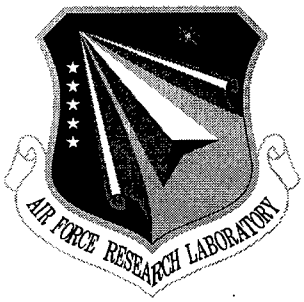
AFRL-IF-RS-TR-2001-70
Final Technical Report
May 2001



ACTIVE VIRTUAL NETWORK MANAGEMENT PREDICTION

General Electric Corporate Research and Development

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. G420

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-70 has been reviewed and is approved for publication.

APPROVED:

SCOTT S. SHYNE
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ACTIVE VIRTUAL NETWORK MANAGEMENT PREDICTION

Stephen Bush

Contractor: General Electric Corporate Research and Development
Contract Number: F30602-98-C-0230
Effective Date of Contract: 05 June 1998
Contract Expiration Date: 31 November 2000
Short Title of Work: Active Virtual Network Management Prediction

Period of Work Covered: Jun 98 - Nov 00

Principal Investigator: Stephen Bush
Phone: (518) 387-6827
AFRL Project Engineer: Scott S. Shyne
Phone: (315) 330-4819

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Scott S. Shyne, AFRL/IFGA, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MAY 2001	Final Jun 98 - Nov 00

4. TITLE AND SUBTITLE
ACTIVE VIRTUAL NETWORK MANAGEMENT PREDICTION

5. FUNDING NUMBERS
C - F30602-98-C-0230
PE - 62301E
PR - G420
TA - 00
WU - 01

6. AUTHOR(S)
Stephen Bush

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
General Electric Corporate Research Development
Bldg KW - C512
One Research Circle
Niskayuna NY 12309

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2001-70

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Scott S. Shyne/IFGA/(315) 330-4819

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In active networking, applications cannot only determine the protocol functions as necessary at the endpoints of a communication path, but can also inject new protocols into the network for the network nodes to execute on their behalf. The nodes of the network, called active nodes, are programmable entities. Application code executes on these nodes to implement new protocols and services. This project has designed, prototyped, and experimentally validated a prediction mechanism that uses the new capabilities of active networks to add prediction to network management, know as Active Virtual Network Management Prediction (AVNMP).

14. SUBJECT TERMS
Active Network, Network Management, Prediction

15. NUMBER OF PAGES
214

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

CONTENTS

TABLE OF FIGURES

LISTING OF TABLES

1

INTRODUCTION

Active networking is a novel approach to network architecture in which network nodes – the switches, routers, hubs, bridges, gateways etc., – perform customized computation on the packets flowing through them. The network is called an "active network" because new computations are injected into the nodes dynamically, thereby altering the behavior of the network. Packets in an active network can carry fragments of program code in addition to data. Customized computation is embedded in the packet's code, which is executed on the network nodes. By making the computation application-specific, applications utilizing the network can customize network behavior to suit their requirements and needs.

The active network model provides a user-driven customization of the infrastructure, allowing new services to be deployed at a faster pace than can be sustained by vendor-driven consensus or through standardization. The essential feature of active networks is the programmability of its infrastructure. New capabilities and services can be added to the networking infrastructure on demand. This creates a versatile network that can easily adapt to future needs of applications. The ability to program new services into the network will lead to a user-driven innovation process in which the availability of the new services will be dependent on their acceptance in the marketplace. In short, active networking enables the rapid deployment of novel and innovative services and protocols into the network. For example, a video conferencing application can inject a custom packet-filtering algorithm into the network that, in times of congestion, filters video packets and allows only audio packets to reach the receivers. Under severe congestion conditions, the algorithm compresses audio packets to reduce network load and alleviate congestion. This enables the application to handle performance degradation due to network problems gracefully and in an application-specific manner.

In active networking, applications cannot only determine the protocol functions as necessary at the endpoints of a communication path, but can also inject new protocols into the network for the network nodes to execute on their behalf. The nodes of the network, called active nodes, are programmable entities. Application code executes on these nodes to implement new protocols and services. This project has designed, prototyped, and experimentally validated a prediction mechanism that uses the new capabilities of active networks to add prediction to network management, known as Active Virtual Network Management Prediction (AVNMP).

1.1 OUTLINE OF THE REPORT

Chapter 2 discusses the motivation for a reference model that addresses limitations of the current network management framework and leverages the powerful features of active networking to develop an integrated framework. The later part of Chapter 2 prepares the reader for AVNMP, which is the focus of the remainder of the report.

The report provides a close-up view of a novel application enabled by active network technology. It describes the life-cycle of an active networking protocol from conception to implementation. The application chosen implements the predictive aspect of the active management framework discussed in Chapter 2 and is called Active Virtual Network Management Prediction. In current network management, managed entities are either polled to determine their health or they send unsolicited messages indicating failed health. By the time such messages are generated, much less received, by a centralized system manager, the network has already failed. Active Virtual Network Management Prediction has resulted from research in developing proactive system management, in other words, to solve a potential problem before it impacts the system. Active Virtual Network Management Prediction accomplishes this by modeling network devices within the network itself and running that model ahead of real time. Active Virtual Network Management Prediction is also self-correcting. Thus, managed devices can be queried for events which are likely to happen in the future; problems are detected ahead of time. The chapters of the report are organized as follows:

- Chapter 2: Management Reference Model
- Chapter 3: AVNMP Architecture
- Chapter 4: Detailed Example of AVNMP Operation
- Chapter 5: Algorithmic Description of AVNMP
- Chapters 6-7: Performance Measurements and Analysis of AVNMP
- Chapter 8: Experimental Validation of AVNMP
- Chapter 9: Summary and Concluding Remarks
- Chapter 10: Glossary
- Chapter 11: References

Chapter 3 describes the architecture of the AVNMP framework and explains how various features of an active network can be leveraged to create a novel management strategy. Chapter 3 includes examples of Driving Processes for specific applications, while Chapter 4 provides a detailed operational example of AVNMP. Chapter 5 discusses the background and origin of the algorithm used by AVNMP and includes an Appendix on some of the implementation details. Chapter 6 quantifies the performance of AVNMP, deriving equations for AVNMP performance and overhead. Chapter 7 considers the challenges faced by any system attempting to predict its own behavior and some of the unique characteristics of AVNMP in meeting those challenges. Chapter 8 presents an experimental validation of AVNMP.

This project has challenged itself to consider the benefits of Active Networking and to apply those benefits towards the management of Active Networks. The inherently distributed nature of communication networks and the computational power unleashed by the Active Networking paradigm have been used to mutual benefit in the development of the Active Virtual Network Management Prediction mechanism. Both load and CPU prediction capability have been explored using AVNMP. Active Networks benefit from AVNMP by continuously providing information about potential problems before they occur. AVNMP benefits from Active Networks in many ways. The first and most practical is the ease of development and deployment of this novel protocol. This could not have been accomplished so quickly or easily given today's closed, proprietary network device processing. Another benefit is the fact that network packets now have the unprecedented ability to control their own processing. Great advantage is taken of this new capability in AVNMP. Virtual messages, varying widely in content and processing, can adjust their predicted values as they travel through the network. Finally, Active Networks add a level of robustness that cannot be found in today's networks. This robustness is due to the ability of the

AVNMP system components, which are themselves active packets, to easily migrate from one node to another in the event of failure -- or the prediction of failure provided by AVNMP!

2

MANAGEMENT REFERENCE MODEL

This chapter discusses the goals and requirements for an active network management framework. The active network management framework refers to the minimum model that describes components and interactions necessary to support management within an active network. This is motivated by comparing management in current networks with the possibilities enabled to support management within active networks. Towards this objective, an overview of the current network management model is discussed as a prelude to discussing the active network management model.

In the current communications model, managed devices are viewed abstractly as protocol layer two and protocol layer three network devices that forward data from source towards destination end-systems. The actions taken by these devices are predefined and fixed for each protocol layer and packet type as shown in Figure 2.1. The figure shows non-active data packets transporting management requests to the managed device and a possible management response is shown leaving the managed device.

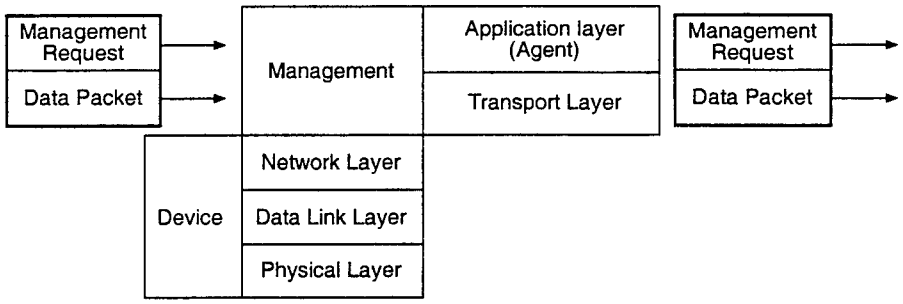


Figure 2.1. Current Management Model

The current management model, as illustrated and implemented by such protocols as the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP), requires that network devices have a management agent that responds to management requests. Devices must be addressable and respond directly to remote management commands. The model assumes that network nodes are instrumented with the ability to respond to requests for pre-configured data points of management information. Management information needs to be gathered for behavior of protocols in the higher layers of the stack, e.g., application data. This requires instrumenting more than just the bottom two or three standard protocol layers.

Therefore, management has never been a natural fit to the current non-active communications model for intermediate network devices. It was initially considered difficult and uncommon for any type of standards-based management to exist because of the large number of non-interoperable proprietary attempts to solve the problem. Thus, the goal had been to implement a standard management framework that was robust and would be ubiquitously deployed across the Internet. The Simple Network Management Protocol had filled this role to some extent; however, active networks allow for a better solution.

In the current management model, shown in Figure 2.2, high-level queries are entered into, or generated from, a central management station that breaks the query down into low-level requests for data from managed entities. The current management model requires that all data values that would be needed for management must be predetermined and pre-defined in an information store called a Management Information Base (MIB). Each data point has a predetermined type, size, and access level and is called a Management Information Base Object. The result is that the Management Information Base contents, that is, the collection of Objects, must be painstakingly designed and agreed upon far ahead of time before they can be widely used. Even after accomplishing this, elements of the Management Information Base have static, inflexible types. This is antithetical to the objective of the active network framework, which seeks to minimize committee-based agreements. In an active network framework, elements of the Management Information Base have the potential to be dynamically defined and used by applications. The static data type of a Management Information Base may be reasonable for network hardware, but becomes less appropriate as higher layers of the protocol stack and applications are instrumented.

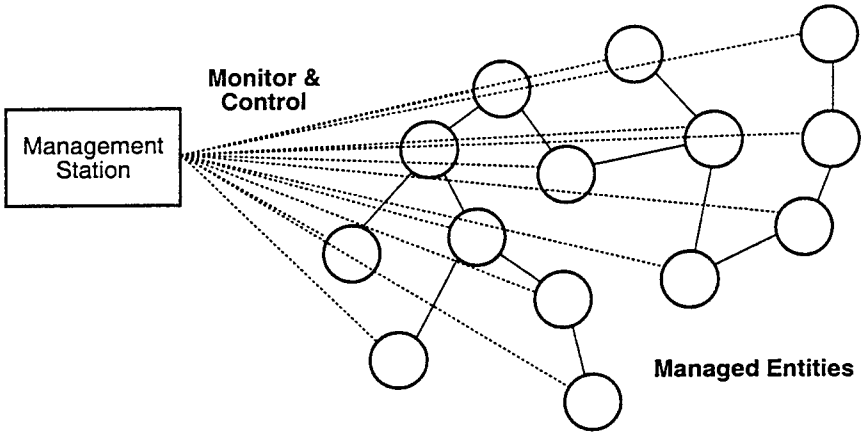


Figure 2.2. Current Centralized Management Model.

The current management model leads to a poor network control architecture. Large delays are incurred as agents send raw data to a central management station that takes time to refine and process the primitive data and perhaps respond with a Simple Network Management Protocol *Set Request* control action. However, the current management model has been primarily concerned with monitoring rather than control, in part because control has been hampered by the long transfer delay times to the centralized management station. While management *Set Requests* can

be used for control purposes, few Management Information Bases today utilize the set commands for any type of real-time control.

As the Simple Network Management Protocol in current non-active networks has made steps toward providing reliable, integrated network management, the demand for more systems integrated management and control increases. Perhaps the demand is fed by the success of the Simple Network Management Protocol and by the explosion in the size of communication networks and number of applications utilizing them. Network administrators are pushing to extend network management ever higher towards and into the application layer. Integration is a primary driver. Clearly, applications, end-systems and the network all need to be managed in an integrated fashion.

The paradigm of instrumenting network elements is not the best solution for managing higher layer protocols and applications, especially in an active network wherein applications have a direct interaction with network elements. One reason is increased complexity. Network hardware devices and low-level network protocol layers behave in precise, well defined ways. On the other hand, active applications can interact with network protocols and other applications in myriad ways. This complexity in interaction requires a proportional increase in the number of management data points. Instrumenting every network device and end-system to support management of every application is not a scalable or feasible option. This could lead to other problems, such as redundant management data points, because two applications interacting with each other utilize the same management agent capability.

Another characteristic of the current management paradigm is that intermediate nodes are not designed to support management algorithms on the nodes themselves. However, fully integrated system management has always been the goal. Values from data points from all managed elements, including applications, are simply transported to a centralized management station where all refinement and processing takes place. Proxy agents are sometimes used to manage devices that have non-standard or non-existing management interfaces. Proxy agents serve as intermediary translators between the management standard and the operations of which the device is capable for management. The only other place that processing could be done within the network is in the managed object's agent. However, the prevailing philosophy has been that the managed object should be fully devoted to its primary task of forwarding data, not management, and therefore the agent is designed and implemented to be as simple and efficient a process as possible. The agent simply responds to requests for management data point values and generates unsolicited trap messages, hopefully, infrequently and only under extreme conditions.

Active networking affords an opportunity to take a new look at the network management problem and communications in general from a different perspective. It is a perspective that flips the traditional networking paradigm on its head. By allowing general-purpose computation on traditional intermediate network systems, it is no longer required that application processing, including network management processing, be restricted to end-systems. Optimum management efficiency can be achieved because processing can be allocated to intermediate network resources. This allows for a larger set of feasible solutions to the allocation of processing resources. For example, the old philosophy of keeping communications as simple possible has resulted in a plethora of highly specialized protocols illustrated by the large number of Internet Engineering Task Force Requests for Comments that are extant. In terms of network management, the old philosophy has caused enormous inefficiency by requiring large amounts of data to be transported to centralized management stations, even in instances when the data turns out to be of limited or no value. The active network model provides a communications model

that is a better fit to the management model. In the active network reference model, intermediate-system active devices have the ability to accept and process any packet as a natural part of its packet processing, including network management packets. In fact, in the new management paradigm, management can be integrated into the processing framework itself; that is, packets are the application and manage themselves.

In the new management model shown in Figure 2.3, it is possible for a high-level management query in executable form to be sent directly to the managed active application. Because the managed application is active, it is implemented via active packets. The management query active packet interacts with the active application's packets in order to determine the result of the query. Given active network protocol composition, methods dynamically bound to an application no longer require a Management Information Base with static data point definitions to return predefined values, but instead, access local data points, compute a result from the local data and return only the final result, or some set of data culled from the local data that can lead to the final result, which may be computed in another part of the network. Many management systems today operate by polling a value and setting a threshold that trips an alarm when the threshold is crossed. Frequent queries result in wasted bandwidth if the threshold is rarely reached. The only information required in such cases is the alarm. In the active network management environment, the threshold crossing detection can be dynamically bound as a method in the managed active application.

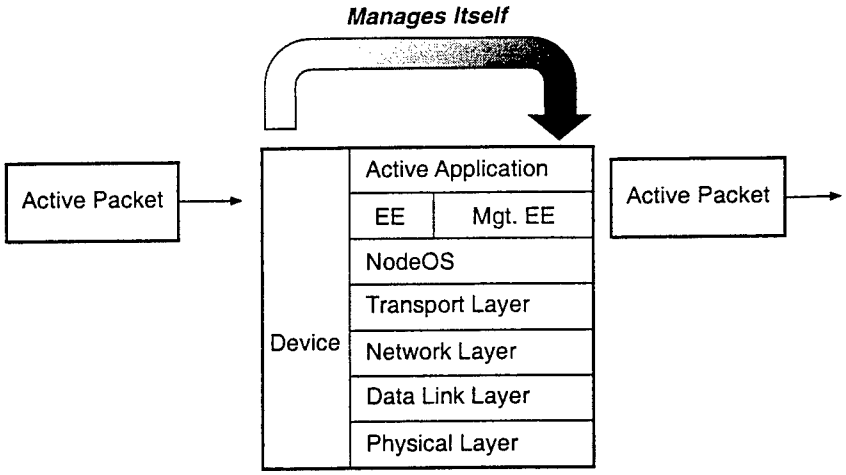


Figure 2.3. Active Management Model.

The old management philosophy requires that a *Set Request* be used for control purposes. This results in long delays when the controller is a centralized management station as compared to the active network model that enables local computation and control. Delays are clearly dangerous in a control system. Thus, using the Simple Network Management Protocol *Set Request* is also highly inefficient for dynamic control purposes. Active networks allow more distributed control for management purposes than in today's management model and an opportunity for a new management paradigm. The control algorithm is bound directly in the managed active application, thus reducing the delay incurred by dealing with a centralized management station. The active nature of the network also allows a framework in which efficient

prediction of network behavior is possible. The Active Virtual Network Management Algorithm described in the next part of this report takes advantage of the active network to provide a model-based predictive management control framework. This requires a form of introspection that is possible in the new management model. Introspection is enabled because applications can control and manage themselves to a greater degree with active networks than ever before in the old management philosophy. The following example shows that data in an active network management model example has the ability to be queried by standards based network management protocols. A small agent is encapsulated with the active data as shown in Figure 2.5. When the data is queried, the agent responds with the values maintained by that specific agent's Management Information Base (MIB). The converse of this is shown in Figure 2.4, which illustrates active data containing a management client capable of querying management agents. This concept has been prototyped in Java in the active network testbed at General Electric Corporate Research and Development.

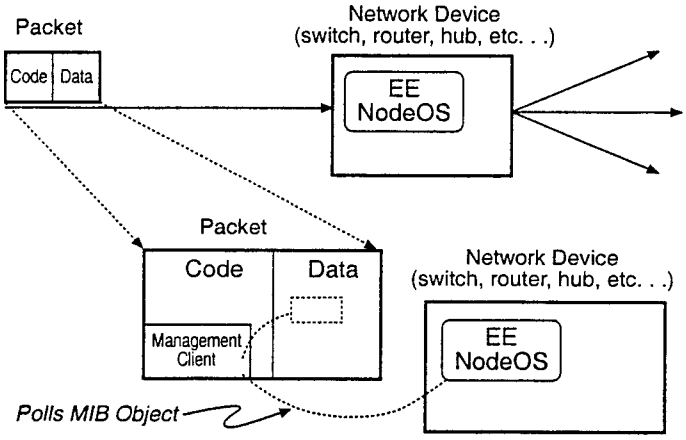


Figure 2.4. An Overview of the Traveling SNMP Client.

It has been recognized that, even in the new active management model, systems administrators require an integrated view of the entire managed system. However, note that integrated does not necessarily imply centralized. Also, note that the functionality of an integrated management view has changed dramatically from the old management model. The old model management view consisted of displaying values from static data types that are predefined. This is in contrast to the new that consists of controlling the algorithms (methods) that are bound to managed active entities and displaying results from those algorithms. Thus, the new management model deals with methods rather than data types.

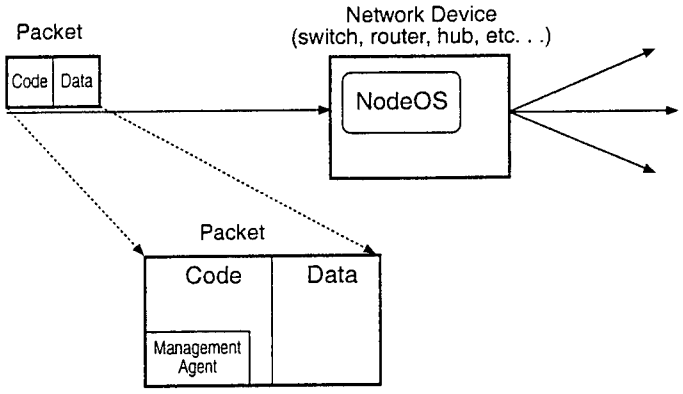


Figure 2.5. Queryable Data.

In the view of the authors, the new active network management goals should consist of:

- Automated self-management and control of applications.
- Ability to dynamically add/remove management features across all active applications.
- A richer integrated management view of the network and applications than in the old network management model.
- Decentralized and distributed management within the network for increased efficiency.
- Extreme reliability in the face of network failure.
- Support for integrated management of legacy applications

A few words of explanation are in order to justify why these goals are worthy of pursuit. Clearly, network management benefits from being as automated as possible. The words "self-management" are used because it is assumed that the system is able to determine best how to manage and control itself. An integrated view is the most concise and logical for human consumption and allows quick identification of correlated events. This assumes that a security policy mechanism is in place for network managers to gain access only to their own views of the system. The goal is that active networks will allow a richer semantic view of the integrated system. We want the system to be decentralized and distributed since that provides the most efficient use of resources and better response times. In addition, it can allow for graceful degradation of performance as resources fail. Finally, management is most critical when the system is failing. Thus the management system must be as robust and reliable as possible; that is, it should be the last service to fail. A framework within active networks that supports these goals is useful. However, care must be taken in developing a framework that does not preclude the development of general-purpose innovative management techniques enabled by active networking. The Active Virtual Network Management Prediction Algorithm is a step towards an active network management framework by enabling model-based predictive control, as discussed in the next section.

2.1 TOWARDS AN ACTIVE NETWORK MANAGEMENT FRAMEWORK

The previous section discussed the goals of a new framework for network management. Consider what is required from the framework in order to achieve these goals. Automated self-management and control of applications require application developers to provide monitoring and access into their applications. While an application may be self-managing and autonomous, it cannot be a completely closed system. The application needs information about other applications and the network that it resides upon. The application may need to negotiate with other applications for resources. The management interface between applications could be accomplished through definitions as is the case in today's non-active networks; however, more is possible with an active network. For example, the Management Information Base could itself become an active entity. Model-based predictive control is a particular mechanism enabled by the Active Virtual Network Management Prediction Algorithm described in detail in the next part of this report. A fully autonomous, self-managed application requires:

- Inter-application semantic specification
- Inner-loop control mechanisms
- Negotiation capability
- Managed data semantic correlation
- Security policy

The negotiation capability and inter-application semantic specification are of primary interest here because they require some form of semantic knowledge and goal seeking capability. While dealing with semantic knowledge and goal achieving research are major efforts in their own right, the new architecture should facilitate and encourage their development. The integrated management view requires that all the management information from each managed entity be brought together and presented to a single user. This means that a policy must be in place to control access to information and the data must have the ability to correlate itself with other data for an integrated view. This requires a security policy and managed data semantic correlation.

There are several spheres of management in the active network management model: the Execution Environment (EE), the Active Application (AA), and the Network management algorithm, where a network Management Application (MA) is a new management feature to be added to all Active Applications (AA). The ability to dynamically bind methods into active applications is an assumed feature in active networks. The actual mechanisms for inserting methods into an existing and executing application are discussed in (Zegura, 1998). A brief summary of example methods is presented in Table 2.1. Self-organizing management code, knowing when, where, and how to insert itself into the managed active application, is a goal that is partially met by the Active Virtual Network Management Prediction Algorithm. The Active Virtual Network Management Prediction framework discussed in detail in the next part of this report demonstrates the fundamental requirements of the new active network management framework, namely:

- Access to managed device monitoring and control.
- Insertion of monitoring elements into arbitrary locations of active applications.
- Injection of executable models onto managed nodes and/or into managed active applications.

- Injection/interception of management packets within the network.

Table 2.1. Active Network Composition Methods.

Composition Type	Reference
Functional	Hicks et al., 1999
Dataflow	Da Silva et al., 1998
Slots	Samrat Bhattercharjee, Kennth L. Calvert and Ellen W. Zegura, 1998
Signaling Extensions	Braden et al., 2000

2.2 PREDICTION IN NETWORK MANAGEMENT

Network management is evolving from a static model of simply monitoring the state of the network to a more dynamic, feature-rich model that contains analysis, device and line utilization, and fault-finding capabilities. The management marketplace is rich in software to help monitor and analyze performance. However, a severe limitation of current state-of-the-art network management techniques is that they are inherently reactive. They attempt to isolate the problem and determine solutions after the problem has already occurred. An example of this situation is the denial of service attack on Internet portal Yahoo!'s servers on February 7, 2000. Network managers were only able to detect the attack and respond to it long after it crippled their servers. To prevent such occurrences, network management strategies have to be geared towards assessing and predicting potential problems based on current state. Another limitation of current management software is "effect-chasing." Effect chasing occurs when a problem causes a multitude of effects that management software misdiagnoses as causes themselves. Attempts to solve the causes instead of the problem result in wasted effort. Recent advances in network management tools have made use of artificial intelligence techniques for drilling down to the root cause of problems. Artificial Intelligence techniques sift through current data and use event correlation after the problem occurs to isolate the problem. While this provides a reasonable speedup in problem analysis, finding a solution can still be time-consuming because these tools require enough data to form their conclusions. Therefore, proactive management is a necessary ingredient for managing future networks. Part of the proactive capability is provided by analyzing current performance and predicting future performance based on likely future events and the network's reaction to those events. This can be a highly dynamic, computationally intensive operation. This has prevented management software from incorporating prediction capabilities. But distributed simulation techniques take advantage of parallel processing of information. If the management software can be distributed, it is possible to perform computation in parallel and aggregate the results to minimize computation overhead at each of the network nodes. Secondly, the usefulness of optimistic techniques has been well documented for improving the efficiency of simulations. In optimistic logical process synchronization techniques, also known as Time Warp (Bush et al., 1999; Bush, 1999), causality can be relaxed in order to trade model fidelity for speed. If the system that is being simulated can be queried in real time, prediction accuracy can be verified and measures taken to keep the simulation in line with actual performance. Networks present a highly dynamic environment in which new

behaviors can be introduced as new applications inject new forms of data. The network management software would have to be highly adaptive to model these behaviors and analyze their effects.

Active networking provides an answer to this problem. Active networking offers a technology wherein applications can inject new protocols into the network for the network nodes to execute on behalf of the application. A network is defined to be an active network if it allows applications to inject customized programs into the network to modify the behavior of the network nodes. The nodes of the network, called active nodes, are programmable entities. Application code is embedded inside a special packet called a SmartPacket. When the SmartPacket reaches the appropriate active node, the code is extracted and executed at the node to implement new services. Active networking thus enables modification of a running simulation by injecting packets modeling the behavior of a new application into the network. This research presents a new proactive network management framework by combining the three key enabling technologies: (1) distributed simulation, (2) optimistic synchronization, and (3) active networking. The next section provides an introduction to the predictive framework and describes its various components.

2.2.1 Temporal Overlay

The approach taken by AVNMP is to inject an optimistic parallel distributed simulation of the network into the active network. This can be viewed as a virtual overlay network running temporally ahead of the actual network. A virtual network, representing the actual network, can be viewed as overlaying the actual network. A motivating factor for this approach is apparent when AVNMP is viewed as a model-based predictive control technique where the model resides inside the system to be controlled. The environment is an inherently parallel one; using a technique that takes maximum advantage of parallelism enhances the predictive capability. A well-known problem with parallel simulation is the blocking problem, in which processors are each driven by messages whose queues are attached to the processor. The message time-stamps are within the message. The message value is irrelevant. It is possible that one processor could execute a message with a given time stamp, then it could receive the next message with an earlier time-stamp. This is a violation of causality and could lead to an inaccurate result. There have been many proposed solutions to this problem. However, many solutions depend on the processor that is likely to receive messages out of order, waiting until the messages are guaranteed to arrive in the proper order. This increases delay and thus reduces the overall system performance. The AVNMP Algorithm makes use of a well-known optimistic approach that allows all processors to continue processing without delay, but with the possibility that a processor may have to rollback to a previous state. In addition the AVNMP Algorithm dynamically keeps the predictions within a given tolerance of actual values. Thus the model-based predictive system gains speedup due to parallelism while maintaining prediction accuracy.

The AVNMP system is comprised of Driving Processes, Logical Processes, and streptichrons, which are active virtual messages. The Logical Processes and Driving Processes execute within an Active Network Execution Environment (EE) on each active network node. The Logical Process manages the execution of the virtual overlay on a single node and is primarily responsible for handling rollback. Rollback can be induced by out-of-order Virtual Message arrivals and by prediction inaccuracy. A tolerance is set on the maximum allowable deviation between the predicted values and the actual values. If this tolerance is exceeded, a rollback to wallclock time occurs. The Logical Processes' notions of time only increment as

virtual messages are executed. A sliding lookahead window is maintained so that a specified distance bounds the Logical Processes' virtual time progression into the future. The Driving Process monitors the input to that portion of the network enhanced by AVNMP and generates the Virtual Messages that drive the AVNMP Logical Processes forward in time. The driving process monitors the actual application via a general management frame developed within the active network environment. The driving process samples the values to be predicted and generates a prediction. The actual mechanism used for predicting output from any application is application dependent and de-coupled from the system. However, a simple curve-fitting algorithm based upon past history has worked adequately well.

2.2.2 Enhanced Message Capabilities

A Streptichron (from Classical Greek meaning to "bend time") is an active packet facilitating prediction that implements any of the active mechanisms described in this section. The streptichron can use this capability to refine its prediction as it travels through the network. In the initial AVNMP architecture, there was a one-to-one correspondence between virtual messages and real messages. While this correspondence works well for adding prediction to protocols using a relatively small portion of the total bandwidth, it is clearly beneficial to reduce message load, especially when attempting to add prediction of the bandwidth itself. There are more compact forms of representing future behavior within an active packet besides a virtual message. For relatively simple and easily modeled systems, only the model parameters need be sent and used as input to the logical process on the appropriate intermediate device. Note that this assumes that the intermediate network device's Logical Process is simulating the device operation and contains the appropriate model. However, because the payload of a virtual message is exactly the same as a real message, it can be passed to the actual device, and the result from the actual device is intercepted and cached. In this case, the Logical Process is a thin layer of code between the actual device and virtual messages primarily handling rollback. An entire executable model can be included within an active packet generated by the DP and executed by the Logical Process. When the active packet reaches the target device, the model provides virtual input messages to the Logical Process, and the payload of the virtual message is passed to the actual device as previously described. Autoanaplasis ("self adjust") is the self-adjusting characteristic of streptichrons. For example, in load prediction, streptichrons use the transit time to check prior predictions. General-purpose code contained within the packet is executed on intermediate nodes as the packet is forwarded to its destination. For example, a packet containing a prediction of traffic load may notice changes in traffic that influence the value it carries as the packet travels towards its destination. The active packet updates the prediction accordingly.

Time is critical in the architecture of the AVNMP Algorithm system; thus, most classes are derived from class Date. Class AvnmpTime handles relative time operations. Class Gvt uses the active GvtPackets class to calculate global virtual time. Class AvnmpLP handles the bulk of the processing including rollback. Class Driver generates and injects real and virtual messages into the system. The PP class either simulates or accesses an actual device on behalf of the Logical Process. The PP class may not need to simulate the device because the payload of a virtual message is exactly the same as a real message; thus, the payload of the virtual message can be passed to the actual device and the result from the actual device is intercepted and cached. In this case, the Logical Process is a thin layer of code between the actual device accessed by the PP class. The GvtPacket class implements the Global Virtual Time packet that is exchanged by all

logical and driving processes to determine global virtual time. The AvnmpPacket class is derived from KU_SmartPacket_V2 and is the class from which GvtPacket and Streptichron classes are derived. Magician is a toolkit that provides a framework for creating SmartPackets as well as an environment for executing the SmartPackets. Magician is implemented in Java version 1.1. Version 1.1 was primarily chosen because it was the first version to support serialization. Serialization preserves the state of an object so that it can be transported or saved, and re-created at a later time. Therefore, in Magician, the executing entity is a Java object whose state is preserved as it traverses the active network. Magician adheres to the Active Network Encapsulation Protocol (ANEP) (Alexander et al., 1997) format when sending the Java class definitions and the Java object itself over the network. The details about the architecture of an active node in Magician and the exact format of a Magician SmartPacket are described in (Kulkarni et al., 1998). AVNMP runs as an active application (AA) inside the Magician environment. AVNMP queries Magician's state to perform resource monitoring and for load computation. Communication between different packets belonging to AVNMP and with other active applications like an SNMP-based real-time plotter takes place through smallstate, named caches that applications can create for storage, from which information can be retrieved. The remainder of this report discusses AVNMP and some of the surprising temporal complexities it introduces in greater levels of detail. While active networking provides the benefits previously discussed, it also adds to the complexity of the network. The additional complexity of active networks makes network and systems management a challenging and interesting problem because it is a problem in which distributed computing can now more easily be brought to bear because distributed computing algorithms can be more easily implemented and more quickly deployed in an active network. It will no longer suffice for network analysts to focus solely on traditional network performance characteristics such as load,delay, and throughput. Because active networking enables application computation to be performed within the network, the network performance must be optimized in tandem with applications. Delays through the network may be slightly longer because of computation, yet more work is done on behalf of the application. Thus metrics that include a closer association with applications are required. The next part of this report explains the design and development of active networks that are capable of predicting their own behavior and serve as a predictive active network management framework.

2.3 PREDICTIVE SYSTEMS DISCUSSION

Imagine a time in the future where someone digs up a crusty old technical document. This document makes its way into the hands of a few bright minds of the time who instantly recognize it to be the foundation work of the late 20th century on a fledgling technology called "active networking" that evolved into the current communications infrastructure. These bright minds parse the document and attempt to figure out the reasoning behind the decisions outlined in the manuscript. The names of the characters in the dialog are purposely reminiscent of ancient Greece, foreshadowing the issues of rollback and tangled hierarchies to be discussed in later chapters.

Glaucon: "It seems clear to me that to perform any type of prediction requires a projection forward in time at a rate faster than wallclock time. I suppose that a closed system, one that is totally self-contained, could be run forward in time very accurately. This is because it would have no interaction with elements running at wallclock time."

Thrasymachus: "I don't believe it. Even a completely closed system could exhibit chaotic behavior. And besides, even if a perfectly closed system existed, it would be of no use to anyone since we could not interact with it."

Socrates: "This sounds like an interesting topic. I am not as intelligent as either of you, so please help me follow where this discussion may lead. I believe, Glaucon, that you are searching for a simplified, ideal model in which to formulate predictive capability for an active network. Am I correct?"

Glaucon: "You are correct, Socrates."

Socrates: "We exist within the Universe and often attempt to predict events about ourselves within the Universe: weather, investments, political and military results -- consider the cleverly planned, but ill-fated attempts of Athens against Sparta.* We were part of that event, yet would have been hard pressed to have predicted its outcome. Is it better to be within the system or outside of the system for which you are attempting to compute a prediction?"

Thrasymachus: "Your question, Socrates, is a moot one. We can never truly be outside of a system and still interact with it. The mere act of measurement changes a system, however negligible. We can never know the truth. Even the supposed perfect abstraction that we use to model the world, Mathematics, cannot fully and completely describe itself, as Gödel has shown."

Glaucon: "Thrasymachus, do not be such a downer. We have come a long way in understanding the world around us. The scientific method of observation, hypothesis, and experimental validation continue to yield many new insights. Let us continue the quest for a predictive system, while realizing that perfection may not be possible in practice."

Socrates: "Well said, Glaucon. In fact, you have mentioned the scientific method. I think there is more to what you have said than you may realize. What is the fundamental activity in developing a hypothesis? Or, let me state it this way: How does one determine the best hypothesis if more than one appear equally valid in experimental validation?"

Glaucon: "One would prefer the simpler hypothesis. We seek to reduce complexity in our understanding of the world around us."

Socrates: "Excellent. How does one measure complexity?"

Thrasymachus: "Socrates, I believe I know where you heading with this line of reasoning...and it is pointless. Complexity is the size of the smallest algorithm or program that describes the information of which you wish to measure the complexity. However, this complexity is, in general, uncomputable. So again, you're leading us to a dead end as usual."

Glaucon: "Wait Thrasymachus, I wish to see where this would lead. What could this possibly have to do with Active Networks or predictive network management?"

Socrates: "What was the new feature that active networks had added to communication that never existed before?"

Glaucon: "Executable code within packets executed by intermediate nodes within the network."

---

* This is the Pelopennesian War (431-404 B.C.) in which the defeat of the formerly liberal and free-thinking Athens by Sparta led to Athen's defeatist attitude and subsequent trial and execution of Socrates.

Socrates: "Exactly. Active networks are much more amenable to algorithmic information. In other words, it becomes much easier to transmit algorithms than it ever had before active networks."

Thrasymachus: "Fine. I know where you are going here. You are going to say that we can now transmit executable models once, rather than passive data many times. But think of the overhead. What would you gain by transmitting a huge executable model of a system to a destination when it interacts only rarely with that destination?"

Glaucon: "I see your point Thrasymachus. We need to know when it is advantageous to transmit the model, and when to transmit only the passive data from that model. But how does all of this relate to predictive network management?"

Socrates: "In order to obtain predictive capability from an active network, we can inject a model of the network into the network itself. Sounds very Gödelian...if there is such a word."

Thrasymachus (sarcastic tone): "Very good. Now what about the effect that the model has upon the network? How can the model predict its own impact upon the network? Shall we inject a model of the model into the model? This is all nonsense. The system could never be perfectly accurate and the overhead would make it too slow."

Socrates: "Thrasymachus, is the complexity of a network node smaller than the length of the actual code on network node itself?"

Thrasymachus: "Unless the node and its code have been optimized to perfection, the complexity will be smaller. This is obvious."

Socrates: "Will the model injected into the network be more, or less complex than the node itself?"

Thrasymachus: "Less complex, Socrates. As we have already determined, the purpose of science is to find the least complex representation of a phenomenon. That is what a model represents."

Socrates: "Thrasymachus, will you agree that a communication network is by its very nature a highly distributed entity?"

Thrasymachus: "Clearly, the network is widely distributed."

Socrates: "Thus, an application that takes advantage of that large spatial area would benefit greatly, would it not?"

Thrasymachus: "Agreed."

Glaucon: "Are you suggesting, Socrates, that we use space to gain time in implementing our lower complexity models?"

Socrates: "Certainly that should allow the models injected into the network to project ahead of wallclock time."

Thrasymachus: "I see that you are attempting to trade off space, fidelity, and complexity in order to gain time, but this still sounds like a very tough problem and the devil will be in the details. Synchronization algorithms cannot gain the full processing power of all the processors in the distributed system. This is because messages must arrive in the proper order causing some parts of the system to slow down more than others waiting for messages to arrive in order."

Glaucon: "Optimistic distributed simulation algorithms do not slow down a priori. They assume messages arrive in the proper order and processing always continues full speed. If a message does arrive out of order at a processor, the processor must rollback to a previously known valid state, send out anti-messages to cancel the effects of now possibly invalid messages that it had sent, and continue processing from the rollback time incorporating the new message in its proper order."

Socrates: "If each processor executes at its own speed based upon its input messages, then each processor must have its own notion of time."

Glaucon: "That is correct. Each processor has its own Local Virtual Time."

Thrasymachus: "Let me understand this more concretely by a tangible analogy. Let us suppose that messages are ideas, processors are mind, and time is the advancement of knowledge. Each person advances his or her knowledge by listening to and combining ideas, thus generating new ideas for others to improve upon."

Socrates: "Very good. Now suppose one was to discover a previously unknown work by say.the philosopher Heraclitus. Suppose also that this work was so advanced for its time that it changed my thinking on previous work that I had done. I would need to go back to that previous work, remember what I had been thinking at that time, incorporate the new idea from Heraclitus, and generate a new result."

Thrasymachus: "But from society's perspective, this would not be enough. You would need to remember to whom you had communicated your previous ideas and give them the new result. This may cause those people, in turn, to modify their own past work."

Socrates: "Exactly. One can see the advancement of philosophy moving faster in some people and slower in others. The people in whom it moves slowest can impede the advancement for society in general. If the ideas (messages) could be transmitted and received in proper order of advancement among individuals, then progress by society would be fastest; rather than having to waste time and energy to go back and correct for new ideas."

Thrasymachus: "This sounds fantastic if the messages happen to arrive in causal order, that is, in the order in which they should be received. It also sounds terribly inefficient if messages arrive out-of-order."

Socrates: "Perhaps Complexity Theory can be of help here. It is known that the true measure of complexity of a string is reached when the program that describes the string is the smallest program that returns the string. As the program becomes smaller, it becomes more random. Thus, the program optimized for size is the more random program. Can this be true of time as well? Is the most compressed, thus most efficient, virtual time also the most random?"

Glaucon: "I am beginning to grasp what you are saying. If the rollbacks occur in random sequence, then perhaps the network is optimized; if there is any non-randomness, or pattern in the rollback sequence, then there is an opportunity to optimize the causality in some manner."

Thrasymachus (sarcastically): "Wonderful, another dead-end. There are no perfect tests for randomness. You can't even detect it, much less optimize it using this method."

Socrates: "Unfortunately, Thrasymachus, you are correct. If there were answers to the deep problems of randomness and complexity, and their relationship to time and space, these would result in great benefits to mankind."

The next part of this report attempts to address the concepts raised in this discussion. Chapters 3 and 4 discuss an implementation of the distributed network prediction framework that is included on the CD in this report. This framework enables the rollback mechanism explained by Socrates and Thrasymachus above. Chapter 5 discusses in detail the work on synchronization algorithms leading towards AVNMP. Chapter 6 builds the theory for relating performance, accuracy, and overhead of such a system. Chapter 7 considers many of Thrasymachus' arguments against the existence of such a predictive system.

Notes

[1][Bush et al., 1999] and [Bush, 2000] provide early thoughts on this concept.

3

AVNMP ARCHITECTURE

This chapter begins by describing the Active Virtual Network Management Prediction architecture and follows with an operational example. While the system attributes predicted by the Active Virtual Network Management Prediction Algorithm are generic, the focus of this report is load prediction. In the discussion that follows, new meaning is given to seemingly familiar terms from the area of parallel simulation. Terminology borrowed from previous distributed simulation algorithm descriptions has a slightly different meaning in Active Virtual Network Management Prediction; thus it is important that the terminology be precisely understood by the reader.

The Active Virtual Network Management Prediction Algorithm can be conceptualized as a model-based predictive control technique where the model resides inside the system being controlled. As shown in Figure 3.1, a virtual network representing the actual network can be viewed as overlaying the actual network. The system being controlled is a communications network comprised of many intermediate devices, each of which is an active network node. This is an inherently parallel system; the predictive capability is enhanced by using a technique that takes maximum advantage of parallelism.

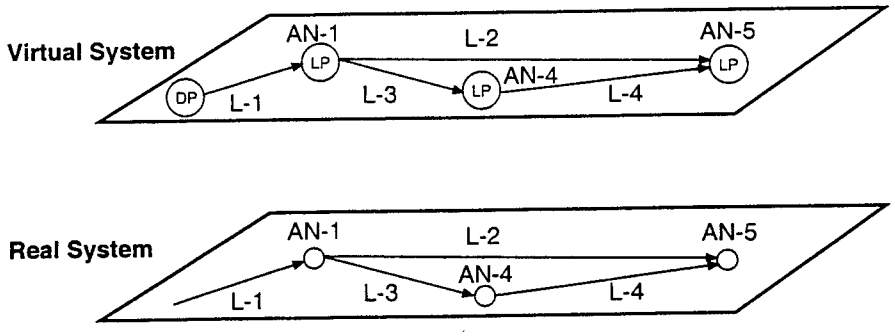


Figure 3.1. Virtual Overlay.

A well-known problem with parallel simulation is the blocking problem illustrated in Figure 3.2, where processors A, B, C, and D are each driven by messages whose queues are shown attached to the processor. The message time-stamps are indicated within the message. The message value is irrelevant. Notice that processor D could execute the message with time stamp 9, then it could receive the next message with time-stamp 6. This is a violation of causality and

could lead to an inaccurate result. There have been many proposed solutions to this problem which are described in greater detail in the following chapters of this report. However, many solutions depend on the processor that is likely to receive messages out of order waiting until the messages are guaranteed to arrive in the proper order. This adds delay and thus reduces the overall system performance. The Active Virtual Network Management Prediction Algorithm follows a well-known optimistic approach that allows all processors to continue processing without delay, but with the possibility that a processor may have to rollback to a previous state. In addition the Active Virtual Network Management Prediction Algorithm dynamically keeps the predictions within a given tolerance of actual values. Thus the model-based predictive system gains speed up due to parallelism while maintaining prediction accuracy.

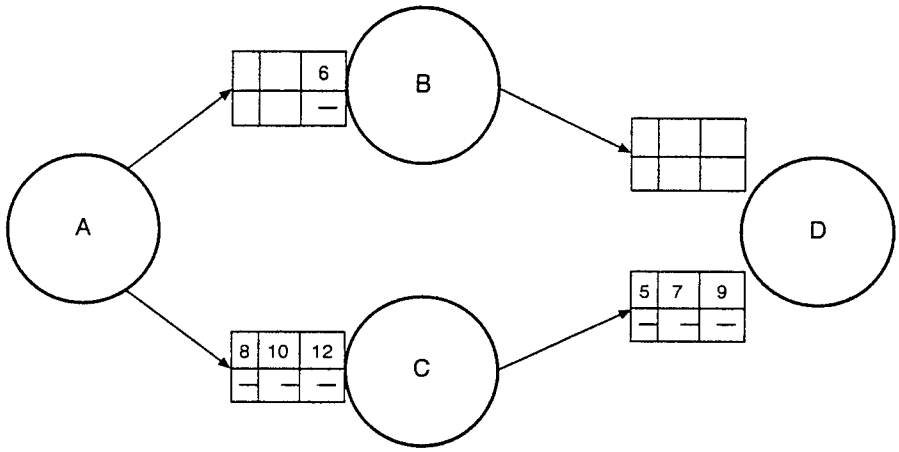


Figure 3.2. Blocked Process.

3.1 AVNMP ARCHITECTURAL COMPONENTS

The Active Virtual Network Management Prediction algorithm encapsulates each Physical Process within a Logical Process. A Physical Process is nothing more than an executing task defined by program code. The Logical Process can be thinly designed to use the physical processes' software. If that is not possible, then the entire model can be designed into the Logical Process. An example of a Physical Process is the packet forwarding process on a router. A Logical Process consists of a Physical Process and additional data structures and instructions that maintain and correct operation as the system executes ahead of wallclock time as illustrated in Figure 3.3. As an example, the packet forwarding Physical Process is encapsulated in a Logical Process that maintains load values in its State Queue and handles rollback due to out-of-order input messages or out-of-tolerance real messages as explained later. A Logical Process contains a Send Queue (QS) and State Queue (SQ) within an active packet. In this implementation, the packet is encapsulated inside a Magician SmartPacket which follows the Active Network Encapsulation Protocol (Alexander et al., 1997) format. The Receive Queue maintains newly arriving messages in order by their Receive Time (TR). The Receive Queue is an object residing in an

active node's smallstate. Smallstate is state left behind by an active packet. The Magician (Kulkarni et al., 1998) execution environment is used in the implementation described in this report. The Magician execution environment allows any kind of information to be stored in smallstate including Java objects; the Receive Queue is a Java object maintaining active virtual message ordering and scheduling. The Send Queue maintains copies of previously sent messages in order of their send times. The Send Queue is necessary for the generation of anti-messages for rollback described later. The state of a Logical Process is periodically saved in the State Queue. An important part of the architecture for network management is that the state queue of the Active Virtual Network Management Prediction system **is** the network Management Information Base. The Active Virtual Network Management Prediction values are the Simple Network Management Protocol Management Information Base Object values. They are the values expected to occur in the future. The current version of the Simple Network Management Protocol (Rose, 1991) has no mechanism for a managed object to report its future state; currently all results are reported assuming the state is valid at the current time. In working on predictive Active Network Management there is a need for managed entities to report their state information at times in the future. These times are unknown to the requester. A simple means to request and respond with future time information is to append the future time to all Management Information Base Object Identifiers that are predicted. This requires making these objects members of a table indexed by predicted time. Thus a Simple Network Management Protocol client that does not know the exact time of the next predicted value can issue a **get-next** command appending the current time to the known object identifier. The managed object responds with the requested object valid at the closest future time as shown in Figure 3.4.

