past state maintained by the operational network. After updating predicted MIB variables, the LP consults the node's routing table and forwards incoming virtual messages on to other LPs, if required.

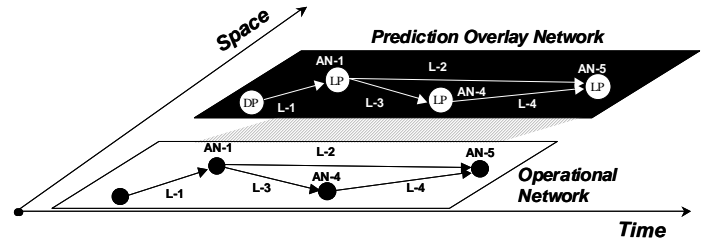


Figure 2. AVNMP as a Prediction Overlay Network

The prediction-overlay network then generates and routes simulated network traffic that attempts to run ahead in virtual time of operational network traffic (time dimension in Figure 2). While the operational network advances in real time, the LP in the prediction-overlay network advances in virtual time, receiving virtual messages and estimating future load. Periodically, the LP compares the actual and predicted MIB values for corresponding intervals in real and virtual time. If the values agree within an error tolerance, then the simulation remains ahead of real time and continues to advance. If not, then the LP rolls virtual time back to the current real time, discarding predictions for future MIB state, and then simulation resumes. AVNMP contains some special processing to cancel virtual messages that might be in transit across the prediction overlay network during a rollback, but we omit these details.

PREDICTING CPU USAGE

Once message load (in number and size) can be predicted, then CPU usage for conventional networks can be estimated. Since each node has a rating for per-

message and per-byte throughput, a linear extrapolation should provide reasonable estimates for CPU use. Unfortunately, this simple approach cannot work for active applications because individual packets can require substantially different processing. Worse, identical packets can require different CPU time on various nodes. To enable AVNMP to predict CPU usage in active networks, we decided to investigate adaptive CPU modeling techniques as an alternative to the more rigid approaches used by most active-network execution environments to constrain CPU use by active packets.

Every execution environment employs some approach to prevent erroneous or maliciously coded active packets from consuming excessive CPU time on a node or in the network. Some execution environments assign a fixed resource limit to individual active packets; some environments assign a maximum bound to any active packet; some environments combine these approaches. Such rigid techniques can cause significant problems for active applications because the CPU time needed to execute an active packet can vary from node-to-node. Sometimes valid active packets can be terminated too soon because they exhaust their CPU limit. In other cases, erroneous or malicious active packets can “steal” excessive CPU time because they are permitted to execute for too long. Similarly, should AVNMP use these fixed limits to estimate CPU time usage for an active application, the estimates may be too low or too high, depending on the capabilities of individual network nodes.

These problems motivated us to investigate adaptive models to represent the CPU use of active applications in a form that can be scaled among heterogeneous nodes [3]. We suspected that such adaptive models might improve the ability of AVNMP to predict CPU usage in heterogeneous active networks, while also reducing the number of simulation rollbacks. To investigate these hypotheses, we collaborated in an experiment, discussed next.

AN EXPERIMENT AND RESULTS

As shown in Figure 3, we constructed a four-node, heterogeneous active network, consisting of source (200 MHz Pentium Pro/64 Mbytes) and destination nodes (450 MHz Pentium II/128 Mbytes) separated by two intermediate nodes: one faster (333 MHz Pentium II/128 Mbytes) and one slower (120 MHz Pentium/64 Mbytes). All nodes included Magician running on a Java virtual machine (jdk 1.2.2) supported by Linux (release 2.2.7). The operational active network comprised these nodes connected to a switched 10-Mbps Ethernet, which included a few other nodes that were not part of the experiment. We configured the experiment nodes to run

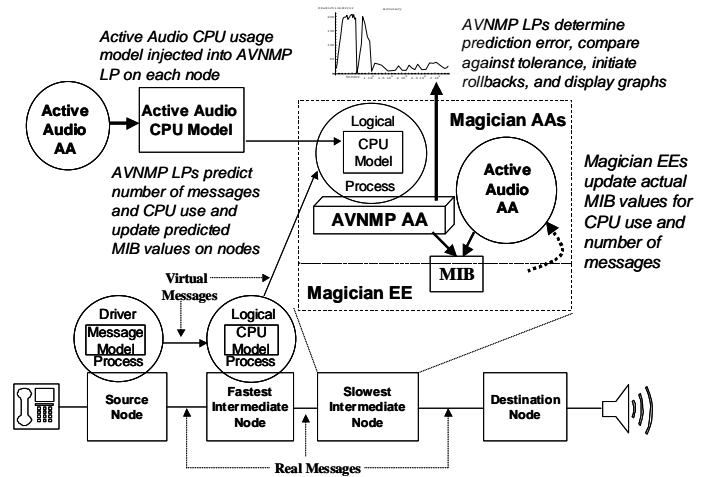


Figure 3. Experiment Configuration

an active audio application. The prediction overlay network included AVNMP deployed as an active application on each node, with a DP injected into the source node and an LP injected into the destination and each intermediate node. The DP included a message model to generate virtual message traffic and a CPU model to simulate processor use associated with each virtual message. Each LP included a CPU model to simulate processor use for each arriving virtual message.