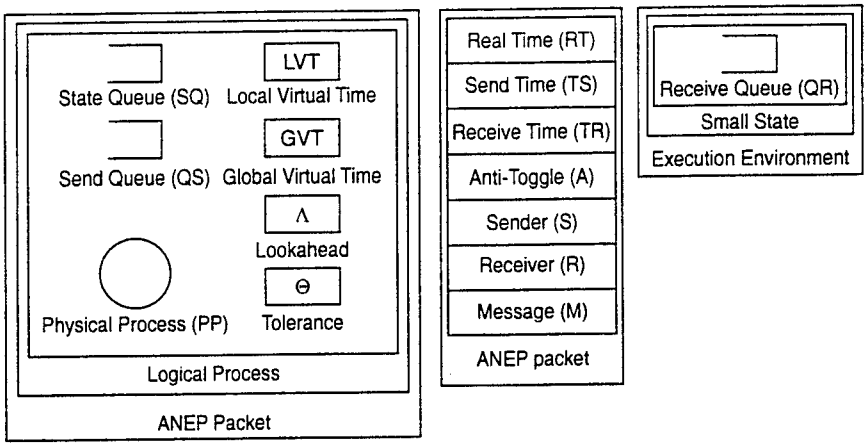


Figure 3.3. Active Global Virtual Time Calculation Overview.

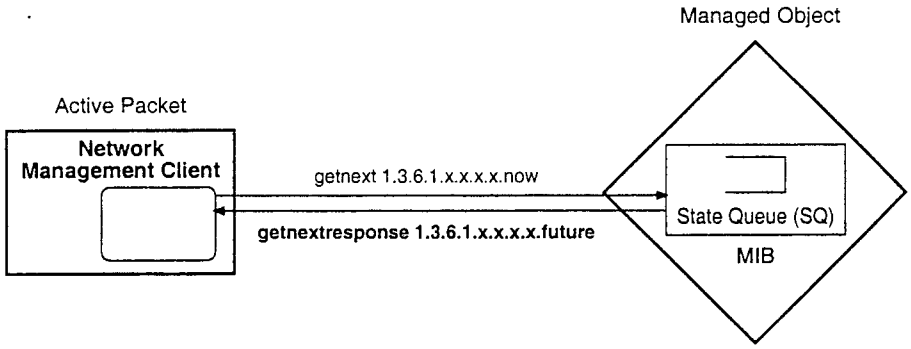


Figure 3.4. Legacy Network Management Future Time Request Mechanism.

The Logical Process also contains its notion of time, known as Local Virtual Time (LVT), and a Tolerance (Θ). Local Virtual Time advances to the of the next virtual message that is processed. Tolerance is the allowable deviation between actual and predicted values of incoming messages. For example, when a real message enters the load prediction Logical Process, the current load values are compared with the load values cached in the State Queue of the Logical Process. If predicted load values in the State Queue are out of tolerance, then corrective action is taken in the form of a rollback as explained later. Also, the Current State (CS) of a Logical Process is the current state of the structures and Physical Process encapsulated within a Logical Process.

3.1.1 Global Virtual Time

The Active Virtual Network Management Prediction system contains a notion of the complete system time known as Global Virtual Time (GVT) and a sliding window of length *Lookahead time* (Λ). Global Virtual Time is required primarily for the purpose of throttling forward prediction in Active Virtual Network Management Prediction; that is, it governs how far into the future the system predicts. There have been several proposals for efficient determination of Global Virtual Time, for example (Lazowaska and Lin, 1990) The algorithm in (Lazowaska and Lin, 1990) allows Global Virtual Time to be determined in a message-passing environment as opposed to the easier case of a shared memory environment. Active Virtual Network Management Prediction allows only message passing communication among Logical Processes. The algorithm in (Lazowaska and Lin, 1990) also allows normal processing to continue during the determination phase. A logical process that needs to determine the current Global Virtual Time does so by broadcasting a Global Virtual Time update request to all processes. Note that Global Virtual Time is the minimum of all logical process Local Virtual Times and the minimum message receive time that is in the system. An example is shown in Figure 3.5. The *Active Global Virtual Time Request Packet* notices that the logical process with a Global Virtual Time of 20 is greater than the last logical process that the *Active Global Virtual Time Request Packet* passed through and thus destroys itself. This limits unnecessary traffic and computation. The nodes that receive the *Active Global Virtual Time Request Packet* forward the result to the initiator of the Global Virtual Time request. As the *Active Global Virtual Time Request Packets* return to the initiator, the last packet is maintained in the cache of each logical process. If the value of the is

greater than or equal to the value in the cache, then the packet is dropped. Again, this reduces traffic and computation at the expense of space.

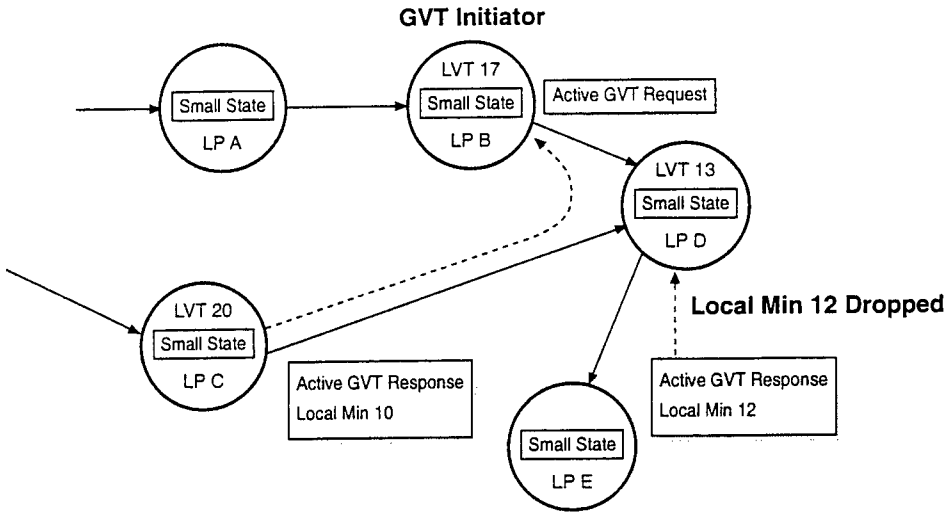


Figure 3.5. Active Global Virtual Time Calculation Overview.

3.1.2 AVNMP Message Structure

Active Virtual Network Management Prediction messages contain the Send Time (TS), Receive Time (TR), Anti-toggle (A) and the actual message object itself (M). The message is encapsulated in a Magician SmartPacket which follows the ANEP standard. The Receive Time is the time this message is predicted to be valid at the destination Logical Process. The Send Time is the time this message was sent by the originating Logical Process. The "A" field is the anti-toggle field and is used for creating an anti-message to remove the effects of false messages as described later. A message also contains a field for the current Real Time (RT). This is used to differentiate a real message from a virtual message. A message that is generated and time-stamped with the current time is called a real message. Messages that contain future event information and are time-stamped with a time greater than the current wallclock time are called virtual messages. If a message arrives at a Logical Process out of order or with invalid information, it is called a false message. A false message causes a Logical Process to rollback. The structures and message fields are shown in Table 3.1, Table 3.2 and in Figure 3.3. The Active Virtual Network Management Prediction algorithm requires a driving process to predict future events and inject them into the system. The driving process acts as a source of virtual messages for the Active Virtual Network Management Prediction system. All other processes react to virtual messages.

3.1.3 Rollback

A rollback is triggered either by messages arriving out of order at the Receive Queue of a Logical Process or by a predicted value previously computed by this Logical Process that is beyond the allowable tolerance. In either case, rollback is a mechanism by which a Logical Process returns to a known correct state. The rollback occurs in three phases. In the first phase, the state is restored to a time strictly earlier than the Receive Time of the false message. In the second phase, anti-messages are sent to cancel the effects of any invalid messages that had been generated before the arrival of the false message. An anti-message contains exactly the same contents as the original message with the exception of an anti-toggle bit which is set. When the anti-message and original message meet, they are both annihilated. The final phase consists of executing the Logical Process forward in time from its rollback state to the time the false message arrived. No messages are canceled or sent between the time to which the Logical Process rolled back and the time of the false message. These messages are correct; therefore, there is no need to cancel or re-send them, which improves performance and prevents additional rollbacks. Note that another false message or anti-message may arrive before this final phase has completed without causing problems. The Active Virtual Network Management Prediction Logical Process has the contents shown in Table 3.1, the message fields are shown in Table 3.2, and the message types are listed in Table 3.3 where t is the wallclock time at the receiving Logical Process.

Table 3.1. AVNMP Logical Process Structures

Structure	Description
Receive Queue (QR)	Ordered by message receive time (TR)
Send Queue (QS)	Ordered by message send time (TS)
Local Virtual Time	$LVT = \inf RQ$
Current State (CS)	State of the logical and physical process
State Queue (SQ)	States (CS) are periodically saved
Sliding Lookahead Window (SLW)	$SLW = (t, t + \Lambda)$
Tolerance (Θ)	Allowable deviation

Table 3.2 AVNMP Message Fields.

Field	Description
Send Time (TS)	LVT of sending process when message is sent
Receive Time (TR)	Scheduled time to be received by receiving process
Anti-toggle (A)	Identifies message as normal or antimessage
Message (M)	The actual contents of the message
Real Time (RT)	The wallclock time at which the message originated

Table 3.3 AVNMP Message Types.

Virtual Message	$RT > t$
Real Message	$RT \leq t$

3.1.4 Space-Time Trade-offs

The partitioning of physical processes into logical processes has an effect on the performance of the system. Active networks allow the possibility of physical processes to dynamically merge into logical process. In addition, both virtual and anti-messages can be fused on their way to their destination. There are several ways that this can occur. The first is a straightforward combination of data within the virtual messages when they reach a common node. Another fusion technique is to maintain a cache in each node of the last message that traveled through the node on the way to the message's destination for each source/destination pair. When a message arrives at a node to be forwarded towards its destination, it can check whether a message had been previously cached and if its Receive Time is greater than that of the current message. If so, this message knows it is going to cause a rollback. The message then checks whether it would have affected the result, for example, via a semantic check. If it would have had no effect, the message is discarded. In the specific case of load prediction, the change in load that the out-of-order message creates within the system can be easily checked. If many messages discover they would cause rollback on the way towards their destination, the destination logical process could perhaps be moved closer to the offending message generator logical process. If the message is a real message and the cached message is virtual and their times are not too far apart, a check can be made at that point as to whether a rollback is needed. If no rollback is needed, the real message can be dropped.

Virtual messages can be cached as they travel to their destination logical process. The cache uses a key consisting of the source-destination node of the message. Only the last message for that source-destination pair is cached. When the next message passes through the intermediate node matching that source-destination pair, the new message compares itself with the cached message. This is shown in Figure 3.6. If one exists and has a larger time-stamp, then a rollback is highly likely, and steps can be taken to mitigate the effects of the rollback. After the comparison, the old message is replaced in the cache with the new message. If many such rollback indications appear in the path of a virtual message, the destination process can be slowed or move itself to a new spatial location to mitigate the temporal effects of causality violations. Also, if a new message passing through an intermediate node is real, and the cached message is virtual, and they are within the same tolerance of time and value, the real message will destroy itself since it is redundant.

Logical Processes, because they are active packets, can move to locations that will improve performance. Logical Processes can even move between the network and end systems. In an extreme case of process migration, the Logical Processes are messages that install themselves only where needed to simulate a portion of the network as shown in Figure 3.7. Notice that choosing to simulate a single route always results in a feed-forward network.

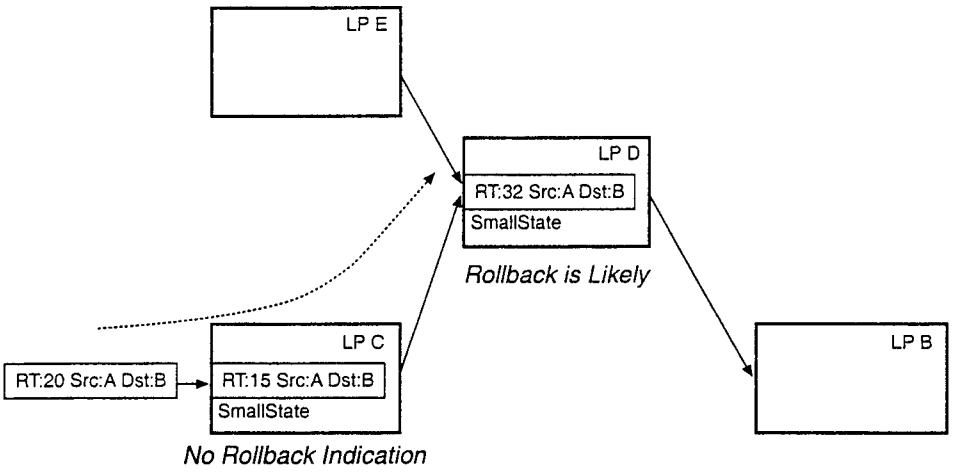


Figure 3.6. Active Rollback Mitigation.

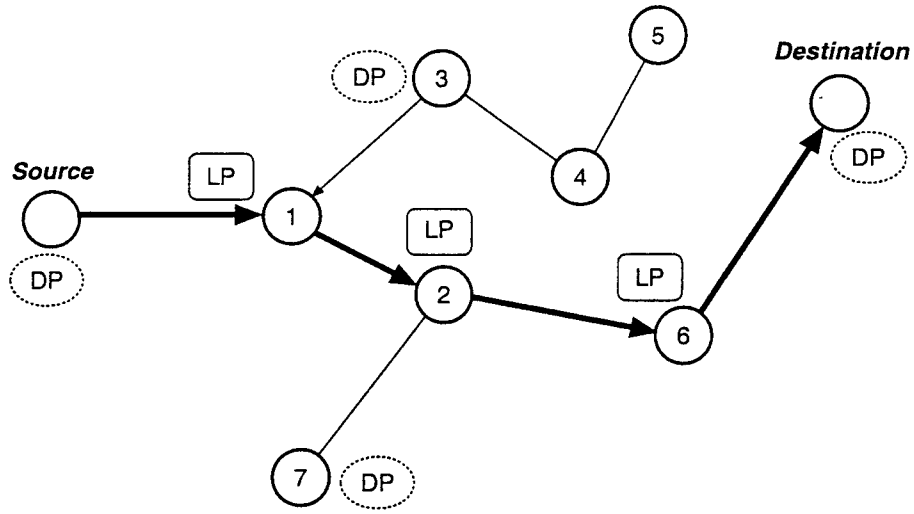


Figure 3.7. Partial Spatial Network Prediction.

3.1.5 Enhanced Message Capabilities

The active packet allows the virtual message to be enhanced with more processing capability. The virtual message can use this capability to refine its prediction as it travels through the network. In the Active Virtual Network Management Prediction architecture described thus far, there is a one-to-one correspondence between virtual messages and real messages. While this correspondence works well for adding prediction to protocols using a relatively small portion of the total bandwidth, it would clearly be beneficial to reduce message load, especially when attempting to add prediction of the bandwidth itself. There are more compact forms of representing future behavior within an active packet besides a virtual message. For relatively simple and easily modeled systems, only the model parameters need be sent and used as input to the logical process on the appropriate intermediate device. Note that this assumes that the intermediate network device's Logical Process is simulating the device operation and contains the appropriate

model. However, because the payload of a virtual message is exactly the same as a real message, it can be passed to the actual device and the result from the actual device is intercepted and cached. In this case, the Logical Process is a thin layer of code between the actual device and virtual messages primarily handling rollback. An entire executable load model can be included within an active packet generated by the DP and executed by the Logical Process. When the active packet reaches the target intermediate device, the load model provides virtual input messages to the and the payload of the virtual message passed to the actual device as previously described. A Streptichron is an active packet facilitating prediction as shown in Definition 3.1, which implements any of the above mechanisms.

$$\text{Streptichron} \triangleq \begin{cases} \text{Input (Monte-Carlo) Model} \\ \text{Model Parameters (Self-Adjusting)} \\ \text{Virtual Message (Self-Adjusting)} \end{cases} \tag{3.1}$$

Autoanaplasis is the self-adjusting characteristic of streptichrons. For example, in load prediction, use the transit time to check prior predictions. Figure 3.8 shows an overview of autoanaplasis. General purpose code contained within the packet is executed on intermediate nodes as the packet is forwarded to its destination.

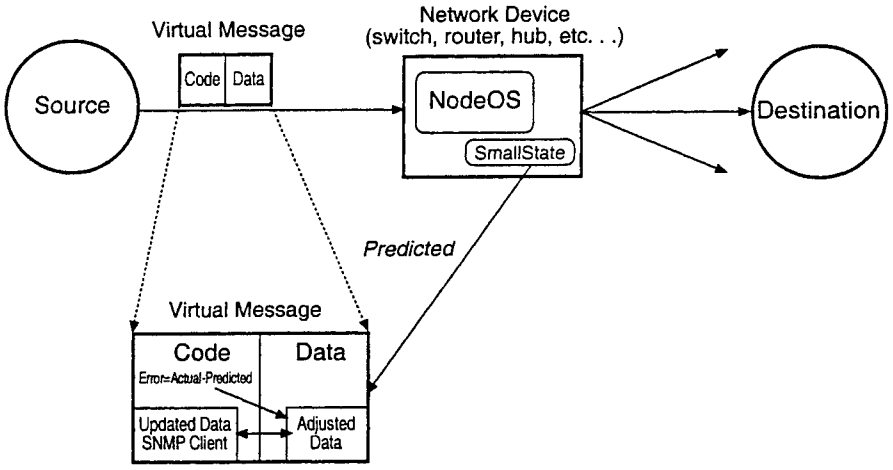


Figure 3.8. Self Adjusting Data.

For example, a packet containing a prediction of traffic load may notice changes in traffic that influence the value it carries as the packet travels towards its destination. The active packet updates the prediction accordingly.

3.1.6 Multiple Future Event Architecture

It is possible to anticipate alternative future events using a direct extension of the basic Active Virtual Network Management Prediction algorithm (Tinker and Agra, 1990). The driving process generates multiple virtual messages, one for each possible future event with corresponding probabilities of occurrence, or a ranking, for each event. Instead of a single Receive Queue for each Logical Process, multiple Receive Queues for each version of an event are cre-

ated dynamically for each Logical Process. The logical process can dynamically create Receive Queues for each event and give priority to processing messages from the most likely versions' Receive Queues. This enhancement to Active Virtual Network Management Prediction has not been implemented. This architecture for implementing alternative futures, while a simple and natural extension of the Active Virtual Network Management Prediction algorithm, creates additional messages and increases the message sizes. Messages require an additional field to identify the probability of occurrence and an event identifier. Alternative future events can also be considered at a much lower level, in terms of perturbations in packet arrivals. Perturbation Analysis is described in more detail in (Ho, 1992).

3.1.7 Magician and AVNMP

The Active Virtual Network Management Prediction Algorithm has been built upon the Magician (Kulkarni et al., 1998) Execution Environment. This section discusses the development and architecture at the Execution Environment level. As discussed in the beginning of this report, Magician is a Java-based Execution Environment that was used to implement the Active Virtual Network Management Prediction Algorithm because at the time this project started, Magician had the greatest flexibility and capability. This included the ability to send active packets as Java objects. Figure 3.9 shows the Java class structure of the Active Virtual Network Management Prediction Algorithm implementation. Time is critical in the architecture of the system; thus, most classes are derived from class *Date*. Class *AvnmpTime* handles relative time operations. Class *Gvt* uses active the *GvtPackets* class to calculate global virtual time. Class *AvnmpLP* handles the bulk of the processing including rollback. Class *Driver* generates and injects real and virtual messages into the system. The *PP* class either simulates, or accesses, an actual device on behalf of the Logical Process. The *PP* class may not need to simulate the device because the payload of a virtual message is exactly the same as a real message; thus, the payload of the virtual message can be passed to the actual device and the result from the actual device is intercepted and cached. In this case, the Logical Process is a thin layer of code between the actual device accessed by the *PP* class. The *GvtPacket* class implements the Global Virtual Time packet which is exchanged by all logical and driving processes to determine global virtual time. Currently only the virtual message form of a streptichron has been implemented. The active packets have been implemented in both ANTS (Tennenhouse et al., 1997) and SmartPackets (Kulkarni et al., 1998).

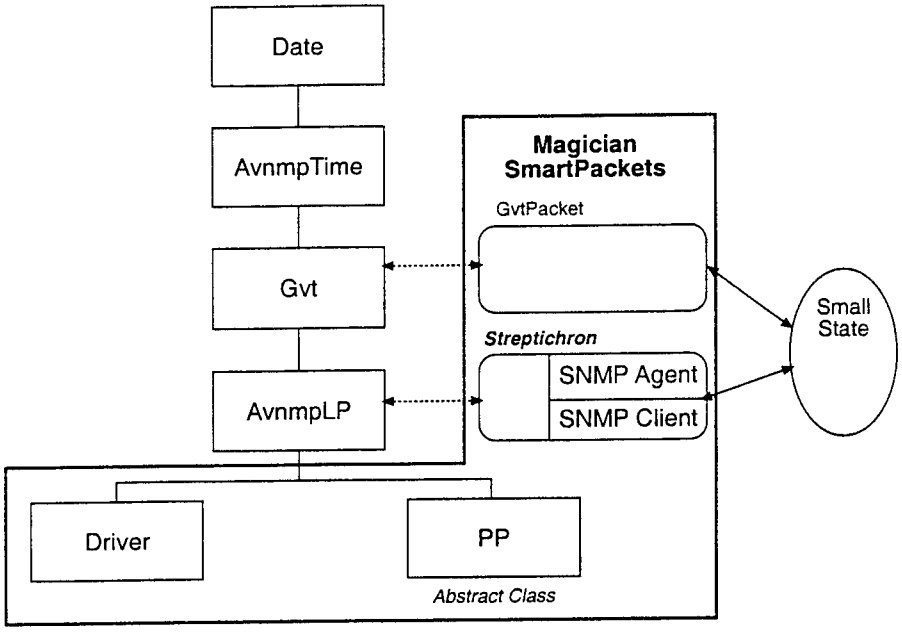


Figure 3.9. Active Virtual Network Management Protocol Class Hierarchy.

3.2 EXAMPLE DRIVING PROCESSES

3.2.1 Flow Prediction

Network flows are comprised of streams of packets. The ultimate goal for network management of flows is to allocate resources in order to provide the best quality of service possible for all user flows within the network. However, knowledge of how best to allocate resources is greatly aided by knowledge of future usage. Active Virtual Network Management Prediction provides that future usage information. The Active Virtual Network Management Prediction driving processes generate virtual load messages. The manner in which the prediction is accomplished is irrelevant to Active Virtual Network Management Prediction. Some example techniques could include a Wavelet-based technique described in (Ma and Ji, 1998) or simple regression models (Pandit and Wu, 1983).

3.2.2 Mobility Prediction

Proposed mobile networking architectures and protocols involve predictive mobility management schemes. For example, an optimization to a Mobile IP-like protocol using IP-Multicast is described in (Seshan et al., 1996). Hand-offs are anticipated and data is multicast to nodes within the neighborhood of the predicted handoff. These nodes intelligently buffer the data so that no matter where the mobile host (MH) re-associates after a handoff, no data will be lost. Another example (Liu et al., 1995) (Liu, 1996) proposes deploying mobile floating agents, which decouple services and resources from the underlying network. These agents would be pre-assigned and pre-connected to predicted user locations.

The Active Virtual Network Management Prediction driving process for mobile systems requires accurate position prediction. A non-active form of Active Virtual Network Management

Prediction has been used for a rapidly deployable wireless mobile network as described in (Bush, 1997). Previous mobile host location prediction algorithms have focused on an aggregate view of mobile host location prediction, primarily for such purposes as base-station channel assignment and base-station capacity planning. Examples are a fluid flow model (Thomas et al., 1988) and the method of Hong and Rappaport (Hong and Rappaport, 1986). A location prediction algorithm accurate enough for individual mobile host prediction has been developed in (Liu and Jr., 1995). A brief overview of the algorithm follows because the algorithm in (Liu and Jr., 1995) is an ideal example of a driving process for Active Virtual Network Management Prediction and demonstrates the speedup that Active Virtual Network Management Prediction is capable of providing with this prediction method. The algorithm allows individual mobile hosts to predict their future movement based on past history and known constraints in the mobile host's path.

All movement ($\{M(k,t)\}$) is broken into two parts, regular and random motion. A Markov model is formed based on past history of regular and random motion and used to build a prediction mechanism for future movement as shown in Equation 3.1. The regular movement is identified by $S_{k,t}$ where S is the state (geographical cell area) identified by state index k at time t and the random movement is identified similarly by $X(k,t)$. $M(k,t)$ is the sum of the regular and random movement.

$$\{M(k,t)\} = \{S_{k,t} \mid k \leq K, t \in T\} + \{X(k,t) \mid k \leq K, t \in T\} \qquad (3.1)$$

$$\{X(k,t)\} = \{M(k,t)\} - (\{M_c(k,t) | k \leq K, t \in T\} + \{M_t(k,t) \mid k \leq K, t \in T\}) \qquad (3.2)$$

The mobile host location prediction algorithm in (Liu and Jr., 1995) determines regular movement as it occurs, then classifies and saves each regular move as part of a movement track or movement circle. A movement circle is a series of position states that lead back to the initial state, while a movement track leads from one point to another distinct point. A movement circle can be composed of movement tracks. Let M_c denote a movement circle and M_t denote a movement track. Then Equation 3.2 shows the random portion of the movement.

The result of this algorithm is a constantly updating model of past movement classified into regular and random movement. The proportion of random movement to regular movement is called the randomness factor. Simulation of this mobility algorithm in (Liu and Jr., 1995) indicates a prediction efficiency of 95%. The prediction efficiency is defined as the rate over the regularity factor. The prediction accuracy rate is defined in (Liu and Jr., 1995) as the probability of a correct prediction. The regularity factor is the proportion of regular states, $\{S_{k,t}\}$, to random states $\{X(k,t)\}$. The theoretically optimum line in (Liu and Jr., 1995, p. 143) may have been better labeled the deterministic line. The deterministic line is an upper bound on prediction performance for all *regular* movement. The addition of the random portion of the movement may increase or decrease actual prediction results above or below the deterministic line. A theoretically optimum (deterministic) prediction accuracy rate is one with a randomness factor of zero and a regularity factor of one. The algorithm in (Liu and Jr., 1995) does slightly worse than expected for completely deterministic regular movement, but it improves as movement becomes more random. As a prediction algorithm for Active Virtual Network Management Prediction, a state as defined in (Liu and Jr., 1995) is chosen such that the area of the state corresponds exactly to the Active Virtual Network Management Prediction tolerance, then based on the prediction accuracy rate in the graph shown in (Liu and Jr., 1995, p. 143) the probability of being out of tolerance is less than 30% if the random movement ratio is kept below 0.4. An out-of-tolerance proportion of less than 30% where virtual messages are transmitted at a rate of $\lambda_{vm} = 0.03$ per millisecond results in a significant speedup as shown in Chapter 6.

3.2.3 Vulnerability Prediction

Network vulnerability to information warfare attack can be quantified and vulnerability paths through the network can be identified. General Electric Corporate Research & Development has a patent disclosure on such a system. The results of this vulnerability system are used to identify the most likely path of an attack, thus predicting the next move of a knowledgeable attacker.

Once an attack has been detected, the network command and control center can respond to the attack by repositioning safe-guards and by modifying services used by the attacker. However, cutting-off services to the attacker also impacts legitimate network users, and a careful balance must be maintained between minimizing the threat from the attack and maximizing service to customers. For example, various stages of an attack are shown in Figure 3.10. Since the allocation of resources never changes throughout the attack in this specific scenario, the vulnerability of the target increases significantly with each step of the attack.

A probabilistic and maximum flow analysis technique for quantifying network vulnerability have been developed at General Electric Corporate Research & Development (Bush and Barnett, 1998). The results from that work are the probability of an attacker advancing through multiple vulnerabilities and the maximum flow or rate. Using this information, the logical processes in Figure 3.11 can predict when and where the attacker is likely to proceed and can update the graphical interface with this information before the attack is successful. This allows time for various countermeasures to be taken or the opportunity to open an easier path for the attacker to a "fish bowl," a portion of the network where attackers are unknowingly steered in order to watch their activity. Virtual messages are exchanged between the Information Warfare Command and Control and the logical processes in Figure 3.11.

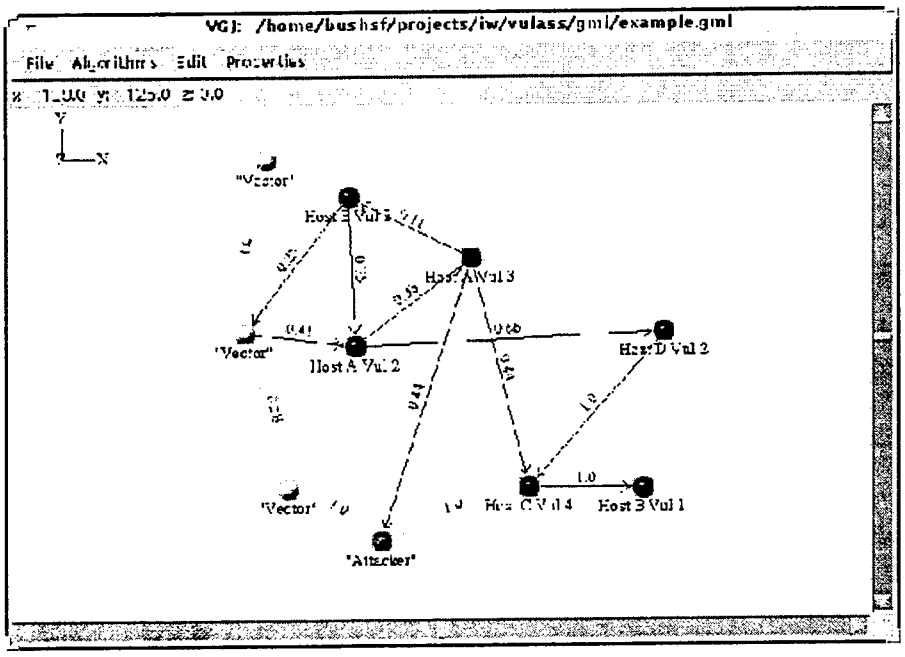


Figure 3.10. An Example of an Attack in Progress.

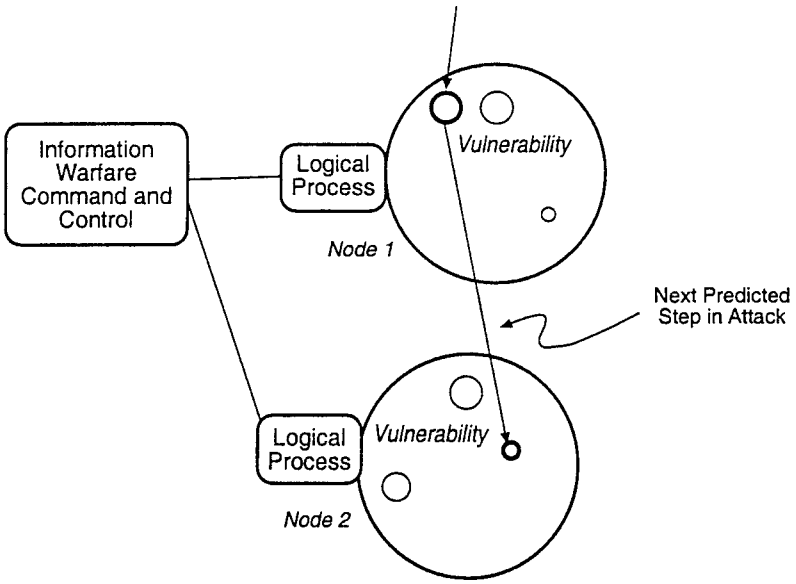


Figure 3.11. An Overview of Information Warfare Attack Prediction.

4

AVNMP OPERATIONAL EXAMPLES

The driving processes can make local predictions about load, vulnerability (Bush and Barnett, 1998), and mobile location (Bush, 1997). Load can be used to predict local QoS, congestion, and faults. The focus of this report is on the development and application of the Active Virtual Network Management Prediction algorithm and not the predictive methods within the driving processes. The primary purpose of Active Virtual Network Management Prediction is to distribute local changes throughout the network in both space and time.

Various predictive techniques can be used such as regression-based methods based on past history or similar techniques in the Wavelet domain. Since the Active Virtual Network Management Prediction implementation follows good modular programming style, the driving process has been decoupled from the actual prediction algorithm. Active Virtual Network Management Prediction has been tested by executing it in a situation where the outputs and internal state are known ahead of time as a function of the driving process prediction. The prediction within the driving processes is then corrupted and the Active Virtual Network Management Prediction output examined to determine the effect of the incorrect predictions on the system.

4.1 AVNMP OPERATIONAL EXAMPLE

A specific operational example of the Active Virtual Network Management Prediction Algorithm used for load prediction and management is shown in Figures 4.2 through 4.10. This particular execution log is from the operation of Active Virtual Network Management Prediction running on a simple three node network with an active end-system and two active intermediate nodes: AH-1, AN-1, AN-2. The legend used to indicate Active Virtual Network Management Prediction events is shown in Figure 4.1.

The Active Virtual Network Management Prediction system illustrated throughout this report has been developed using the Magician (Kulkarni et al., 1998) active network execution environment; the driving processes, logical processes, and virtual messages are implemented as Magician Smartpackets.

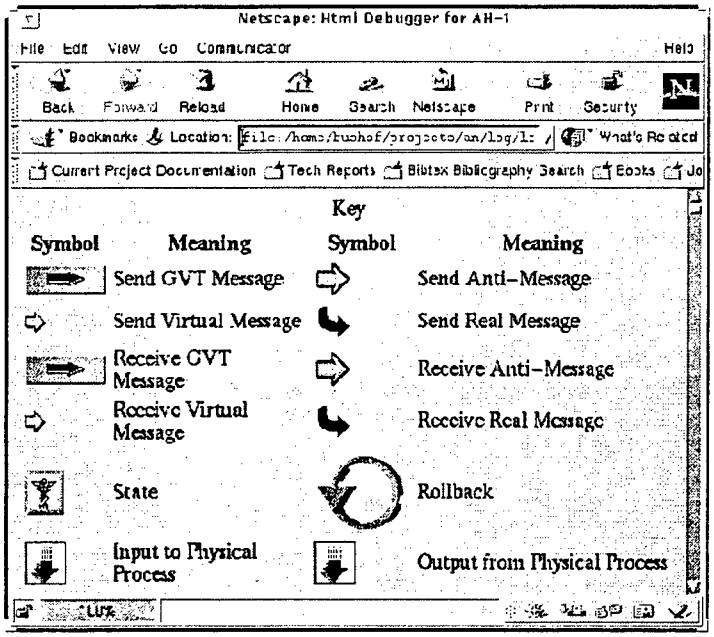


Figure 4.1. Legend of Operational Events.

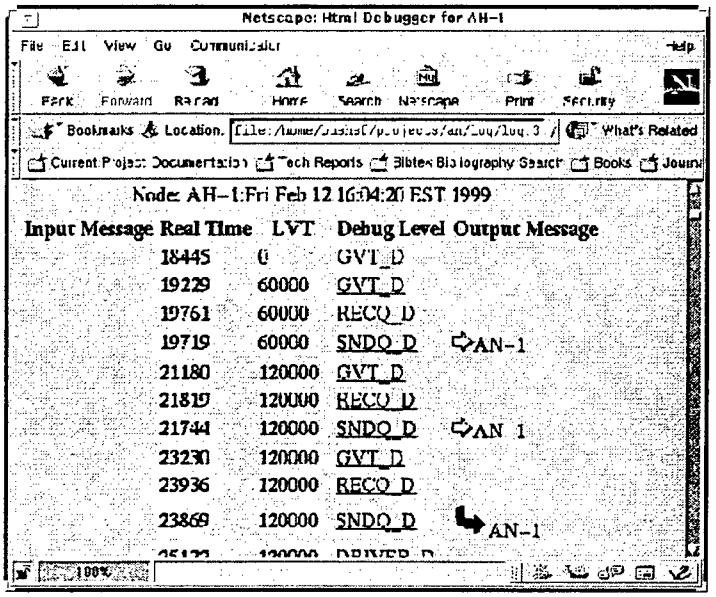


Figure 4.2. Active Node AH-1 Driving Process.

The logical process and driving process are injected into the network. The logical process automatically spawns copies of itself onto intermediate nodes within the network while the driving processes migrate to end-systems and begin taking load measurements in order to predict load and inject virtual messages. At the start of the Logical Process's execution, Local Virtual

Time and Global Virtual Time are set to zero, lookahead (Λ) is set to 60,000 microseconds, and a (Θ) of 1,000 bytes/second is allowed between predicted and actual values for this process.

The description of the algorithm begins with an Active Virtual Network Management Prediction enabled network that has just been turned on and is generating real messages. The real messages in this case are randomly generated Magician Smartpackets running over a local area network. The driving process, is located on active node, AH-1. The driving process generates predictions about usage in the near future and injects virtual messages based on those predictions as shown in Figure 4.2. Figure 4.2 illustrates the log format used for the top-level view of all the Active Virtual Network Management Prediction Logical Processes. The left-most column shows incoming messages, the next column shows the wallclock time in microseconds, the next column shows the Local Virtual Time, the next column is a link to more detailed information about the event, and the right-most column shows any output messages that are generated. Both the input and output messages indicate the type of message by the legend shown in Figure 4.1 and are labeled with the source or destination of the message. Active node AH-1 shows two virtual active packets and one real active packet sent to AN-1.

4.1.1 Normal Operation Example

In Figure 4.3, active node AN-1 has begun running and receives the first virtual message from AH-1. AN-1's Logical Process must first determine whether it is virtual or real by examining the field. If the active packet is a virtual active packet, the Logical Process compares the message with its Local Virtual Time to determine whether a rollback is necessary due to an out-of-order message. If the message has not arrived in the past relative to the Logical Process's Virtual Time, the message then enters the Receive Queue in order by Receive Time. The Logical Process takes the next message from the Receive Queue, updates its Local Virtual Time, and processes the message (shown below the current view in Figure 4.3. Figure 4.4 shows the AN-1 state after receiving the first virtual message.

If an outgoing message is generated, as shown in Figure 4.5, a copy of the message is saved in the State Queue, the Receive Time is set, and the Send Time is set to the current Local Virtual Time. The message is then sent to the destination Logical Process. If the virtual message arrived out of order, the Logical Process must rollback as described in the previous section. Figure 4.6 shows AN-1's Local Virtual Time, Send Queue contents, contents, and contents after the received virtual message has been processed and forwarded. Figure 4.7 shows AN-1's state after sending the first virtual message.

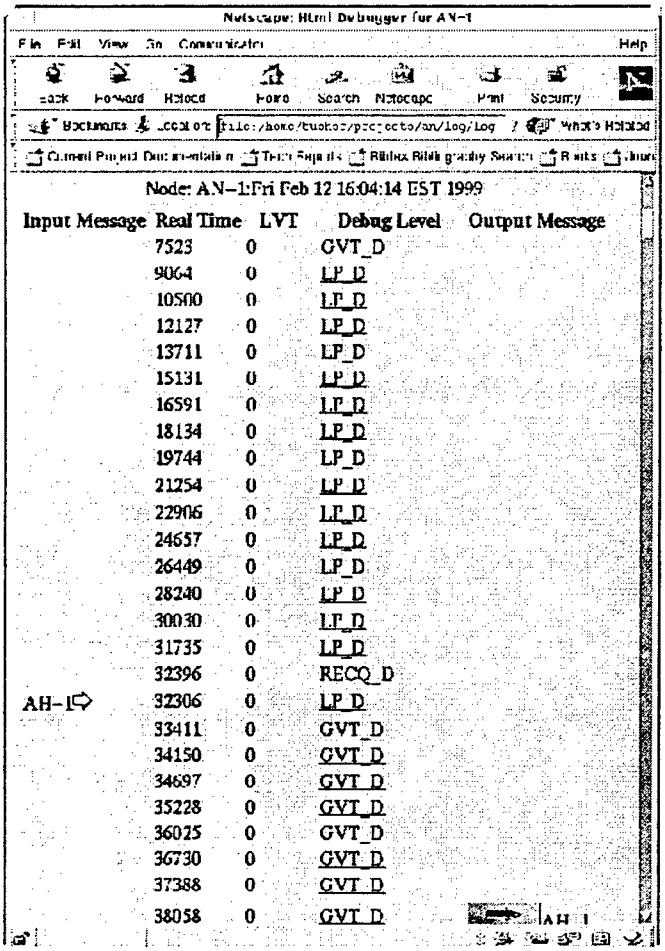


Figure 4.3. Active Node AN-1 Receives a Virtual Message.

4.1.2 Out-of-Tolerance Rollback Example

An example of out of tolerance rollback is illustrated in Figure 4.8. A real message arrives and its message contents are compared with the closest saved state value. The message value is out of tolerance; therefore, all state queue values with times greater than the receive time of the real message are discarded.

The send queue message anti-toggle is set and the anti-message is sent. The invalid states are discarded. The rollback causes the Logical Process to go back to time 120000 because that is the time of the most recent saved state that is less than the time of the out-of-tolerance message's Receive Time.

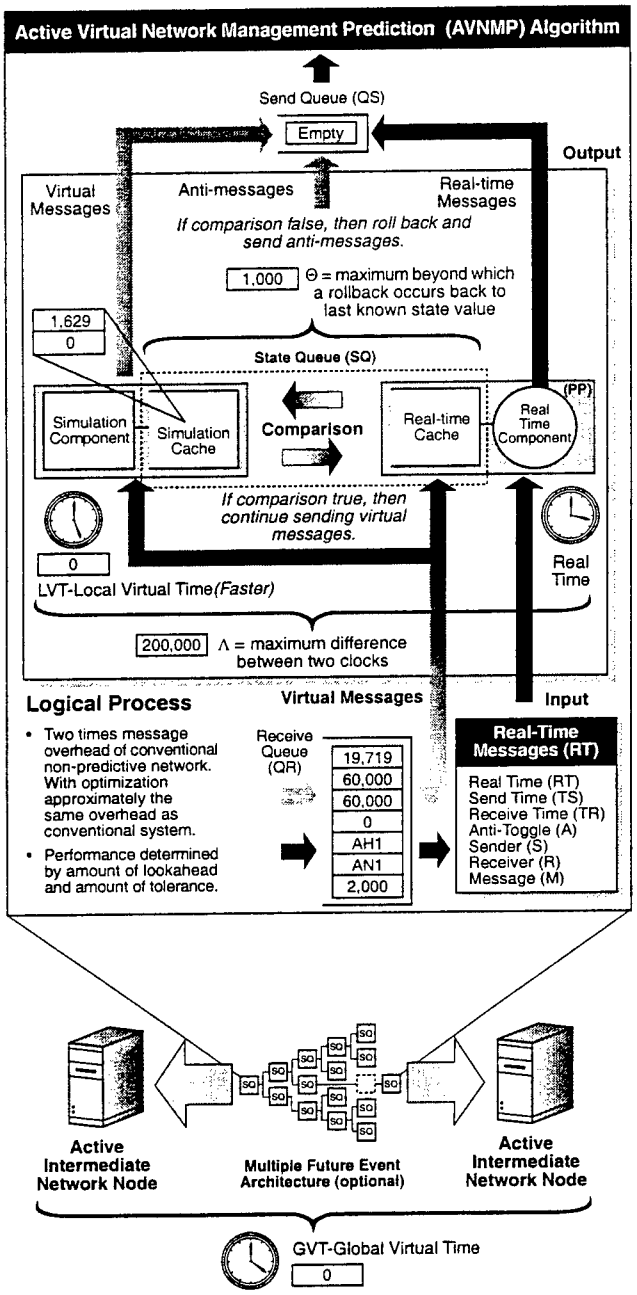


Figure 4.4. Active Node AN-1 After Receiving Virtual Message.

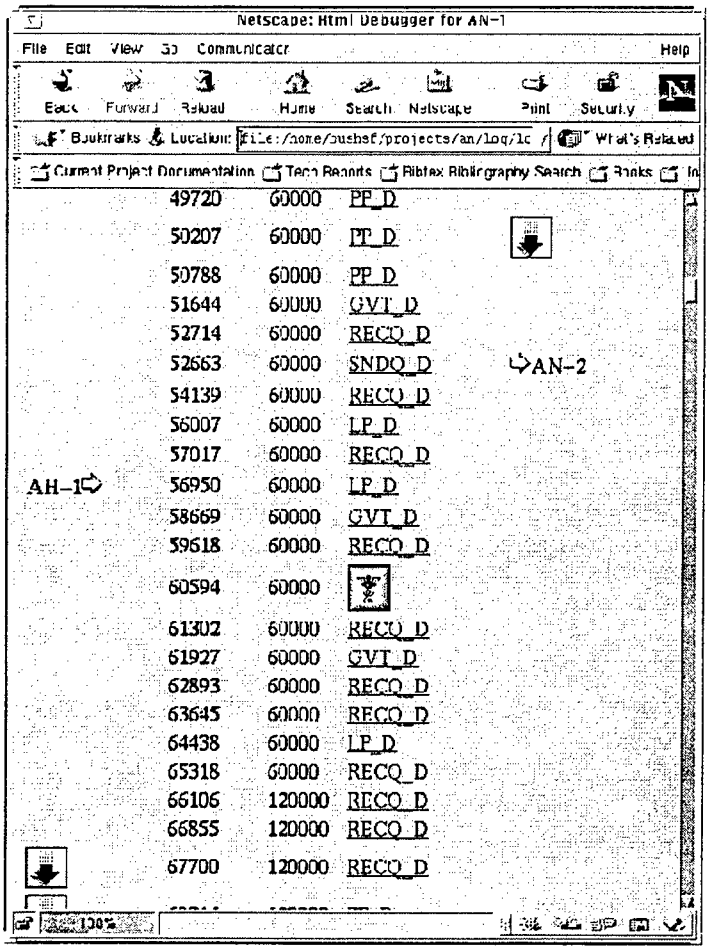


Figure 4.5. Active Node AN-1 Sends a Virtual Message.

Figure 4.12 shows the first virtual message received by AN-2. Figure 4.11 shows the AN-1 state after the first rollback. The anti-messages are the messages in the Send Queue that are crossed out. When these messages are sent as anti-messages, the anti-toggle bit is set. Also shown in Figure 4.11 is the discarded State Queue element that is no longer valid.

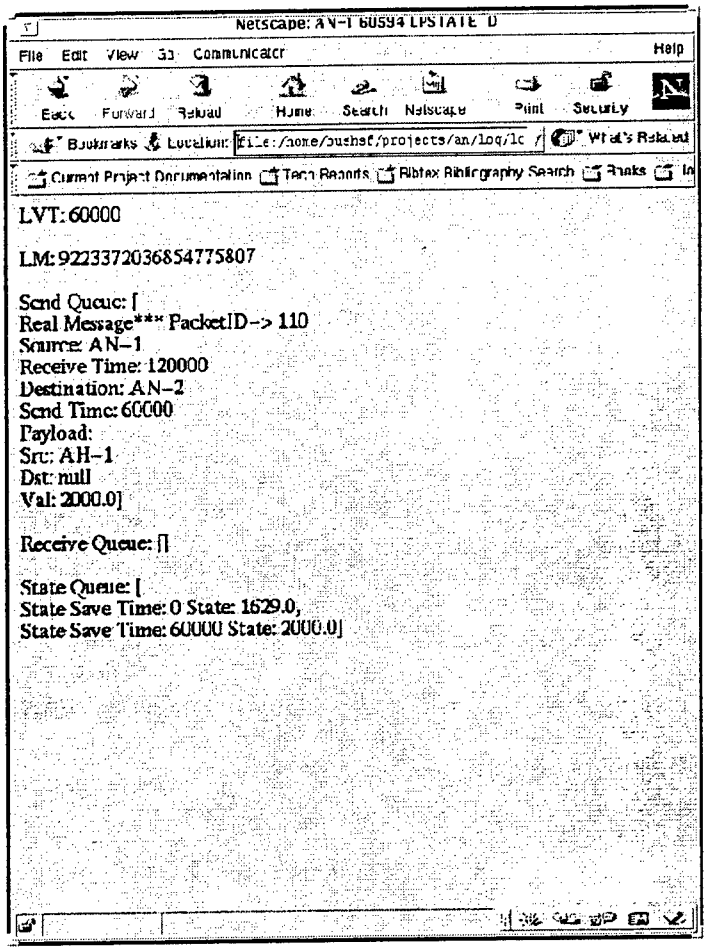


Figure 4.6. Active Node AN-1 Queue Contents after First Virtual Message Arrival.

4.1.3 Example Performance Results

Figure 4.13 shows the Local Virtual Time of node AN-1 versus wallclock time. Note that the logical process on AN-1 quickly predicted load 200,000 milliseconds ahead of wallclock time and then maintained the 200,000 millisecond lookahead. The sudden downward spikes in the plot are rollbacks.

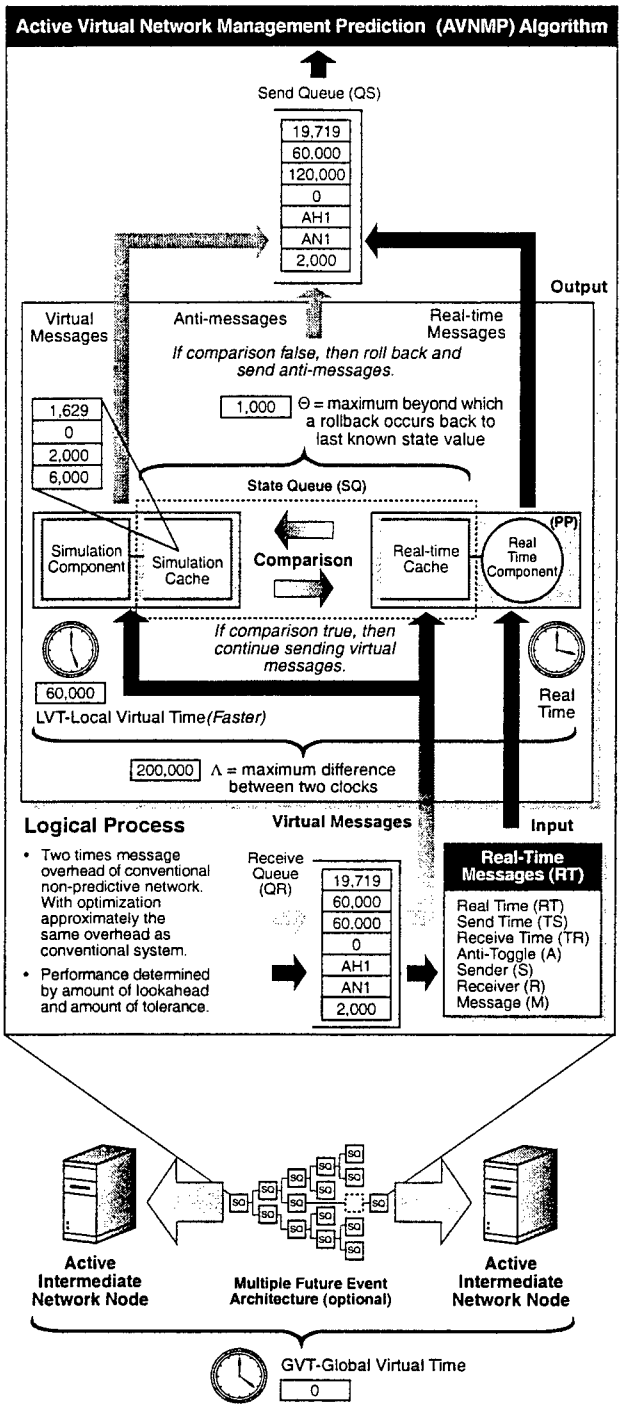


Figure 4.7. Active Node AN-1 after Sending Virtual Message.

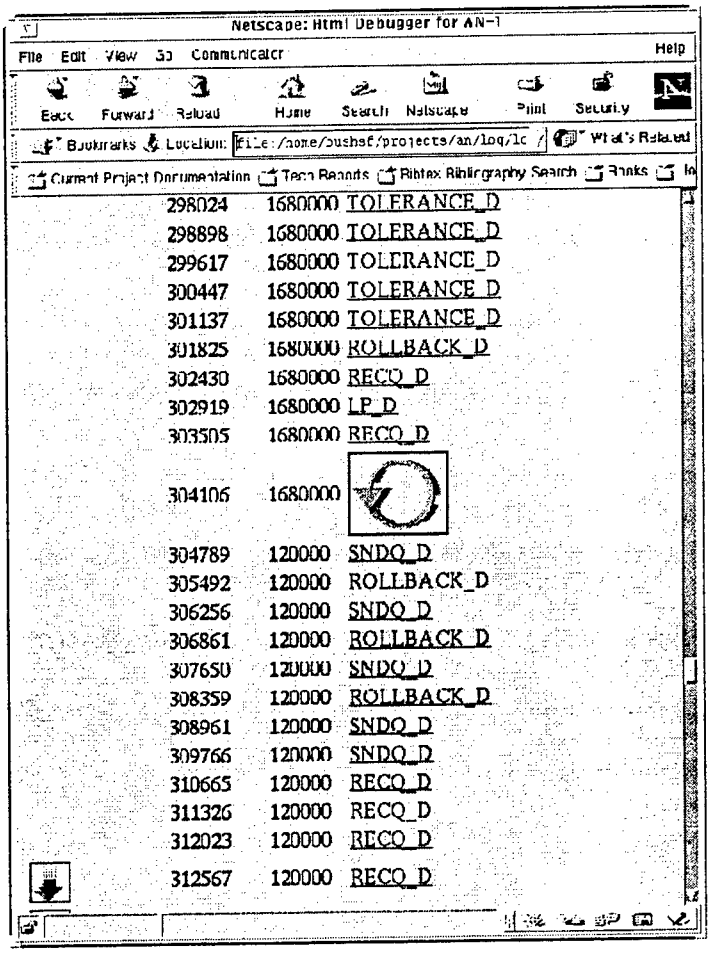


Figure 4.8. Active Node AN-1 Out-of-Tolerance Rollback Occurs.

A more complete view can be seen in the three-dimensional graph of Figure 4.14. The predicted values are shown as a function of wallclock time and LVT. This data was collected by SNMP polling an active execution environment that was enhanced with AVNMP. The valleys between the peaks are caused by the polling delay. A diagonal line on the LVT/Wallclock plane from the front right corner to the back left corner separates LVT in the past from LVT in the future; future LVT is towards the back of the graph, past LVT is in the front of the graph. Starting from the front, right hand corner, examine slices of fixed wallclock time over LVT; this shows both the past values and the predicted value for that fixed wallclock time.

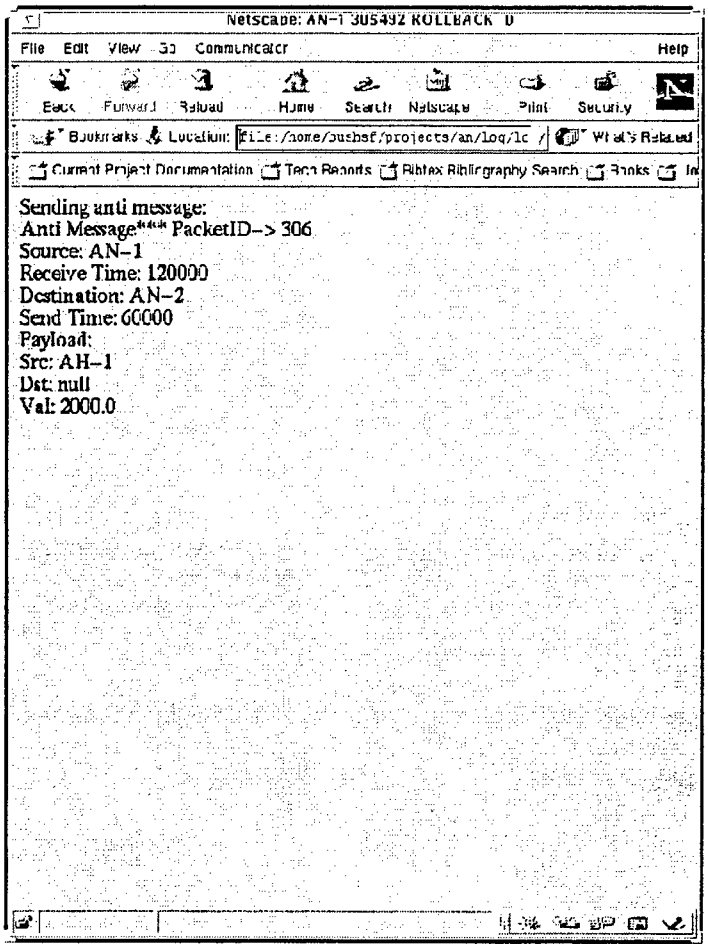


Figure 4.9. Active Node AN-1 Anti-Message Sent after First Rollback.

As wallclock time progresses, the system corrects for out-of-tolerance predictions. Thus, LVT values in the past relative to wallclock are corrected. By examining a fixed LVT slice, the prediction accuracy can be determined from the graph.

This chapter described the architecture and operation of the Active Virtual Network Management Prediction Algorithm. The performance of the algorithm is impacted by the accuracy of the predictions generated by the driving processes. The architecture is execution environment independent; however, the implementation used Magician. The next section discusses the driving processes in more detail. The remaining chapters of the report include analysis of the effect upon the system of driving process parameters such as virtual message generation rate, the ratio of virtual to real messages, and the prediction stepsize.

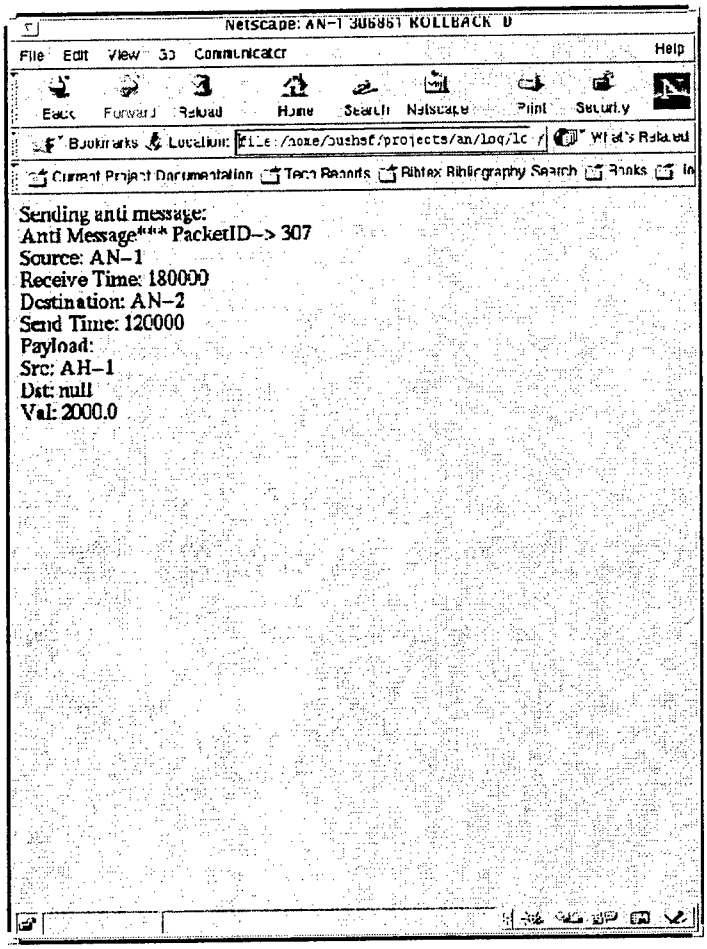


Figure 4.10. Active Node AN-1; Another Anti-Message Sent after First Rollback.

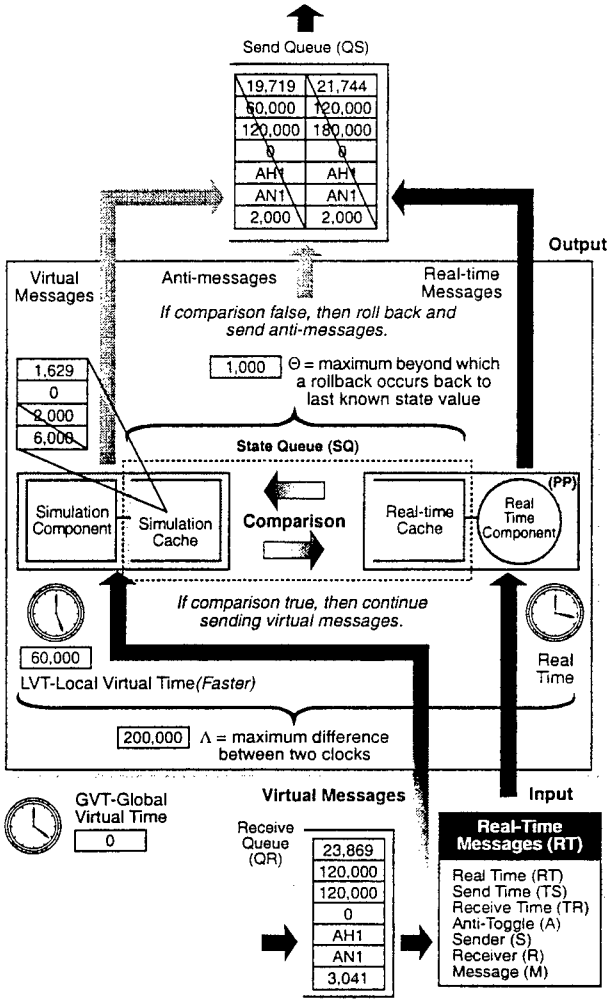


Figure 4.11. Active Node AN-1 after Rollback.

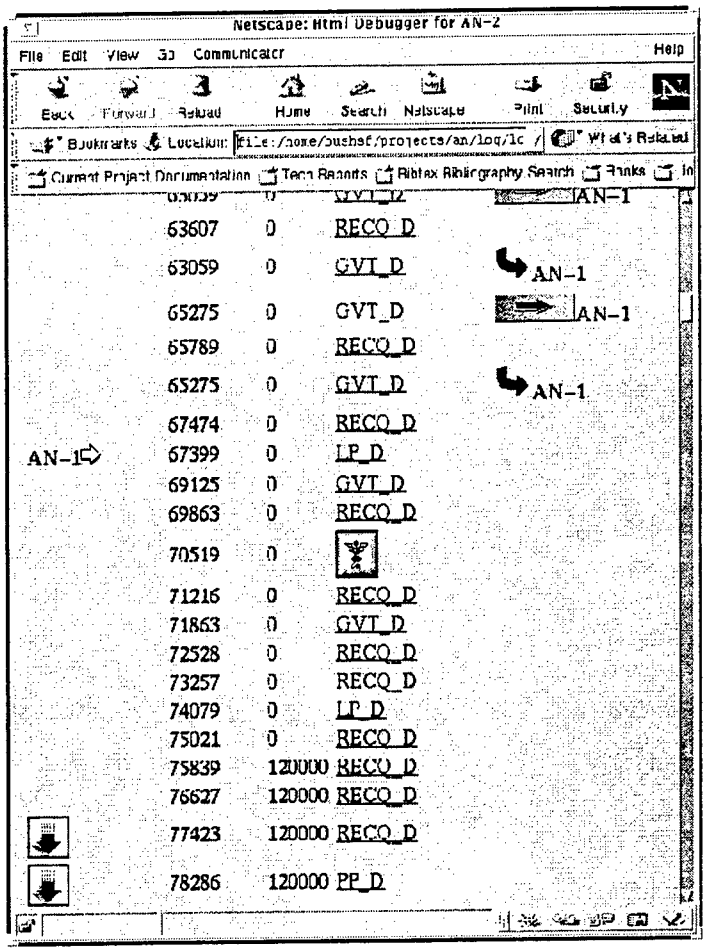


Figure 4.12. Active Node AN-2 First Virtual Message Received.

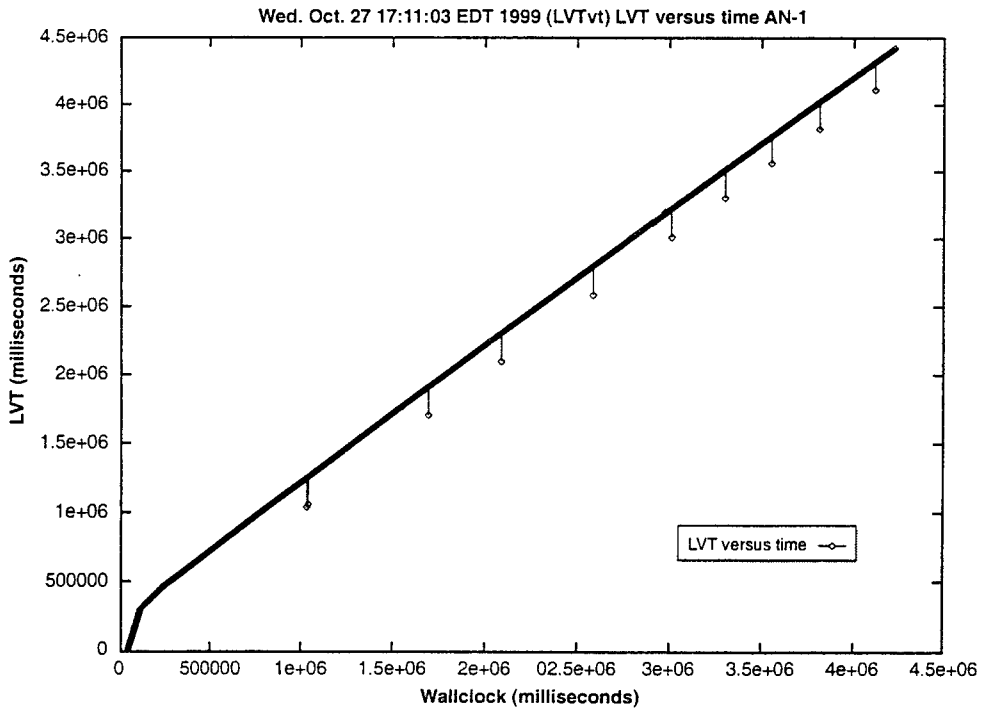


Figure 4.13. Active Node AN-1 LVT versus Wallclock.

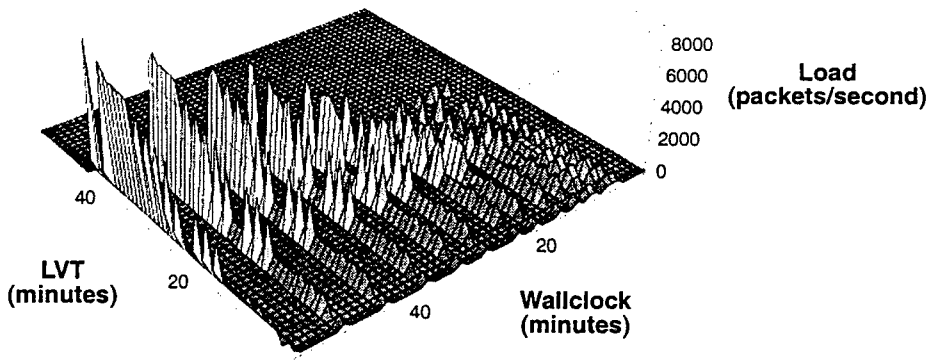


Figure 4.14. Three-Dimensional Graph Illustrating Predicted Load Values as a Function of Wallclock Time and LVT.

5

AVNMP ALGORITHM DESCRIPTION

One of the major contributions of this research is to recognize and define an entirely new branch of the Time Warp Family Tree of algorithms. Active Virtual Network Management Prediction integrates real and virtual time at a fundamental level allowing processes to execute ahead in time. The Active Virtual Network Management Prediction algorithm must run in real-time, that is, with hard real-time constraints.

5.1 FUNDAMENTALS OF DISTRIBUTED SIMULATION

Consider the work leading towards the predictive Active Virtual Network Management Prediction algorithm starting from a classic paper on synchronizing clocks in a distributed environment (Lamport, 1978). A theorem from this paper limits the amount of parallelism in any distributed simulation algorithm:

Rule 1: *If two events are scheduled for the same process, then the event with the smaller timestamp must be executed before the one with the larger timestamp.*

Rule 2: *If an event executed at a process results in the scheduling of another event at a different process, then the former must be executed before the latter.*

A parallel simulation method, known as CMB (Chandy-Misra-Bryant), that predates Time Warp (Jefferson and Sowizral, 1982) is described in (Chandy and Misra, 1979). CMB is a conservative algorithm that uses Null Messages to preserve message order and avoid deadlock. Another method developed by the same author does not require Null Message overhead, but includes a central controller to maintain consistency and detect and break deadlock. There has been much research towards finding a faster algorithm, and many algorithms claiming to be faster have compared themselves against the CMB method.

5.2 BASICS OF OPTIMISTIC SIMULATION

The basic Time Warp Algorithm (Jefferson and Sowizral, 1982) was a major advance in distributed simulation. Time Warp is an algorithm used to speedup Parallel Discrete Event Simulation by taking advantage of parallelism among multiple processors. It is an optimistic method because all messages are assumed to arrive in order and are processed as soon as possible. If a message arrives out-of-order at a Logical Process, the Logical Process rolls back to

a state that was saved prior to the arrival of the out-of-order message. Rollback occurs by sending copies of all previously generated messages as anti-messages. Anti-messages are exact copies of the original message, except and anti-bit is set within the field of the message. When the anti-message and real message meet, both messages are removed. Thus, the rollback cancels the effects of out-of-order messages. The rollback mechanism is a key part of Active Virtual Network Management Prediction, and algorithms that improve Time Warp and rollback also improve Active Virtual Network Management Prediction. There continues to be an explosion of new ideas and protocols for improving Time Warp. An advantage to using a Time Warp based algorithm is the ability to leverage future optimizations. There have been many variations and improvements to this basic algorithm for parallel simulation. A collection of optimizations to Time Warp is provided in (Fujimoto, 1990). The technical report describing Time Warp (Jefferson and Sowizral, 1982) does not solve the problem of determining Global Virtual Time; however, an efficient algorithm for the determination of Global Virtual Time is presented in (Lazowaska and Lin, 1990). This algorithm does not require message acknowledgments, thus increasing the performance, yet the algorithm works with unreliable communication links.

An analytical comparison of CMB and Time Warp is the focus of (Lin and Lazowska, 1990). In this paper the comparison is done for the simplified case of feed-forward and feedback networks. Conditions are developed for Time Warp to be conservative optimal. Conservative optimal means that the time to complete a simulation is less than or equal to the critical path (Berry and Jefferson, 1985) through the event-precedence graph of a simulation.

5.3 ANALYSIS OF OPTIMISTIC SIMULATION

A search for the upper bound of the performance of Time Warp versus synchronous distributed processing methods is presented in (Felderman and Kleinrock, 1990). Both methods are analyzed in a feed-forward network with exponential processing times for each task. The analysis in (Felderman and Kleinrock, 1990) assumes that no Time Warp optimizations are used. The result is that Time Warp has an expected potential speedup of no more than the natural logarithm of P over the synchronous method where P is the number of processors.

A Markov Chain analysis model of Time Warp is given in (Gupta et al., 1991). This analysis uses standard exponential simplifying assumptions to obtain closed form results for performance measures such as the fraction of processed events that commit, speedup, rollback recovery, expected length of rollback, probability mass function for the number of uncommitted processed events, probability distribution function of the local virtual time of a process, and the fraction of time the processors remain idle. Although the analysis appears to be the most comprehensive analysis to date, it has many simplifying assumptions such as no communications delay, unbounded buffers, constant message population, message destinations are uniformly distributed, and rollback takes no time. Thus, the analysis in (Gupta et al., 1991) is not directly applicable to the time sensitive nature of Active Virtual Network Management Prediction.

Further proof that Time Warp out-performs is provided in (Lipton and Mizell, 1990). This is done by showing that there exists a simulation model that out-performs CMB by exactly the number of processors used, but that no such model in which CMB out-performs Time Warp by a factor of the number of processors used exists.

A detailed comparison of the CMB and Time Warp methods is presented in (Lin, 1990). It is shown that Time Warp out-performs conservative methods under most conditions. Improvements to Time Warp are suggested by reducing the overhead of state saving information and the introduction of a global virtual time calculation. Simulation study results of Time Warp are presented in(Turnbull, 1992). Various parameters such as communication delay, process delay, and process topology are varied, and conditions under which Time Warp and CMB perform best are determined.

The major contribution of this section is to recognize and define an entirely new branch of the Time Warp Family Tree of algorithms, shown in Figure 5.1, that integrates real and virtual time at a fundamental level. The Active Virtual Network Management Prediction algorithm must run in real-time, that is, with hard real-time constraints. Real-time constraints for a time warp simulation system are discussed in (Ghosh et al., 1993). The focus in (Ghosh et al., 1993) is the R-Schedulability of events in Time Warp. Each event is assigned a real-time deadline ($d_{Ei,T}$) for its execution in the simulation. *R*-Schedulability means that there exists a finite value (*R*) such that if each event's execution time is increased by R, the event can still be completed before its deadline. The first theorem from (Ghosh et al., 1993) is that if there is no constraint on the number of such false events that may be created between any two successive true events on a Logical Process, Time Warp cannot guarantee that a set of R-schedulable events can be processed without violating deadlines for any finite R. There has been a rapidly expanding family of Time Warp algorithms focused on constraining the number of false events discussed next.

5.4 CLASSIFICATION OF OPTIMISTIC SIMULATION TECHNIQUES

Another contribution of this section is to classify these algorithms as shown in Figures 5.1, 5.2, 5.3 and Table 5.1. Each new modification to the Time Warp mechanism attempts to improve performance by reducing the expected number of rollbacks. Partitioning methods attempt to divide tasks into logical processes such that the inter- communication is minimized. Also included under partitioning are methods that dynamically move Logical Processes from one processor to another in order to minimize load and/or inter-Logical Process traffic. Delay methods attempt to introduce a minimal amount of wait into Logical Processes such that the increased synchronization and reduced number of rollbacks more than compensates for the added delay. Many of the delay algorithms use some type of windowing method to bound the difference between the fastest and slowest processes.

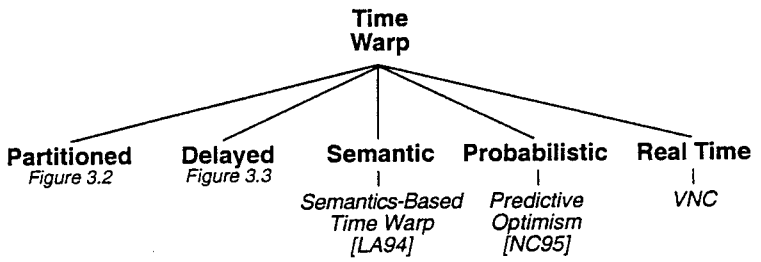


Figure 5.1. Time Warp Family of Algorithms

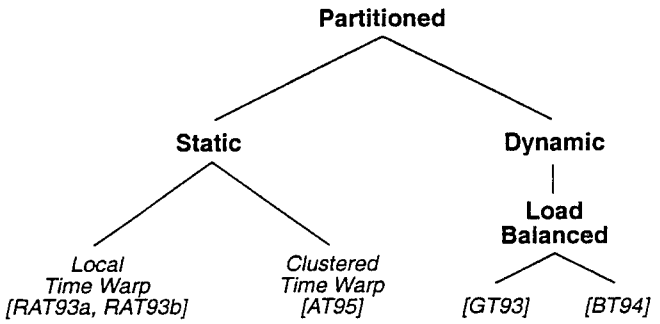


Figure 5.2. Partitioned Algorithms

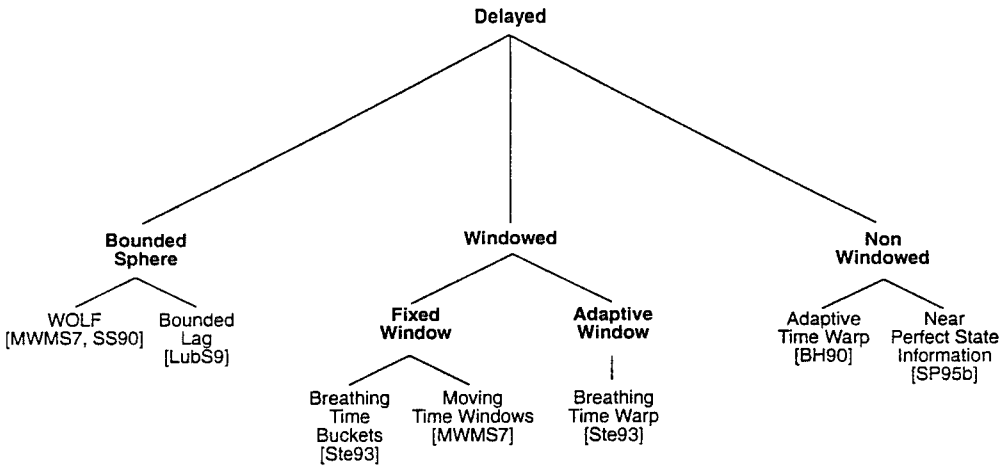


Figure 5.3 Delaying Algorithms

Table 5.1 Time Warp Family of Algorithms.

Class	Sub Class	Sub Class	Description	Example
Probabilistic			Predict msg arrival time.	Predictive Optimism ((Leong and Agrawal, 1994))
Semantic			Contents used to reduce rollback.	Semantics Based Time Warp ((Leong and Agrawal, 1994))
Partitioned			Inter-LP comm minimized.	
	Dynamic		LPs change mode dynamically.	
		Load Balanced	LPs migrate across hosts.	((Glazer and Tropper, 1993), and (Boukerche and Tropper, 1994))
	Static		LPs cannot change mode while executing.	Clustered Time Warp ((Avril and Tropper, 1995)) Local Time Warp ((Rajaei et al., 1993a, Rajaei et al., 1993b))
Delayed			Delays reduce rollback.	
	Windowed		Windows reduce rollback.	
		Adaptive Window	Windows adapt to reduce rollback.	Breathing Time Warp ((Steinman, 1993))
		Fixed Window	Window does not adapt.	Breathing Time Buckets ((Steinman, 1993)) Moving Time Windows ((Madisetti et al., 1987))
	Bounded		Based on earliest time inter-LP effects occur.	Bounded Lag ((Lubachevsky, 1989))
	Sphere			WOLF ((Madisetti et al., 1987, Sokol and Stucky, 1990))
	Non-Windowed		Non-Window method to reduce rollback.	Adaptive Time Warp ((Ball and Hoyt, 1990)) Near Perfect State Information ((Srinivisan and Paul F. Reynolds, 1995b))

$$\alpha(i) = \min_{j \in S\downarrow(i,B) \wedge j \neq i} \{d(j,i) + \min\{T(j), d(i,j) + T(i)\}\} \qquad (5.1)$$

The bounded sphere class of delay mechanisms attempts to calculate the maximum number of nodes that may need to be rolled back because they have processed messages out of order. For example, $S\downarrow(i, B)$ in (Lubachevsky et al., 1989) is the set of nodes affected by incoming messages from node i in time B, while $S\uparrow(i, B)$ is the set of nodes affected by outgoing messages from node i in time B. The downward pointing arrow in $S\downarrow(i, B)$ indicates incoming messages, while the upward pointing arrow in $S\uparrow(i, B)$ indicates outgoing messages.

Another approach to reducing rollback is to use all available semantic information within messages. For example, commutative sets of messages are messages that may be processed out-of-order yet they produce the same result. Finally, probabilistic methods attempt to predict certain characteristics of the optimistic simulation, usually based on its immediate past history, and take action to reduce rollback based on the predicted characteristic. It is insightful to review a few of these algorithms because they not only trace the development of Time Warp based algorithms but also because they illustrate the "state of the art" in preventing rollback, attempts at improving performance by constraining lookahead, partitioning of Logical Processes into sequential and parallel environments, and the use of semantic information. All of these techniques and more may be applied in the Active Virtual Network Management Prediction algorithm.

The Bounded Lag algorithm (Lubachevsky, 1989) for constraining rollback explicitly calculates, for each Logical Process, the earliest time that an event from another Logical Process may affect the current Logical Process's future. This calculation is done by first determining the

($S\downarrow(i, B)$), which is the set of nodes that a message may reach in time B. This depends on the minimum propagation delay of a message in simulation time from node i to node j, which is $d(i,j)$. Once $S\downarrow(i, B)$ is known, the earliest time that node i can be affected, $\alpha(i)$, is shown in Equation 5.1, where $T(i)$ is the minimum message receive time in node i's message receive queue. After processing all messages up to time $\alpha(i)$, all Logical Processes must synchronize.

The Bounded Lag algorithm is conservative because it synchronizes Logical Processes so that no message arrives out of order. The problem is that a minimum $d(i,j)$ must be known and specified before the simulation begins. A large $d(i,j)$ can negate any potential parallelism, because a large $d(i,j)$ implies a large $\alpha(i)$, which implies a longer time period between synchronizations. A filtered rollback extension to Bounded Lag is described in (Lubachevsky et al., 1989). Filtered Rollback allows $d(i,j)$ to be made arbitrarily small, which may possibly generate out of order messages. Thus the basic rollback mechanism described in (Jefferson and Sowizral, 1982) is required.

A thorough understanding of rollbacks and their containment is essential for Active Virtual Network Management Prediction. In (Lubachevsky et al., 1989), rollback cascades are analyzed under the assumption that the Filtered Rollback mechanism is used. Rollback activity is viewed as a tree; a single rollback may cause one or more rollbacks that branch out indefinitely. The analysis is based on a "survival number" of rollback tree branches. The survival number is the difference between the minimum propagation delay $d(j,i)$ and the delay in simulated time for an event at node i to affect the history at node, j $t(i,j)$. Each generation of a rollback caused by an immediately preceding node's rollback adds a positive or negative survival number. These rollbacks can be thought of as a tree whose leaves are rollbacks that have "died out." It is shown that it is possible to calculate upper bounds, namely, infinite or finite number of nodes in the rollback tree.

A probabilistic method is described in (Noble and Chamberlain, 1995). The concept in (Noble and Chamberlain, 1995) is that optimistic simulation mechanisms are making implicit predictions as to when the next message will arrive. A purely optimistic system assumes that if no message has arrived, then no message **will** arrive and computation continues. However, the immediate history of the simulation can be used to attempt to predict when the next message will arrive. This information can be used either for partitioning the location of the Logical Processes on processors or for delaying computation when a message is expected to arrive.

In (McAffer, 1990), a foundation is laid for unifying conservative and optimistic distributed simulation. Risk and aggressiveness are parameters that are explicitly set by the simulation user. Aggressiveness is the parameter controlling the amount of non-causality allowed in order to gain parallelism, and risk is the passing of such results through the simulation system. Both aggressiveness and risk are controlled via a windowing mechanism similar to the sliding lookahead window of the Active Virtual Network Management Prediction algorithm.

A unified framework for conservative and optimistic simulation called ADAPT is described in (Jha and Bagrodia, 1994). ADAPT allows the execution of a "sub-model" to dynamically change from a conservative to an optimistic simulation approach. This is accomplished by uniting conservative and optimistic methods with the same Global Control Mechanism. The mechanism in (Jha and Bagrodia, 1994) has introduced a useful degree of flexibility and described the mechanics for dynamically changing simulation approaches; (Jha and Bagrodia, 1994) does not quantify or discuss the optimal parameter settings for each approach.

A hierarchical method of partitioning Logical Processes is described in (Rajaei et al., 19993a, Rajaei et al., 19993b). The salient feature of this algorithm is to partition Logical Processes into clusters. The Logical Processes operate as in Time Warp. The individual clusters interact with each other in a manner similar to Logical Processes.

The CTW is described in (Avril and Tropper, 1995). The CTW mechanism was developed concurrently but independently of Active Virtual Network Management Prediction. This approach uses Time Warp between clusters of Logical Processes residing on different processors and a sequential algorithm within clusters. This is in some ways similar to the SLogical Process described later in Active Virtual Network Management Prediction. Since the partitioning of the simulation system into clusters is a salient feature of this algorithm, CTW has been categorized as a partitioned algorithm in Figure 5.2. One of the contributions of (Avril and Tropper, 1995) in CTW is an attempt to efficiently control a cluster of Logical Processes on a processor by means of the CE. The CE allows the Logical Processes to behave as individual Logical Processes as in the basic time warp algorithm or as a single collective Logical Process. The algorithm is an optimization method for the Active Virtual Network Management Prediction SLogical Processes.

Semantics Based Time Warp is described in (Leong and Agrawal, 1994). In this algorithm, the Logical Processes are viewed as abstract data type specifications. Messages sent to a Logical Process are viewed as function call arguments and messages received from Logical Processes are viewed as function return values. This allows data type properties such as commutativity to be used to reduce rollback. For example, if commutative messages arrive out-of-order, there is no need for a rollback since the results will be the same.

Another means of reducing rollback, in this case by decreasing the aggressiveness of Time Warp, is given in (Ball and Hoyt, 1990). This scheme involves voluntarily suspending a processor whose rollback rate is too frequent because it is out-pacing its neighbors. Active Virtual Network Management Prediction uses a fixed sliding window to control the rate of forward emulation progress; however, a mechanism based on those just mentioned could be investigated.

The NPSI Adaptive Synchronization Algorithms for Parallel Discrete Event Synchronization are discussed in (Srinivisian and Paul F. Reynolds, 1995a) and (Srinivisian and Paul F. Reynolds, 1995b). The adaptive algorithms use feedback from the simulation itself in order to adapt. Some of the deeper implications of these types of systems are discussed in Appendix 8. The NPSI system requires an overlay system to return feedback information to the Logical Processes. The NPSI Adaptive Synchronization Algorithm examines the system state (or an approximation of the state), calculates an error potential for future error, and then translates the error potential into a value that controls the amount of optimism.

Breathing Time Buckets described in (Steinman, 1992) is one of the simplest fixed window techniques. If there exists a minimum time interval between each event and the earliest event generated by that event (T), then the system runs in time cycles of duration T. All Logical Processes synchronize after each cycle. The problem with this approach is that T must exist and must be known ahead of time. Also, T should be large enough to allow a reasonable amount of parallelism, but not so large as to lose fidelity of the system results.

Breathing Time Warp (Steinman, 1993) attempts to overcome the problems with Breathing Time Buckets and Time Warp by combining the two mechanisms. The simulation mechanism operates in cycles that alternate between a Time Warp phase and a Breathing Time Buckets

phase. The reasoning for this mechanism is that messages close to GVT are less likely to cause a rollback, while messages with time-stamps far from GVT are more likely to cause rollback. Breathing Time Warp also introduces the *event horizon*, that is the earliest time of the next new event generated in the current cycle. A user-controlled parameter controls the number of messages that are allowed to be processed beyond GVT. Once this number of messages is generated in the Time Warp phase, the system switches to the Breathing Time Buckets phase. This phase continues to process messages, but does not send any new messages. Once the event horizon is crossed, processing switches back to the Time Warp phase. One can picture the system taking in a breath during the Time Warp phase and exhaling during the Breathing Time Buckets phase.

An attempt to reduce roll-backs is presented in an algorithm called WOLF (Mandisetti et al., 1987, Sokol and Stucky, 1990). This method attempts to maintain a sphere of influence around each rollback in order to limit its effects.

The Moving Time Window (Sokol et al., 1988, Sokol and Stucky, 1990) simulation algorithm is an interesting alternative to Time Warp. It controls the amount of aggressiveness in the system by means of a moving time window MTW. The trade-off in having no roll-backs in this algorithm is loss of fidelity in the simulation results. This could be considered as another method for implementing the Active Virtual Network Management Prediction algorithm.

An adaptive simulation application of Time Warp is presented in (Tinker and Agra, 1990). The idea presented in this paper is to use Time Warp to change the input parameters of a running simulation without having to restart the entire simulation. Also, it is suggested that events external to the simulation can be injected even after that event has been simulated.

Hybrid simulation and real system component models are discussed in (Bagrodia and Shen, 1991). The focus in (Bagrodia and Shen, 1991) is on PIPS Components of a performance specification for a distributed system that are implemented while the remainder of the system is simulated. More components are implemented and tested with the simulated system in an iterative manner until the entire distributed system is implemented. The PIPS system described in (Bagrodia and Shen, 1991) discusses using MAY or Maisie as a tool to accomplish the task, but does not explicitly discuss Time Warp.

5.5 REAL-TIME CONSTRAINTS IN OPTIMISTIC SIMULATION

The work in (Ghosh et al., 1993) provides some results relevant to Active Virtual Network Management Prediction. It is theorized that if a set of events is R-schedulable in a conservative simulation, and $R \geq \rho + c\ t + \sigma$ where ρ is the time to restore an state, c is the number of Processes, t is the time the simulation has been running, and σ is the real time required to save an state, then the set of events can run to completion without missing any deadline by an NFT Time Warp strategy with lazy cancellation. NFT Time Warp assumes that if an incorrect computation produces an incorrect event ($E_{i,T}$), then it must be the case that the correct computation also produces an event ($E_{i,T}$) with the same timestamp[1]. This result shows that conditions exist in a Time Warp algorithm that guarantee events are able to meet a given deadline. This is encouraging for the Active Virtual Network Management Prediction algorithm since clearly events must be completed before real-time reaches the predicted time of the event for the cached

results to be useful in Active Virtual Network Management Prediction. Finally, this author has not been the only one to consider the use of Time Warp to speed up a real-time process. In Tennenhouse and Bose, 1995), the idea of temporal decoupling is applied to a signal processing environment. Differences in granularity of the rate of execution are utilized to cache results before they are needed and to allocate resources more effectively.

This section has shown the results of research into improving Time Warp, especially in reducing rollback, as well as the limited results in applying Time Warp to real time systems. Improvements to Time Warp and the application to real time systems are both directly applicable to Active Virtual Network Management Prediction. Now consider the Active Virtual Network Management Prediction Algorithm in more detail.

5.6 PSEUDOCODE SPECIFICATION FOR AVNMP

The Active Virtual Network Management Prediction algorithm requires both Driving Processes and Logical Processes. Driving Processes predict events and inject virtual messages into the system. Logical Processes react to both real and virtual messages. The Active Virtual Network Management Prediction Algorithm for a driving process is shown in Figure 5.4. The operation of the driving process and the logical process repeat indefinitely. If the Driving Process has not exceeded its lookahead time, a new value Δ time units into the future is computed by the function $C(t)$ and the result is assigned to the message (M) and sent. The receive time, which is the time at which this message value is to be valid, is assigned to (M).

```
repeat
      if GVT ≤ t + Λ
            then                        /* not yet reached lookahead */
                  M.val ← C(LVT + Δ) /* compute next message
value */
                  M.rt ← LVT + Δ   /* set packet receive time */
                  Send(M)
End pseudo-code.
```

Figure 5.4. AVNMP Driving Process Algorithm.

The Active Virtual Network Management Prediction Algorithm for a Logical Process is specified in Figure 5.5. Note that inf is infimum. The next message from the Receive Queue is checked to determine whether the message is real. If the message is real, the next line in the pseudo-code retrieves the state that was saved closest to the receive time of the message and checks whether the values of the saved state are within tolerance. If the tolerance is exceeded, the process rolls back. Also, if the message is received in the past relative to this process's Local Virtual Time (LVT), the process rolls back as shown. The pre-computed and cached value in the State Queue is committed. Committing a value is an irreversible action because it cannot be rolled back once committed. If the process's Local Virtual Time has not exceeded its time as determined, then the virtual message is processed. The function $C_1(M, LVT)$ represents the computation of the new state. The function $C_1(M, LVT)$ returns the state value for this Logical Process and updates the LVT to the time at which that value is valid. The function C_2(M, LVT)

represents the computation of a new message value. The appendix to this chapter takes another look at the algorithm and begins to tie the algorithm to the code provided on the CD included with this report.

```
LVT ← 0
repeat
      M ← inf M.tr ∈ QR /* retreive message with lowest receive
time */
      CS(t).val ← C1(M, t)/* compute based on new message and
update current state */
      if (M.rt ≤ t) and (|SQ(t).val − Θ| > CS(t).val)
            then Rollback()          /* rollback if real message and
out-of-tolerance */
      if M.rt < LVT then Rollback()/* rollback if virtual message
and out-of-order */
      if M.rt ≤ t then Commit(SQ : SQ.t ≈ M.rt)
      if LVT + Λ ≤ GVT then    /* not looking far enough ahead
yet */
            SQ.val ← C1(M, LVT)/* update the state queue with
the predicted state */
            SQ.t ← LVT/* record the time of the predicted event
*/
            M.val ← C2(M, LVT)  /* generate any new messages
based on previous input message */
            M.rt ← LVT              /* set message receive time */
            QS ← M                  /* save copy in send queue */
            Send(M)
End pseudo-code.
```

Figure 5.5. AVNMP Logical Process Algorithm.

APPENDIX: AVNMP IMPLEMENTATION

This section discusses enhancing an existing Physical Process (PP)with AVNMP. The web-based tutorial in the CD included with this report provides a step-by-step explanation of how to enhance an application with AVNMP. This appendix provides a more detailed look at the internals of the AVNMP Driving and Logical Processes required in order to perform the enhancement. Notation for Communicating Sequential Processes (CSP) (Hoare, 1981) will serve as an intermediate description before looking at the details of the Java code. In CSP "X?Y" indicates process X will wait until a valid message is received into Y, and "X!Y" indicates X sends message Y. A guard statement is represented by "X→Y," which indicates that condition X must be satisfied in order for Y to be executed. Concurrent operation is indicated by "X||Y," which means that X operates in parallel with Y. A "*" preceding a statement indicates that the statement is repeated indefinitely. An alternative command, represented by "X||Y," indicates that either X or Y may be executed assuming any guards (conditions) that they may have are satisfied. If X and Y can both be executed, then only one is randomly chosen to execute. A familiar example used to illustrate CSP is shown in Algorithm 5.A.1. This is the bounded buffer problem in which a finite size buffer requests more items from a consumer only when the buffer will not run out of capacity.

Assume a working PP abstracted in Algorithm 5.A.2 where *S* and *D* represent the source and destination of real and virtual messages. Algorithm 5.A.3 shows the PP converted to a AVNMP LP operating with a monotonically increasing LVT. Note that the actual AVNMP Class function names are used; however, all the function arguments are not shown in order to simplify the explanation. Each function is described in more detail later. The input messages are queued in the Receive Queue as shown in Algorithm 5.A.3 by recvm(). In non-rollback operation the function getnextvm() returns the next valid message from the Receive Queue to be processed by the PP. When the PP has a message to be sent, the message is place in the State Queue by sendvm(). While a message is flowing through the process, the process saves its state periodically. Normal operation of the AVNMP as just described may be interrupted by a rollback. If recvm() returns a non-zero value, then either an out-of-order or out-of-tolerance message has been received. In order to perform the rollback, getstate() is called to return the proper state to which the process must rollback. It is the application's responsibility to ensure that the data returned from getstate() properly restores the process state. Anti-messages are sent by repeatedly calling rbq() until rbq() returns a null value. With each call of rbq(), an anti-message is returned which is sent to the destination of the original message.

5.A.1. AVNMP Class Implementation

Figure 5.A.4 lists a selection of the main classes and their primary purpose in the AVNMP system. A complete list of the classes and their descriptions can be found on the CD in README.html. The classes are the primary classes for understanding the operation of the AVNMP system.

```
X::
buffer:(0..9) portion;
in,out:integer; in := 0; out := 0;
*[in < out + 10; producer?buffer(in mod 10) →
  in := in + 1;
| | out < in; consumer?more() →
  consumer!buffer(out mod 10); out := out + 1;
]
```

Figure 5.A.1. A CSP Example

```
PP::
*[S?input;
  output := process(input);
  D!output]
```

Figure 5.A.2. A Physical Process

```
PP::
*[S?input;
[recvm(input)!=0 → getstate();
*[rbq()!=NULL → S!AvnmpDriverRb;D!rbq()]| |
[recvm(input)==0 →
savestate();
input := getnextvm();
output := process(input);
sendvm(output);
D!output]
]
]
```

Figure 5.A.3. The Logical Process

avnmp.java.lp.AvnmpRecQueue Receive a message, determine whether virtual or real, rollback

avnmp.java.lp.AvnmpSndQueue Send a virtual message, save a copy

avnmp.java.lp.AvnmpQueue All queue related functions

avnmp.java.lp.AvnmpLP Roll back to given time

avnmp.java.lp.AvnmpStateQueue Save previous state

avnmp.java.lp.AvnmpTime Local virtual time maintenance functions

avnmp.java.lp.AvnmpPacket The virtual message

avnmp.java.dp.Driver The driving process

avnmp.java.pp.PP The physical process

avnmp.java.pp.PayLoad The real message

Figure 5.A.4. AVNMP Class Files

Predict() → output → getvm()

Figure 5.A.5. The Driving Process

input → process → output

Figure 5.A.6. The Logical Process

5.A.2 AVNMP Logical Process Implementation

This class implements the AVNMP logical process. The general idea is to have a working process modified in Figure 5.A.6. Figure 5.A.7 shows the "normal" operation, while Figure 5.A.8 shows the operation of the process when a rollback occurs.

input → getvm(); getnext() →

$$\begin{cases} \text{process} \rightarrow \text{sendvm()} \rightarrow \text{output} \\ \text{savestate()} \end{cases}$$

Figure 5.A.7. AVNMP Normal Operation

if(getvm() ≠ 0)getstate() → process()→ rbq → output

Figure 5.A.8. AVNMP Rollback Operation

Notes

[1]This simplification makes the analysis in (Ghosh et al., 1993) tractable. This assumption also greatly simplifies the analysis of Active Virtual Network Management Prediction. The Active Virtual Network Management Prediction algorithm is simplified because the state verification component of Active Virtual Network Management Prediction requires that saved states be compared with the real-time state of the process. This is done easily under the assumption that the T (timestamp) values of the two events $E_{i,Tv}$ and $E_{i,Tr}$ are the same.