We conducted two experiment runs. In the first run the DP and LPs predict a fixed CPU time for each virtual message on every node. In the second run, the average CPU time predicted for each virtual message differs on each node. Table 1 shows the relevant experiment parameters at each node.

Table 1. Relevant Experiment Parameters for Each Node

Experiment Parameter	Fixed CPU Time Model		Adaptive CPU Time Model	
	Fastest Intermediate Node	Slowest Intermediate Node	Fastest Intermediate Node	Slowest Intermediate Node
Avg. CPU Time (ms and clock cycles)	7 (2,340,750)	7 (833,980)	3 (900,000)	12.5 (1,500,000)
Error Tolerance (+/-10%) (clock cycles)	234,075	83,398	90,000	150,000
Audio Stream Size (bytes)	91,105	91,105	91,105	91,105
Avg. Measurement Interval (seconds)	10.8	13.9	9.3	14

We assigned 7 ms per packet for the fixed CPU time model. This figure was obtained by measuring the active audio application executing on the source node. Note that 7 ms equates to a different number of clock cycles on each node, depending on processor speed. To obtain variable predictions for the adaptive CPU time model, we simulate that each virtual message consumes CPU time chosen from a statistical distribution, as determined

by analyzing an execution trace generated when the active audio application runs on the source node. Each time an application returns from a system call, the execution trace records an event. Each event identifies the previous and current system call, and includes the amount of CPU time used in the current call, as well as the CPU time used in the execution environment between the two calls. Elsewhere, we explain how this execution trace can be used to model CPU-time use as a function of application behavior, specifically transitions between system calls and the execution environment [3].

To adapt the variable time model to account for node differences, we calibrate [9] the ability of each node to execute system calls and an active-application benchmark within Magician. Further, we select the source node as a reference, and distribute its calibration results to all nodes. Since the variable-time CPU model is a function of the interactions between system calls and execution environment, a straightforward algorithm can transform the CPU model from local to reference form, based on the relative calibration performance of the local and reference nodes. After sending this application model to another node, the transformation can be simply inverted. As a result, the CPU model generated on the source node can be transformed into terms understood on any network node. Using this approach, the adaptive CPU time model predicts that each active audio packet will take 3 ms on the fastest intermediate node and 12.5 ms on the slowest intermediate node. Our hypothesis: because an adaptive model more accurately represents CPU use in an application, as compared against a fixed-time model, AVNMP should require fewer tolerance rollbacks; thus, the prediction-overlay network should provide better look-ahead into virtual time.

For both experiment runs we fixed the error tolerance at 10%, which means that AVNMP initiates tolerance rollbacks whenever the measured CPU use (averaged over 1000 messages) differs from the predicted CPU use by more than 10%. This tolerance, computed relative to predicted CPU use, equates to a different number of clock cycles for each node and run. Using a wider error tolerance would likely mask any improvements from the adaptive CPU time model.

In conducting each run, the audio application emitted a stream of 91,105 bytes, and the intermediate nodes periodically measured the cumulative tolerance rollbacks and the virtual time. As shown in Table 1, the average measurement interval varied on each node due to the stochastic nature of thread scheduling in Java. Table 2 compares the results we obtained from our experiment runs.

Over the audio streaming period, we can compare AVNMP performance for the same nodes when using the fixed CPU-time model vs. the adaptive CPU-time

model. For both the fastest and slowest intermediate node, the adaptive CPU-time model induces fewer tolerance rollbacks. In the case of the fastest intermediate node, this permitted AVNMP to reach a greater maximum virtual time during the audio streaming period. In the case of the slowest intermediate node, AVNMP reached the same maximum virtual time, regardless of CPU model.

Table 2. Comparing AVNMP Performance with Fixed vs. Adaptive CPU-Time Models

Metric	Fixed CPU Time Model		Adaptive CPU Time Model	
	Fastest Intermediate Node	Slowest Intermediate Node	Fastest Intermediate Node	Slowest Intermediate Node
Maximum Virtual Time (seconds)	640	360	984	360
Tolerance Rollbacks	96	25	74	0

APPLICATIONS

This paper describes two complementary innovations: (1) the use of active-network technology to deploy a shadow, prediction-overlay network that can estimate the future state of an operational network and (2) the use of application-level CPU-time models that can adapt to account for varying capabilities among heterogeneous network nodes. While complementary, the fundamental innovations can be applied independently to address existing and future needs within distributed systems.

In the case of AVNMP, effective prediction of future network state can enable network managers to adapt network configurations in response to traffic overloads before the overloads actually occur. In addition, predicting resource demands at a node can enable the operating system to better schedule node resources and to provide better admission control decisions. Further, predicting resource use along network paths can help network management systems to estimate the quality of service available to distributed applications. In fact, one can envision the use of such predictive capabilities to respond to application queries seeking a network path that guarantees a specified quality of service.