6

ALGORITHM ANALYSIS

The purpose of this section is to analyze the performance of the Active Virtual Network Management Prediction Algorithm. As discussed in detail in previous chapters, current network management is centralized, as shown in Figure 6.1. On the other hand, the Active Virtual Network Management Prediction Algorithm distributes management. Figure 6.2 shows an active network testbed consisting of three active nodes. The active nodes are labeled AN-1, AN-4, and AN-5, and the links are labeled L-1, L-2, L-3, and L-4. One of the goals of this section is to investigate the benefits of the new active network based distributed management model. The characteristics of the Active Virtual Network Management Prediction Algorithm analyzed in this section are speedup, lookahead, accuracy, and overhead. Speedup is the ratio of the time required to perform an operation without the Active Virtual Network Management Prediction Algorithm to the time required with the Active Virtual Network Management Prediction Algorithm. Lookahead is the distance into the future that the system can predict events. Accuracy is related to the rate of convergence between the predicted and actual values. Bandwidth overhead is the ratio of the amount of additional bandwidth required by the Active Virtual Network Management Prediction Algorithm system to the amount of bandwidth required without the Active Virtual Network Management Prediction Algorithm system, and processing overhead is the reduction in network capacity due to active packet execution.

Because the Logical Processes of the Active Virtual Network Management Prediction Algorithm system are asynchronous, they can take maximum advantage of parallelism. However, messages among processes may arrive at a destination process out-of-order as illustrated in Figure 3.2. As shown in Figure 6.2, a virtual network representing the actual network can be viewed as overlaying the actual network for analytical purposes.

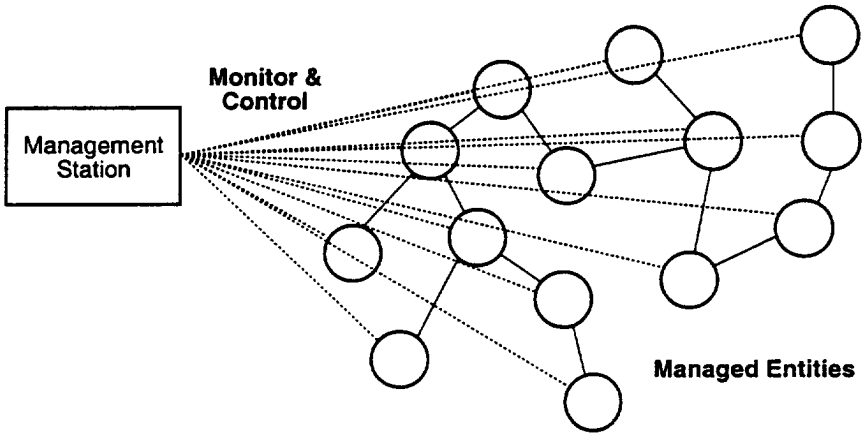


Figure 6.1. Centralized Network Management Model.

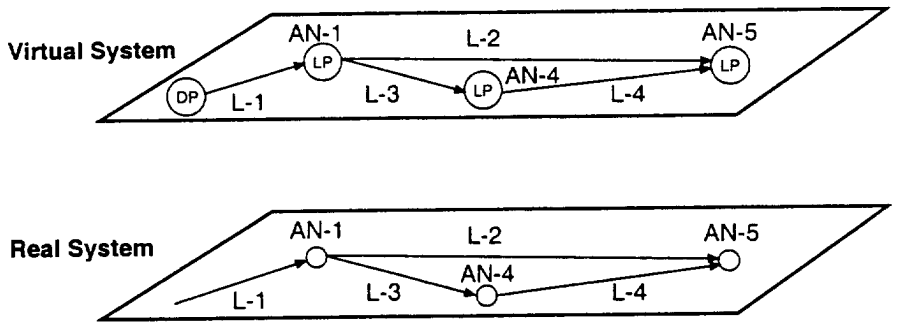


Figure 6.2. AVNMP as a Virtual Overlay for Network Management.

Virtual messages may not arrive at a logical process in the order of Receive Time for several reasons. The first reason is that in an optimistic parallel model, virtual messages are executed as soon as they arrive at a logical process. Thus, in an optimistic simulation of a complex network, virtual messages do not block or delay to enforce causality. This leads to a possibility of messages arriving out-of-order even if the virtual message links have no transmission delay. Petri-Net theory is used to analyze this type of out-of-order message arrival. Petri-Nets are commonly used for synchronization analysis. In Petri-Nets, "places," usually shown as circles, represent entities such as producers, consumers, or buffers, and "transitions," shown as squares, allow "tokens," shown as dots, to move from one place to another. In this analysis, tokens represent the Active Virtual Network Management Prediction Algorithm messages and Petri-Net places represent the Active Virtual Network Management Prediction Algorithm Logical Processes. Characteristics of Petri-Nets are used to determine the likelihood of out-of-order messages.

Another source of out-of-order virtual message arrival at a logical process is due to congestion or queuing delay. The actual messages in Figure 6.2 can cause the virtual messages along a particular link to arrive later than virtual messages arriving along another link to the same logical process. However, the Active Virtual Network Management Prediction Algorithm can predict that the congestion and thus the late virtual message arrival are likely to occur. The accuracy of this prediction depends in part upon the acceptable tolerance setting of the prediction. The relationship of the tolerance to prediction accuracy and late virtual message arrival likelihood are discussed later in this chapter. If a Logical Process predicts congestion along an input link, then the Logical Process delays itself until some virtual message arrives along that link, thus avoiding a possible rollback. The likelihood of the occurrence of out-of-order messages and out-of-tolerance messages is required by an equation that is developed in this chapter to describe the speedup of the Active Virtual Network Management Prediction Algorithm. After analyzing the speedup and lookahead, the prediction accuracy and overhead are analyzed. This chapter considers enhancements and optimizations such as implementing multiple future events, eliminating the calculation, and elimination of real messages when they are not required.

Performance analysis of the Active Virtual Network Management Prediction algorithm must take into account accuracy as well as distance into the future that predictions are made. An

inaccurate prediction can result in committed resources that are never used and thus wasted, or in not committing enough resources when needed, thus causing a delay. Unused resource allocation must be minimized. Active Virtual Network Management Prediction does not require permanent over-allocation of resources; however, the Active Virtual Network Management Prediction algorithm may make a false prediction that *temporarily* establishes resources that may never be used. An Active Virtual Network Management Prediction system whose tolerances are reduced in order to produce more accurate results will have fewer unused allocated resources; however, the tradeoff is a reduction in speedup.

$$U_{AVNMP} = \eta \Phi_s - \alpha \Phi_w - \beta \Phi_b \tag{6.1}$$

Equation (6.1) quantifies the advantage of using Active Virtual Network Management Prediction where η is the expected speedup using Active Virtual Network Management Prediction over a non-Active Virtual Network Management Prediction system, Φ_s is the marginal utility function of the configuration speed, and α is the expected quantity of wasted resources other than overhead, and Φ_w is the marginal utility function of the allocated but unused resource. An example of a resource that may be temporarily wasted due to prediction error is a Virtual Circuit in a mobile wireless network that may be established temporarily and never used. The expected overhead is represented by β and Φ_b is the marginal utility function of bandwidth and processing.

The marginal utility functions Φ_s, Φ_w and Φ_b are subjective functions that describe the value of a particular service to the user. The functions Φ_s, Φ_w and Φ_b may be determined by monetary considerations and user perceptions. The following sections develop propositions that describe the behavior of the Active Virtual Network Management Prediction algorithm and from these propositions equations for η, α and β are defined.

6.1 PETRI-NET ANALYSIS FOR THE AVNMP ALGORITHM

In this section the probability of message arrival at a Logical Process is determined, the expected proportion of messages ($E[X]$) and the probability of rollback due to messages (P_{oo}) is analyzed, and a new and simpler approach to analyzing Time-Warp based algorithms in general and the Active Virtual Network Management Prediction Algorithm in particular is developed. The contribution is unique because most current optimistic analysis has been explicitly time-based, yielding limited results except for very specific cases. The approach is topological; timing is implicit rather than explicit. A C/E is used in this analysis because it is the simplest form of a Petri-Net that is ideal for studying the Active Virtual Network Management Prediction Algorithm synchronization behavior.

A C/E network consists of condition and transition elements that contain tokens. Tokens reside in condition elements. When all condition elements leading to a transition element contain a token, several changes take place in the network. First, the tokens are removed from the conditions that triggered the event, the event occurs, and finally tokens are placed in all condition outputs from the transition that was triggered. Multiple tokens in a condition and the uniqueness of the tokens is irrelevant in a C/E Net. In this analysis, tokens represent virtual messages,

conditions represent processes, and transitions represent interconnections. The notation from (Reisig, 1985) is used: $\Sigma = (B,E;F,C)$ is a C/E Net where B is the set of conditions, E is the set of transitions, and $F \subseteq (B \times E) \cup (E \times B)$ where $\cup$ is union and $\times$ is the cross product of all conditions and transitions. A marking is the set of conditions containing tokens at any given time during C/E operation and C is the set of all possible sets of markings of Σ. The input conditions to a transition are written as "pre–e" and the output conditions are written as "post–e." Let $c \subseteq C$, then a transition $e \in E$ is triggered when pre–$e \subseteq (c \subseteq B)$ and post–$e \cap c = \varnothing$. If c is the current set of enabled conditions and after the next transition (e) the new set of enabled conditions is c', then this is represented more compactly as $c[e\rangle c'$. C/E networks provide insight into liveness, isomorphism, reachability, a method for determining synchronous behavior, and behavior based on the topology of the Active Virtual Network Management Prediction Algorithm Logical Process communication. Every Finite State Machine has an equivalent C/E Net (Peterson, 1981, p. 42).

Some common terminology and concepts are defined next that are needed for a topological analysis of the Active Virtual Network Management Prediction Algorithm. These terms and concepts are introduced in a brief manner and build upon one another. Their relationship with the Active Virtual Network Management Prediction Algorithm will soon be made clear. The following notation is used: "$\neg$" means "logical not," "$\exists$" means "there exists," ``$\forall$" means "for each," "$\wedge$" means "logical and,", "$\vee$" means "logical or," "$\in$" means that an element is a member of a set, "$\equiv$" means "defined as," and "$\rightarrow$" defines a mapping or function. Also, $a \prec b$ indicates an ordering between two elements, a and b, such that a precedes b in some relation. "$\Rightarrow$" means "logical implication" and "$\leftrightarrow$" means "logical equivalence."

A region of a particular similarity relation $(\cdot)$ of $B \subseteq A$ means that $\forall a,b \in B : a \cdot b$ and $\forall a \in A : a \notin B \Rightarrow \exists b \in B : \neg (a \cdot b)$. This means that the relation is "full" on B and B is a maximal subset on which the relation is full. In other words, a graph of the relation $(\cdot)$ would show B as the largest fully connected subset of nodes in A.

Let "$\underline{\text{li}}$" represent a such that $a \ \underline{\text{li}}\ b \leftrightarrow (a \prec b) \vee (b \prec a) \vee (a \equiv b)$. Let "$\underline{\text{co}}$" represent a concurrent ordering $a \ \underline{\text{co}}\ b \leftrightarrow \neg (a \ \underline{\text{li}}\ b) \vee (a \equiv b)$. Figure 6.3 illustrates a region of $\underline{\text{co}}$ that contains $\{a, c\}$ and of $\underline{\text{li}}$ that contains $\{a, b, d\}$ where $\{a, b, c, d\}$ represents Logical Processes and the relation is "sends a message to." Trivially, if every process in the Active Virtual Network Management Prediction Algorithm system is a region of $\underline{\text{li}}$ then regardless of how many driving processes there are, no synchronization is necessary since there exist no processes. If no synchronization is needed, then virtual messages cannot arrive out-of-order; thus no rollback will occur.

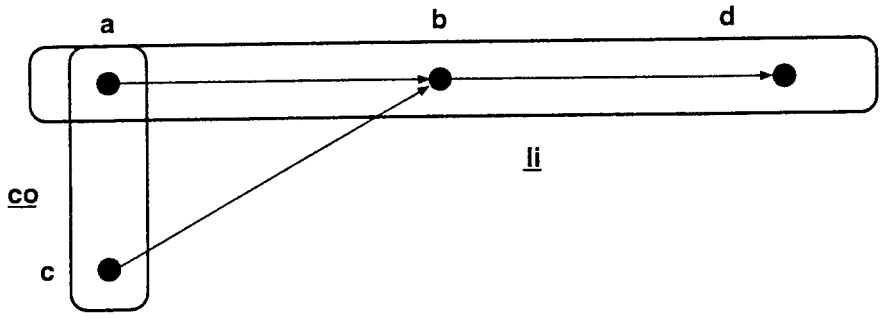


Figure 6.3. Demonstration of li and co.

Let D be the set of driving processes and R be the set of the remaining processes in the Active Virtual Network Management Prediction Algorithm system. Then $D \prec R \leftrightarrow \forall d \in D\ \forall r \in R : (d \prec r) \vee (d\ \underline{\text{co}}\ r)$. In order for the virtual messages that originate from D to be used, $D \prec R$ where R are the remaining non-driving processes. This is again assumed to be "sends a message to."

In the remaining definitions, let A, B, and C be arbitrary sets where $B \subseteq A$ used for defining additional operators. Let $B \preceq C \equiv \forall b \in B\ \forall c \in C: b \prec c \vee b\ \underline{\text{co}}\ c$. Let $B^- \equiv \{\ a \in A \mid \{a\} \preceq B\ \}$ and $B^+ \equiv \{\ a \in A \mid B \preceq \{a\}\ \}$ where | means "such that." Also, let $[\ \overline{B}\ ] \equiv \{b \in B \mid \forall b' \in B: (b\ \underline{\text{co}}\ b') \vee (b \prec b')\ \}$ and $\underline{B} \equiv \{\ b \in B \mid \forall b' \in B: (b\ \underline{\text{co}}\ b') \vee (b' \prec b)\ \}$. This is illustrated in Figure 6.4,where all nodes are in the set A and B is the set of nodes that lie within the circle. B^- is the set $\{a,b,c,d,f\}$ and $[\ \overline{B}\ ]$ is the set $\{b\}$.

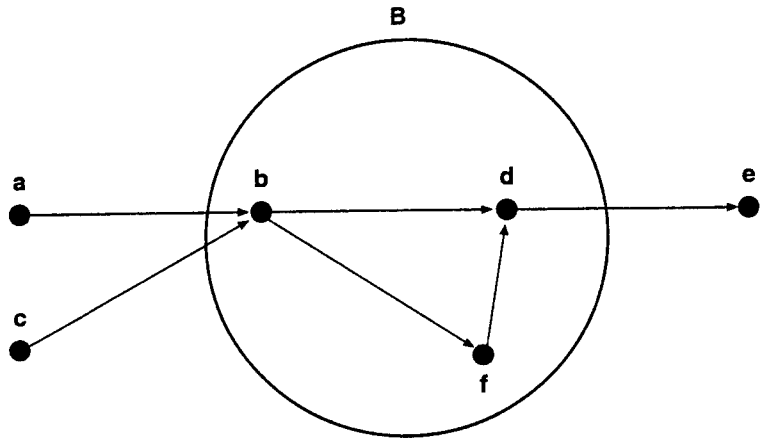


Figure 6.4. Illustration of B^- and $[\ \overline{B}]$.

An occurrence network (K) is a network that is related to the operation of a particular network (Σ). The occurrence network (K) begins as an empty C/E network; conditions and

events are added to K as Σ operates. K represents a particular sample of operation of Σ. There can be multiple events in Σ that are capable of firing, but only one event is chosen to fire; thus it is possible that a particular Σ will not always generate the same occurrence net (K) each time it operates. Note that K has some special properties. The condition elements of K have one and only one transition, because only one token in Σ may fire from a given condition. Also, K is cycle free because K represents the operation of Σ.

A few more definitions are required before the relation described above between K and Σ can be formally defined. This relationship is called a Petri-Net process. Once the Petri-Net process is defined, a measure for the "out-of-orderness" of messages can be developed based on synchronic distance. A *line* is a subset that is a region of $\underline{\text{li}}$ and a *cut* is a subset that is a region of $\underline{\text{co}}$. A slice ("$\underline{\text{sl}}$") is a cut of an occurrence network (K) containing condition elements, and $\underline{\text{sl}}$ (K) is the set of *all* slices of K. The of $\underline{\text{co}}$ shown in Figure 6.3 illustrates a cut where nodes represent conditions and the relation defines an event from one condition to another in a C/E Network.

A formal definition of the relation between an occurrence net and a C/E net is given by a Petri-Net process. A Petri-Net process (p) is defined as a mapping from a network K to a C/E Network Σ, $p : K \rightarrow \Sigma$, such that each slice of K is mapped injectively (one-to-one) into a marking and ($p(\text{pre}{-}t) = \text{pre}{-}p(t)) \wedge (p(\text{post}{-}t) = \text{post}{-}p(t))$. Also note that p^{-1} is used to indicate the inverse mapping of p. Think of K as a particular sample of the operation of a C/E Network. A C/E Network can generate multiple processes. Another useful characteristic is whether a network is K-dense. A network is K-dense if and only if every $\underline{\text{sl}}$ (K) has a non-empty intersection with every region of $\underline{\text{li}}$ in K. This means that each intersects every sequential path of operation.

All of the preceding definitions have been leading towards the development of a measure for the "out-of-orderness" of messages that does not rely on explicit time values or distributions. In the following explanation, a measure is developed for the synchronization between events. Consider D_1 and D_2 that are two slices of K and M is a set of events in a C/E Network. $\mu(M, D_1, D_2)$ is defined as $| M \cap D^+_2 \cap D^-_1 | - | M \cap D^-_1 \cap D^+_2 |$. Note that $\mu(M, D_1, D_2) = -\mu(M, D_2, D_1)$. Thus $\mu(M, D_1, D_2)$ is a number that defines the number of events between two specific slices of a net.

Let $(p{:}K \rightarrow \Sigma) \in \pi_\Sigma$ where π_Σ is the set of all finite processes of Σ. A term known as "variance" is defined that describes the number of events across all slices of a net (K). The variance of T_Σ is $\nu(p, T_1, T_2) \equiv \max\{\mu(p^{-1}(T_1), D_1, D_2) - \mu(p^{-1}(T_2), D_1, D_2) \mid D_1, D_2 \in \underline{\text{sl}}\ (K)\}$. Also, note that $\nu(p, T_1, T_2) = \nu(p, T_2, T_1)$ where and $T_1, T_2 \subseteq T_\Sigma$. This defines a measure of the number of events across all slices of a net (K).

The synchronic distance ($\sigma(T_1, T_2) = \sup\{\ \nu(p, T_1, T_2) \mid p \in \pi_\Sigma\ \}$) is the supremum of the variance in all finite processes. This defines the measure of "out-of-orderness" across all possible K. By determining the synchronic distance, a measure for the likelihood of rollback in the Active Virtual Network Management Prediction Algorithm can be defined that is dependent on the topology and is **independent of time**. Further details on syn chronic distance and the relation of synchronic distance to other measures of synchrony can be found in (Voss et al, 1987). A more intuitive method for calculating the synchronic distance is to insert a virtual condition into the C/E net. This condition has no meaning or effect on operation. The condition is allowed to hold multiple tokens and begins with enough tokens so that it can emit a token whenever a condition connected to its output transition is ready to fire. The virtual condition has inputs from all

members of T_1 and output transitions of all members of T_2. The synchronic distance is the maximum variation in the number of tokens in the virtual condition. The greater the possibility of rollback, the larger the value of $\sigma(T_1, T_2)$. A simple example in Figure 6.5 intuitively illustrates what the synchronic distance means. Using the virtual condition method to calculate the synchronic distance between $\{a, b\}$ and $\{c, d\}$ in the upper C/E Network, the synchronic distance is found to be two. By adding two more conditions and another transition to the C/E network, the synchronic distance of the lower C/E Network shown in Figure 6.5 is one. The larger the value of $\sigma(T_1, T_2)$, the less synchronized the events in sets T_1 and T_2. If these events indicate message transmission, then the less synchronized the events, the greater the likelihood that the messages based on events T_1 and T_2 are out-of-order. This allows the likelihood of message arrival at a Logical Process to be determined based on the inherent synchronization of a system. However, a completely synchronized system does not gain the full potential provided by optimistic parallel synchronization.

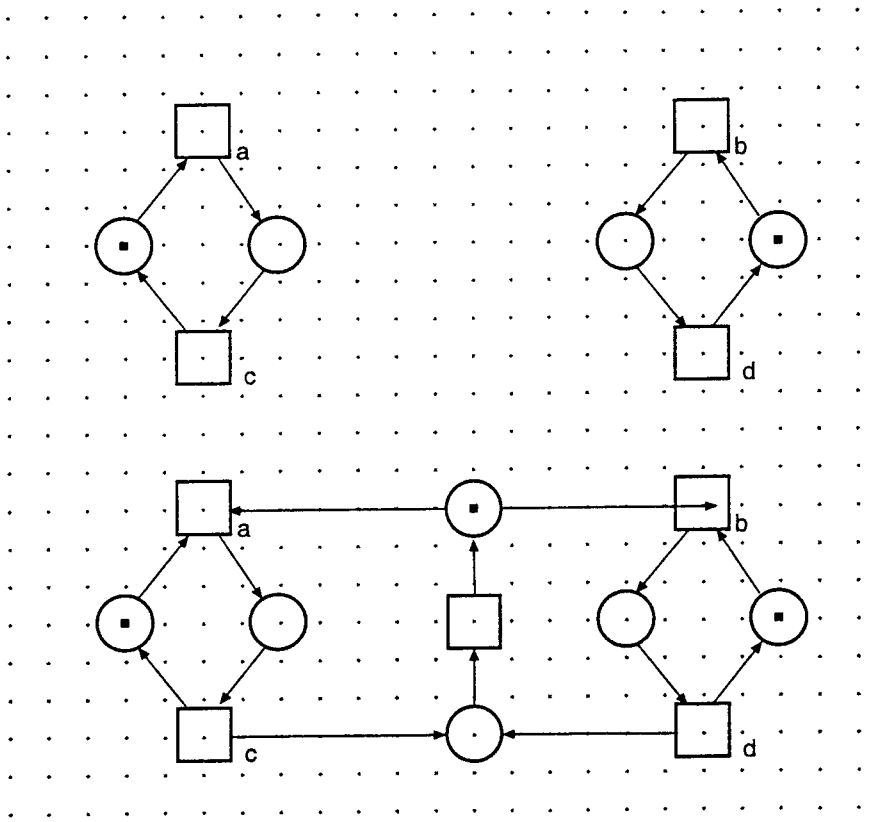


Figure 6.5. Example of Synchronic Distance.

A P/T Network is similar to a C/E network except that a P/T Net allows multiple tokens in a place and multiple tokens may be required to cause a transition to fire. Places are defined by the set S and transitions by the set T. The operation of a network can be described by a matrix. The rows of the matrix represent places and the columns represent transitions. The last column of the

matrix represents the current number of tokens in a place. Each element of the matrix contains the number of tokens that either leave (negative integer) or enter (positive integer) a place when the transition fires. When a transition fires, the column corresponding to the transition is added to the last column of the matrix. The last column of the matrix changes as the number of nodes in each place change. The matrix representation of a P/T Network is shown in Matrix 6.2, where $LP_n \in S$, $c_n \in T$ and $w_{i,j}$ is the weight or number of tokens required by link j to fire or the number of tokens generated by place i. Note that LP_n and c_n bordering Matrix 6.2 indicate labels for rows and columns. Note also that there exists a duality between places and transitions such that places and transitions can be interchanged (Peterson, 1981, p. 13). P/T networks can be extended from the state representation of C/E networks to examine problems involving quantities of elements in a system, such as producer/consumer problems. The places in this analysis are analogous to Logical Processes because they produce and consume both real and virtual messages. Transitions in this analysis are analogous to connections between Logical Processes, and tokens to messages. The weight, or number of tokens, is $-w_{i,j}$ for outgoing tokens and $w_{i,j}$ for incoming tokens. The current marking, or expected value of the number of tokens held in each place, is given in column vector $\vec{m}_N$. A transition to the next state is determined by $\vec{m}_{N+1} = \vec{m}_N + \vec{c}_i$ where $\vec{c}_i$ is the column vector of the transition that fired and N is the current matrix index.

$$M_N = \begin{array}{c} \\ LP_1 \\ LP_2 \\ LP_3 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & \cdots & m_N \end{array} \\ \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & n_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & n_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & n_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \end{array} \tag{6.2}$$

A global synchronic distance value is shown in Equation 6.3 where T consists of the set of all transitions. The global synchronic distance is used to define a normalized measure. The global measure is the maximum in a P/T network and $\sigma_n(I_1,I_2) \in [0,1]$ is a normalized value shown in Equation 6.4 where $\{I_n\}$ is a set of all incoming transitions to a particular place. A probability of being within tolerance is defined in vector $\vec{p}$ shown in Matrix 6.5. Each LP_i along the side of Matrix 6.5 indicates a LP and the $1 - P_{ot}$ along the top of Matrix 6.5 indicates p_i values that are the individual probabilities that the tolerance is not exceeded. The probability of out-of-tolerance rollback is discussed in more detail in Section 4.1. Let (LP_i, c_j) be the transition from LP_i across connection c_j. After each transition of M_N from (LP_i, c_j), the next value of n_i that is the element in the i^{th} row of the last column of M_N is $\sigma_n(I_1,I_2)\, p_i^{n_i}$.

$$GSV = \max_{f_1, f_2 \subset T} \{\sigma(f_1, f_2)\} \tag{6.3}$$

$$\sigma_n(I_1, I_2) = 1.0 - \frac{\sigma(I_1, I_2)}{GSV} \tag{6.4}$$

$$\vec{p} = \begin{matrix} \\ LP_1 \\ LP_2 \\ LP_3 \\ \vdots \end{matrix} \begin{matrix} 1-P_{ot} \\ \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \end{pmatrix} \end{matrix} \tag{6.5}$$

It is possible for the synchronic distance to be infinite. One way to avoid an infinite synchronic distance is to use weighted synchronic distances. A brief overview of weighted synchronic distances is given in this section. (Andre et al., 1979) introduces capacity Petri-Nets (CPN). Capacity Petri-Nets have place values that hold a multiple number of tokens but with a maximum capacity. A transition cannot fire if it results in a place exceeding its pre-specified capacity. The capacity has an effect upon the synchronic distance. A place between two sets of transitions enforces a synchronic distance equal to the capacity of that place. This is directly apparent because an intuitive method for determining synchronic distance is to add a place with inputs from one set of transitions and outputs to the other set. The synchronic distance is the maximum number of tokens that can appear in the place given all possible firing sequences. In (Goltz and Reisig, 1982) weighted synchronic distances are introduced. Synchronic distance as originally defined can in many instances become infinite even though it is apparent a regular structure exists in the Petri-Net. In (Goltz, 1987) the concept of synchronic distance is introduced along with weighted synchronic distance. (Silva and Colom, 1988) builds on the relationship between synchronic invariants and linear programming. In (Silva and Murata, 1992) measures related to synchronic distances are discussed, namely bounded-fairness. Bounded-fair relations are concerned with the number of times a transition fires before another transition can fire. Marked graphs form a subset of Petri-Nets. The synchronic distance matrix of a marked graph holds the synchronic distances between every vertex in the marked graph. In (Mikami et al., 1993, Tamura and Abe, 1996) necessary and sufficient conditions are given for a matrix to represent a marked graph.

As $p_i^{n_i}$ approaches zero, the likelihood of an out-of-tolerance induced rollback increases. As $\sigma_n(I_1,I_2)\ p_i^{n_i}$ becomes very small, the likelihood of a rollback increases either due to a violation of causality or an out-of-tolerance state value. Synchronic distance is a metric and furthermore the $\sigma_n(I_1,I_2)$ value is treated as a probability because it has the axiomatic properties of a probability. The axiomatic properties are that $\sigma_n(I_1,I_2)$ assigns a number greater than or equal to zero to each synchronic value, $\sigma_n(I_1,I_2)$ has the value of one when messages are always in order, and $\sigma_n(A) + \sigma_n(B) = \sigma_n(A \cup B)$, where A and B are mutually exclusive sets of transitions.

A brief example is shown in Figure 6.6. The initial state shown in Figure 6.6 is represented in Matrix 6.6. The Global Synchronic Value of this network is four. The tolerance vector for this example is shown in Vector 6.7. Consider transition a shown in Figure 6.6; it is enabled since tokens are available in all of its inputs. The element in the $\vec{p}$ column vector shown in Vector 6.7 is taken to the power of the corresponding elements of the column vector $\vec{a}$ in Matrix 6.6 that are greater than zero ($p_i^{n_i}$). This is the probability that all messages passing through transition a arrive within tolerance. All columns of rows of $\vec{a}$ that are greater than zero that have greater than zero values form the input set ($\{I_n\}$) for $\sigma_n(I_1,I_2)$. Since transition a has only one input, $\sigma_n(\{a\})$ is

one. When transition a fires, column vector $\vec{a}$ is added to column vector $\vec{m}_0$ to generate a new vector $\vec{m}_1$. Matrix 6.8 results after transition a fires. Continuing in this manner, Matrix shows the result after transition b fires. Since $\sigma_n(\{b\})$ is one, row LP_4 of $\vec{m}_2$ is 0.3.

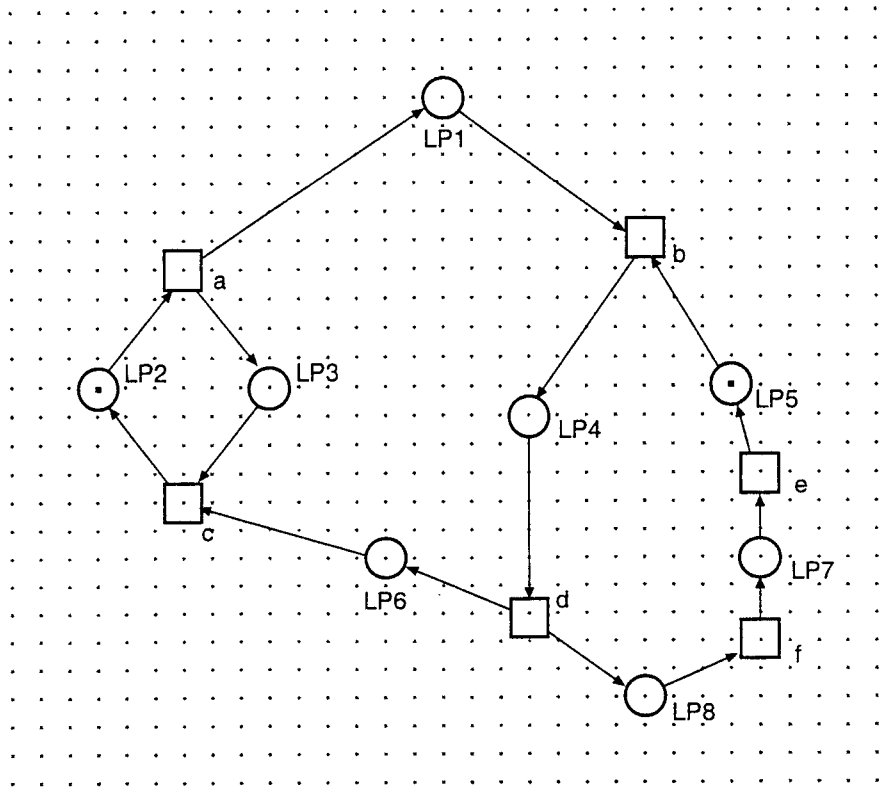


Figure 6.6. Example of P_{∞} Analysis.

$$M_0 = \begin{array}{c} \\ LP_1 \\ LP_2 \\ LP_3 \\ LP_4 \\ LP_5 \\ LP_6 \\ LP_7 \\ LP_8 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & m_0 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{pmatrix} \end{array} \tag{6.6}$$

$$\vec{p} = \begin{array}{c} \\ LP_1 \\ LP_2 \\ LP_3 \\ LP_4 \\ LP_5 \\ LP_6 \\ LP_7 \\ LP_8 \end{array} \begin{array}{c} 1-P_{ot} \\ \begin{pmatrix} 0.7 \\ 0.2 \\ 0.3 \\ 0.4 \\ 0.6 \\ 0.4 \\ 0.2 \\ 0.1 \end{pmatrix} \end{array} \tag{6.7}$$

$$M_1 = \begin{array}{c} \\ LP_1 \\ LP_2 \\ LP_3 \\ LP_4 \\ LP_5 \\ LP_6 \\ LP_7 \\ LP_8 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & m_1 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 1 & 0 & 0.7 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0.3 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{pmatrix} \end{array} \tag{6.8}$$

$$M_2 = \begin{array}{c} \\ LP_1 \\ LP_2 \\ LP_3 \\ LP_4 \\ LP_5 \\ LP_6 \\ LP_7 \\ LP_8 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & m_2 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0.3 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0.3 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{pmatrix} \end{array} \tag{6.9}$$

The analysis presented in this section reduces the time and topological complexities characteristic of more explicit time analysis methods to simpler and more insightful matrix manipulations. The method presented is used in the following section to determine the probability of rollback due to messages, $P_{oo} = 1 - \sigma_n(I_1,I_2)$.

Also, the worst case proportion of out-of-order messages (X) is calculated as follows. The ($\sigma(I_1,I_2)$) is a measure of the maximum difference in the rate of firing among transitions. The maximum possible value of $\sigma(I_1,I_2)$ that can occur is the rate of the slowest firing transition in sets I_1,I_2. Equation 6.10 shows the relationship between $E[X]$ and the rate at which transition I fires.

$$E[X] \leq \min_{\{Transition \in I_1, I_2\}} \{rate\,(Transition)\} \tag{6.10}$$

6.1.1 T-Invariants

An alternative analysis of the likelihood of out-of-order message arrival at a logical process and quantitative synchronization analysis can be derived from invariants in the Petri-Net representation of the Active Virtual Network Management Predication system. T-invariants are transition vectors whose values are the number of times each transition fires in order to obtain the same marking. P-variants are sets of places that always contain the same number of tokens. In (J. Martinez and Silva, 1982) an algorithm is given to determine all the invariants of generalized and capacity Petri-Nets.

Figure 6.7 provides an example of a sample active network not yet enhanced with Active Virtual Network Management Prediction. The active network nodes are illustrated as well as the end-systems and the active packet. A Petri-Net representation of this network is derived as follows. The logical processes are injected into the network and persist at the active nodes to be AVNMP-enhanced as shown in Figure 6.8. The Active Virtual Network Management Prediction system was developed using the Magician (Kulkarni et al., 1998) execution environment; the driving processes, logical processes, and virtual messages are implemented as active packets. The driving processes reside at the edge of the region to be enhanced with AVNMP. Virtual messages now enter the picture.

This analysis considers the number of transition firings as the local virtual time. Thus, the logical processes are transitions. The token represents an update to the local virtual time of the logical process driven by the receive time of a virtual message that has been processed. Thus, in the transition from Figure 6.8 to 6.9, the driving processes become token generators and logical processes become Petri-Net transitions. The active packets that were virtual messages become Petri-Net tokens.

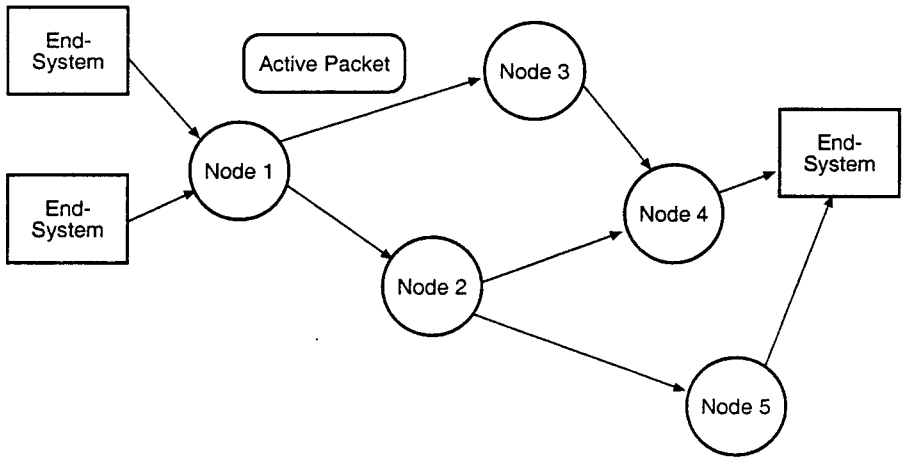


Figure 6.7. Active Network Configuration for T-Invariant Analysis.

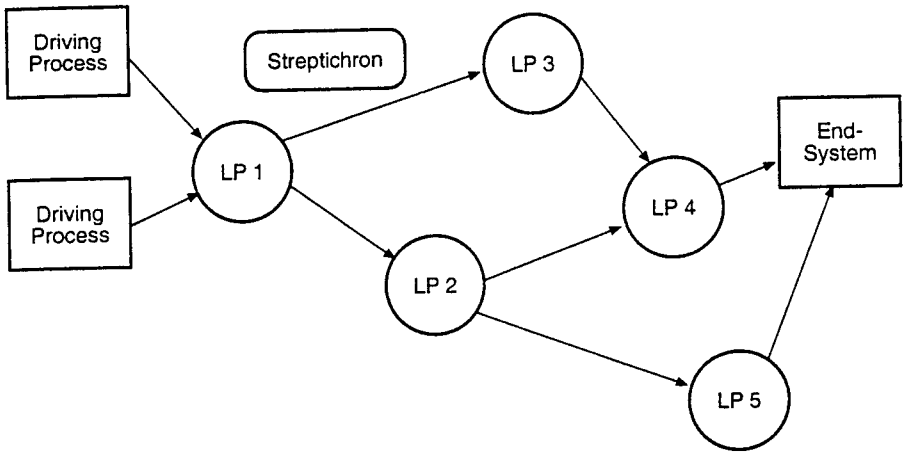


Figure 6.8. Active Network with AVNMP for T-Invariant Analysis.

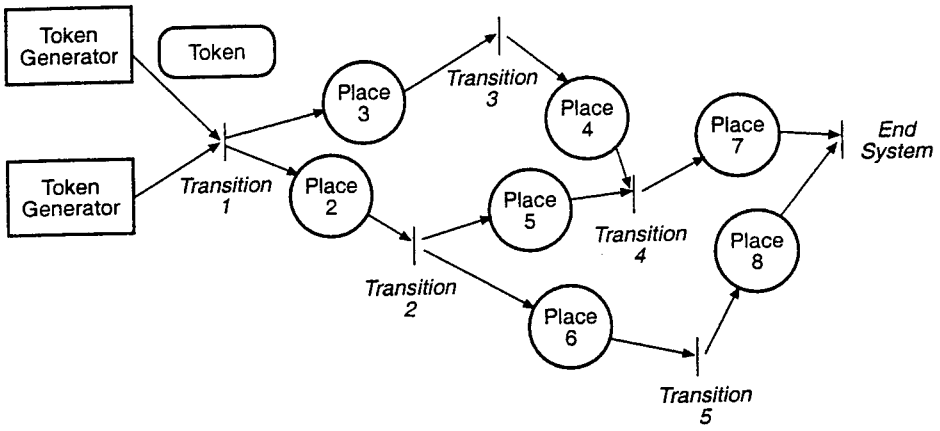


Figure 6.9. Petri-Net Representation of Active Network with AVNMP for T-Invariant Analysis.

A rollback occurs when an incoming virtual message has a less than the logical process's. The receive time of a virtual message is determined by the local virtual time of the sending logical process. It is assumed that the receive time cannot be less than the local virtual time of the sending logical process. Let T_j be the total number of transition firings for logical process j. When a token arrives at logical process j from logical process i, a rollback does not occur as long as $T_i < T_j$. A logical process can receive virtual messages from more than one logical process. Let T_i^* be the set of all inputs to logical process j. Then $\forall T_k \in T_i^*: T_k < T_j$. If N is a matrix form of the Petri-Net as used in the previous section, Matrix 6.6 for example, and x is a vector of transitions, the T-Invariant is computed as shown in Equation 6.11. Based upon the set of x that satisfy 6.11, it is possible to determine whether j will rollback, and if so, how many of the possible invariants cause a rollback.

$$N \cdot x = 0 \tag{6.11}$$

6.2 EXPECTED SPEEDUP: η

This section analyzes the primary benefit of Active Virtual Network Management Prediction, namely expected lookahead into the future. This depends on the rate that the system can generate and handle predictions. This rate is referred to as speedup, because when these values were cached and used, they increase the rate at which the system executes. There are many factors which influence speedup including out-of-order message probability, out-of-tolerance state value probability, rate of virtual messages entering the system, task execution time, task partitioning into Logical Processes, rollback overhead, prediction accuracy as a function of the distance into the future which predictions are attempted, and the effects of parallelism and optimistic synchronization. All of these factors are considered, beginning with a direct analysis using definitions from optimistic simulation.

The definition of Global Virtual Time (*GVT*) can be applied to determine the relationship among expected task execution time (τ_{task}), the real time at which the state was cached (t_{SQ}), and real time (t). Consider the value (V_v), which is cached at real time t_{SQ} in the SQ resulting from a particular predicted event. For example, refer to Figures 5.16 through 5.20 and notice that state queue values may be repeatedly added and discarded as Active Virtual Network Management Prediction operation proceeds in the presence of rollback. As rollbacks occur, values for a particular predicted event may change, converging to the real value (V_r). For correct operation of Active Virtual Network Management Prediction, V_v should approach V_r as t approaches $GVT(t)$ where $GVT(t)$ is the *GVT* of the Active Virtual Network Management Prediction system at time t. Explicitly, this is $\forall \varepsilon > 0\ \exists \delta > 0$ s.t. $|f(t) - f(GVT(t))| < \varepsilon \Rightarrow 0 < |GVT(t) - t| < \delta$ where $f(t) = V_r$ and $f(GVT(t)) = V_v$. $f(t)$ is the prediction function of a driving process. The purpose and function of the driving process has been explained in Section 7. Because Active Virtual Network Management Prediction always uses the correct value when the predicted time (τ) equals the current real time (t) and it is assumed that the predictions become more accurate as the predicted time of the event approaches the current time, the reasonable assumption is made that $\lim_{\tau \to t} f(\tau) = V_v$. In order for the Active Virtual Network Management Prediction system to always look ahead, $\forall t\ GVT(t) \geq t$. This means that $\forall n \in \{LPs\}$ and $\forall t\ LVT_{lpn}(t) \geq t$ and $\min_{m \in \{M\}} \{ m \} \geq t$ where m is the receive time of a message, M is the set of messages in the entire system and LVT_{lpn} is the of the n^{th} Logical Process. In other words, the Local Virtual Time of each must be greater than or equal to real time and the smallest message not yet processed must also be greater than or equal to real time. The smallest message could cause a rollback to that time. This implies that $\forall n,t\ LVT_{dpn}(t) \geq t$. In other words, this implies that the Logical Virtual Time of each driving process must be greater than or equal to real time. An out-of-order rollback occurs when $m < LVT(t)$. The largest saved state time such that $t_{SQ} < m$ is used to restore the state of the Logical Process, where t_{SQ} is the real time the state was saved. Then the expected task execution time (τ_{task}) can take no longer than $t_{SQ} - t$ to complete in order for *GVT* to remain ahead of real time. Thus, a constraint between expected task execution time (τ_{task}), the time associated with a state value (t_{SQ}), and real time (t) has been defined. What remains to be considered is the effect of out-of-tolerance state values on the rollback probability and the concept of stability in Active Virtual Network Management Prediction.

6.2.1 Rollback Rate

Stability in Active Virtual Network Management Prediction is related to the ability of the system to reduce the number of rollbacks. An unstable system is one in which there exist enough rollbacks to cause the system to take longer than real-time to reach the end of the Sliding Lookahead Window. This window has a length of *Lookahead* time units. One end of the window follows the current wallclock time and the other is the distance to which the system should predict. Rollback is caused by the arrival of a message that should have been executed in the past and by out-of-tolerance states. In either case, messages that had been generated prior to the rollback are false messages. Rollback is contained by sending anti-messages to cancel the effects of false messages. The more quickly the anti-messages overtake the effect of false messages, the more efficiently rollback is contained.

One cause of rollbacks in Active Virtual Network Management Prediction is real messages that are out of tolerance. Those processes that require a higher degree of tolerance are most likely to rollback. A worst case probability of out-of-tolerance rollback for a single process, shown in Equation 6.12, is based on Chebycheff's Inequality (Papoulis, 1991) from basic probability. The variance of the data is σ^2 and Θ is the acceptable tolerance for a configuration process. Therefore, the performance gains of Active Virtual Network Management Prediction are reduced as a function of P_{ot}. At the cost of increasing the accuracy of the driving process(es), that is, decreasing σ^2 in Proposition 1, P_{ot} becomes small thus increasing the performance gain of Active Virtual Network Management Prediction.

Proposition 1

The probability of rollback of an LP is

$$P_{ot} \leq \frac{\sigma^2}{\Theta^2} \tag{6.12}$$

where P_{ot} is the probability of out-of-tolerance rollback for an LP, σ^2 is the variance in the amount of error, and Θ is the tolerance allowed for error.

The expected time between rollbacks for the Active Virtual Network Management Prediction system is critical for determining its feasibility. The probability of rollback for all processes is the probability of out-of-order message occurrence and the probability of out-of-tolerance state values ($P_{rb} = P_{oo} + P_{ot}$). The received message rate per is R_m and there are N Logical Processes. The expected inter-rollback time for the system is shown in Equation 6.13.

Proposition 2

The expected inter-rollback time is

$$T_{rb} = \frac{1}{\lambda_{rb}} = \frac{1}{R_m N P_{rb}} \tag{6.13}$$

where T_{rb} is the expected inter-rollback time, λ_{rb} is the expected rollback rate, R_m is the received message rate per , there are N es, and P_{rb} is the probability of rollback per process.

6.2.2 Single Processor Logical Processes

Multiple Logical Processes on a single processor lose any gain in concurrency since they are being served by a single processor; however, the Logical Processes can maintain the Active Virtual Network Management Prediction lookahead if partitioned properly. The single processor logical processes receive virtual messages expected to occur in the future as well as real messages. Because single processor logical processes reside on a single processor, they are not operating in parallel as logical processes do in an optimistic simulation system; thus a new term needs to be applied to a task partitioned into Logical Processes on a single processor. Each partition of tasks into Logical Processes on a single processor is called a Single Processor Logical Process (SLP). In the upper portion of Figure 6.10, a task has been partitioned into two logical processes. The same task exists in the lower portion of Figure 6.10 as a single Logical Process. If task B must rollback because of an out-of-tolerance result, the entire single Logical Process must rollback, while only the Logical Process for task B must rollback in the multiple case. Thus partitioning a task into multiple Logical Processes saves time compared to a single task. Thus, **without considering parallelism**, lookahead is achieved by allowing the sequential system to work ahead while individual tasks within the system are allowed to rollback. Only tasks that deviate beyond a given pre-configured tolerance are rolled back. Thus entire pre-computed and cached results are not lost due to inaccuracy; only parts of pre-computed results must be re-computed. There are significant differences in the behavior of SLP, MLP, and hybrid systems. Each system needs to be analyzed separately.

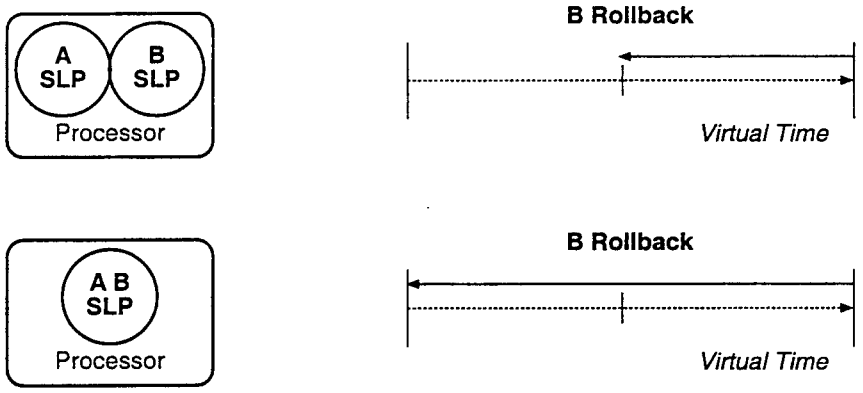


Figure 6.10. Single and Multiple Processor Logical Process System.

Consider the optimal method of partitioning a single processor system into Single Processor Logical Processes in order to obtain speedup over a single process. Assume n tasks, $task_1$, ..., $task_n$, with expected execution times of τ_1, ..., τ_n, and that $task_n$ depends on messages from $task_{n-1}$ with a tolerance of Θ_n. This is the largest error allowed in the input message such that the output is correct. Using the results from Proposition 4.1, it is possible to determine a partitioning of tasks into logical processes such that speedup is achieved over operation of the same tasks encapsulated in a single Logical Process. Figure 6.11 shows possible groupings of the same set of six tasks into logical processes. It is hypothesized that the tasks that are most likely to rollback and those that take the greatest amount of time to execute should be grouped together within Single Processor Logical Processes to minimize the rollback time. There are 2^{n-1} possible groupings of tasks into Single Processor Logical Processes, where n is the number of tasks and

message dependency among the tasks is maintained. Those tasks least likely to rollback and those that execute quickly should be grouped within a single Single Processor to reduce the overhead of rollback. For example, if all the tasks in Figure 6.11 have an equal probability of rollback and $\tau_2 >> \max\{ \tau_1, \tau_3, \ldots \}$ then the tasks should be partitioned such that $task_2$ is in a separate Single Processor : ($task_1$ | $task_2$ | $task_3$ ... $task_n$) where "|" indicates the grouping of tasks into sequential logical processes.

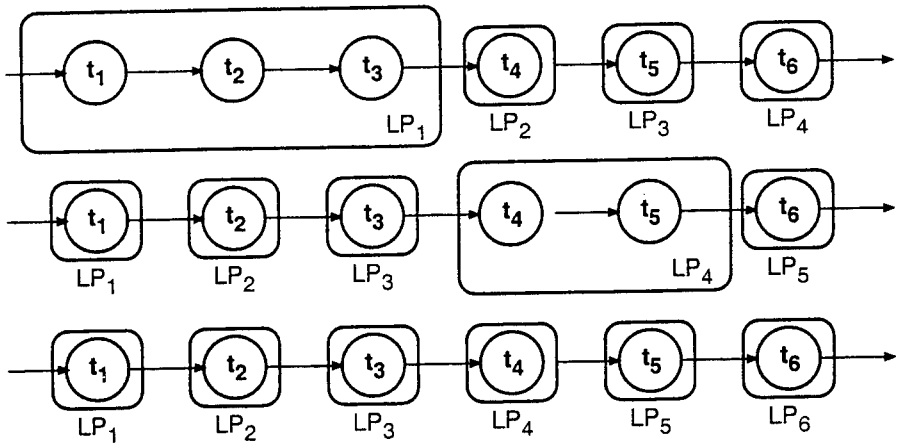


Figure 6.11. Possible Partitioning of Tasks into Logical Processes on a Single Processor.

For example, the expected execution time for five tasks with equal probabilities of rollback of 0.1 are shown in Figure 6.12. It is assumed that these tasks communicate in order starting from Task 1 to Task 5 in order to generate a result. In Figure 6.12, the x-axis indicates the boundary between task partitions as the probability of rollback of task 5 is varied. With an x-value of 3, the solid surface shows the expected execution time for the first three tasks combined within a single and the remainder of the tasks encapsulated in separate Logical Processes. The dashed surface shows the first three tasks encapsulated in separate Logical Processes and the remainder of the tasks encapsulated within a Logical Process. The graph in Figure 6.12 indicates a minimum for both curves when the high probability rollback tasks are encapsulated in separate Logical Processes from the low probability of rollback tasks. As the probability of rollback increases, the expected execution time for all five processes is minimized when Task 5 is encapsulated in a separate Logical Process.

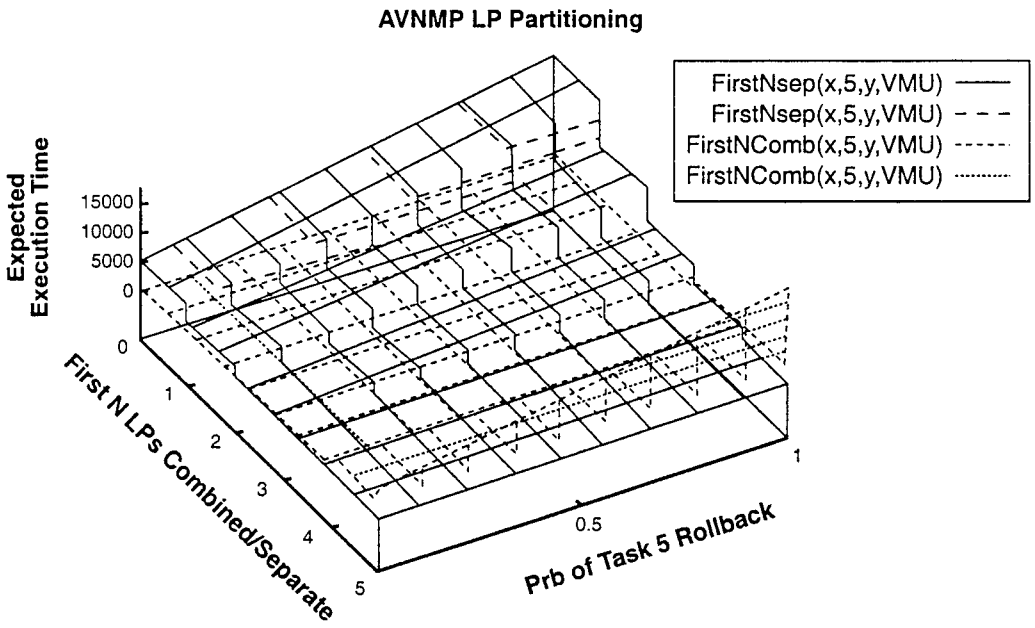


Figure 6.12. Optimal Single Processor Logical Process Partitioning.

6.2.2.1 Task Partition Analysis

Consider an example of Active Virtual Network Management Prediction used for traffic prediction. Assume the computation time is exponentially distributed with mean $[1/(\mu_{r2})]$. As a simplified example, assume the packet forwarding operation for a router of type A is also exponentially distributed with mean $[1/(\mu_{r1})]$. The router of type B has a rollback probability of P_{r2} and takes time τ_{r2} to rollback. The router of type A has a rollback probability of P_{r1} and takes time τ_{r1} to rollback. If both operations are encapsulated by a single logical process, then the expected time of operation is shown in Equation 6.14. If each operation is encapsulated in a separate logical process, then the expected time is shown in Equation 6.15. Equations 6.14 and 6.15 are formed by the sum of the expected time to execute the task, which is the first term, and the rollback time, which is the second term. The probability of rollback in the combined Logical Process is the probability that either task will rollback. Therefore, the expected execution time of the tasks encapsulated in separate Logical Processes is smaller since $\tau_{separate} < \tau_{combined}$.

$$\tau_{combined} = \left(\frac{1}{\mu_{r_2}} + \frac{1}{\mu_{r_1}}\right) + \left(\frac{1}{\mu_{r_2}} + \frac{1}{\mu_{r_1}}\right)\left(P_{r_2} + P_{r_1}\right)\left(\pi_{r_2} + \pi_{r_1}\right) \tag{6.14}$$

$$\tau_{separate} = \left(\frac{1}{\mu_{r_2}} + \frac{1}{\mu_{r_2}} P_{r_2}\pi_{r_2}\right) + \left(\frac{1}{\mu_{r_1}} + \frac{1}{\mu_{r_1}} P_{r_1}\pi_{r_1}\right) \tag{6.15}$$

The grouping of tasks into Single Processor Logical Processes can be done dynamically, that is, while the system is in operation. This dynamic adjustment is currently outside the scope of this research but related to optimistic simulation load balancing (Glazer, 1993, Glazer and

Tropper, 1993) and the recently developed topic of optimistic simulation dynamic partitioning (Boukerche and Tropper, 1994, Konas and Yew, 1995).

6.2.3 Single Processor Logical Process Prediction Rate

The Local Virtual Time is a particular Logical Process's notion of the current time. In optimistic simulation the Local Virtual Time of individual processes may be different from one another and generally proceed at a much faster rate than real time. Thus, the rate at which a Single Processor system can predict events (prediction rate) is the rate of change of the Single Processor Logical Process's Local Virtual Time with respect to real time. Assume a driving process whose virtual message generation rate is λ_{vm}. The Local Virtual Time is increased by the expected amount Δ_{vm} every $[1/(\lambda_{vm})]$ time units. The expected time spent executing the task is τ_{task}. The random variables X and Y are the proportion of messages that are out-of-order and out-of-tolerance respectively. The expected real time to handle a rollback is τ_{rb}. Then the Single Processor Logical Process's Local Virtual Time advances at the expected rate shown in Proposition 3.

Proposition 3 *[Single Processor Logical Process Speed] The average prediction rate of a single logical processor system is*

$$S_{cache} = \frac{LVT}{t} = \lambda_{vm}\left(\Delta_{vm} - \tau_{task} - (\tau_{task} + \tau_{rb})E[X] - \left(\Delta_{vm} - \frac{1}{\lambda_{vm}} \right) E[Y] \right) \tag{6.16}$$

where the virtual message generation rate is λ_{vm}, the expected lookahead per message is Δ_{vm}, the proportion of out-of-order messages is X, the proportion of out-of-tolerance messages is Y, τ_{task} is the expected task execution time in real time, τ_{rb} is the expected rollback overhead time in real time, LVT is the Local Virtual Time, and t is real time.

In Proposition 3, the expected lookahead per message (Δ_{vm}) is reduced by the real time taken to process the message (τ_{task}). The expected lookahead is also reduced by the time to re-execute the task (τ_{task}) and the rollback time (τ_{rb}) times the proportion of occurrences of an out-of-order message ($E[X]$) that results in the term (τ_{task} + τ_{rb}) $E[X]$. Finally, the derivation of the (Δ_{vm} – $[1/(\lambda_{vm})]$) $E[Y]$ term is shown in Figure 6.13. In Figure 6.13, a real message arrives at time t. Note that real time t and Local Virtual Time are both shown on the same time axis in Figure 6.13. The current Local Virtual Time of the process is labeled at time $LVT(t)$ in Figure 6.13. The dotted line in Figure 6.13 represents the time $\Delta_{vm} - [1/(\lambda_{vm})]$ that is subtracted from the when an out-of-tolerance rollback occurs. The result of the subtraction of $\Delta_{vm} - [1/(\lambda_{vm})]$ from the LVT(t) results in the Local Virtual Time returning to real time as required by the algorithm. The virtual message inter-arrival time is $[1/(\lambda_{vm})]$. Note that the (Δ_{vm} – $[1/(\lambda_{vm})]$) $E[Y]$ term causes the speedup to approach 1 based on the frequency of out-of-tolerance rollback ($E[Y]$).

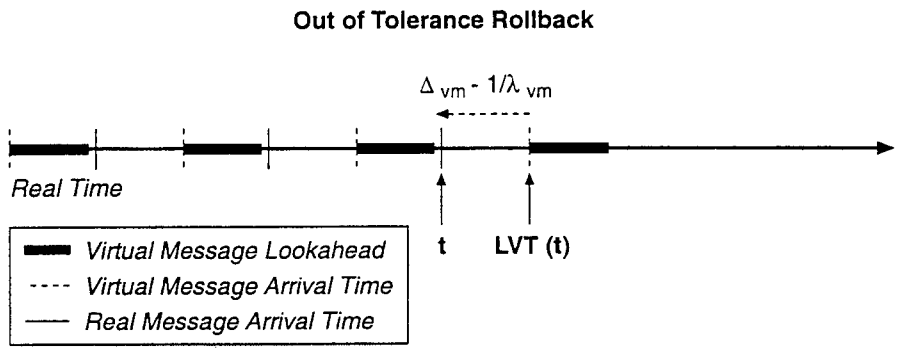


Figure 6.13. Out-of-Tolerance Rollback.

$$\begin{aligned} \nabla f(x^*) + \lambda^T \nabla h(x^*) + \mu^T \nabla g(x^*) = 0 \\ \mu^T g(x^*) = 0 \\ \mu \geq 0 \end{aligned} \tag{6.17}$$

6.2.4 Sensitivity

If the proportion of out-of-tolerance messages, Y, cannot be reduced to zero, the virtual message generation rates and expected virtual message lookahead times can be adjusted in order to improve speedup. Given the closed form expression for Active Virtual Network Management Prediction speedup in Proposition 3, it is important to determine the optimal values for each parameter, particularly λ_{vm} and Δ_{vm} and in addition, the sensitivity of each parameter. Sensitivity information indicates parameters that most affect the speedup. The parameters that most affect the speedup are the ones that yield the best results if optimized.

One technique that optimizes a constrained objective function and that also determines the sensitivity of each parameter within the constraints is the Kuhn-Tucker method (Luenberger, 1989, p. 314). The reason for using this method rather than simply taking the derivative of Equation 6.16 is that the optimal value must reside within a set of constraints. Depending on the particular application of Active Virtual Network Management Prediction, the constraints may become more complex than those shown in this example. The constraints for this example are discussed in detail later. The sensitivity results appear as a by-product of the Kuhn-Tucker method. The first order necessary conditions for an extremum using the Kuhn-Tucker method are listed in Equation 6.17. The second order necessary conditions for an extremum are given in Equation 6.18, where L must be positive semi-definite over the active constraints and L, F, H, and G are Hessians. The second order sufficient conditions are the same as the first order necessary conditions and the Hessian matrix in Equation 6.18 is positive definite on the subspace $M = \{y: \nabla h(x)\, y = 0, \nabla g_j(x)\, y = 0 \text{ for all } j \in J\}$, where $J = \{j: g_j(x) = 0, \mu_j \geq 0\}$. The sensitivity is determined by the Lagrange multipliers, λ^T and μ^T. The Hessian of the objective function and of each of the inequality constraints is a zero matrix; thus, the eigenvalues L in Equation 6.18 are zero and the matrix is clearly positive definite, satisfying both the necessary and sufficient conditions for an extremum.

$$L(x^*) = F(x^*) + \lambda^T H(x^*) + \mu^T G(x^*) \tag{6.18}$$

The function f in Equation 6.17 is the Active Virtual Network Management Prediction speedup given in Equation 6.16. The matrix h does not exist, because there are no equality constraints, and the matrix g consists of the inequality constraints that are specified in Equation 6.20.

Clearly the upper bound constraints on $E[X]$ and $E[Y]$ are the virtual message rate. The constraints for τ_{task} and τ_{rb} are based on measurements of the task execution time and the time to execute a rollback. The maximum value for λ_{vm} is determined by the rate at which the virtual message can be processed. Finally, the maximum value for Δ_{vm} is determined by the required caching period. If Δ_{vm} is too large, there may be no state in the SQ with which to compare an incoming real message.

From inspection of Equation 6.16 and the constraint shown in Equation 6.19, the constraints from are $\Delta_{vm} = 45.0$, $\tau_{task} = 5.0$, $\tau_{rb} = 1.0$, $E[X] = 0.0$, $E[Y] = 0.0$ that results in the optimal solution shown in Equation 6.22. The Lagrange multipliers μ_1 through μ_6 show that $E[Y]$ ($-\mu_6 = -8.0$), λ_{vm} ($-\mu_1 = -40.0$), and $E[X]$ ($-\mu_5 = -1.2$) have the greatest sensitivities. Therefore, reducing the out-of-tolerance rollback has the greatest effect on speedup. However, the effect of optimistic synchronization on speedup needs to be studied.

$$l\lambda_{vm} = \frac{1}{\tau_{task} + (\tau_{task} + \tau_{rb})E[X] + \left(\Delta_{vm} - \frac{1}{\lambda_{vm}}\right)E[Y]} \tag{6.19}$$

$$0.0 \le \lambda_{vm} \le \frac{1}{\tau_{task} + (\tau_{task} + \tau_{rb})E[X] + \left(\Delta_{vm} - \frac{1}{\lambda_{vm}}\right)E[Y]} \tag{6.20}$$

$$0.1 \le \Delta_{vm} \le 45.0 \tag{8.21}$$

$$\begin{aligned} 5.0 \le \tau_{task} &\le 10.0 \\ 1.0 \le \tau_{rb} &\le 2.0 \\ 0.0 \le E[X] &\le 1.0 \\ 0.0 \le E[Y] &\le 1.0 \\ \lambda_{vm} = 1.0, \mu_1 &= 40.0 \\ \Delta_{vm} = 45.0, \mu_2 &= 0.2 \\ \tau_{task} = 0.0, \mu_3 &= 0.2 \\ \tau_{rb} = 0.0, \mu_4 &= 0.0 \\ E[X] = 0.0, \mu_5 &= 1.2 \\ E[Y] = 0.0, \mu_6 &= 8.0 \end{aligned} \tag{6.22}$$

6.2.5 Sequential Execution Multiple Processors

At the time of this writing, a comparison of optimistic synchronization with sequential synchronization cannot be found in the literature because there has been little work on techniques that combine optimistic synchronization and a real time system with the exception of hybrid systems such as the system described in (Bagrodia and Shen, 1991). The hybrid system described in (Bagrodia and Shen, 1991) is used as a design technique in which distributed simulation LPs are gradually replaced with real system components allowing the emulated system to be executed as the system is built. It does not focus on predicting events as in Active Virtual Network Management Prediction. This section examines sequential execution of tasks, which corresponds with non-Active Virtual Network Management Prediction operation as shown in Figure 6.14 in order to compare it with the Active Virtual Network Management Prediction algorithm in the next section. As a specific example, consider K virtual messages with load prediction values passing through P router forwarding processes and each process has an exponential processing time with average $[1/(\mu)]$. In the sequential case, as might be done within the centralized manager as shown in Figure 6.1, the expected completion time should be K times the summation of P exponential distributions. The summation of P exponential distributions is a Gamma Distribution as shown in the sequential execution probability distribution function in Equation 6.23. The average time to complete K tasks is shown in Equation 6.24.

$$f_T(x|P,\mu)=\begin{cases}\dfrac{\mu^P}{\Gamma(P)}x^{P-1}\exp^{-\mu x} & x>0\\ 0 & x\le 0\end{cases} \tag{6.23}$$

$$T_{seq}=K\int_0^{\infty}xf_T(x|P,\mu)\,dx \tag{6.24}$$

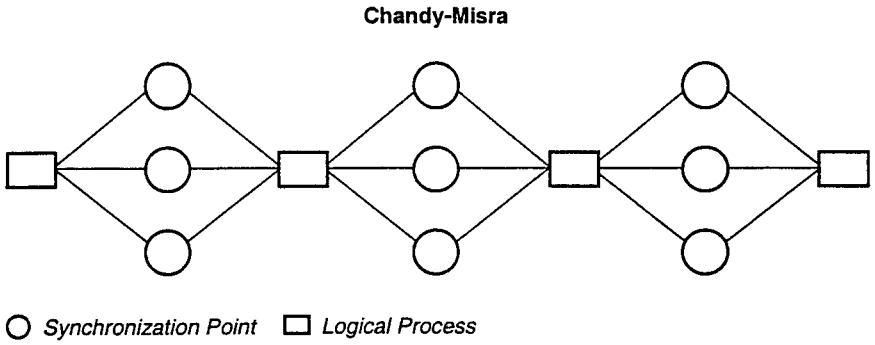


Figure 6.14. Sequential Model of Operation.

6.2.6 Asynchronous Execution Multiple Processors

Assume that an ordering of events is no longer a requirement. This represents the asynchronous Active Virtual Network Management Prediction case and is shown in Figure 6.15. Note that this is the analysis of speedup due to parallelism only, not the lookahead capability of asynchronous Active Virtual Network Management Prediction. This analysis of speedup

assumes messages arrive in correct order and thus there is no rollback. However, this also assumes that there are no optimization methods such as lazy cancellation. Following (Felderman and Kleinrock, 1990) the expected completion time is approximated by the maximum of P K-stage Erlangs where P is the number of processes which can execute in parallel at each stage of execution. A K-stage Erlang model represents the total service time as a series of exponential service times, where each service time is performed by a process residing on an independent processor in this case. There is no need to delay processing within the K-stage model because of inter-process dependencies, as there is for synchronous and sequential cases. Equation 6.25 shows the pdf for a K-stage Erlang distribution.

$$f_T(x) = \frac{\mu e^{-\mu x}(\mu x)^{K-1}}{(K-1)!} \tag{6.25}$$

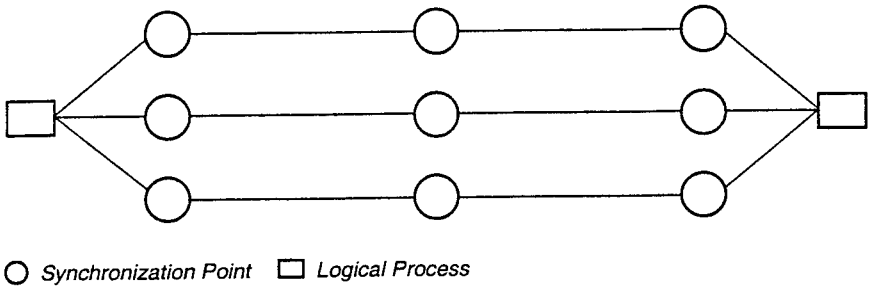


Figure 6.15. Active Virtual Network Management Prediction Model of Parallelism.

As pointed out in (Felderman and Kleinrock, 1990), the probability that a K-stage Erlang takes time less than or equal to t is 1 minus the probability that the K-stage Erlang distribution takes time greater than t, which is simply one minus the probability that there are K arrivals in the interval $[0,t]$ from a Poisson process at rate μ. This result is shown in Equation 6.26.

$$F_T(x) = 1 - e^{-\mu x} \sum_{i=0}^{K-1} \frac{(\mu x)^i}{i!} \tag{6.26}$$

$$T_{async} = \int_0^\infty [1 - F_T(x)]\, dx \tag{6.27}$$

The expected value is shown in Equation 6.27. This integral is hard to solve with a closed form solution and (Felderman and Kleinrock, 1990) instead try to find an approximate equation. This study attempts to be exact by using Equation 6.27 and solving it numerically (Kleinrock, 1975, p. 378). In Equation 6.28 $S_{parallel}$ is the speedup of optimistic synchronization over strictly sequential synchronization and is graphed in Figure 6.16 as a function of the number of processors. The speedup gained by parallelism ($S_{parallel}$) augments the speedup due to lookahead (S_{cache}) as shown in Equation 6.29, where the (PR) is the Active Virtual Network Management

Prediction speedup and X and Y are random variables representing the proportion of out-of-order and out-of-tolerance messages respectively.

$$S_{parallel} = \frac{T_{seq}}{T_{async}} \tag{6.28}$$

$$PR_{X,Y} = \lambda_{vm}\left(\Delta_{vm}S_{parallel} - \tau_{task} - (\tau_{task} + \tau_{rb})X - \left(\Delta_{vm}S_{parallel} - \frac{1}{\lambda_{vm}}\right)Y\right) \tag{6.29}$$

There is clearly a potential speedup in Active Virtual Network Management Prediction in contrast to a single processor model of the network. The Active Virtual Network Management Prediction algorithm implementation is able to take advantage of both Single Processor Logical Processes (Slogical Process) lookahead without parallel processing and speedup due to parallel processing because Active Virtual Network Management Prediction has been implemented on many nodes throughout the network and each node has its own processor. Note that while Clustered Time Warp (Avril, 1996), which was developed concurrently but independently of Active Virtual Network Management Prediction, uses a similar concept to Single Processor Logical Processes and Logical Process, it does not consider a real-time system as in Active Virtual Network Management Prediction.

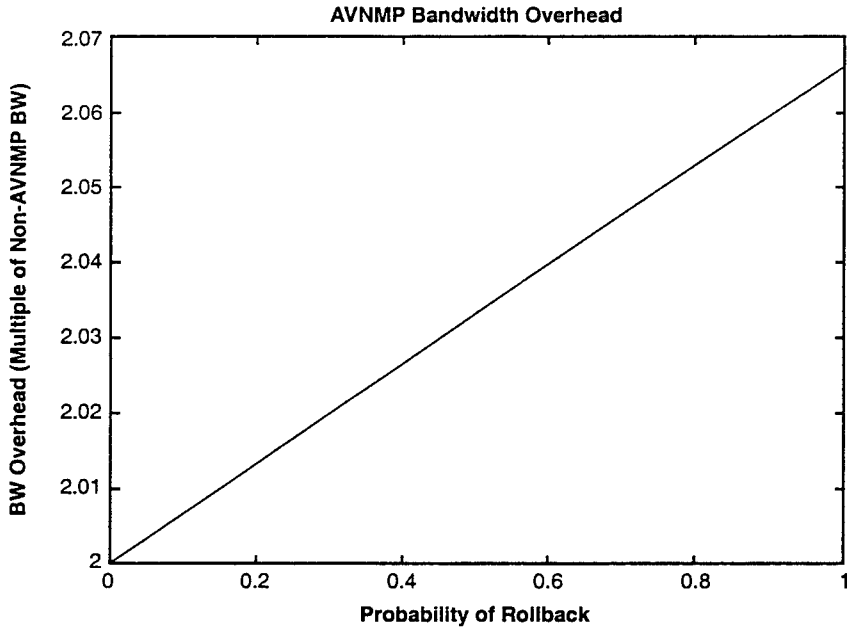


Figure 6.16. Speedup of AVNMP over Non-AVNMP Systems Due to Parallelism.

6.2.7 Multiple Processor Logical Processes

The goal of Active Virtual Network Management Prediction is to provide accurate predictions quickly enough so that the results are available before they are needed. Without taking advantage of parallelism, a less sophisticated algorithm than Active Virtual Network

Management Prediction could run ahead of real-time and cache results for future use. This is done in the Sequential Processor system, which assumes strict synchronization between processes whose prediction rate is defined in Proposition 3. With such a simpler mechanism, P_{oo} and $E[X]$ are always zero. However, simply predicting and caching results ahead of time does not fully utilize inherent parallelism in the system as long as messages between Logical Processes remain strictly synchronized. Strict synchronization means that processes must wait until all messages are insured to be processed in order. Any speedup to be gained through parallelism comes from the same mechanism as in optimistic parallel simulation; the assumption that messages arrive in order by TR, thus eliminating unnecessary synchronization delay. However, messages arrive out-of-order in Active Virtual Network Management Prediction for the following reasons. A general-purpose system using the Active Virtual Network Management Prediction algorithm may have multiple driving processes, each predicting at different rates into the future. Another reason for out-of-order messages is that Logical Processes are not required to wait until processing completes before sending the next message. Also, processes may run faster for virtual computations by allowing a larger tolerance. Finally, for testing purposes, hardware or processes may be replaced with simulated code, thus generating results faster than the actual process would. Thus, although real and future time are working in parallel with strict synchronization, no advantage is being taken of parallel processing. This is demonstrated by the fact that, with strict synchronization of messages, the same speedup (S_{cache}) as defined in Proposition 3 occurs regardless of whether a single processor or multiple processors are used. What differentiates Active Virtual Network Management Prediction is the fact that it takes advantage of inherent parallelism in the system as compared to a sequential non-Active Virtual Network Management Prediction pre-computation and caching method. Thus it is better able to meet the deadline imposed by predicting results before they are required. To see why this is true, consider what happens as the overhead terms in Proposition 3, $\tau_{task} - (\tau_{task} + \tau_{rb})\ E[X] - (\Delta_{vm} S_{parallel} - [1/(\lambda_{vm})])\ E[Y]$, approach Δ_{vm}. The prediction rate becomes equal to real-time and can fall behind real-time as $\tau_{task} - (\tau_{task} + \tau_{rb})\ E[X] - (\Delta_{vm} S_{parallel} - [1/(\lambda_{vm})])\ E[Y]$ becomes larger. Optimistic synchronization helps to alleviate the problem of the prediction rate falling behind real-time. Optimistic synchronization has another advantageous property, super-criticality. A super critical system is one that can compute results faster than the time taken by the critical path through the system. This can occur in Active Virtual Network Management Prediction using the lazy cancellation optimization as discussed in Section 7. Super-criticality occurs when task execution with false message values generates a correct result. Thus prematurely executed tasks do not rollback and a correct result is generated faster than the route through the critical path.

The Active Virtual Network Management Prediction algorithm has two forms of speedup that need to be clearly defined. There is the speedup in availability of results because they have been pre-computed and cached. There is also the speedup due to more efficient usage of parallelism. The gain in speedup due to parallelism in Active Virtual Network Management Prediction can be significant given the proper conditions. In order to prevent confusion about the type of speedup being analyzed, the speedup due to pre-computing and caching results is defined as S_{cache} and the speedup due to parallelism is defined as $S_{parallel}$. Speedup due to parallelism among multiple processors in Active Virtual Network Management Prediction is gained from the same mechanism that provides speedup in parallel simulation, that is, it is assumed that all relevant messages are present and are processed in order by receive time. The method of maintaining message order is optimistic in the form of rollback. The following sections look at $S_{parallel}$ due to a multiprocessor configuration system.

6.2.8 AVNMP Prediction Rate with a Fixed Lookahead

There are three possible cases to consider when determining the speedup of Active Virtual Network Management Prediction over non-lookahead sequential execution. The speedup given each of these cases and their respective probabilities needs to be analyzed. These cases are illustrated in Figures 6.17 through 6.19. The time that an event is predicted to occur and the result cached is labeled $t_{virtual\ event}$, the time a real event occurs is labeled $t_{real\ event}$, and the time a result for the real event is calculated is labeled $t_{no-avnmp}$. In Active Virtual Network Management Prediction, the virtual event and its result can be cached before the real event, as shown in Figure 6.17, between the real event and the time the real event result is calculated as shown in Figure 6.18, or after the real event result is calculated as shown in Figure 6.19. In each case, all events are considered relative to the occurrence of the real event. It is assumed that the real event occurs at time t. A random variable called the lookahead (LA) is defined as $LVT - t$. The virtual event occurs at time $t - LA$. Assume that the task that must be executed once the real event occurs takes τ_{task} time. Then without Active Virtual Network Management Prediction the task is completed at time $t + \tau_{task}$.

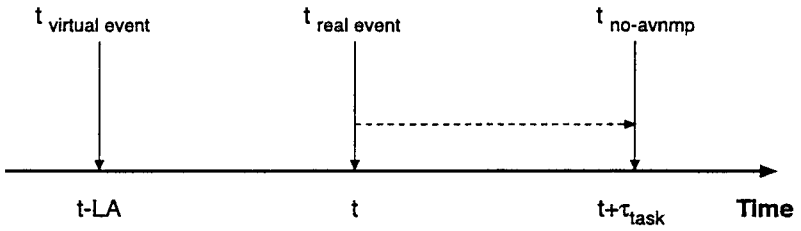


Figure 6.17. AVNMP Prediction Cached before Real Event.

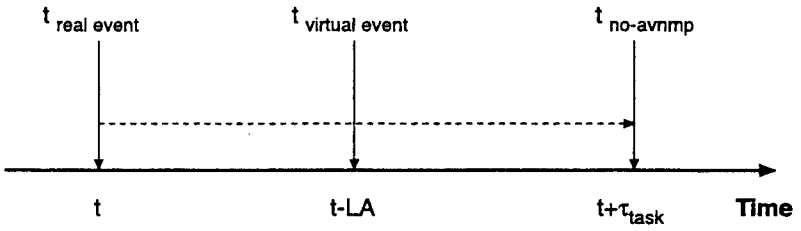


Figure 6.18. AVNMP Prediction Cached Later than Real Event.

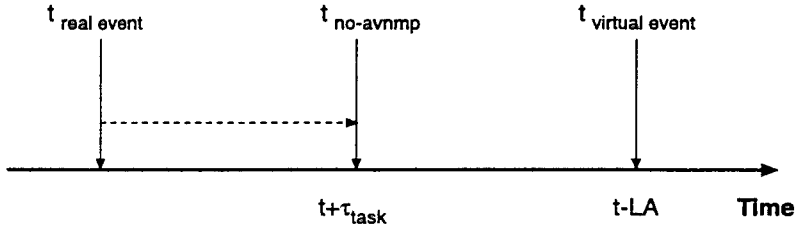


Figure 6.19. AVNMP Prediction Cached Slower than Real Time.

The prediction rate has been defined in Equation 6.29 and includes the time to predict an event and cache the result in the State Queue. Recall that in Section 2 the expected value of X has been determined based on the inherent synchronization of the topology. It was shown that X has an expected value that varies with the rate of hand-offs. It is clear that the proportion of out-of-order messages is dependent on the architecture and the partitioning of tasks into Logical Processes. Thus, it is difficult in an experimental implementation to vary X. It is easier to change the tolerance rather than change the architecture to evaluate the performance of Active Virtual Network Management Prediction. For these reasons, the analysis proceeds with $PR_{X,Y|X=E[X]}$. Since the prediction rate is the rate of change of Local Virtual Time with respect to time, the value of the Local Virtual Time is shown in Equation 6.30, where C is an initial offset. This offset may occur because Active Virtual Network Management Prediction may begin running C time units before or after the real system. Replacing LVT in the definition of LA with the right side of Equation 6.30 yields the Equation for lookahead shown in Equation 6.31.

$$LVT_{X,Y|x=E[X]} = \lambda_{vm}\left(\Delta_{vm}S_{parallel} - \tau_{task} - (\tau_{task} + \tau_{rb})E[X] - \left(\Delta_{vm}S_{parallel} - \frac{1}{\lambda_{vm}}\right)Y\right)t + C \tag{6.30}$$

$$LA_{X,Y|X=E[X]} = \left(LVT_{X,Y|X=E[X]} - 1\right) + C \tag{6.31}$$

The probability of the event in which the Active Virtual Network Management Prediction result is cached before the real event is defined in Equation 6.32. The probability of the event for which the Active Virtual Network Management Prediction result is cached after the real event but before the result would have been calculated in the non-Active Virtual Network Management Prediction system is defined in Equation 6.33. Finally, the probability of the event for which the Active Virtual Network Management Prediction result is cached after the result would have been calculated in a non-Active Virtual Network Management Prediction system is defined in Equation 6.34.

$$P_{cache} = P\left[LA_{X,Y|X=E[X]} > \tau_{task}\right] \tag{6.32}$$

$$P_{late} = P\left[0 \le LA_{X,Y|X=E[X]} \le \tau_{task}\right] \tag{6.33}$$

$$P_{slow} = P\left[LA_{X,Y|X=E[X]} < 0\right] \tag{6.34}$$

The goal of this analysis is to determine the effect of the proportion of out-of-tolerance messages (Y) on the speedup of an Active Virtual Network Management Prediction system. Hence we assume that the proportion Y is a binomially distributed random variable with parameters n and p where n is the total number of messages and p is the probability of any single message being out of tolerance. It is helpful to simplify Equation 6.31 by using γ_1 and γ_2 as defined in Equations 6.36 and 6.37 in Equation 6.35.

$$LA_{X,Y|X=E[X]} = \gamma_1 - \gamma_2 Y \tag{6.35}$$

$$\gamma_1 = \left(\lambda_{vm}\Delta_{vm}S_{parallel} - \lambda_{vm}\tau_{task} - \lambda vm(\tau_{rb} + \tau_{task})E[X] - 1\right)t + C \tag{6.36}$$

$$\gamma_2 = \lambda_{vm}\left(\Delta_{vm} S_{parallel} - \frac{1}{\lambda_{vm}} + \tau_{rb}\right)t \tag{6.37}$$

The early prediction probability as illustrated in Figure 6.17 is shown in Equation 6.38. The late prediction probability as illustrated in Figure 6.18 is shown in Equation 6.38. The probability for which Active Virtual Network Management Prediction falls behind real time as illustrated in Figure 6.19 is shown in Equation 6.40. The three cases for determining Active Virtual Network Management Prediction speedup are thus determined by the probability that Y is greater or less than two thresholds.

$$P_1(t) = P_{cache} \quad X,Y|X=E[X] = P\left[Y < \frac{\gamma_1 - \tau_{task}}{\gamma_2}\right] \tag{6.38}$$

$$P_2(t) = P_{late} \quad X,Y|X=E[X] = P\left[\frac{\gamma_1 - \tau_{task}}{\gamma_2} \le Y \frac{\gamma_1}{\gamma_2}\right] \tag{6.39}$$

$$P_3(t) = P_{slow} \quad X,Y|X=E[X] = P\left[Y > \frac{\gamma_1}{\gamma_2}\right] \tag{6.40}$$

The three probabilities in Equations 6.38 through 6.40 depend on (Y) and real time because the analysis assumes that the lookahead increases indefinitely, which shifts the thresholds in such a manner as to increase Active Virtual Network Management Prediction performance as real time increases. However, the Active Virtual Network Management Prediction algorithm holds processing of virtual messages once the end of the Sliding Lookahead Window is reached. The hold time occurs when $LA = \Lambda$ where Λ is the length of the Sliding Lookahead Window. Once Λ is reached, processing of virtual messages is discontinued until real-time reaches Local Virtual Time. The lookahead versus real time including the effect of the Sliding Lookahead Window is shown in Figure 6.20. The dashed arrow represents the lookahead which increases at rate PR. The solid line returning to zero is lookahead as the Logical Process delays. Because the curve in Figure 6.20 from 0 to t_L repeats indefinitely, only the area from 0 to t_L need be considered. For each $P_i(t)$ i = 1,2,3, the time average over the lookahead time (t_L) is shown by the integral in Equation 6.41.

$$P_{X,Y|X=E[X]} = \frac{1}{t_L}\int_0^{t_L} P_i t\,(dt) \tag{6.41}$$

$$\eta \equiv P_{cache\ X|X=E[X]} C_r + \left(P_{late\ X|X=E[X]} + P_{slow\ X|X=E[X]}\right) PR_{X,Y|X=E[X]} \tag{6.42}$$

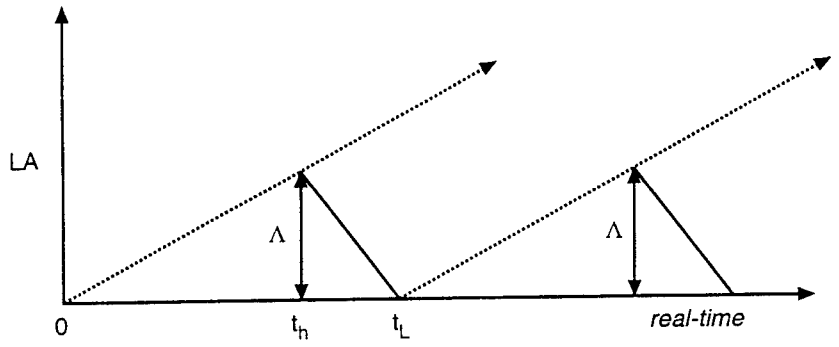


Figure 6.20. Lookahead with a Sliding Lookahead Window.

The probability of each of the events shown in Figures 6.17 through 6.19 is multiplied by the speedup for each event in order to derive the average speedup. For the case shown in Figure 6.17, the speedup (C_r) is provided by the time to read the cache over directly computing the result. For the remaining cases the speedup is $PR_{X,Y|X = E[X]}$ that has been defined as $[(LVT_{X,Y|X = E[X]})/t]$ as shown in Equation 6.42. The analytical results for speedup are graphed in Figure 6.21. A high probability of out-of-tolerance rollback in Figure 6.21 results in a speedup of less than one. Real messages are always processed when they arrive at a Logical Process. Thus, no matter how late Active Virtual Network Management Prediction results are, the system continues to run near real time. However, when Active Virtual Network Management Prediction results are very late due to a high proportion of out-of-tolerance messages, the Active Virtual Network Management Prediction system is slower than real time because out-of-tolerance rollback overhead processing occurs. Anti-messages must be sent to correct other Logical Processes that have processed messages which have now been found to be out of tolerance from the current Logical Process. This causes the speedup to be less than one when the out-of-tolerance probability is high. Thus, $PR_{X,Y|X = E[X]}$ will be less than one for the "slow" predictions shown in Figure 6.19.

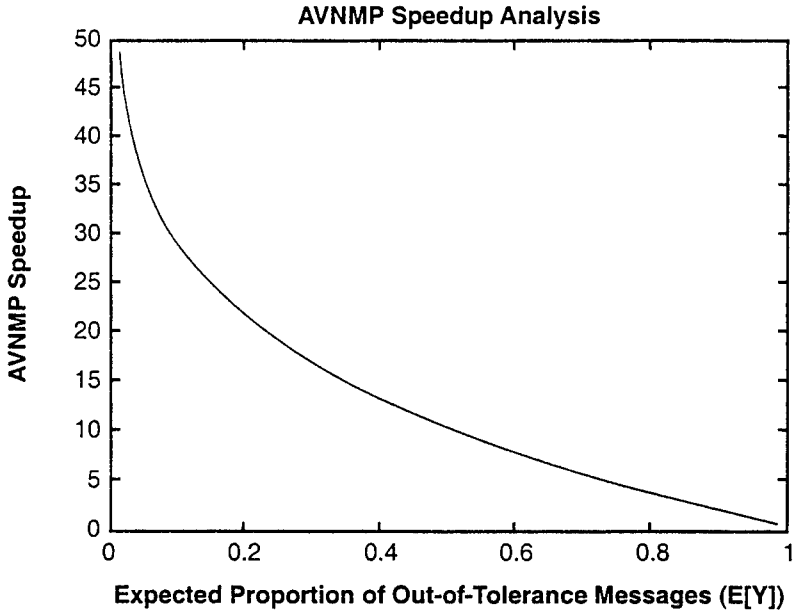


Figure 6.21. AVNMP Speedup.

6.3 PREDICTION ACCURACY

This section derives the prediction accuracy and bandwidth overhead of AVNMP and uses these relationships along with the expected speedup from the previous section to analyze the performance of AVNMP.

6.3.1 Prediction of Accuracy: α

Accuracy is the ability of the system to predict future events. A higher degree of accuracy will result in more "cache hits" of the predicted state cache information. Smaller tolerances should result in greater system accuracy, but this comes at the cost of a reduction in speedup.

Assume for simplicity that the effects of non-causality are negligible for the analysis in this section. The effects of causality are discussed in more detail in Section 6.2. A Logical Process may deviate from the real object it represents either because the Logical Process does not accurately represent the actual entity or because events outside the scope of the predictive system may effect the entities being managed. Ignore events outside the scope of the predictive system for this analysis and consider only the deterministic error from inaccurate prediction of the driving process. The error is defined as the difference between an actual message value at the current time (v_t) and a message value that had been predicted earlier (v_{tp}). Thus the **M**essage **E**rror is $ME = v_t - v_{tp}$. Virtual message values generated from a driving process may contain some error. It is assumed that the error in any output message generated by a process is a function of any error in the input message and the amount of time it takes to process the message. A larger processing time increases the chances that external events may have changed before the processing has completed.

Two functions of total **AC**cumulated message value error (AC(·)) in a predicted result are described by Equations 6.43 and 6.44 and are illustrated in Figure 6.22. ME_{lp0} is the amount of

error in the value of the virtual message injected into the predictive system by the driving process (lp_0). The error introduced into the value of the output message produced by the computation of each is represented by the **C**omputation **E**rror function $CE_{lpn}(ME_{lpn-1}, t_{lpn})$. The real time taken for the n^{th} Logical Process to generate a message is t_{lpn}. The error accumulates in the State Queue at each node by the amount $CE_{lpn}(ME_{lpn-1}, t_{lpn})$, which is a function of the error contained in the input message from the predecessor and the time to process that message. Figure 6.22 shows a driving process (DP) generating a virtual message that contains prediction error (ME_{lp0}). The virtual message with prediction error (ME_{lp0}) is processed by node LP_1 in t_{lp1} time units resulting in an output message with error, $ME_{lp1} = CE_{lp0}(ME_{lp0}, t_{lp1})$.

Proposition 4

The accumulated error in a message value is Equation 6.43 and Equation 6.44.

$$AC_n(n) = \sum_{i=1}^{N} CE_{lp_i}\left(ME_{lp_{i-1}}, t_{lp_i}\right) \tag{6.43}$$

$$AC_t(\tau) = \lim_{\sum t_{lp_i} \to \tau} \sum_{i=1}^{n} CE_{lp_i}\left(ME_{lp_{i-1}}, t_{lp_i}\right) \tag{6.44}$$

$$\alpha \equiv \Pr\left[\tan^{-1}\left(\frac{\sigma}{d}\right) > \Theta\right] \tag{6.45}$$

Where Ce_{lpi} is the computational error added to a virtual message value, ME_{lpi} is the virtual message input error, and $t_{lp}i$ is the real time taken to process a virtual message.

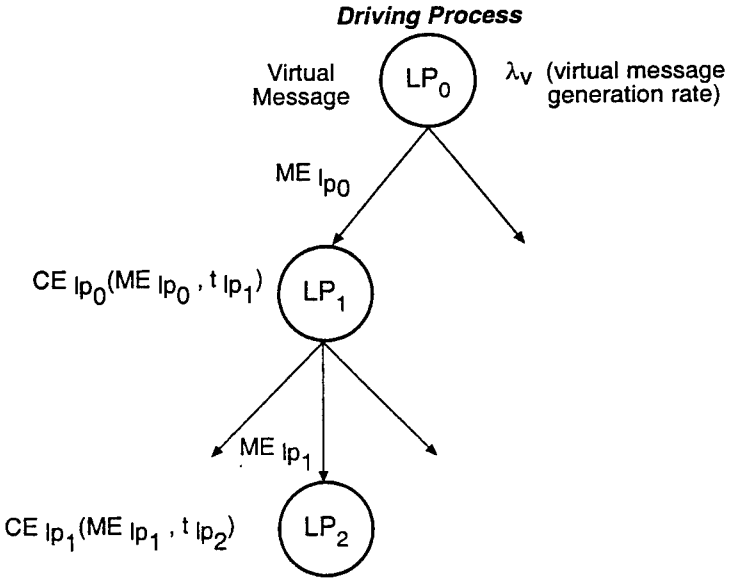


Figure 6.22. Accumulated Message Value Error.

As shown in Proposition 4, $AC_n(n)$ is the total accumulated error in the virtual message output by the n^{th} from the driving process. $AC_t(\tau)$ is the accumulated error in τ real time units from the generation of the initial virtual message from the driving process. Equation is $\lim_{\Sigma tlpi} \rightarrow \tau \sum_{i=1}^{n} AC_n(n)$, where n is the number of computations in time τ. In other words, $AC_t(\tau)$ is the error accumulated as messages pass through *n* Logical Processes in real time τ. For example, if a prediction result is generated in the third Logical Process from the driving process, then the total accumulated error in the result is $AC_n(3)$. If 10 represents the number of time units after the initial message was generated from the driving process, then $AC_t(10)$ would be the amount of total accumulated error in the result. A cache hit occurs when $|AC_t(\tau)| \leq \Theta$, where Θ is the tolerance associated with the last Logical Process required to generate the final result. Equations (6.43) and (6.44) provide a means of representing the amount of error in an Active Virtual Network Management Prediction generated result. Once an event has been predicted and results pre-computed and cached, it would be useful to know what the probability is that the result has been accurately calculated, especially if any results are committed before a real message arrives. The out-of-tolerance check and rollback does not occur until a real message arrives. If a resource is allocated ahead of time based on the predicted result, then this section has defined $\alpha = P[|AC_t(\Lambda)| > \Theta]$ where Θ is the Active Virtual Network Management Prediction tolerance associated with the last Logical Process required to generate the final result.