In the case of adaptive CPU-time models, one can imagine improvements in the safety and efficiency of Internet applications, which increasingly use mobile code, such as applets, scripts, and dynamically linked libraries, to deliver new software to millions of users. Without understanding the CPU time required by dynamically downloaded software, computer operating systems cannot effectively manage system resources or control the execution of mobile code. Unfortunately, since mobile code can be downloaded and executed on a wide variety of computer systems with a vast range of

configurations and capabilities, software developers cannot specify CPU requirements a priori.

FUTURE RESEARCH

While our adaptive CPU-time models appear promising, more research remains before the models can be practically applied. Three issues in particular must be resolved. First, our existing models assume that all application behavior can be measured prior to injecting a model into network nodes. Unfortunately, application behaviors often reflect conditions that cannot be known before a program reaches a node. For this reason, our application model must be enhanced to account for such node-dependent conditions. Second, our models consist of fine-grained histograms, which must be exercised with Monte Carlo simulations in order to predict CPU use. As a result, specific application models can be large and can require substantial computation to produce predictions. To some degree the space-time properties of our model can be modulated; however, the prediction error also varies accordingly. The third issue to be resolved involves error characterization. Before taking decisions based on predictions from CPU-time models, an operating system must consider the possible range of prediction error. We have yet to characterize the error properties of our models. The ability of AVNMP to maintain predictions within a specified error bounds offsets this weakness in the application discussed in this paper.

With regard to AVNMP, we demonstrated the ability to make predictions of message load and CPU usage in a rather small network. We have yet to investigate how shadow simulations might be scaled to larger networks with thousands of MIB variables at each node. Our current system emulates real applications running in a logically isolated prediction-overlay network, which shares physical resources with the operational network. This approach will at minimum double the physical resources required from the operational network. We might be able to discover more efficient techniques to simulate future state. Such efficiency improvements, which are being explored in light of recent advances in the application of Kolmogorov complexity theory [10], could prove crucial when we attempt to simultaneously predict alternative future network states. If we can achieve this goal, then AVNMP might be used to estimate multiple future states in a network, perhaps even assigning a probability to each state. Given such capability, a network manager could simultaneously explore multiple “what-if” scenarios and could initiate network reconfigurations based on the most likely or most critical expected outcomes.

CONCLUSIONS

In this paper, we described AVNMP, a system that uses active-network technology to deploy and manage a distributed simulation running in a virtual, prediction-overlay network. AVNMP can be used to predict the future state of network nodes and to maintain those predictions within specified error bounds. Specifically, we showed that AVNMP could predict both message load and CPU usage for a streaming-audio application running in an active network. Further, our experiment results suggest that in a heterogeneous active network the predictive performance of AVNMP, along with associated overhead, varies depending on the nature of the simulation models injected into the prediction-overlay network. When we injected an adaptive CPU-time usage model for the active audio application, the number of tolerance rollbacks decreased and predictive performance of AVNMP either improved or equaled that obtained using fixed CPU-time models.

REFERENCES

- [1] S. F. Bush and A. B. Kulkarni, Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework. ISBN 0-306-46560-4. Kluwer Academic / Plenum Publishers, (in press).
- [2] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active networks research," *IEEE Communications Magazine*, 35(1) 1, pp. 80-86, 1997.
- [3] V. Galtier, K. L. Mills, Y. Carlinet, S. D. Leigh, and A. Ruhkin, "Expressing Meaningful Processing Requirements among Heterogeneous Nodes in an Active Network," *Proceedings Second International Workshop on Software and Performance*, pp. 20-28, September 2000.
- [4] K. L. Calvert (ed), Architectural Framework for Active Networks, Version 1.0, Draft, July 27, 1999. <http://www.dcs.uky.edu/~calvert/arch-latest.ps>
- [5] L. Peterson (ed.), NodeOS Interface Specification, January 24, 2000. <http://www.dcs.uky.edu/~calvert/nodeos-latest.ps>
- [6] S. Dawson, M. Molteni, L. Ricculli, and S. Tsui, User Guide to ANETD 1.6.3, Sept. 28, 2000. <http://www.csl.sri.com/activate/anetd/doc/html/>
- [7] A.B. Kulkarni, G. J. Minden, R. Hill, Y. Wijata, S. Sheth, H. Pindi, F. Wahhab, A. Gopinath, and A. Nagarajan, "Implementation of a Prototype Active Network", *Proceedings OPENARCH '98*, 1998.
- [8] M. T. Rose, The Simple Book: An Introduction to the Management of TCP/IP Based Internets, Prentice-Hall, 1991.
- [9] Y. Carlinet, V. Galtier, K. Mills, S. Leigh, and A. Ruhkin, "Calibrating an Active Network Node", *Proceedings of the 2nd International Workshop on Active Middleware Services*, pp. 115-125, August 2000.
- [10] A. B. Kulkarni and S. F. Bush, "Active Network Management, Kolmogorov Complexity, and Streptichrons," GE-CRD Technical Report, 2000CRD17.

The work discussed in this paper was funded in part by DARPA, under the auspices of the Active Networks program. Our thanks go to Doug Maughan, the Active Networks program manager.