6.3.2 Bandwidth: β

The amount of overhead in bandwidth required by Active Virtual Network Management Prediction is due to virtual and anti-message load. With perfect prediction capability, there should be exactly one virtual message from the driving process for each real message. The inter-rollback time, $[1/(\lambda_{rb})]$, has been determined in Proposition 3, Equation 6.13. Virtual messages are arriving and generating new messages at a rate of λ_v. Thus, the worst case expected number of messages in the State Queue that will be sent as anti-messages is $[(\lambda_v)/(\lambda_{rb})]$ when a rollback occurs. The bandwidth overhead is shown in Equation 6.46, where λ_v is the virtual message load, λ_r is the real message load, and λ_{rb} is the expected rollback rate. The bandwidth overhead as a function of rollback rate is shown in Figure 6.23. Scalability in Active Virtual Network Management Prediction is the rate at which the proportion of rollbacks increases as the number of nodes increases. The graph in Figure 6.24 illustrates the tradeoff between the number of Logical Processes and the rollback rate given $\lambda_{vm} = 0.03$ virtual messages per millisecond, $\Delta_{vm} =$ 30.0 milliseconds, $\tau_{task} = 7.0$ milliseconds, $\tau_{rb} = 1.0$ milliseconds, $S_{parallel} = 1.5$ and $C_r = 100$ where C_r is the speedup gained from reading the cache over computing the result and Rm = [2/30 ms]. The rollback rate in this graph is the sum of both the out-of-order and the out-of-tolerance rollback rates.

Proposition 5

The expected bandwidth overhead is

$$\beta = \frac{\frac{\lambda_v}{\lambda_{rb}} + \lambda_v + \lambda_r}{\lambda_r} \tag{6.46}$$

where λ_{rb} is the expected rollback rate, λ_v is the expected virtual message rate, and λ_r is the expected real message rate.

6.3.3 Analysis of AVNMP Performance

Equation 6.47 shows the complete Active Virtual Network Management Prediction performance utility. The surface plot showing the utility of Active Virtual Network Management Prediction as a function of the proportion of out-of-tolerance messages is shown in Figure 6.25 where Φ_s, Φ_w, Φ_b are one and λ_{vm} = 0.03 virtual messages per millisecond, Δ_{vm} = 30.0 milliseconds, τ_{task} = 7.0 milliseconds, τ_{rb} = 1.0 milliseconds, $S_{parallel}$ = 1.5 and C_r = 100 where C_r is the speedup gained from reading the cache over computing the result. The wasted resources utility is not included in Figure 6.25 because there is only one level of message generation and thus no error accumulation. The y-axis is the relative marginal utility of speedup over reduction in bandwidth overhead SB = $[(\Phi_s)/(\Phi_b)]$. Thus if bandwidth reduction is much more important than speedup, the utility is low and the proportion of rollback messages would have to be kept below 0.3 per millisecond in this case. However, if speedup is the primary desire relative to bandwidth, the proportion of out-of-tolerance rollback message values can be as high as 0.5 per millisecond. If the proportion of out-of-tolerance messages becomes too high, the utility becomes negative because prediction time begins to fall behind real time.

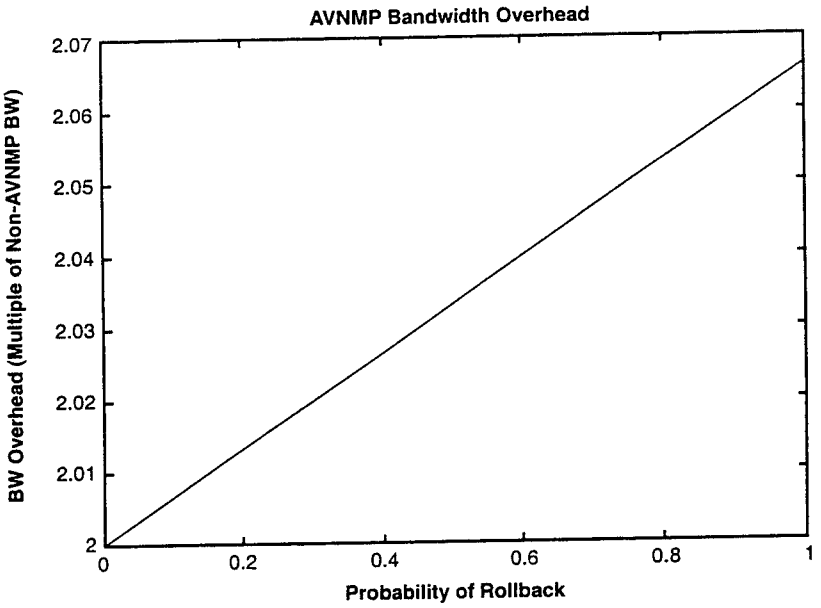


Figure 6.23. AVNMP Bandwidth Overhead.

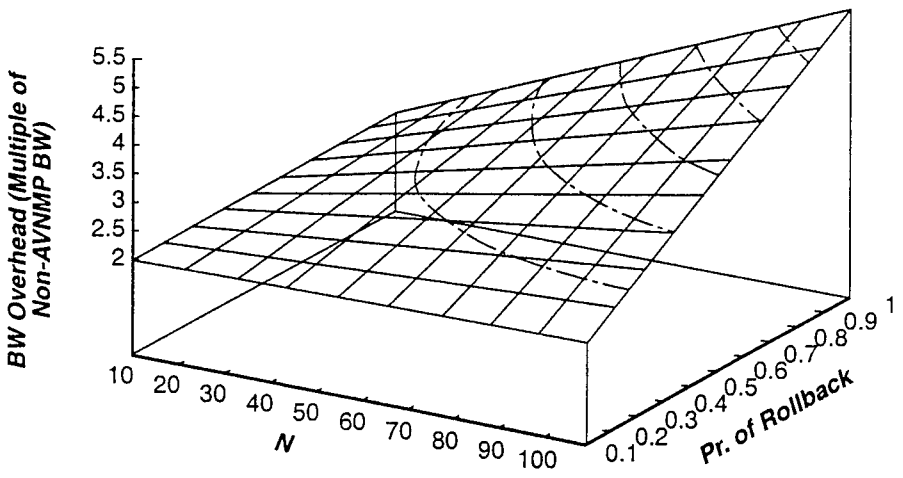


Figure 6.24. AVNMP Scalability.

The effect of the proportion of out-of-order and out-of-tolerance messages on Active Virtual Network Management Prediction speedup is shown in Figure 6.26. This graph shows that out-of-tolerance rollbacks have a greater impact on speedup than out-of-order rollbacks. The reason for the greater impact of the proportion of out-of-tolerance messages is that such rollbacks caused by such messages always cause a process to rollback to real time. An out-of-order rollback only requires the process to rollback to the previous saved state.

Figure 6.27 shows the effect of the proportion of virtual messages and expected lookahead per virtual message on speedup. This graph is interesting because it shows how the proportion of virtual messages injected into the Active Virtual Network Management Prediction system and the expected lookahead time of each message can affect the speedup. The real and virtual message rates are [0.1/ms], Rm = [2/30 ms], λ_{vm} = 0.03 virtual messages per millisecond, Δ_{vm} = 30.0 milliseconds, τ_{task} = 7.0 milliseconds, τ_{rb} = 1.0 milliseconds, $S_{parallel}$ = 1.5 and C_r = 100 where C_r is the speedup gained from reading the cache over computing the result.

$$U_{AVNMP} = \left(P_{cache\ X|X=E[X]}C_r + \left(P_{late\ X|X=E[X]} + P_{slow\ X|X=E[X]}\right)PR_{X,Y|X=E[X]}\right)$$

$$\Phi_s - P\left[\left|AC_t(\Lambda)\right| > \Theta\right]\Phi_w - \left(\frac{\frac{\lambda_v}{\lambda_{rb}} + \lambda_v}{\lambda_r}\right)\Phi_b \qquad (6.47)$$

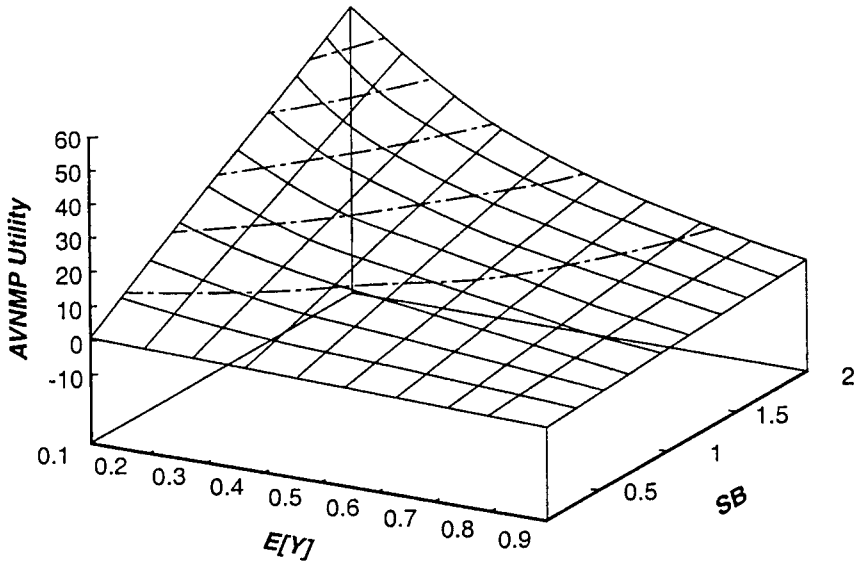


Figure 6.25. Overhead versus Speedup as a Function of Probability of Rollback.

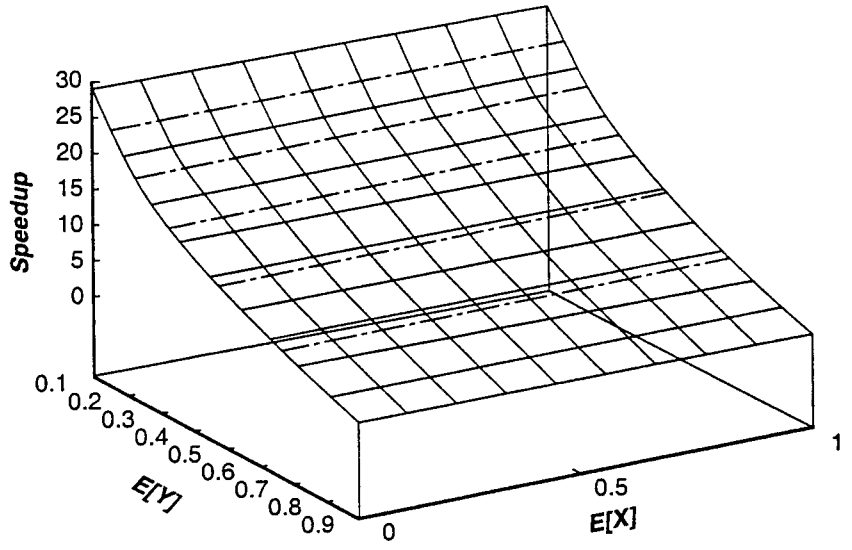


Figure 6.26. Effect of Non-Causality and Tolerance on Speedup.

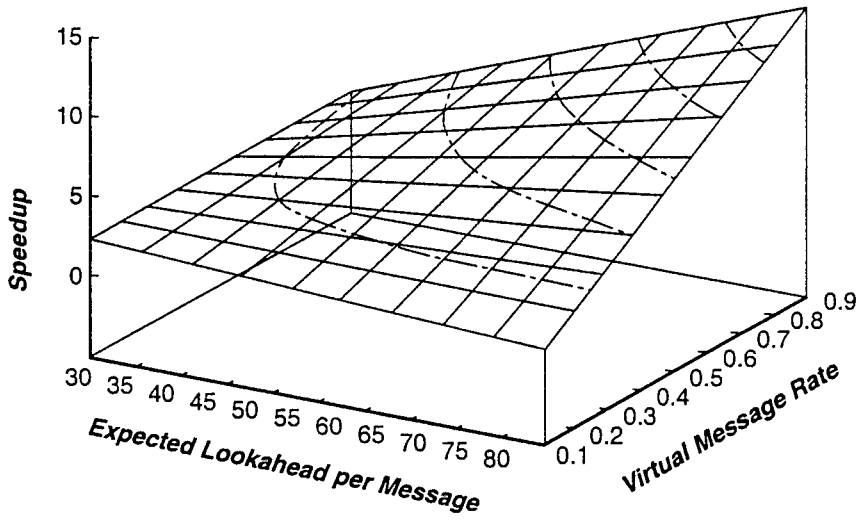


Figure 6.27. Effect of Virtual Message Rate and Lookahead on Speedup.

7

ASPECTS OF AVNMP PERFORMANCE

The following sections discuss other aspects and optimizations of the Active Virtual Network Management Prediction algorithm including handling multiple future events and the relevance of Global Virtual Time to Active Virtual Network Management Prediction. Since all possible alternative events cannot be predicted, only the most likely events are predicted in Active Virtual Network Management Prediction. However, knowledge of alternative events with a lower probability of occurrence allow the system to prepare more intelligently.

Another consideration is the calculation of Global Virtual Time. This requires bandwidth and processing overhead. A bandwidth optimization is suggested in which real packets may be sent less frequently.

7.1 MULTIPLE FUTURE EVENTS

The architecture for implementing alternative futures discussed in Section 7, while a simple and natural extension of the Active Virtual Network Management Prediction algorithm creates additional messages and increases message sizes. Messages require an additional field to identify the probability of occurrence and an event identifier. However, the Active Virtual Network Management Prediction tolerance is shown to provide consideration of events that fall within the tolerances Θ_n where $n \in N$ and N is the number of Logical Processes.

The set of possible futures at time t is represented by the set E. A message value generating an event occurring in one of the possible futures is represented by E_{val}. As messages propagate through the Active Virtual Network Management Prediction system, there is a neighborhood around each message value defined by the tolerance (Θ_n). However, each message value also accumulates error ($AC_n(n)$). Let the neighborhood (E_Δ) be defined such that $E_\Delta \leq |\Theta_n - AC_n(n)|$ for each $n \in \{LPs\}$. Thus, $|E_\Delta + AC_n(n)| \leq \min_{n \in N} \Theta_n$ defines a valid prediction. The infinite set of events in the neighborhood $E_\Delta \leq |\min_{n \in N} \Theta_n - AC_n(n)|$ are valid. Therefore, multiple future events that fall within the bounds of the tolerances reduced by any accumulated error can be implicitly considered.

7.2 GLOBAL VIRTUAL TIME

In order to maintain the lookahead (Λ), for the entire configuration system, it is necessary to know how far into the future the system is currently predicting. The purpose of Global Virtual Time is to determine Λ where Λ is used to stop the Active Virtual Network Management Prediction system from looking ahead once the system has predicted up to the lookahead time. This helps maintain synchronization and saves processing and bandwidth since it is not necessary to continue the prediction process indefinitely into the future, especially since the prediction process is assumed to be less accurate the further it predicts into the future.

Distributed simulation mechanisms require Global Virtual Time in order to determine when to commit events. This is because the simulation cannot rollback beyond Global Virtual Time. In Active Virtual Network Management Prediction, event results are assumed to be cached before real time reaches the Local Virtual Time of a Logical Process. The only purpose for Global Virtual Time in Active Virtual Network Management Prediction is to act as a throttle on computation into the future. Thus, the complexity and overhead required to accurately determine the Global Virtual Time is unnecessary in Active Virtual Network Management Prediction. In the Active Virtual Network Management Prediction system, while the Local Virtual Time of a Logical Process is greater than $t + \Lambda$, the Logical Process does not process virtual messages.

The Global Virtual Time update request packets have the intelligence to travel only to those logical processes most likely to contain a global minimum. An example is shown in Figure 7.1.

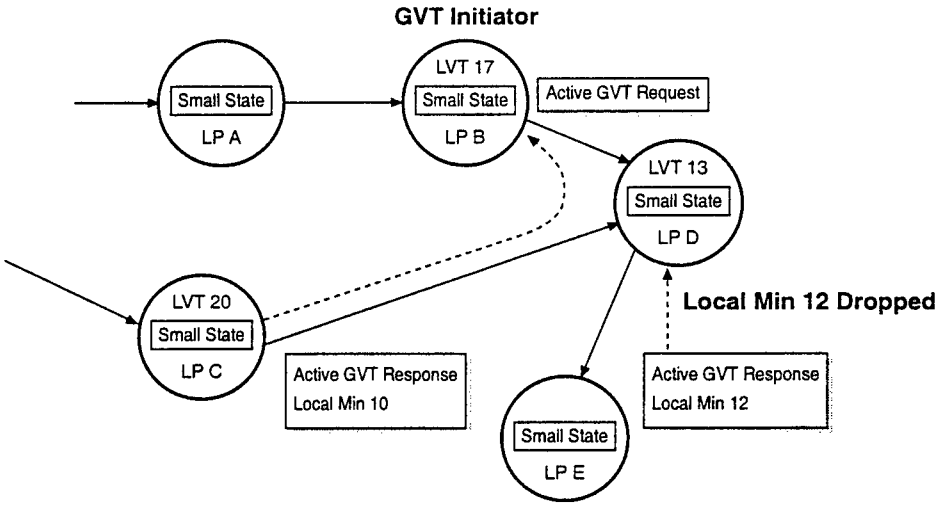


Figure 7.1. Active Global Virtual Time Calculation Overview.

The Active Request packet notices that the logical process with a Global Virtual Time of 20 is greater than the last logical process that the Active Request packet passed through and thus destroys itself. This limits the amount of unnecessary traffic and computation. The nodes that receive the Active Request packet forward the result to the initiator. As the Active Response packets return to the initiator, the last packet is maintained in the cache of each logical process. If

the value of the Active Response packet is greater than or equal to the value in the cache, then the packet is dropped. Again, this reduces the amount of traffic and computation that must be performed.

7.3 REAL MESSAGE OPTIMIZATION

Real messages are only used in the Active Virtual Network Management Prediction algorithm as a verification that a prediction has been accurate within a given tolerance. The driving process need not send a real message if the virtual messages are within the lowest tolerance in the path of a virtual message. This requires that the driving process have knowledge of the tolerance of the destination process. The driving process has copies of previously sent messages in its send queue. If real messages are only sent when an out-of-tolerance condition occurs, then the bandwidth can be reduced by up to 50%. Figure 7.2 compares the bandwidth with and without the real message optimization.

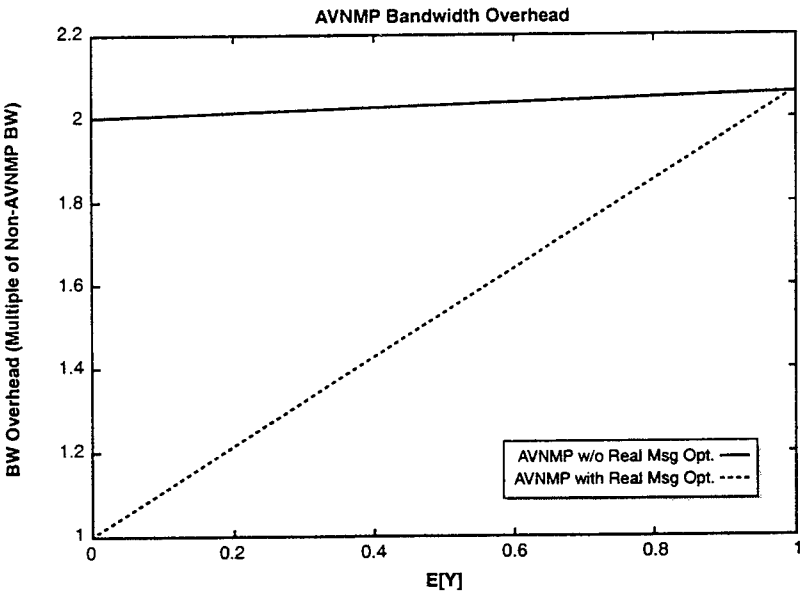


Figure 7.2. Bandwidth Overhead Reduction.

The performance analysis of Active Virtual Network Management Prediction has quantified the costs versus the speedup provided by Active Virtual Network Management Prediction. The costs have been identified as the additional bandwidth and possible wasted resources due to inaccurate prediction. Since the Active Virtual Network Management Prediction algorithm combines optimistic synchronization with a real time system, the probability of non-causal message order was determined. A new approach using Petri-Nets and synchronic distance determined the likelihood of out-of-order virtual messages. The speedup was defined as the expected rate of change

of the Local Virtual Time with respect to real time. The speedup was quantified and a sensitivity analysis revealed the parameters most affecting speedup. The bandwidth was quantified based on the probability of rollback and the expected rollback rate of the Active Virtual Network Management Prediction system. A general analysis of the accumulated error of the Active Virtual Network Management Prediction system followed with the probability of error in the active network. Finally, the consideration of alternative future events, the relevance of Global Virtual Time, and a bandwidth technique were discussed.

Active networks enable an exciting new paradigm for communications. This paradigm facilitates the use of data transmission and computation in ways unimaginable in legacy networks. Hopefully the information provided in this report will give the reader a running start in understanding this new technology and generate new ideas in the reader's mind for novel applications of this technology.

7.4 COMPLEXITY IN SELF-PREDICTIVE SYSTEMS

A fascinating perspective on the topic of self-predictive systems is found in *Gödel, Escher, and Bach: An Eternal Golden Braid*, which is a wonderful look at the nature of Human and Artificial Intelligence. A central point in (Hofstadter, 1980) is that intelligence is a Tangled Hierarchy, illustrated in the famous Escher drawing of two hands – each drawing the other. A hand performing the act of drawing is expected to be a level above the hand being drawn. When the two levels are folded together, a Tangled Hierarchy results, an idea which is expressed much more elegantly in (Hofstadter, 1980). Active Virtual Network Management Prediction as presented in this work is a Tangled Hierarchy on several levels: simulation-reality and also present-future time. One of the hands in the Escher drawing represents prediction based on simulation and the other represents reality, each modifying the other in the Active Virtual Network Management Prediction algorithm. However, there is a much deeper mathematical relationship present in this algorithm that relates to Gödel's Theorem. In a nutshell, Gödel's theorem states that no formal system can describe itself with complete fidelity. This places a formidable limitation on the ability of mathematics to describe itself. The implication for artificial intelligence is that the human mind can never fully understand its own operation, or possibly that if one could fully understand how one thinks while one is thinking, then one would cease to "be." In the much more mundane Active Virtual Network Management Prediction algorithm, a system is in some sense attempting to use itself to predict its own future state with the goal of perfect fidelity. If Gödel's Theorem applies, then perfect fidelity is an impossible goal. However, by allowing for a given tolerance in the amount of error and assuming accuracy in prediction which increases as real time approaches the actual time of an event, this study assumes that a useful self-predictive system can be implemented.

In the course of efforts to fully utilize the power of active networks to build a self-managing communications network, the nature of entanglement and the relationship between modeling and communication becomes of utmost importance. This section provides a general overview of the goal that Active Virtual Network Management Prediction is trying to accomplish as well as its evolution as resources increase; that is, how does such a self-predictive system behave as processing and bandwidth become ever larger and more powerful. An attempt is made to identify new theories required to understand such highly self-predictive systems.

7.4.1 Near-Infinite Resources

Now, imagine stepping across a discontinuity into a world where computing power, bandwidth, and computational ubiquity are nearly infinite. Our vision focuses on effects that near-perfect self-prediction would have upon such a world. First we would have near-perfect optimization of resources since local minima could be pushed far into the horizon. Second, currently wasted effort could be avoided, since the outcome of any action could be determined with very precise limits. Critical missing elements are a theory and applications involving highly predictive systems and components. Further study is needed to explore the exciting new world of near-perfect self-prediction and the relationship between highly predictive systems and communications in particular. Figure 7.3 shows an abstract view of computers embedded within almost all devices. Current engineering organizes computing devices in such a way as to optimize communications performance. In our hypothetical world of near-perfect predictive capabilities, direct communication is less important and, in many cases, no longer required, as discussed later. Instead, computational organization is based on forming systems or islands of near-perfect self-prediction. As shown in Figure 7.4, self-predictive capability is used to enhance the performance of the system, which in turn improves the predictive capability, which again improves the performance of the system, ad infinitum, driving the error towards zero.

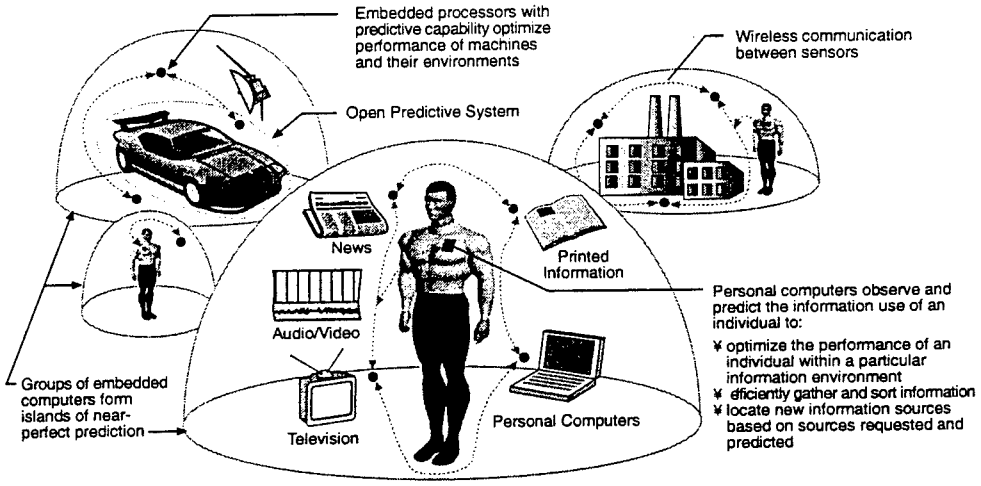


Figure 7.3. Computational organization is based on forming systems or islands of near-perfect self-prediction.

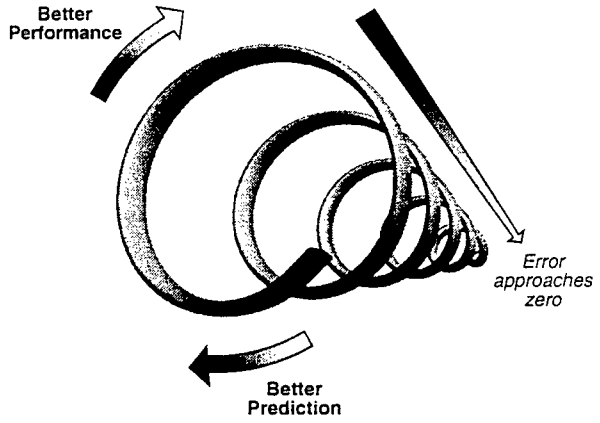


Figure 7.4. This predictive capability is used to drive the error toward zero.

Why do we assume rather than perfect prediction and why do we assume islands rather than perfect prediction everywhere? Clearly, perfect prediction everywhere would take us into a deterministic world where the final outcome of all choices would be known to everyone and the optimal choice could be determined in all cases. In this project it is assumed that limits, however small, exist, such as lack of knowledge about quantum state or of the depths of space. In order to study near-perfect self-predictive islands, the characteristics of such islands need to be identified. It would appear that closed self-predictive islands would be the easiest to understand. The scope of closed self-predictive islands includes all driving forces acting upon the system. Imagine that one has full knowledge of the state of a room full of ping-pong balls and their elasticity. This information can be used to predict the position of the balls at any point in time. However, one is external to the room. The goal is for the balls to predict their own behavior as illustrated in the inner sphere of Figure 7.5. If elasticity represents the dynamics of communication endpoint entities A and B, and movement of the ping-pong balls represents communication, then any exchange of information between A and B is unnecessary since it can be perfectly predicted. Instead of transmitting messages between A and B, an initial transmission of the dynamics of A and B is transmitted to each other, perhaps as active packets within an active network environment. Thus a near-perfect self-predictive island is turned inward upon itself as shown in Figure 7.6. In an active network environment, an executable model can be included within an active packet. When the active packet reaches the target intermediate device, the load model provides virtual input messages to the logical process and the payload of the virtual message is passed to the actual device.

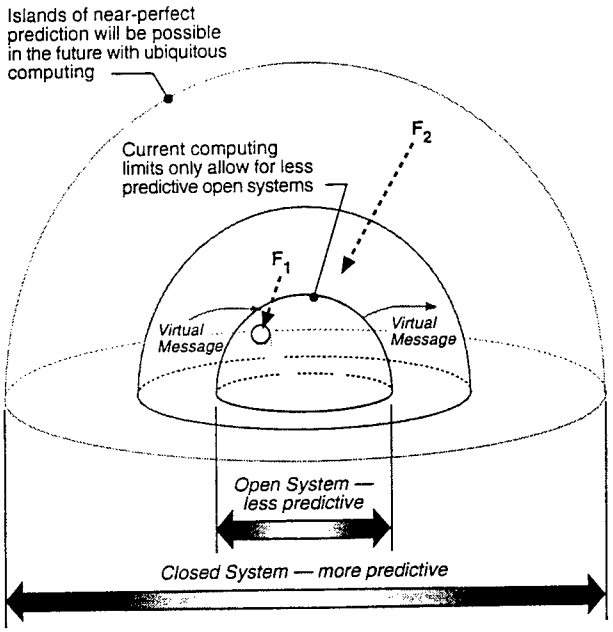


Figure 7.5. Self-predictive islands can improve prediction fidelity by expanding to incorporate more elements.

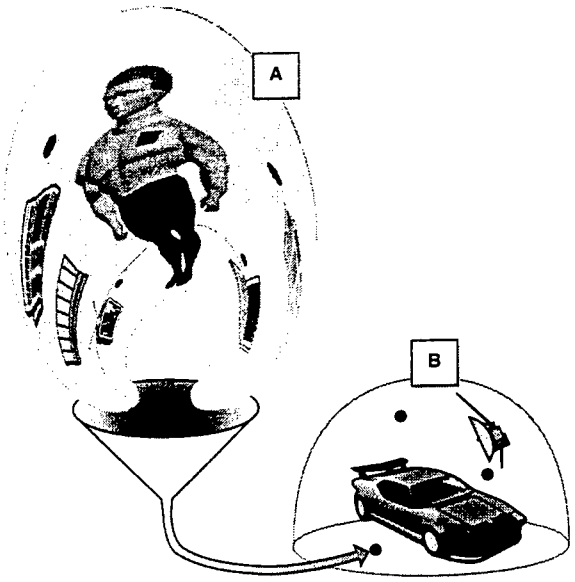


Figure 7.6. Direct communication between A and B is unnecessary as the dynamics of A can be transmitted to B, allowing B to interact with a near-perfect model of A.

Open self-predictive islands will contain inaccuracies in prediction because, by definition, open self-predictive islands include the effects of unknown driving forces upon entities within the of the system. Figure 7.5 shows a force (F_1) acting on the inner system. F_1 is external to the

inner system because it is not included within the system itself or in the virtual messages passed into the system. The system could become closed by either enlarging the scope to include the driving forces within the system, as shown in the figure, or by accepting a level of inaccuracy in the system. Thus we can imagine many initial points of near-perfect self-predictive islands, each attempting to improve prediction fidelity by expanding to incorporate more elements. These are the islands of near-perfect self-prediction.

Recursion is a recurring theme in this work. For example, assume that the inner near-perfect self-predictive island in Figure 7.5 is a wireless mobile communications system and F_1 is the weather. Now assume that ubiquitous computing can be used to include weather observation and prediction, for example, computers within planes, cars, spacecraft, etc. The heat from the circuitry of the wireless system, even though negligible, could have an impact on the weather. This is known as the butterfly effect in Chaos Theory. In recent years the study of chaotic nonlinear dynamical systems has led to diverse applications where chaotic motions are described and controlled into some desirable motion. Chaotic systems are sensitive to initial condition. Researchers now realize that this sensitivity can also facilitate control of system motion. For example, in communications, chaotic lasers have been controlled, as have chaotic diode resonator circuits (Aronson et al., 1994, DiBernardo, 1996). Hence, studying the effects of external forces controlling a chaotic system has become a very important goal and should be a subject for research. By allowing for a given tolerance in the amount of error and assuming accuracy in prediction that increases as real time approaches the actual time of an event, this study assumes that a useful near-perfect self-predictive island can be implemented. The Active Virtual Network Management Prediction project attempts to embed predictive capability within an active network using a self-adjusting Time Warp based mechanism for prediction propagation. This self-adjusting property has been found to be useful in prediction and is referred to as autoanaplasis. In addition to autoanaplasis, it is well known that such systems sometimes exhibit super-criticality, faster than critical path execution. However, due to limited and non-ubiquitous computational power in current technology, prediction inaccuracy causes rollbacks to occur. In a world of near-infinite bandwidth and computing power, the cost of a rollback to a "safe" time becomes infinitesimal. This is one of the many new ideas this project will explore involving the relationship between bandwidth, computing power, and prediction. Given near-infinite bandwidth, the system state can be propagated nearly instantaneously. With nearly infinite and ubiquitous computing, driving processes can be developed with near-perfect accuracy. Let us define near-perfect accuracy of our self-adjusting Time Warp based system in the presence of rollback as the characteristic that a predicted state value (V_v) approaches the real value (V_r) as t approaches $GVT(t_1)$ very quickly, where $GVT(t_1)$ is the Global Virtual Time of the system at time t_1. Explicitly, this is, $\forall \varepsilon > 0, \exists \delta > 0$ s.t. $|f(t) - f(GVT(t_1))| < \varepsilon \rightarrow 0 < |GVT(t_1) - t|$ where $f(t) = V_r$ and $f(GVT(t_1)) = V_v$. $f(t)$ is the prediction function. The effect of should not be ignored. These values are described in more detail in Section5.

7.4.2 Performance Of Near-Perfect Self-Predictive Islands

One focus of study is on the interfaces between systems with various levels of predictive capability. The self-predictive islands formed in Figure 7.3 have various degrees of prediction capability. Our recent theoretical results from the Active Virtual Network Management project indicate that self-predictive islands exhibit high degrees of performance when prediction is accurate, but are brittle when the tolerance for inaccuracy is reached. With respect to network performance as enhanced with Active Virtual Network Management Prediction, systems with little

or no prediction capability appear to be ductile, as they are much better able to tolerate prediction inaccuracy, as shown in Figure 7.8. In other words, performance is moderate, but there are no sudden degradations in performance. This compares favorably to a system with a large lookahead and sudden, near catastrophic degradations in performance.

Thus, an obvious question arises as to what is the optimal grouping of predictive components within a system. What happens when the slope shown in Figure 7.8 becomes nearly vertical? The lookahead into the future is tremendously large in some self-predictive islands and smaller in others. If the lookahead is small in a self-predictive island that feeds into a large lookahead system, then large rollbacks are likely to occur. One focus of study is on the interfaces between systems with various levels of predictive capability and the associated "index of refraction" of performance through the interfaces between islands of near-perfect self-prediction.

Brittle behavior of near-perfect self-predictive islands is shown by point D along curve P_h in Figure 7.9. P_h is the performance curve for a high-performance system with brittle characteristics; P_l is a lower-performance system with ductile characteristics. Clearly, the slope from point D along curve P_h is much steeper than that of point E along curve P_l. The steep decline of performance along P_h can be caused by input parameters that exceed a specified tolerance, or by environmental conditions that exceed specified operating boundaries.

Materials Science	*Near-Perfect Prediction Systems*
Brittle Behavior	Sudden steep decline in performance
Ductile Behavior	Graceful degradation in performance
Stress	Amount parameter exceeds its tolerance
Toughness	System robustness
Hardness	Level of performance within tolerance
Ductility	Level of performance outside of tolerance
Plastic Strain	Degradation from which system cannot recover
Elastic Strain	Degradation from which system can recover
Brittleness	Ratio of hardness over ductility
Deformation	Degradation in performance
Young's Modulus	Amount tolerance is exceeded over degradation

Figure 7.7. Terms Borrowed from Materials Science.

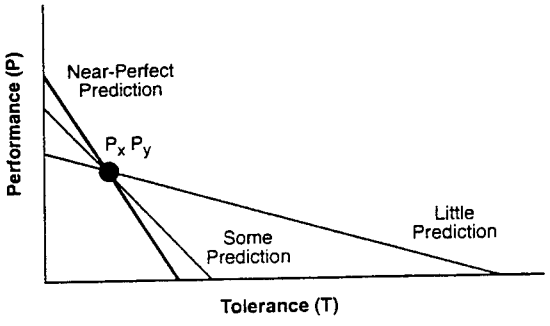


Figure 7.8. Performance of Self-predictive Islands.

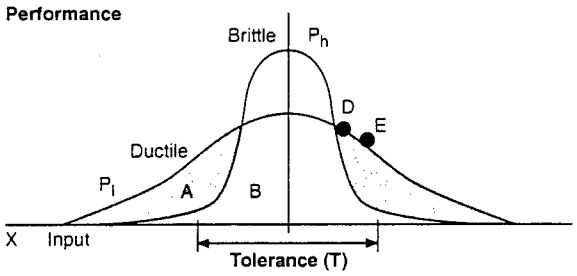


Figure 7.9. A Brittle vs. Ductile System.

Consider a system whose self-predictive islands exhibit various degrees of ductility as defined above. Just as adding impurities to a pure metal causes it to become stronger but more brittle, the addition of more efficient but also more sensitive components to a system, such as a near-perfect self-prediction system, causes the system to increase performance within its operating range, but become less ductile. How do the effects of ductility propagate among the self-predictive islands to influence the ductility of the entire system? Assume the performance response curve is known for each self-predictive island and that the output from one component feeds into the input of the next component as shown in Figure 7.10. The self-predictive islands are labeled S_n and the performance curves as a function of tolerance for error are shown in the illustration immediately above each island. More fundamental research is needed to carry forward this analogy and deliver a theory and models of the relationships among computing, communications, and near-perfect self-prediction.

7.5 SUMMARY

The primary conclusion is that further research is required to understand the nature of entanglement, causality, and the relationship between modeling and communications. For example, Active Network Management Prediction uses a model within a network to enhance the network performance to improve the model's own performance, which thus improves the network's performance thus enhancing the model's performance ad infinitum as shown in Figure 7.4. Furthermore, the Active Virtual Network Management Prediction mechanism uses a Time Warp-like method to ensure causality, yet there is something non-causal about the way Active Virtual Network Management Prediction uses future events to optimize current behavior. This entanglement issue resonates with physicists and those studying the nature of agent autonomy as evident in numerous conferences. Clearly, this needs to be explored in a much deeper manner. Also, formation of islands of near-perfect self-prediction and the need to study the interfaces between those islands was discussed. The idea of wrappers and integration spaces as introduced in (Christopher Landauer and Kirstie L. Bellman, 1996) is likely to provide insight into bringing together complex system components in a self-organizing manner. Another suggestion for the study of predictive interfaces is in a tolerance interaction space (Landauer and Bellman, 1996).

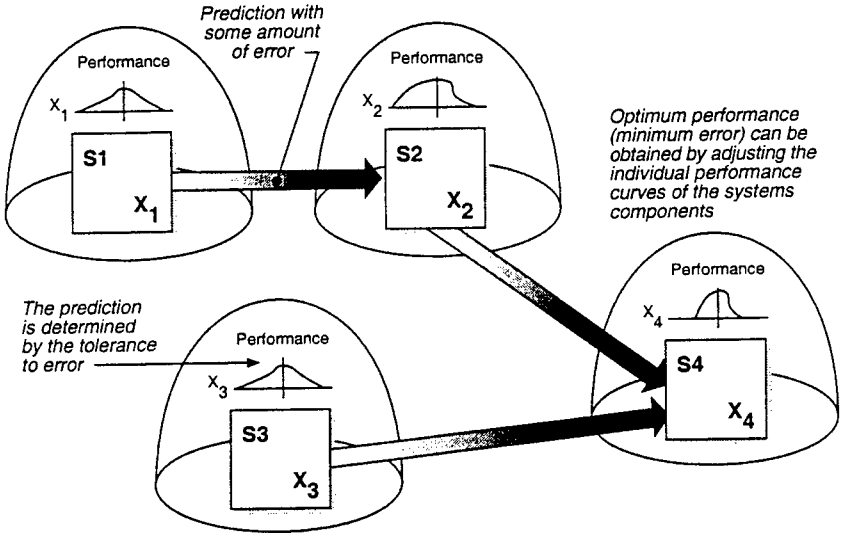


Figure 7.10. Brittle Subsystem Components.

Appendix : AVNMP SNMP MIB

A diagram of the Active Virtual Network Management Prediction SNMP Management Information Base is shown in Figure 7.A.1. This diagram is the authors' interpretation of a Case Diagram, showing the relationship between the primary MIB objects. Many of the MIB objects are for experimental purposes; only the necessary and sufficient SNMP objects based on the authors' experience are included in the Case Diagram. In Figure 7.A.1, the AVNMP process, not shown, can be thought of as being on the top of the figure and the network communication mechanism (not shown) on the bottom of the figure. The vertical arrows illustrate the main path of information flow between the AVNMP process and the underlying network. Lines that cross the main flows indicate counters that accumulate information as each packet transitions between the network and the AVNMP process. Arrows that extend from the main flow are counters where packets are removed from the main flow. The complete AVNMP version 1.1 MIB follows and is included on the CD inmib-avnmp.txt.

```
AVNMP-MIB DEFINITIONS ::= BEGIN

IMPORTS
  MODULE-IDENTITY, OBJECT-TYPE, experimental,
  Counter32, TimeTicks
  FROM SNMPv2-SMI
  DisplayString
    FROM SNMPv2-TC;

avnmpMIB MODULE-IDENTITY                                        10
   LAST-UPDATED "9801010000Z"
   ORGANIZATION "GE CRD"
   CONTACT-INFO
        "Steve Bush bushsf@crd.ge.com"
   DESCRIPTION
        "Experimental MIB modules for the Active Virtual Network
        Management Prediction (AVNMP) system."
   ::= { experimental active(75) 4 }

--
-- Logical Process Table                                        20
--

lP OBJECT IDENTIFIER ::= { avnmpMIB 1 }

lPTable OBJECT-TYPE
      SYNTAX SEQUENCE OF LPEntry
      MAX-ACCESS not-accessible
      STATUS current
      DESCRIPTION                                               30
            "Table of AVNMP LP information."
      ::= { lP 1 }

lPEntry OBJECT-TYPE
      SYNTAX LPEntry
      MAX-ACCESS not-accessible
      STATUS current
```

```
    DESCRIPTION
        "Table of AVNMP LP information."
    INDEX { lPIndex }
    ::= { lPTable 1 }

LPEntry ::= SEQUENCE {
    lPIndex         INTEGER,
    lPID            DisplayString,
    lPLVT           INTEGER,
    lPQRSize        INTEGER,
    lPQSSize        INTEGER,
    lPCausalityRollbacks INTEGER,
    lPToleranceRollbacks INTEGER,
    lPSQSize        INTEGER,
    lPTolerance     INTEGER,
    lPGVT           INTEGER,
    lPLookAhead     INTEGER,
    lPGvtUpdate     INTEGER,
    lPStepSize      INTEGER,
    lPReal          INTEGER,
    lPVirtual       INTEGER,
    lPNumPkts       INTEGER,
    lPNumAnti       INTEGER,
    lPPredAcc       DisplayString,
    lPPropX         DisplayString,
    lPPropY         DisplayString,
    lPETask         DisplayString,
    lPETrb          DisplayString,
    lPVmRate        DisplayString,
    lPReRate        DisplayString,
    lPSpeedup       DisplayString,
    lPLookahead     DisplayString,
    lPNumNoState    INTEGER,
    lPStatePred     DisplayString,
    lPPktPred       DisplayString,
    lPTdiff         DisplayString,
    lPStateError    DisplayString,
    lPUptime        TimeTicks
    }

lPIndex OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The LP table index."
    ::= { lPEntry 1 }

lPID OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The LP identifier."
    ::= { lPEntry 2 }

lPLVT OBJECT-TYPE
```

```
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
                "This is the LP Local Virtual Time."
        ::= { lPEntry 3 }                                          100

lPQRSize OBJECT-TYPE
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
                "This is the LP Receive Queue Size."
        ::= { lPEntry 4 }

lPQSSize OBJECT-TYPE                                               110
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
                "This is the LP send queue size."
        ::= { lPEntry 5 }

lPCausalityRollbacks OBJECT-TYPE
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only                                       120
        STATUS current
        DESCRIPTION
                "This is the number of rollbacks this LP has suffered."
        ::= { lPEntry 6 }

lPToleranceRollbacks OBJECT-TYPE
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION                                                130
                "This is the number of rollbacks this LP has suffered."
        ::= { lPEntry 7 }

lPSQSize OBJECT-TYPE
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
                "This is the LP state queue size."
        ::= { lPEntry 8 }                                          140

lPTolerance OBJECT-TYPE
        SYNTAX INTEGER (0..2147483647)
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
                "This is the allowable deviation between process's
        predicted state and the actual state."
        ::= { lPEntry 9 }
                                                                   150
```

```
lPGVT OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is this system's notion of Global Virtual Time."
    ::= { lPEntry 10 }

lPLookAhead OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is this system's maximum time into which it can
    predict."
    ::= { lPEntry 11 }

lPGvtUpdate OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is the GVT update rate."
    ::= { lPEntry 12 }

lPStepSize OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is the lookahead (Delta) in milliseconds for each
     virtual message as generated from the driving process."
    ::= { lPEntry 13 }

lPReal OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is the total number of real messages received."
    ::= { lPEntry 14 }

lPVirtual OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This is the total number of virtual messages
     received."
    ::= { lPEntry 15 }

lPNumPkts OBJECT-TYPE
    SYNTAX INTEGER (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
```

"This is the total number of all AVNMP packets
received."
::= { lPEntry 16 }

lPNumAnti **OBJECT**-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the total number of Anti-Messages transmitted
by this Logical Process."
::= { lPEntry 17 }

lPPredAcc **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the prediction accuracy based upon time
weighted average of the difference between predicted and real
values."
::= { lPEntry 18 }

lPPropX **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the proportion of out-of-order messages
received at this Logical Process."
::= { lPEntry 19 }

lPPropY **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the proportion of out-of-tolerance messages
received at this Logical Process."
::= { lPEntry 20 }

lPETask **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected task execution wallclock time for this
Logical Process."
::= { lPEntry 21 }

lPErb **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected wallclock time spent performing a
rollback for this Logical Process."

::= { lPEntry 22 }

lPVmRate **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the rate at which virtual messages were processed by this Logical Process."
::= { lPEntry 23 }

lPReRate **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time until next virtual message."
::= { lPEntry 24 }

lPSpeedup **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the speedup, ratio of virtual time to wallclock time, of this logical process."
::= { lPEntry 25 }

lPLookahead **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected lookahead in milliseconds of this Logical Process."
::= { lPEntry 26 }

lPNumNoState **OBJECT**-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the number of times there was no valid state to restore when needed by a rollback or when required to check prediction accuracy."
::= { lPEntry 27 }

lPStatePred **OBJECT**-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the cached value of the state at the nearest time to the current time."
::= { lPEntry 28 }

lPPktPred **OBJECT**-TYPE

```
        SYNTAX DisplayString
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
            "This is the predicted value in a virtual message."
        ::= { lPEntry 29 }

lPTdiff OBJECT-TYPE
        SYNTAX DisplayString
        MAX-ACCESS read-only                                    330
        STATUS current
        DESCRIPTION
            "This is the time difference between a predicted and an
         actual value."
        ::= { lPEntry 30 }

lPStateError OBJECT-TYPE
        SYNTAX DisplayString
        MAX-ACCESS read-only
        STATUS current                                          340
        DESCRIPTION
            "This is the difference between the contents of an application
         value and the state value as seen within the virtual message."
        ::= { lPEntry 31 }

lPUptime OBJECT-TYPE
        SYNTAX DisplayString
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION                                             350
            "This is the time in milliseconds that AVNMP has been
        running on this node."
        ::= { lPEntry 32 }

END
```

8

AVNMP EXPERIMENTAL VERIFICATION

This chapter discusses the experimental validation of the Active Virtual Network Management Prediction Algorithm (AVNMP). The general operation is illustrated in the following four graphs. The bold red curves emphasize expected trends in operation. Figure 8.1 shows the reduction in tolerance versus time that is pre-programmed into each Logical Process. This is done in order to create a greater demand over time for accuracy and thus create a challenging validation of the AVNMP system under gradually increasing stress. In Figure 8.2 the proportion of out-of-tolerance messages is shown as a function of wallclock time. As wallclock time progresses, the tolerance is purposely reduced, causing a greater likelihood of messages exceeding the tolerance. This is done in order to validate the performance of the system as stress, in the form of greater demand for accuracy, is increased. Figure 8.3 shows the prediction error as a function of wallclock time. This graph verifies that the system is producing more accurate predictions as the demand for accuracy increases. However, Figure 8.4 shows the Lookahead decreasing versus wallclock time. The demand for greater accuracy has reduced the distance into the future that the system can predict. Finally, in Figure 8.5, the speedup, which is the virtual time versus wallclock time of the real system, is shown as a function of wallclock time. The speedup is reduced as the demand for accuracy is increased. These graphs serve to show the salient features of AVNMP operation; more detailed results under various conditions follow in this chapter. In the sections that follow, the network management framework of which AVNMP is a part is explained in order to describe the system and its effect upon data collection. Then a comparison and contrast with the analytical results is presented for the case of two different topological AVNMP configurations.

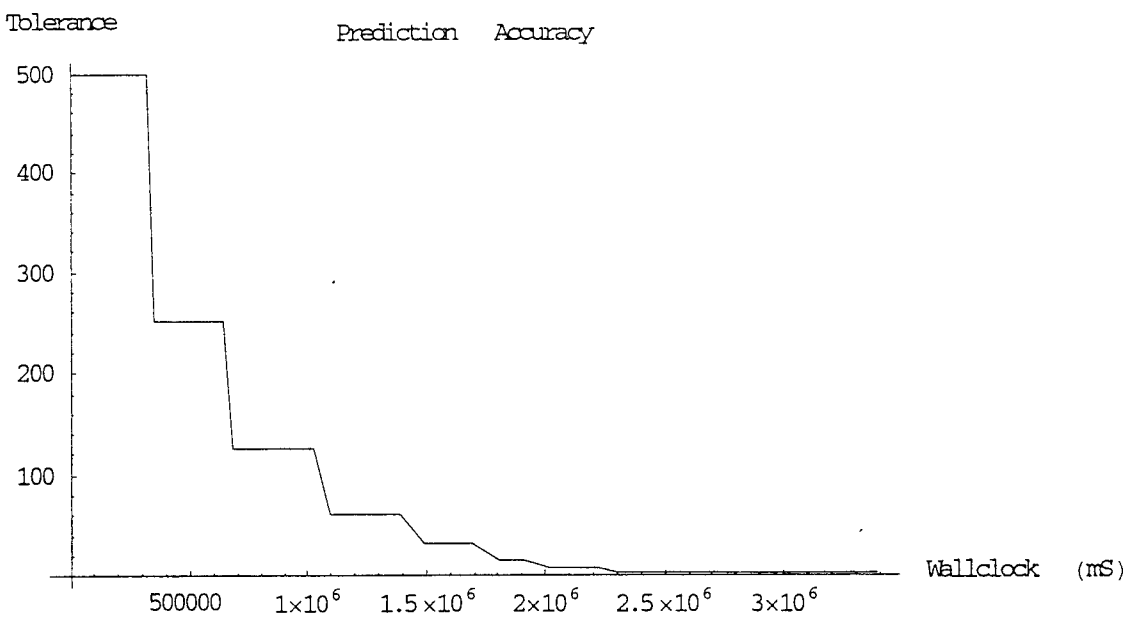


Figure 8.1 Tolerance Setting Decreases as Wallclock Increases Thus Demanding Greater Accuracy...

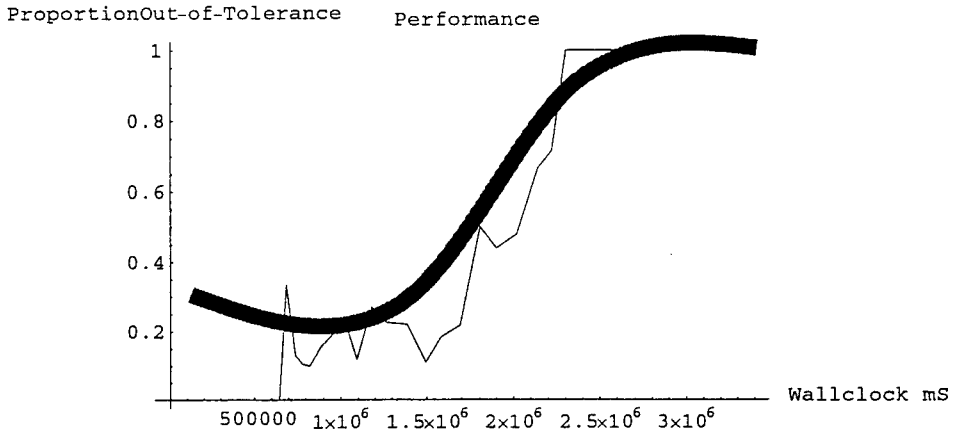


Figure 8.2 ...This Causes the Proportion of Out-of-Tolerance Messages to Increase Due to Greater Demand for Accuracy.

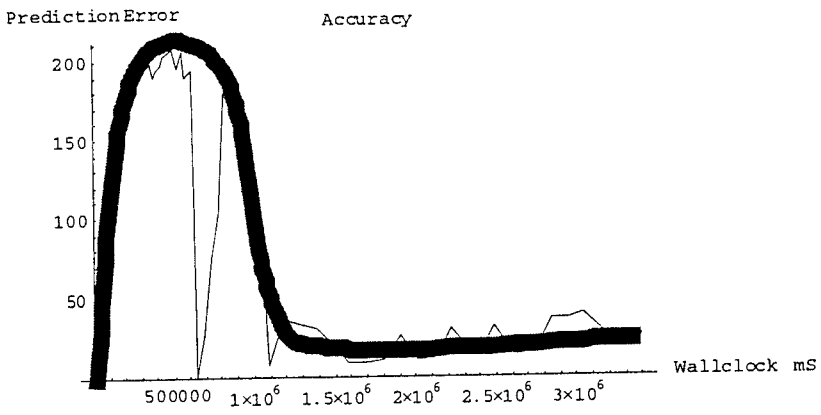


Figure 8.3 ...Predictions Become More Accurate.

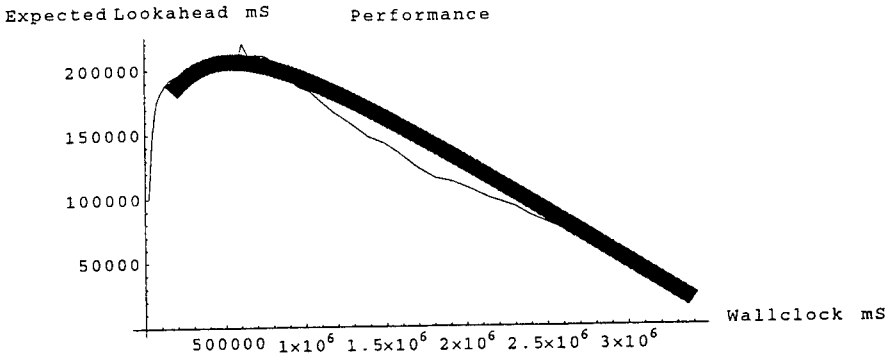


Figure 8.4 ...At the Expense of Lookahead.

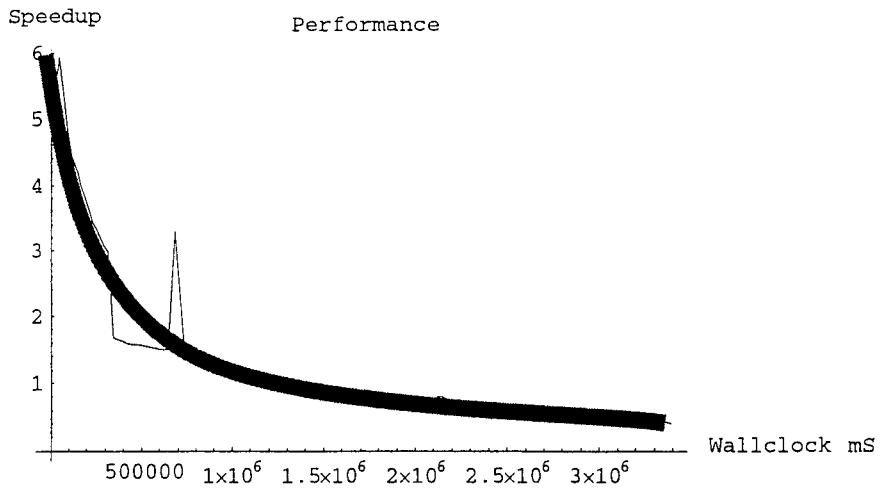


Figure 8.5 ...and Speedup.

8.1 Experimental Environment and Data Collection

The experimental performance data collection takes place in a network with the topology shown in Figure 8.6. The boxes represent active nodes, the lines represent links, and the numbers identify ports. The nodes are Sun Sparcs running the Solaris operating system and the Magician active network execution environment. Figure 8.7 illustrates the framework that is being used to instrument the system with management capability. SmallState is used as a rendezvous location for management between the active SNMP agents and the management clients. A Magician Active Application implemented as a Java class interface collects management from other Magician applications and from the internal Magician Execution Environment and provides an SNMP agent interface.

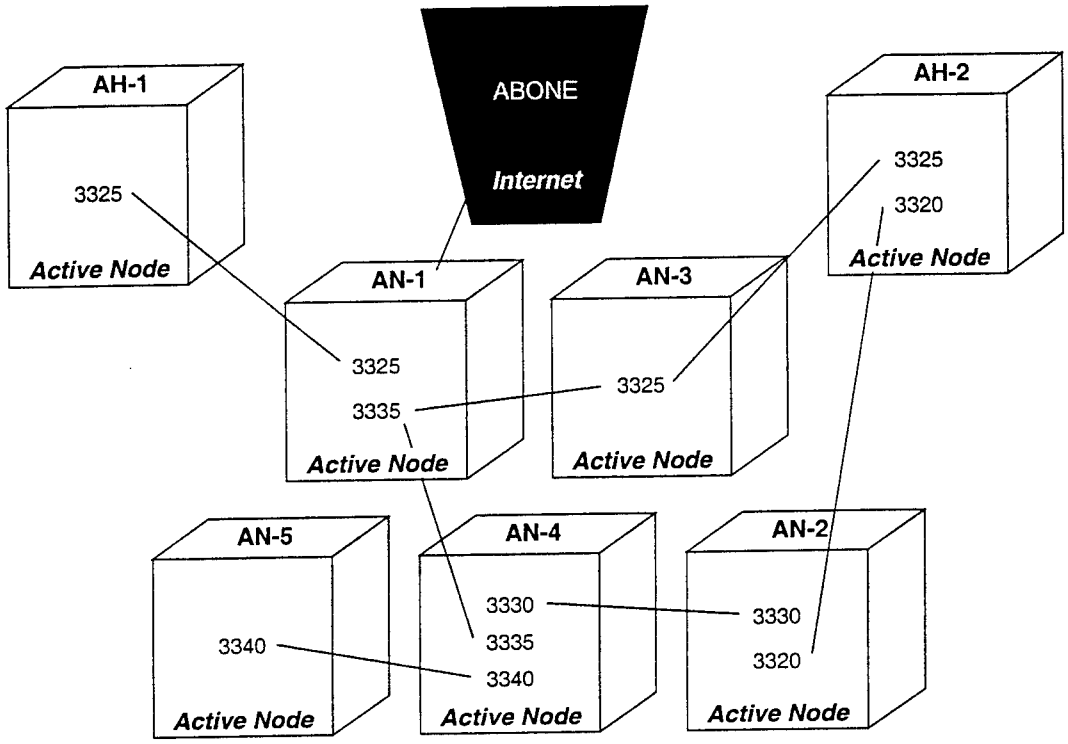


Figure 8.6. The GE CRD Active Network Testbed.

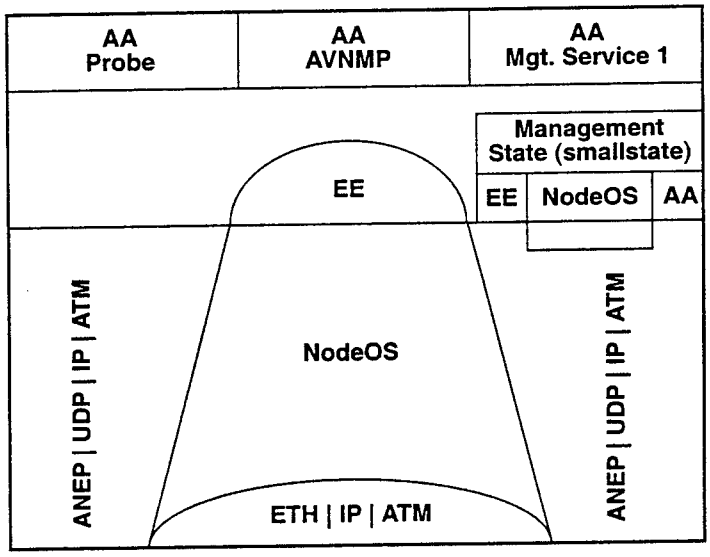


Figure 8.7. Overview of the Management Framework.

Figure 8.8 shows the co-existence of the AVNMP components and the application. The AVNMP Driving Processes gather prediction information from the actual application through the management system. In this case, the information required is the total number of packets generated and the current time. The prediction is based on a simple curve-fitting algorithm. The predicted value is placed into the MIB. The prediction is also propagated to the next hop node. At this node the Physical Process forwards the virtual packet, and the Logical Process provides the virtual time environment in which this takes place. This includes handling rollback when virtual messages arrive out-of-order or real messages are out-of-tolerance with predicted values. The predicted value is again placed within the MIB and the process continues along each node in the path of the data stream.

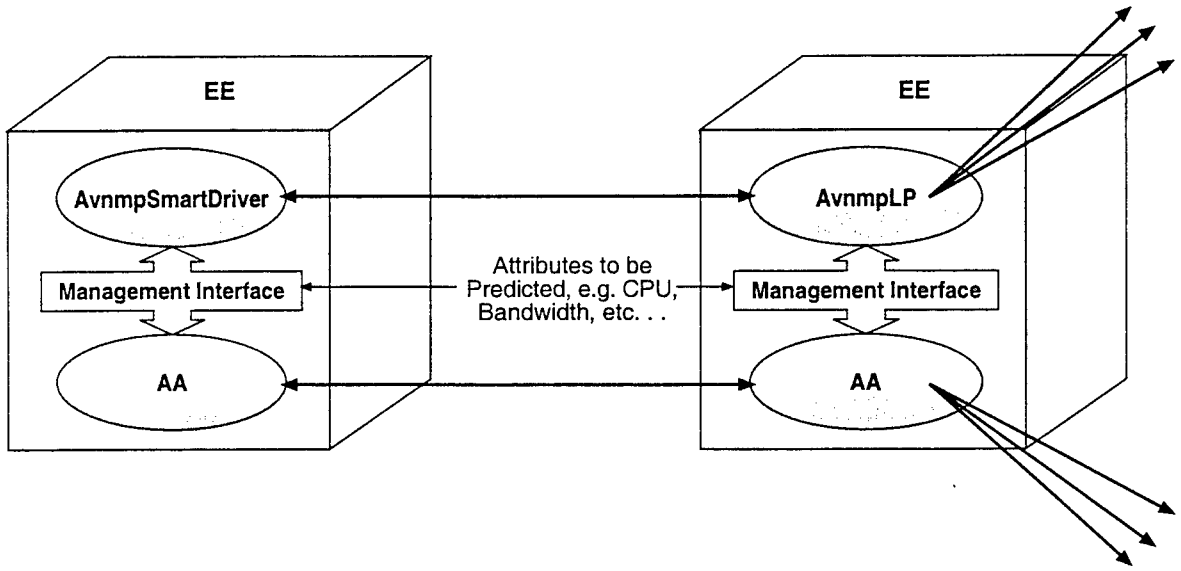


Figure 8.8. Overview of the AVNMP Architecture.

Figure 8.9 shows the AVNMP system in more detail. The active packet contents are illustrated along with the AVNMP components in a Driving Process and Logical Process. The details of the AVNMP system have been described in detail in previous chapters. The important point to note in these figures is that the AVNMP State Queue provides the predicted values for the framework MIB. Because nodes each have their own notion of virtual time and because rollback can occur, the predicted values in the MIB can change. The Management Interface in Figure 8.9 interfaces with internal Magician Execution Environment management data such as CPU utilization as well as Magician application level management information and AVNMP.

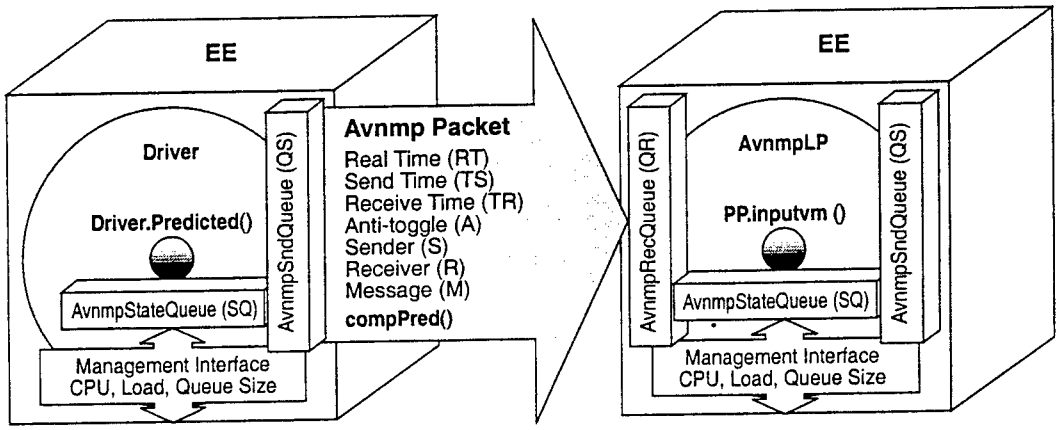


Figure 8.9. The AVNMP Management Interface.

Figure 8.10 provides more detail on the management framework used for validation of the AVNMP algorithm. The Magician active application is provided with a Java Interface that allows implementation of SNMP-like calls to collect and set management values within an application. The SNMP Agent in Figure 8.10 is a separate Magician application that implements the SNMP instrumentation via SmallState (InjectSnmp). Management clients can access the node as though it was a standard SNMP manageable system. This approach was chosen because it appeared to require the least amount of overhead and provided easy access to extant SNMP tools. Note that the communication between the active application and the agent occurs via message passing. Thus the application and the agent need not reside on the same node and the agent can easily be mobile. The SNMP Management Information Base (MIB) can be conceptualized as residing in the SmallState illustrated in the figure.

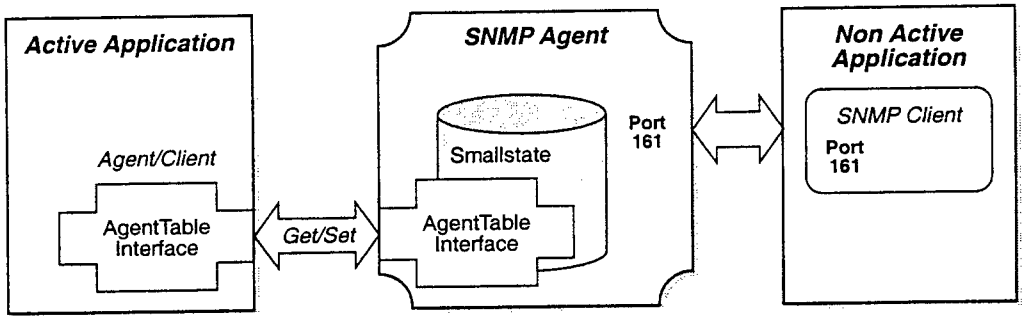


Figure 8.10. AVNMP Architecture in More Detail.

8.2 The Mathematica AVNMP Package

This section presents the development of the relationships used in the analysis of AVNMP. The Mathematica software package has been instrumented with the ability to collect, graph, and analyze the results of the AVNMP experiments. The Mathematica code is interspersed along with the graphs and analytical results and discussion that follows. Mathematica cells appear in

the outlined sections that follow, input code is boldface, and output is in computer typeface font near the bottom of the cells. Equation 1 shows the Mathematica packages and settings used to generate the graphs and equations throughout this document.

```
Needs["Avnmp`"];
Needs["DataRetrieval`"];
Needs["Graphics`MultipleListPlot`"]
Needs["Statistics`DescriptiveStatistics`"]
<<"/home/bushsf/mma/GnuDisplay.m"
<<Statistics`DataManipulation`
<<Graphics`Graphics`
Off[General::spell1]
dir="/home/bushsf/projects/an/snmp/avnmp_stage/10_16_linear/";
```

Equation 1 Mathematica Packages Used to Gather and Manipulate Experimental Data.

Equation 2 defines the time dimension in milliseconds. Equation 3 defines the Lookahead, λ, which is the maximum distance into the future the system is allowed to predict. If a Logical Process progresses beyond λ, it will delay. Equation 4 defines the rate at which the Driving Process generates virtual messages. Equation 5 defines the step size of each virtual message generated by the Driving Processes. Each virtual message will have a timestamp that increments by the amount in Equation 5. Equation 6 is the expected task execution time per virtual message. It is obtained by measurement from data collected during the experiment from the Logical Process. Expected Task execution time is a management object in the AVNMP MIB, along with most of the remaining parameters. Equation 7 is the expected amount of time required to perform a rollback. It is also obtained by measurement from the experimental data from the Logical Processes. Expected Task rollback time is also an AVNMP MIB object. Equation 8 is the expected number of out-of-order rollbacks collected from MIB data during experimental runs. Equation 9 is the mean number of out-of-tolerance rollbacks collected from the experimental runs. The expected number of out-of-tolerance rollbacks is also an AVNMP MIB object.

```
mS = 1./1000.s;
```

Equation 2 Defining the Time Dimension.

```
λ=200000. mS

200 . s
```

Equation 3 Setting the Maximum Lookahead Distance.

```
λvm=(0.5 vM)/(1000. mS)

 0.5 vM
   s
```

Equation 4 Setting the Virtual Message Generation Rate.

$$\Delta vm = 20000.\ mS\ /vM$$
$$\frac{20 \cdot s}{vM}$$

Equation 5 Setting the Step Size for Each Virtual Message.

$$taskT = Mean[Flatten[getData[dir, "lPETask.AN-1"]]]\ mS\ /vM$$
$$\frac{4.1624\ s}{vM}$$

Equation 6 Computing the Mean Task Execution Time.

$$trb = Mean[Flatten[getData[dir, "lPETrb.AN-1"]]]\ mS\ /\ vM$$
$$\frac{14.6617\ s}{vM}$$

Equation 7 Computing the Mean Rollback Time.

```
x=Mean[Flatten[getData[dir, "lPPropX.AN-1"]]]
0.0393805
```

Equation 8 Computing the Mean Number of Out-of-Order Messages.

```
y=Mean[Flatten[getData[dir, "lPPropY.AN-1"]]]
0.370916
```

Equation 9 Computing the Mean Number of Out-of-Tolerance Messages.

In Equation 10 the initial tolerance is set at the given packets per second. This means that a predicted value that differs from the actual value by the above value of packets per second is considered a good prediction. The tolerance is reduced after every time period as specified in Equation 11. The tolerance is reduced in scale by the amount shown in Equation 12 every time period in order to test the system under stress. Thus, the tolerance range for prediction error is narrowed as time progresses as shown in Figure 8.. Every five minutes half reduces the tolerance. This increases the likelihood of out-of-tolerance rollbacks and slows the rate of progress of the Local Virtual Time.

```
initTol=1000.;
```

Equation 10 Setting the Initial Tolerance.

```
runMinutes=5.;
```

Equation 11 Setting the Number of Minutes to Run for Each Tolerance.

```
redTol=.5;
```

Equation 12 Scale the Tolerance by this Amount for Each Run.

8.2.1 Prediction Rate

The derivation of the equations was discussed in previous chapters. In this section a brief sketch of the Mathematica version of those equations is shown because these equations are used in the experimental validation which follows. The rate at which AVNMP can predict is based upon Equation 13. The rate is plotted in Figure 8. using values from an actual execution. This shows the effect that out-of-order messages will have on the performance. In this case, it would take more than 70 percent of the total number of messages being received out-of-order to cause AVNMP to slow down to the point of near real-time speed.

```
S[lvm_, Dvm_, ttask_, trb_, X_, Y_] :=
lvm (Dvm - ttask + trb) X - (Dvm - 1/lvm) Y)
```

Equation 13 AVNMP Speed.

```
Plot[S[λvm, Δvm, taskτ, τrb, x, Y], {Y, .1, 1.},
 AxesLabel -> {"Out-of-Tolerance  Messages", "Speedup"}]
```

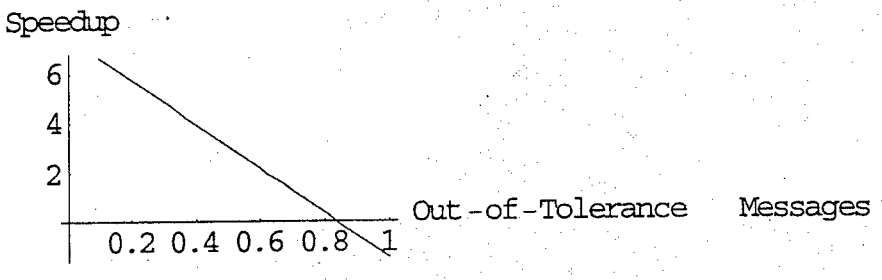


Figure 8.11. AVNMP Speed as a Function of Out-of-Order Messages.

The Local Virtual Time (*LVT*) is derived in Equation 14. *LVT* is a function of *S*, *t*, and *C* where *S* is the prediction rate from Equation 13, *t* is the wallclock time, and *C* is a constant that represents the amount of time the actual system has been in operation before AVNMP is started. *LVT* is plotted in Figure 8.12 as a function of wallclock time and the proportion of out-of-tolerance messages. Fewer out-of-tolerance messages result in a greater predictive distance into the future. Equation 15 defines Lookahead that is graphed in Figure 8.13. Lookahead increases indefinitely with Wallclock time because maximum Lookahead has not incorporated into the equation yet.

```
LVT[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_] :=
  S[lvm, Dvm, ttask, trb, X, Y]t + C
```

Equation 14 The Equation for Local Virtual Time.

```
Plot3D[LVT[λvm, Δvm, 1.0, taskτ, τrb, x, Y, t, 0.], {Y, 0., 1.},
 {t, 0., 100.}, AxesLabel -> {"Y", "t", "LVT"}]
```

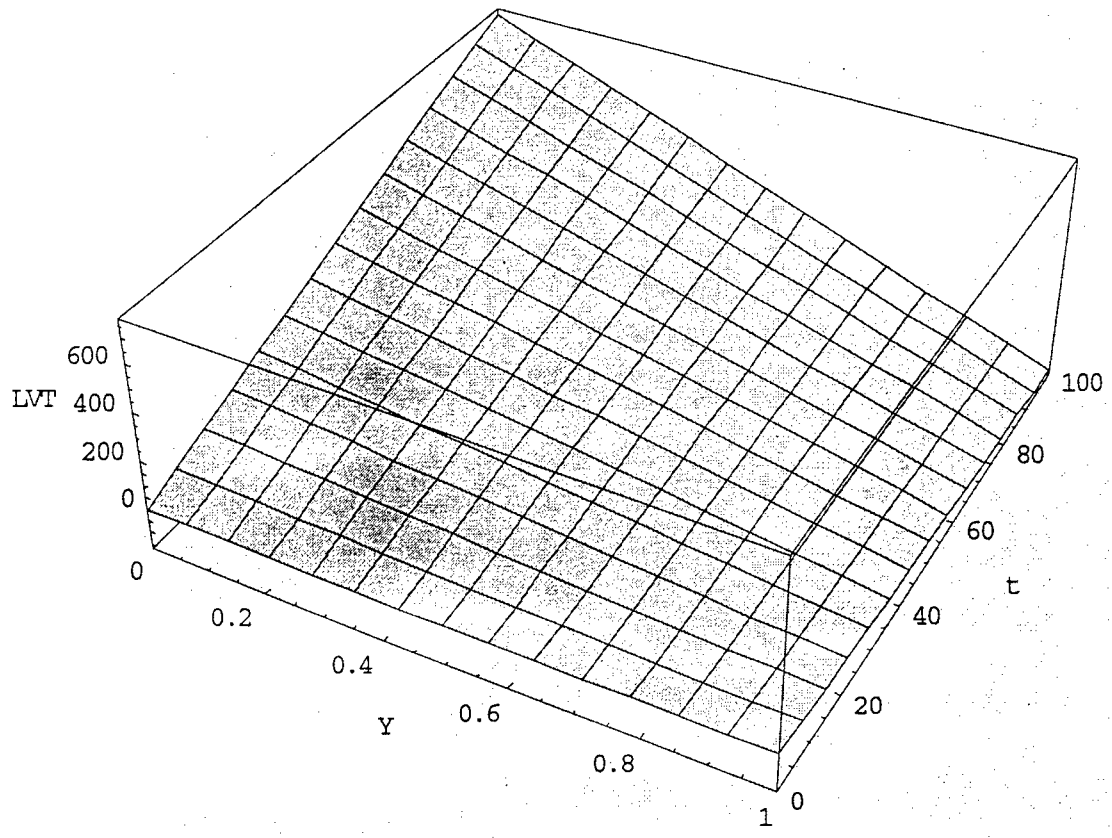


Figure 8.12. AVNMP Performance as a Function of Out-of-Tolerance Message Proportion.

```
LA[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_] :=
   (LVT[lvm, Dvm, Spar, ttask, trb, X, Y, t, C] - 1.) t + C
```

Equation 15 Lookahead.

```
Plot3D[LA[λvm, Δvm, 1.0, taskτ, τrb, x, Y, t, 0.], {Y, 0., 1.},
 {t, 0., 100.}, AxesLabel -> {"Y", "t", "LA"}]
```

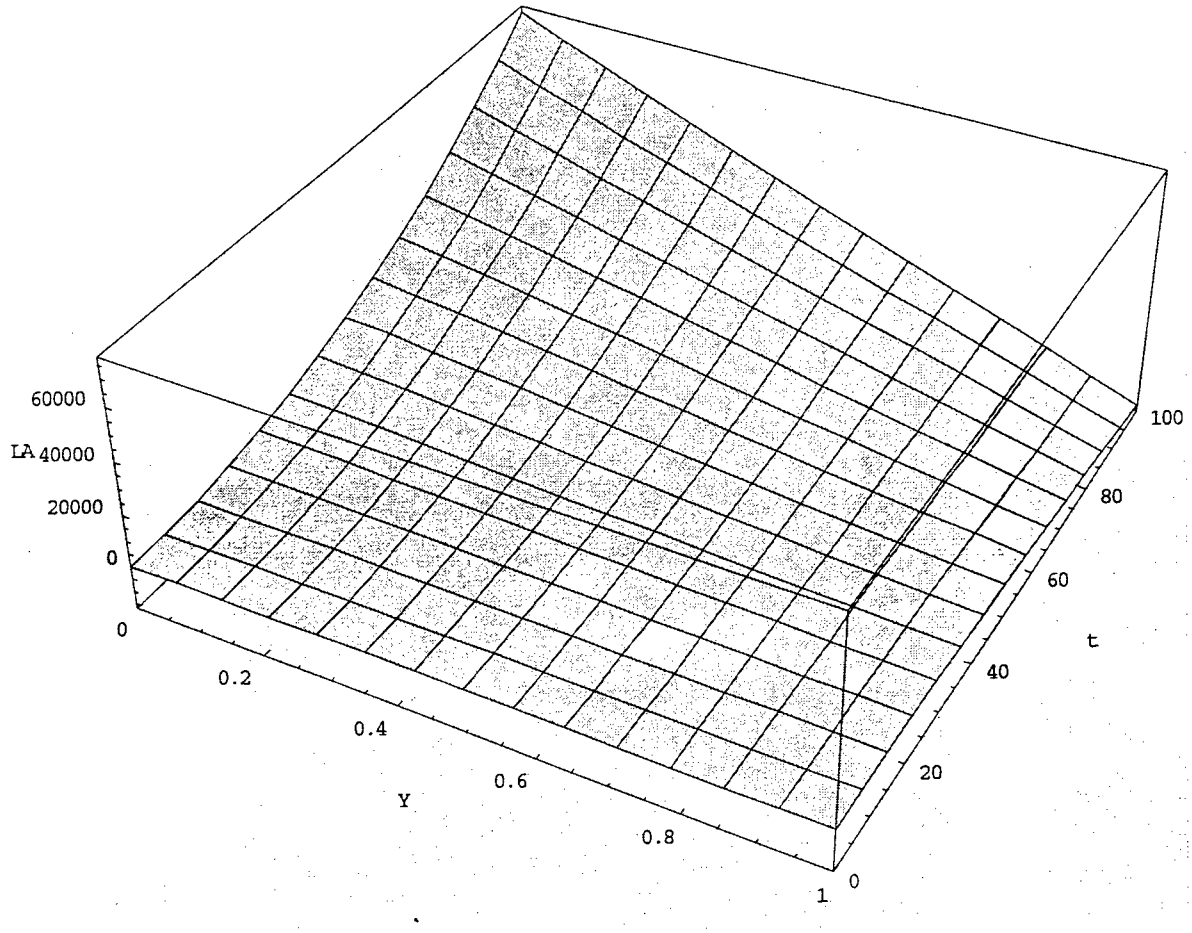


Figure 8.13. Lookahead Performance.

Equation 16 defines an approximate relationship between the probability of a message being out of tolerance given the proportion of out-of-order message proportion and the time to reach that proportion of out-of-tolerance messages. Note that we are assuming exponential amount of error. A sample is graphed in Figure 8.14. The inverse relationship is defined in Equation 17 where the variable *s* is the amount of time into the future at which the event occurs. A sample graphed in Figure 8.15. The proportion of out-of-order messages is calculated given the amount of Lookahead and the tolerance and assuming an error exponential in the amount of time into the future the prediction occurs.

$$\text{InvY}[Y\_, T\_] := -\left(\frac{T}{\text{Cos}[45.\ \text{Degree}]\ \text{Log}[Y]}\right)$$

Equation 16 Probability of Out-of-Tolerance Messages.

```
Plot[InvY[Y, 1.], {Y, 0., 1.}, AxesLabel -> {"Y", "InvY"}]
```

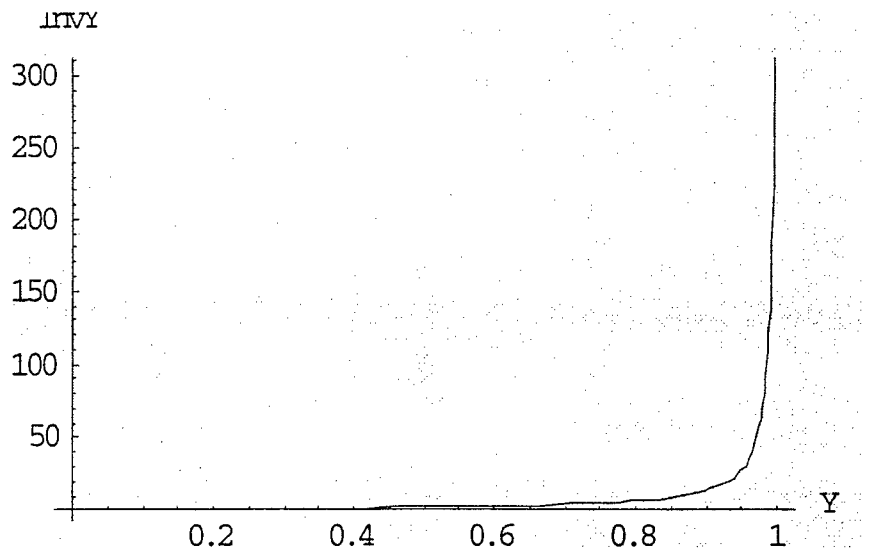


Figure 8.14. Tolerance Setting as a Function of Out-of-Tolerance Proportion.

$$Y[s\_, T\_] := \text{Exp}\left[-\frac{1.}{(\text{Cos}[45.\ \text{Degree}]\ s)}\ T\right]$$

Equation 17 Proportion of Out-of-Order Messages.

```
Plot[Y[s, 1.], {s, 0.1, 1.}, AxesLabel -> {"s", "Y"}]
```

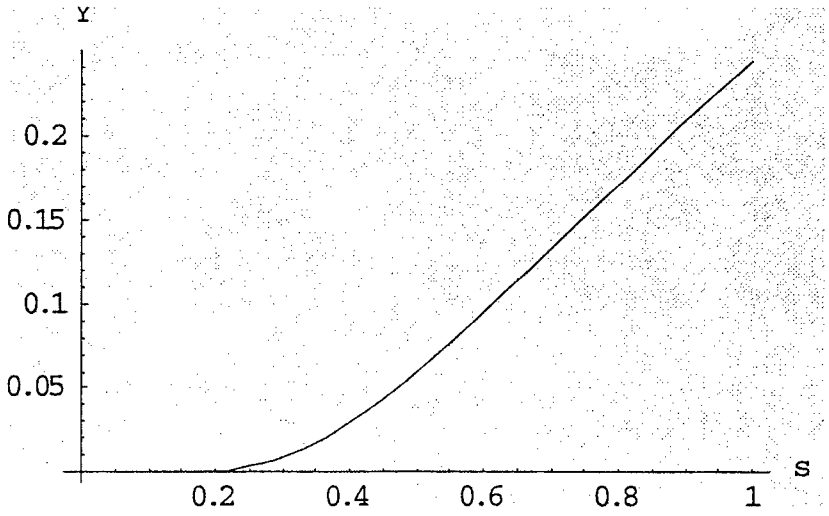


Figure 8.15. Proportion of Out-of-Tolerance Messages as a Function of Distance into the Future.

The function *Gamma* was explained in a previous chapter. Its Mathematica definition is shown in Equation 18 and plotted in Figure 8.16. *Gamma* is used in the denominator of the exponent in defining *Pafter* in Equation 19. Gamma increases with X and is independent of Wallclock time.

$$\text{Gamma1}[\text{lvm\_}, \text{Dvm\_}, \text{Spar\_}, \text{ttask\_}, \text{trb\_}, X\_, Y\_, t\_, C\_] := \text{Dvm Spar} - \frac{\left(\frac{1.}{\text{lvm}}\right) - (\text{ttask} + \text{trb})\, X - \text{ttask}}{\text{Dvm Spar} - \frac{1.}{\text{lvm}} + \text{trb}}$$

Equation 18 Gamma.

```
Plot3D[Gamma1[λvm s / vM, Δvm vM / s, 1.0, taskτ vM / s, τrb vM / s, X,
  Y, t, 0.], {X, .1, 1.}, {t, 0., 10.}, AxesLabel -> {"X", "t", "Gamma1"}]
```

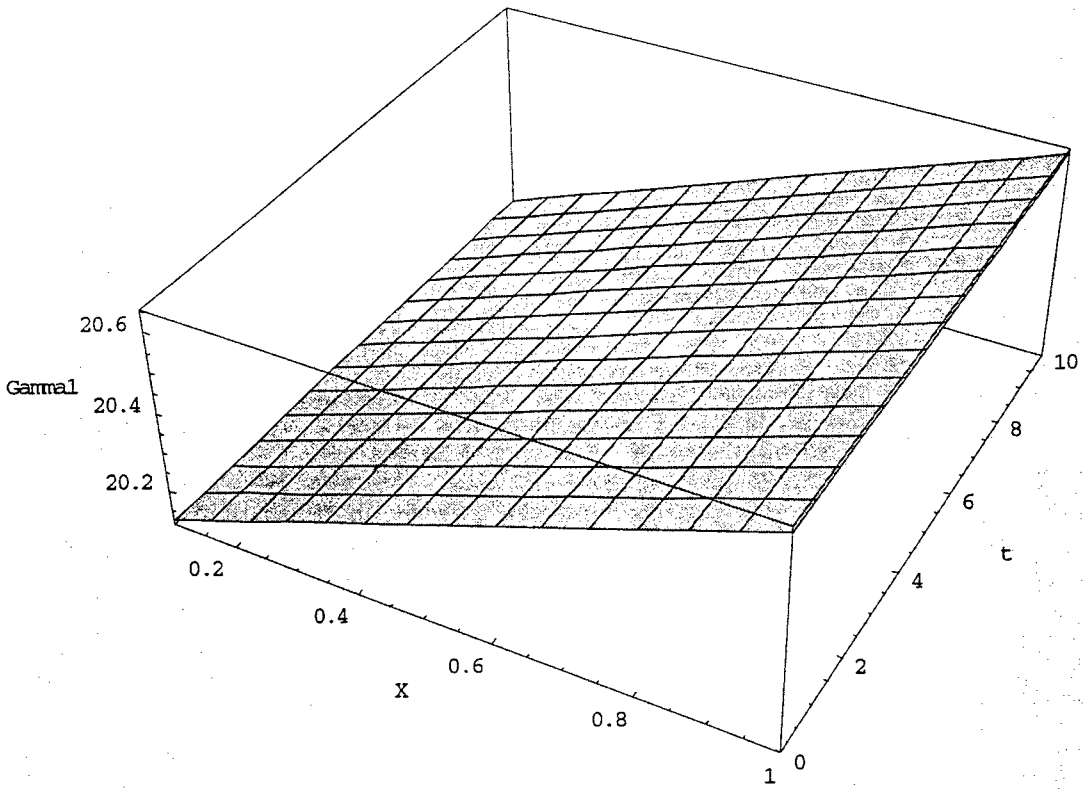


Figure 8.16 Gamma as a Function of Wallclock and Out-of-Order Message Proportion.

Equation 19 defines the probability of an event occurring before it was predicted to occur. In other words, the prediction occurred late. The plot in Figure 8.17 shows that the probability of late prediction appears to be very dependent upon the proportion of out-of-tolerance messages and less so on out-of-order messages. This makes intuitive sense because out-of-order messages

can be corrected with small, quick rollbacks, while out-of-tolerance rollbacks require a rollback to wallclock time.

```
Pafter[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_, T_] :=
 Exp[
  - 1. / (InvY[Y, T] (Gamma1[lvm, Dvm, Spar, ttask, trb, X, Y, t, C] + C))]
```

Equation 19 The Probability of a Prediction Occuring Late.

```
Plot3D[Pafter[λvm s / vM, Δvm vM / s, 1.0, taskτ vM / s, τrb vM / s, X,
  Y, 0., 0., 1.], {X, 0.0001, 1.}, {Y, .001, .99},
 AxesLabel -> {"X", "Y", "Pafter"}]
```

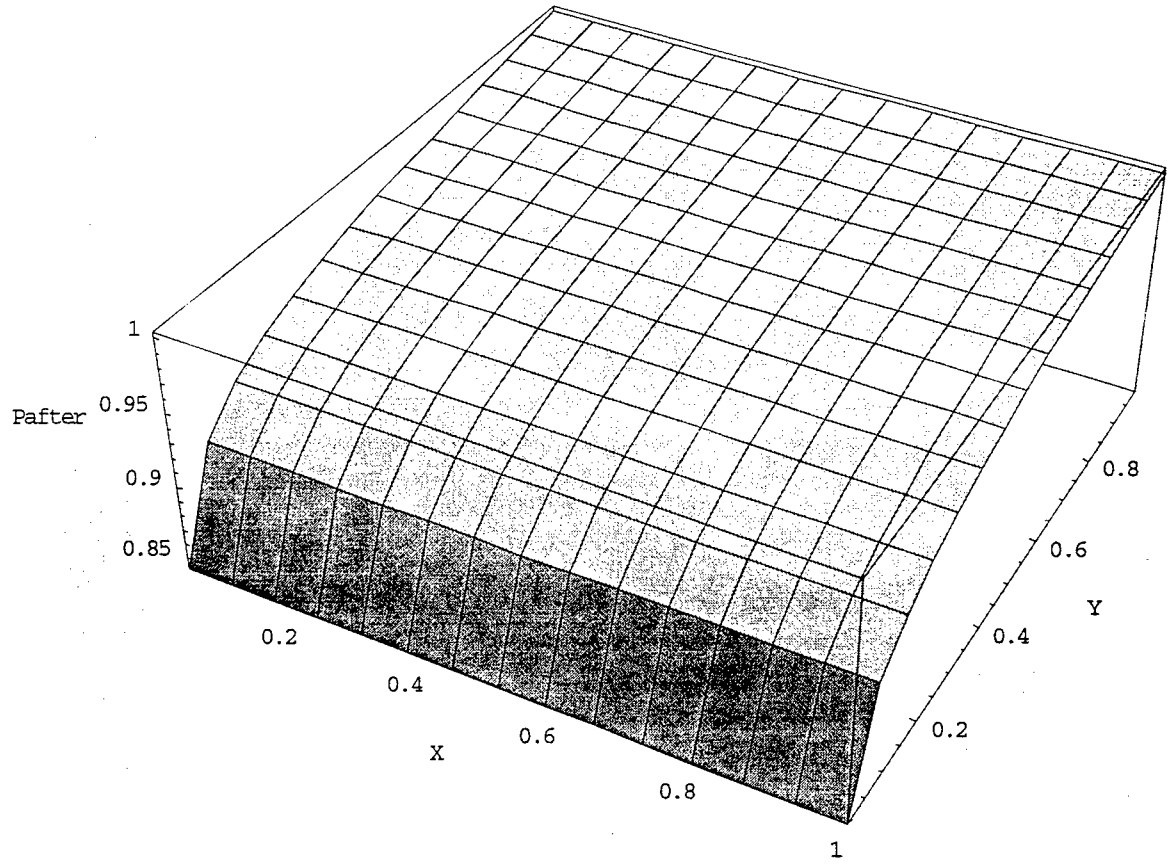


Figure 8.17. Probability of a Late Prediction as a Function of Out-of-Order and Out-of-Tolerance Message Proportions.

Equation 20 is a more accurate definition of the rate at which prediction occurs. It is how much faster LVT advances than wallclock time. This rate is graphed in Figure 8.18 as a function of

out-of-order and out-of-tolerance message proportions. Again, the out-of-tolerance messages clearly have a larger impact on performance. Equation 21 defines the speedup of AVNMP over a non-AVNMP process. Speedup is graphed in Figure 8.19.

$$Prate[lvm\_, Dvm\_, Spar\_, ttask\_, trb\_, X\_, Y\_, t\_, C\_] := lvm\left(Dvm\,Spar - ttask - (ttask + trb)\,X - \left((Dvm\,Spar) - \left(\frac{1.}{lvm}\right) + trb\right)Y\right)$$

Equation 20 The Rate at which AVNMP Predicts.

```
Plot3D[Prate[λvm s / vM, Δvm vM / s, 1.0, taskτ vM/s, τrb vM / s, X, Y, 0., 0.],
 {X, 0.001, 1.}, {Y, 0.001, 1.}, AxesLabel -> {"X", "Y", "Prate"}]
```

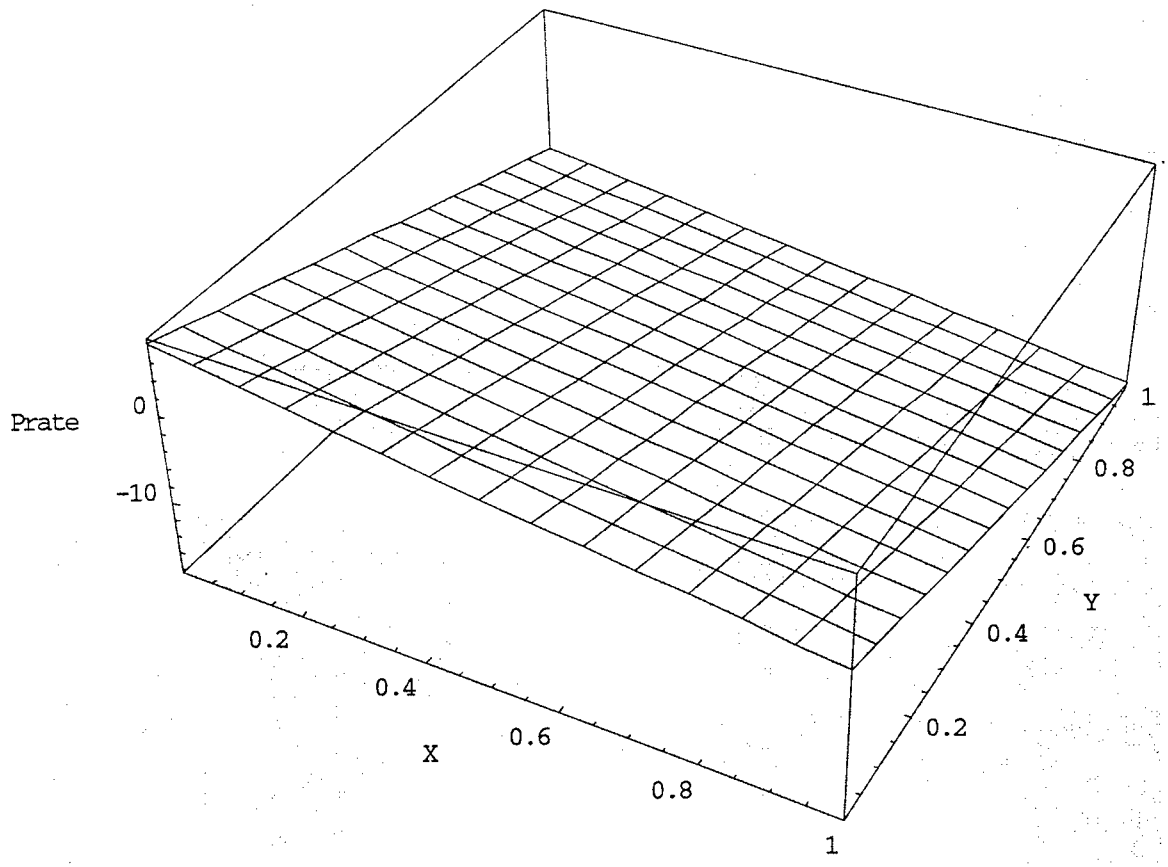


Figure 8.18. AVNMP Prediction Rate as a Function of Out-of-Order and Out-of-Tolerance Messages.

```
Speedup[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_, T_] :=
(1. - Pafter[lvm, Dvm, Spar, ttask, trb, X, Y, t, C, T]) +
Pafter[lvm, Dvm, Spar, ttask, trb, X, Y, t, C, T] Prate[lvm, Dvm, Spar,
ttask, trb, X, Y, t, C]
```

Equation 21 Speedup of AVNMP over the Wallclock Time of the Actual System.

```
Plot3D[Speedup[λvm s / vM, Δvm vM / s, 1.0, taskτ vM/s, τrb vM / s, X,
  Y, 0., 0., 1.], {X, 0.0001, 1.}, {Y, 0., 1.},
 AxesLabel -> {"X", "Y", "Speedup"}]
```

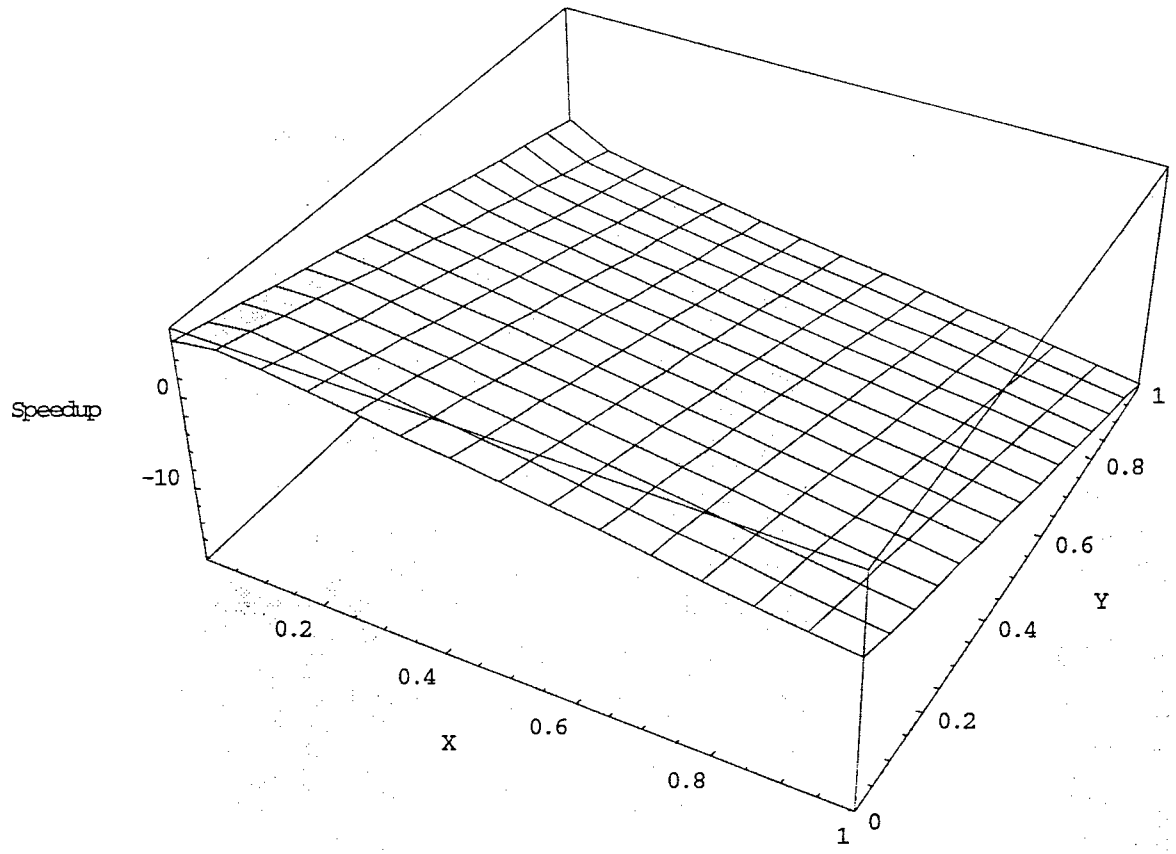


Figure 8.19. Speedup of AVNMP as a Function of Out-of-Order and Out-of-Tolerance Message Proportions.

Next consider the problem from a different perspective. Because AVNMP operates ahead of wallclock time, perhaps the tasks can be given more time to execute without an apparent slowdown in the system. In other words, one would like to know, given certain operating parameters for AVNMP, what is the maximum wallclock time that a task can take to execute. Equation 22 defines the time that a task can take to execute given all the other AVNMP

parameters. The resulting simplified relationship is shown in Equation 23 and graphed in Figure 8.20 and Figure 8.21. This shows that when *LVT/t* is high, a task can take a longer time to execute and the system will complete in the same amount of time. In Figure 8.21, as *LVT/t* increases, expected task execution time can take longer and the system will still compute the result in the same amount of time given no out-of-tolerance or out-of-order rollbacks.

```
ttask[lvm_, Dvm_, Spar_,
     ttask_, trb_, X_, Y_, t_, C_] :=   ttask /.
     Solve[         LVT == LVT[lvm, Dvm, Spar, ttask, trb, X, Y,
        t, C], {ttask}][[1]]
```

Equation 22 Determining Maximum Task Time Given Other AVNMP Parameters.

$$\text{ttask}[\text{lvm\_}, \text{Dvm\_}, \text{Spar\_}, \text{ttask\_}, \text{trb\_}, \text{X\_}, \text{Y\_}, \text{t\_}, \text{C\_}, \text{LVT\_}] := \frac{-C + LVT - Dvm\,lvm\,t + lvm\,t\,trb\,X - t\,Y + Dvm\,lvm\,t\,Y}{lvm\,t\,(1. + X)}$$

Equation 23 Result of Solution to Equation 22 above.

```
Plot3D[ttask[0.03, 40.0, 1.0, 7.0, 1.0, .5, .5, t, 0., LVT],
 {LVT, .0001, 100.}, {t, .0001, 100.},
 AxesLabel -> {"LVT", "t", "Task Time"}]
```

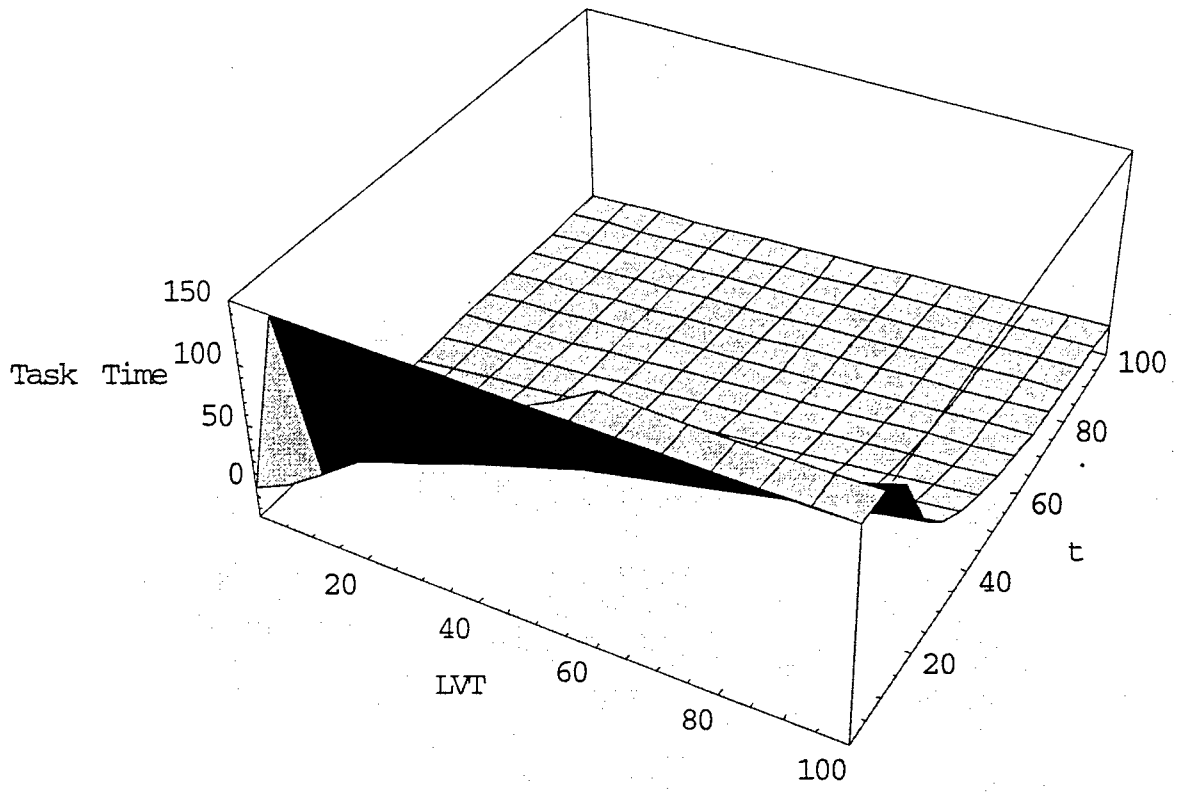


Figure 8.20. Maximum Task Time as a Function of Local Virtual Time and Wallclock Time.

```
Plot[ttask[0.03, 40.0, 1.0, 7.0, 1.0, .5, .5, 0.0001, 0., LVT],
 {LVT, .0001, 100.}, AxesLabel -> {"LVT", "Task Time"}]
```

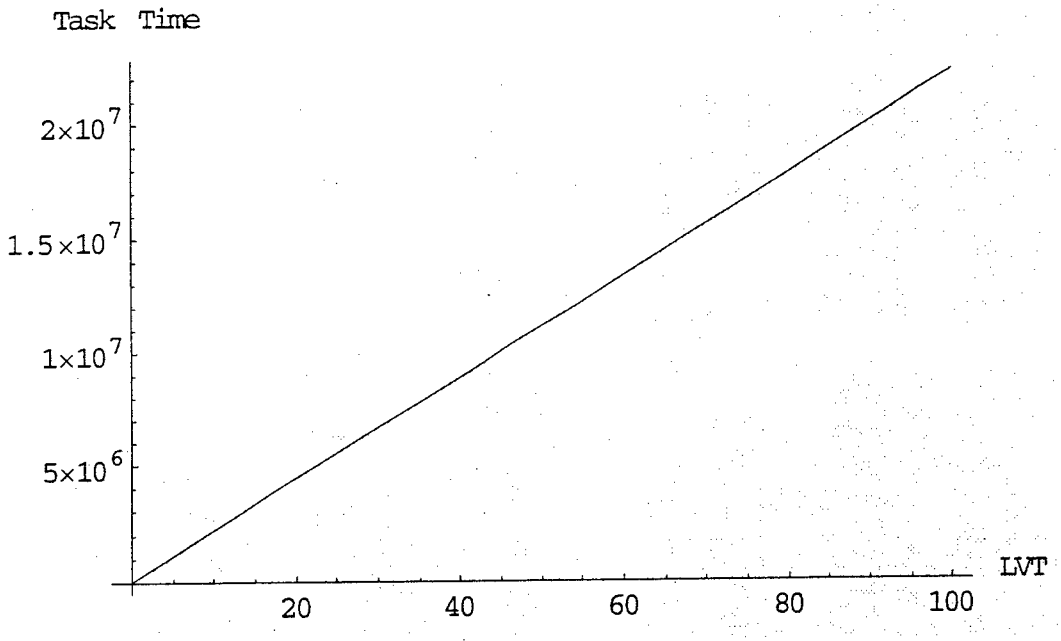


Figure 8.21. Maximum Task Time as a Function of Local Virtual Time.

8.2.2 Deriving Expected Lookahead

Equation 24 derives the wallclock time at the instant when the end of the sliding Lookahead window is reached. This is the maximum allowed Lookahead. *lvm* is the virtual message input rate, *Dvm* is the virtual message Lookahead, *Spar* is the speedup due to parallelism, *ttask* is the task execution time, *trb* is the time to rollback, *X* is the proportion of out of order messages, *Y* is the proportion of out of tolerance message, *t* is the current time, *C* is the fact that AVNMP begins running *C* time units before real message start, and *L* is the maximum Lookahead time. Equation 25 is the wallclock time spent waiting while wallclock time catches up to the *LVT*. Equation 26 is Lookahead at wallclock time *t*. In Equation 27, while wallclock is less than time *th*, Lookahead is *Prate*. Equation 28 is the expected Lookahead of the system.

```
th[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, C_, L_]:= Module[{}, tH /.
Solve[Prate[lvm, Dvm, Spar, ttask, trb, X, Y, 0., C] tH == Dvm
Global`vM/Global`s, {tH}][[1]]]
```

Equation 24 Wallclock Time When End of Sliding LookAhead Window is Reached.

```
tL[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, C_, L_]:= Module[{},(th[lvm, Dvm,
Spar, ttask, trb, X, Y, C, L] Global`s+ L)/Global`s]
```

Equation 25 Time Waiting for Wallclock to Reach Local Virtual Time.

```
La[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_, L_]:= Module[{Tl =
tL[lvm, Dvm, Spar, ttask, trb, X, Y, C, L]}, ([t, Tl]Prate[lvm, Dvm, Spar,
ttask, trb, X, Y,0., C]) ]/; (Mod[t, tL[lvm, Dvm, Spar, ttask, trb, X, Y,C,
L]] <= th[lvm, Dvm, Spar, ttask, trb, X, Y,C, L])
```

Equation 26 Lookahead at a Given Wallclock Time (Part 1).

```
La[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_, t_, C_, L_]:= Module[{Tl =
tL[lvm, Dvm, Spar, ttask, trb, X, Y, C, L]}, ( (L + Dvm Global`vM/Global`s) -
Mod[t,Tl])]/; (Mod[t, tL[lvm, Dvm, Spar, ttask, trb, X, Y, C, L]] > th[lvm,
Dvm, Spar, ttask, trb, X, Y, C, L])
```

Equation 27 Lookahead at a Given Wallclock Time (Part 2).

```
ESLa[lvm_, Dvm_, Spar_, ttask_, trb_, X_, Y_,C_, L_]:= Module[{Tl= tL[lvm, Dvm, Spar, ttask, trb, X, Y,
C, L]}, [La[lvm, Dvm, Spar, ttask, trb, X, Y, t, C, L], {t, 0., Tl}]/Tl]
```

Equation 28 Expected Lookahead.

8.3 Experimental Configurations

Figure 8.22 shows feed-forward deployment of Logical Processes and Driving Processes. The Predictor within the Driving Process is illustrated as well as the Physical Processes encapsulated by the Logical Processes. Attempting to predict load validates the experimental results; the application that is not shown in Figure 8.22 is a simple active packet generator. The Physical Process implements simple forwarding. The experimental goal in this particular validation of AVNMP is to measure its performance predicting the number of packets in both time and space throughout the active network. The configuration values used in the experiment are set as shown in the previous section. These values are used in the analytical results. The AVNMP MIB (shown in Chapter 7) was polled for all values for used in the validation.

In addition to the feed-forward configuration shown in Figure 8.22, another configuration using multiple Driving Processes feeding virtual messages into Logical Processes from diverse locations in the network is experimentally validated. It is important that the Driving Processes synchronize themselves so that they do not induce a continuous causality induced rollback with other Driving Processes. A mechanism to prevent this situation is to gradually increase the Driving Processes *LVT*, and thus its Lookahead, when causality based rollbacks occur. This will cause the Receive Times of the resulting messages to increase such that they are ahead of other Driving Processes' Receive Times, but not so far ahead as to cause the other Driving Processes to rollback. This synchronization mechanism for Driving Processes appears to work reasonably well, as shown in the following graphs. The following sections are labeled by the data graphed and with the AVNMP MIB object identifier name in parenthesis.

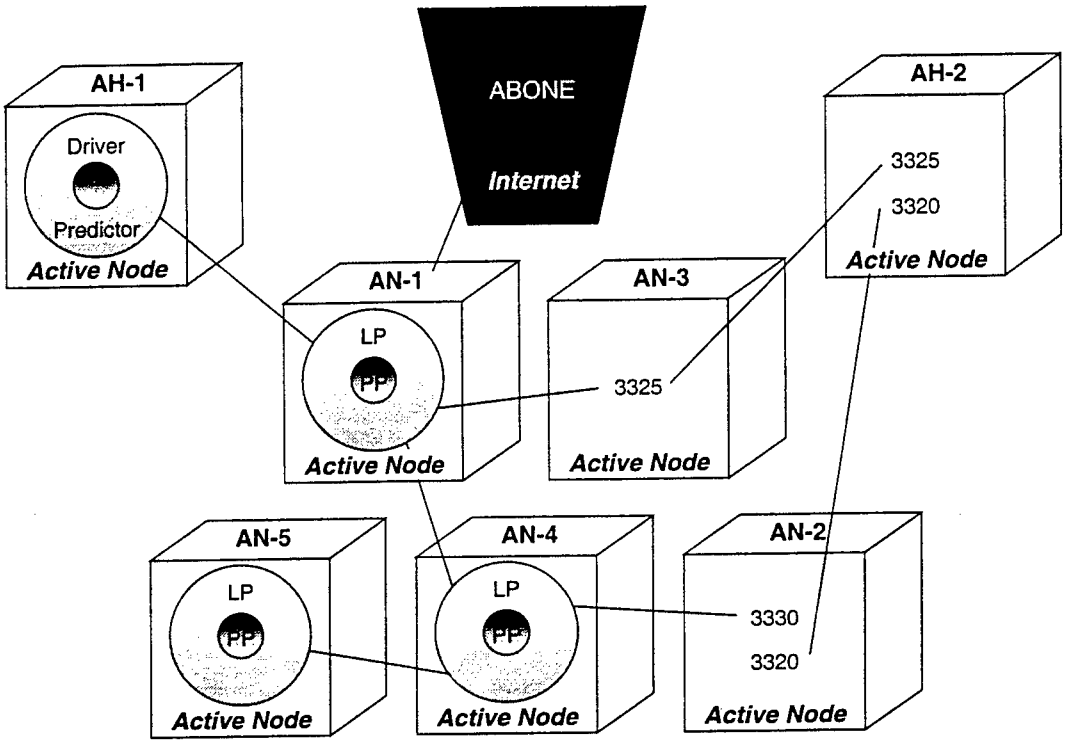


Figure 8.22. Experimental Configuration.

8.3.1 Lookahead (lPELkAhead)

The expected Lookahead is the amount of time from wallclock into the future that the AVNMP system is capable of maintaining within a particular Logical Process. As tolerance increases and rollbacks occur more often, it is anticipated that Lookahead will be reduced. This is actually the case as shown in Figure 8.23 and Figure 8.24.

```
makePlot[dir, "lPUptime.AN-1", "lPELkAhead.AN-1",  {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Expected Lookahead (mS)"}, PlotLabel-
>"Performance"}]
```

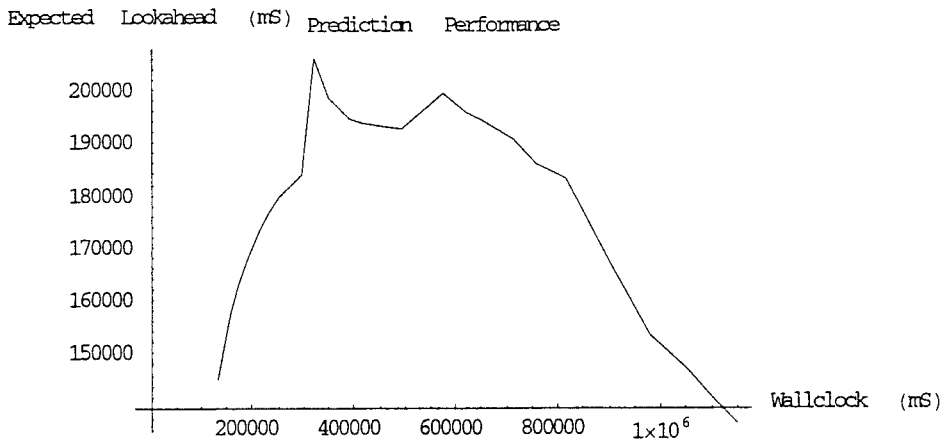


Figure 8.23. Lookahead with Multiple Driving Processes.

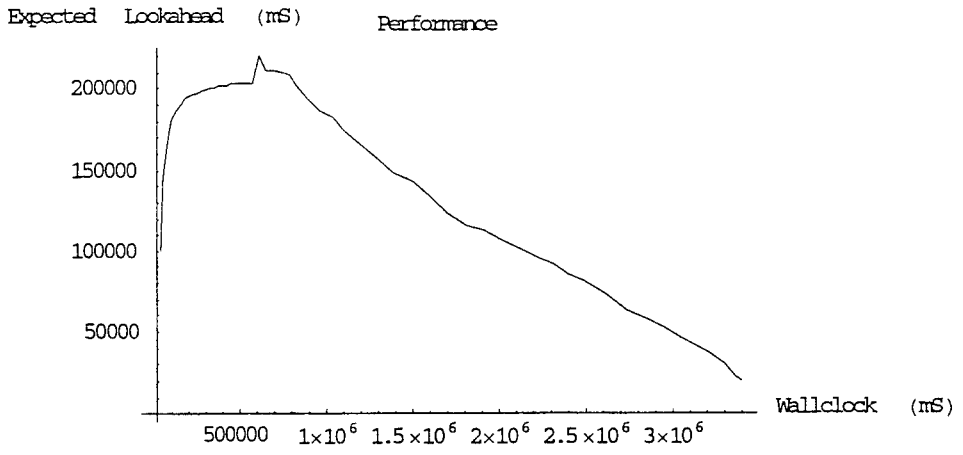


Figure 8.24. Lookahead as a Function of Wallclock.

8.3.2 Proportion Out-of-Tolerance Messages (IPPropY)

As the tolerance is decreased as shown in Figure 8.1, it is anticipated that the number of out-of-tolerance messages will increase and thus the proportion of out-of-tolerance messages should increase. This is shown in Figure 8.25 and Figure 8.26. Figure 8.26 shows the increase in the proportion of out-of-tolerance messages as the tolerance decreases. Figure 8.27 and Figure 8.28 show the proportion of out-of-tolerance messages as a function of the tolerance setting. This verifies that more messages are out-of-tolerance as the tolerance is decreased.

```
makePlot[dir, "1PUptime.AN-1","1PPropY.AN-1",  {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Proportion Out-of-Tolerance"},
PlotLabel->"Performance"}]
```

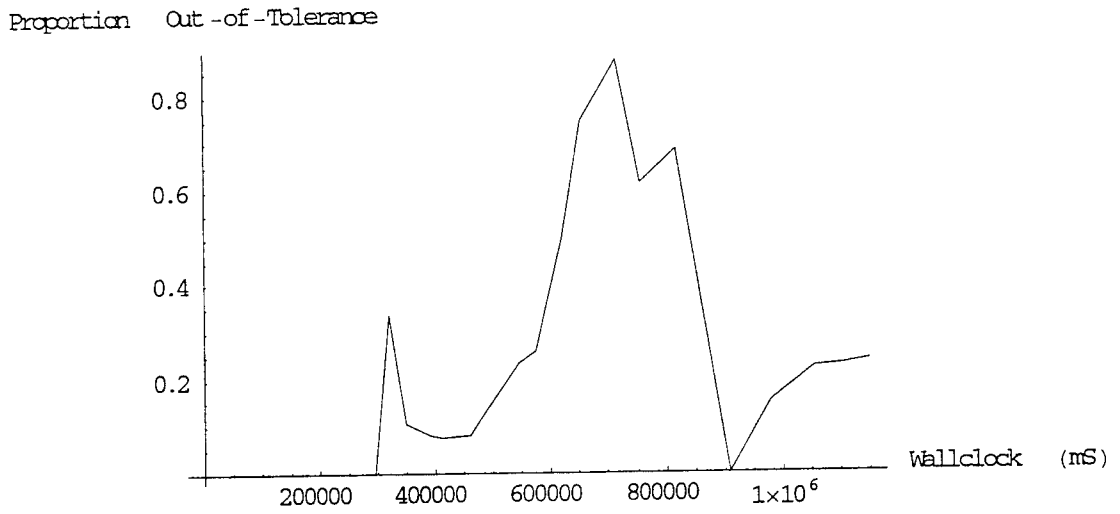


Figure 8.25. Proportion Out-of-Tolerance Messages with Multiple Driving Processes.

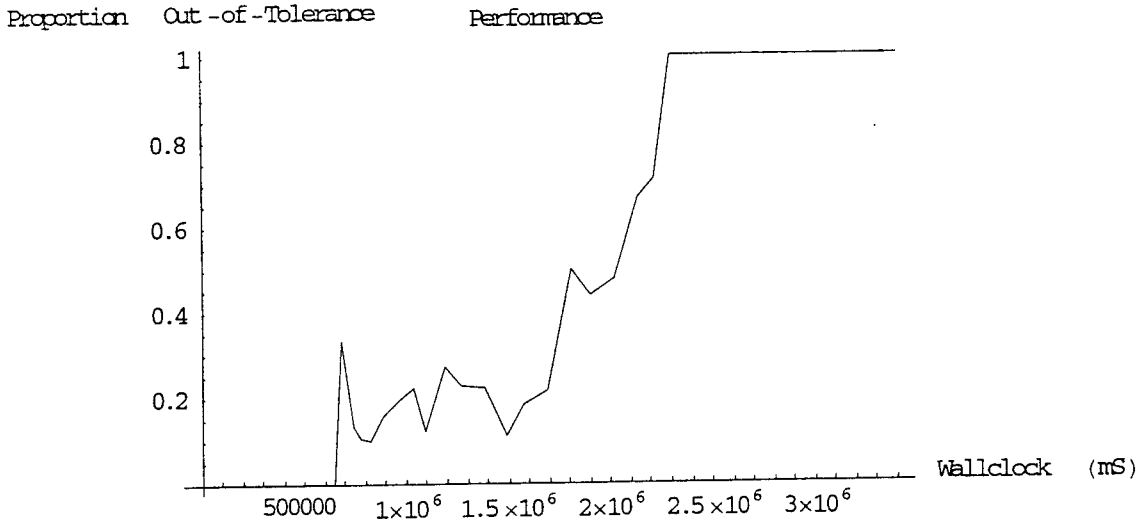


Figure 8.26. Proportion Out-of-Tolerance Messages as a Function of Wallclock.

```
makePlot[dir, "lPActTolerance.AN-1", "lPPropY.AN-1", {PlotJoined->True,
AxesLabel->{"Tolerance (Pkts/mS)", "Proportion Out-of-Tolerance"},
PlotLabel->"Performance"}]
```

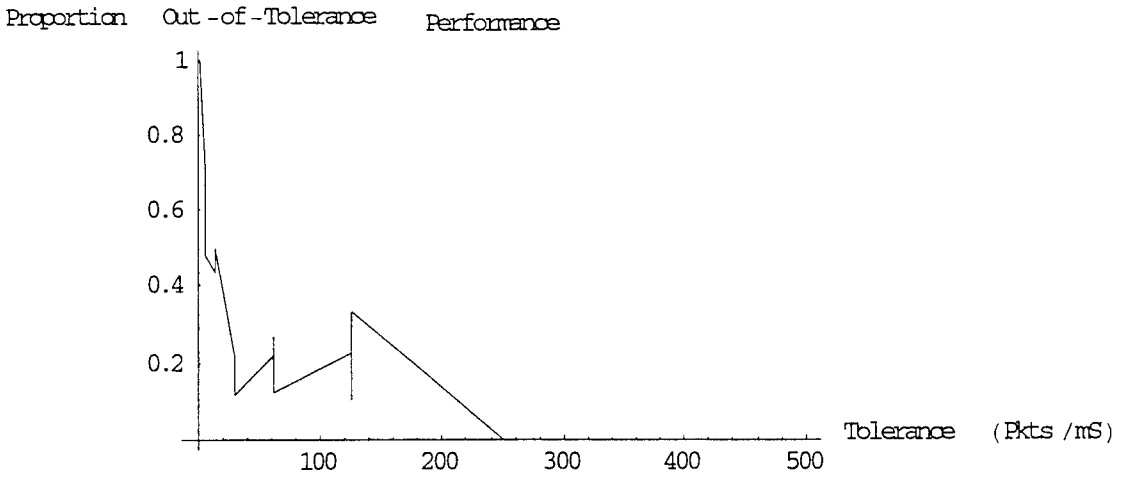


Figure 8.27. Proportion Out-of-Tolerance Messages as a Function of Tolerance.

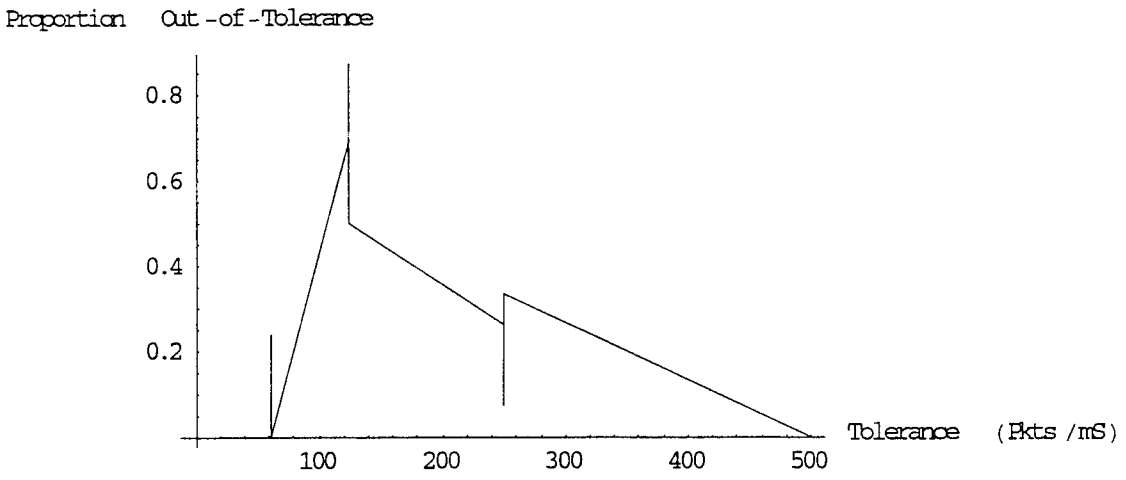


Figure 8.28. Virtual Messages as a Function of Tolerance with Multiple Driving Processes.

8.3.3 Actual Load (loadAppPackets)

An SNMP counter that increases monotonically measures the actual load. Each packet transfer causes the counter to increase by one. Figure 8.29 and Figure 8.30 show the actual application counter value as a function of time. Figure 8.31 and Figure 8.32 show predicted load values from the AVNMP Driving Process. The first prediction set generated a few hundred milliseconds after the AVNMP began running.

```
makePlot[dir, "loadAppUptime.AN-1", "loadAppPackets.AN-1",
{PlotJoined->True, AxesLabel->{"Wallclock (mS)", "Messages"},
PlotLabel->"Load"}]
```

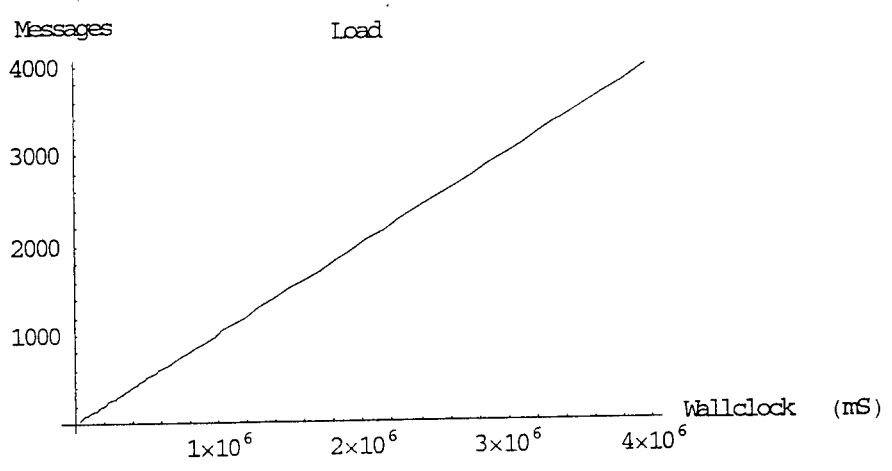


Figure 8.29. Load as a Function of Wallclock.

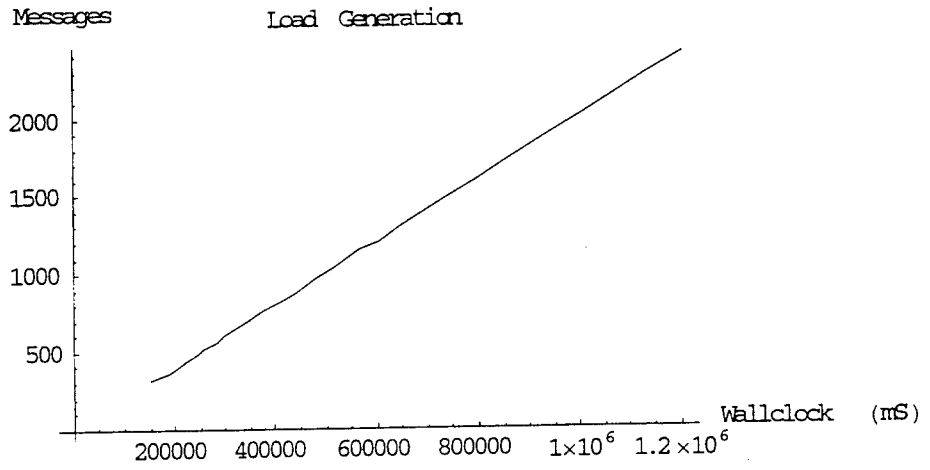


Figure 8.30 Load with Multiple Driving Processes.

```
makePlot[dir, "loadPredictionPredictedTime.AN-1.1",
"loadPredictionPredictedLoad.AN-1.1", {PlotJoined->True, AxesLabel-
>{"Wallclock (mS)", "Messages"}, PlotLabel->"Load Prediction"}]
```

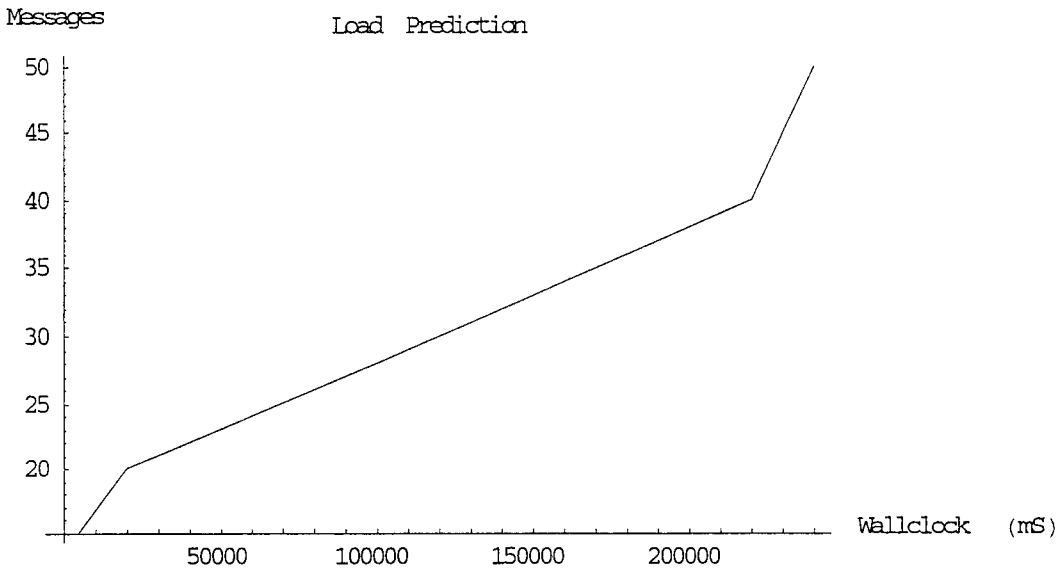


Figure 8.31. Load Prediction as a Function of Wallclock.

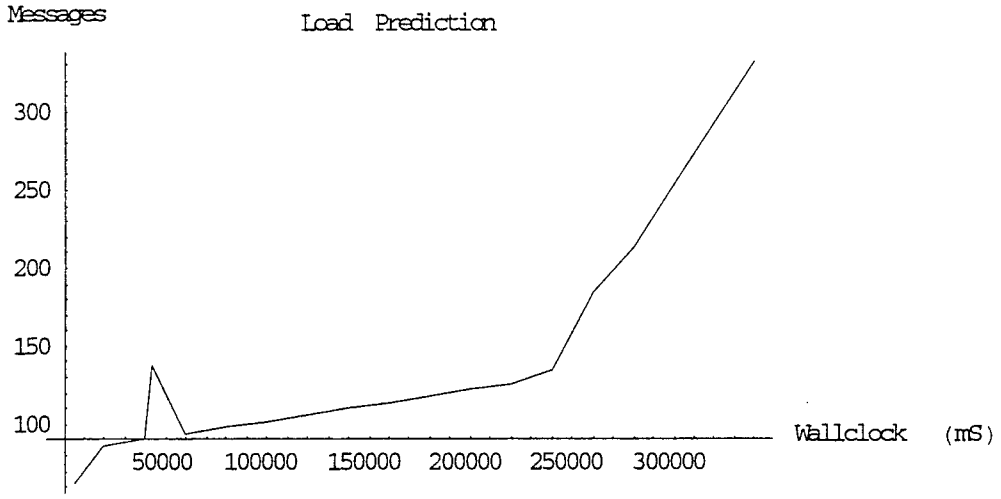


Figure 8.32. Load Prediction with Multiple Driving Processes.

8.3.4 Speedup (lPSpeedup)

This is the expected speedup, *LVT/t*, within an AVNMP Logical Process. The speedup is expected to decrease as the tolerance tightens and the rollbacks increase. This is validated in Figure 8.33 and Figure 8.34. Figure 8.35 and Figure 8.36 show speedup as a function of the proportion of out-of-tolerance messages. As expected the speedup decreases as the proportion of out-of-tolerance messages increases.

```
makePlot[dir, "lPUptime.AN-1","lPSpeedup.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Speedup"}, PlotLabel->"Performance"}]
```

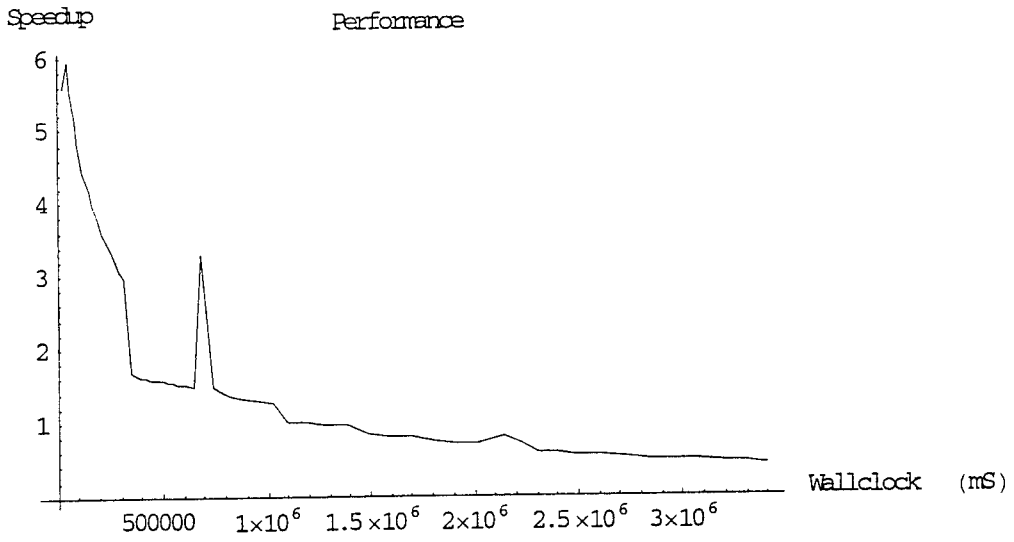


Figure 8.33. Speed as a Function of Wallclock.

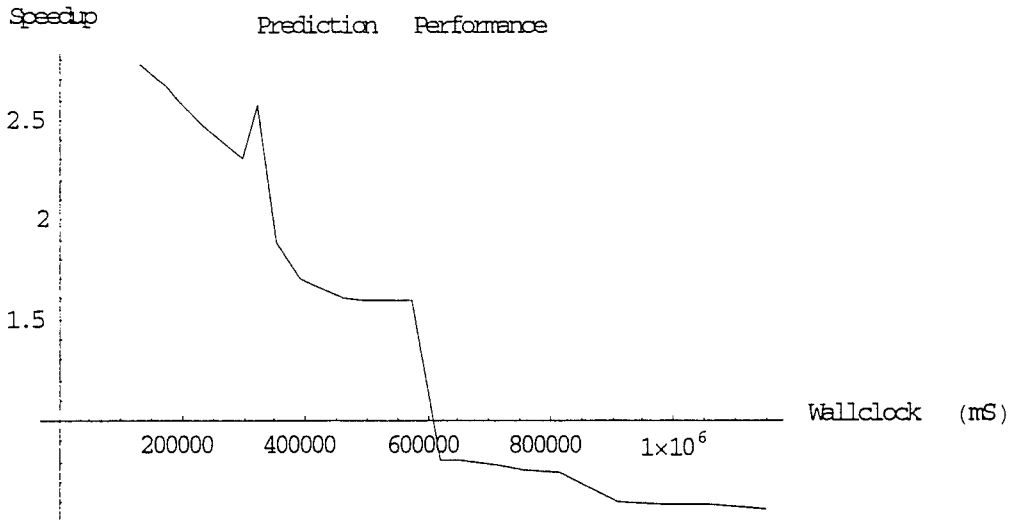


Figure 8.34. Speedup with Multiple Driving Processes.

```
makePlot[dir, "lPPropY.AN-1","lPSpeedup.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Speedup"}, PlotLabel->"Prediction
Performance"}]
```

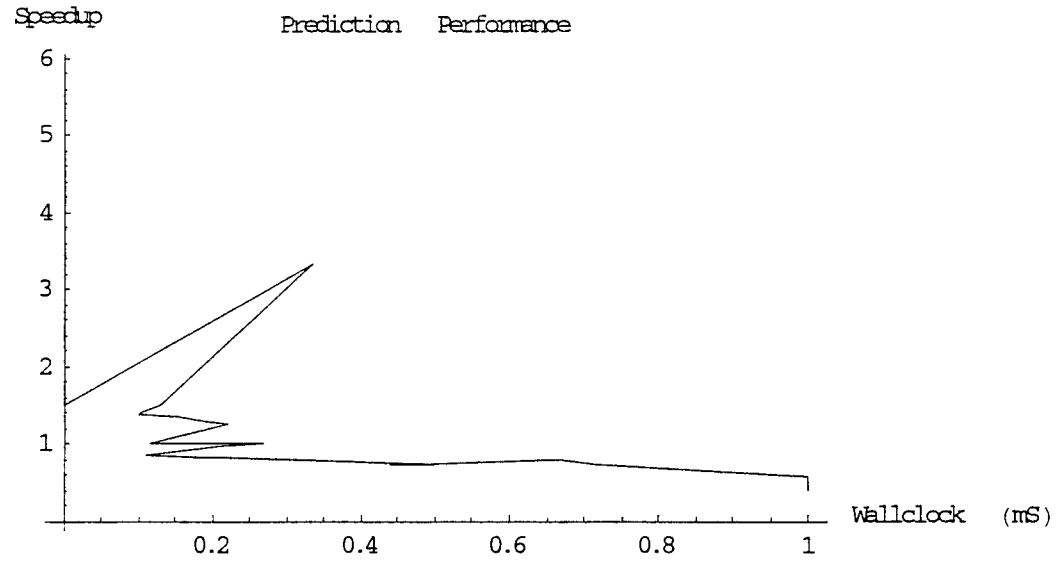


Figure 8.35. Speedup as a Function of Wallclock.

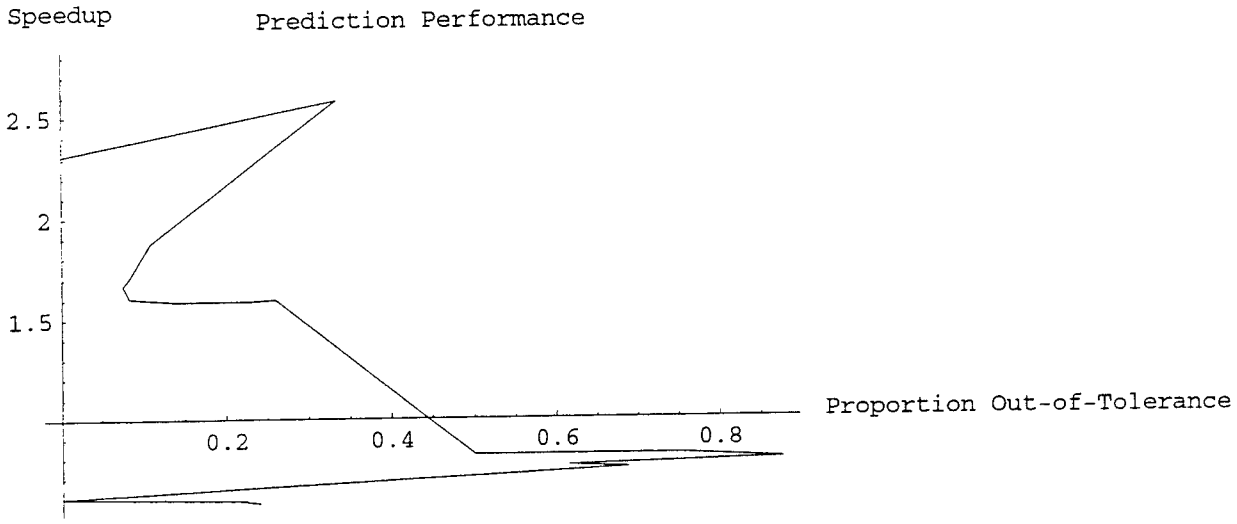


Figure 8.36. Speed as a Function of Proportion Out-of-Tolerance Message with Multiple Driving Processes.

8.3.5 LVT versus Wallclock (lPLVT)

The Local Virtual Time (*LVT*) should maintain a value between wallclock time and the maximum allowed Lookahead. *LVT* starts with a steep positive slope and gradually begins to level off as shown in Figure 8.37 and Figure 8.38. This measurement is made on node AN-1; the node into which two Driving Processes was connected in the multiple Driving Processes experimental validation. *LVT* in the multiple driving process scenarios is more volatile due to the Driving Process synchronization mechanism.

```
makePlot[dir, "lPUptime.AN-1","lPLVT.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "LVT (mS)"}, PlotLabel->"Performance"}]
```

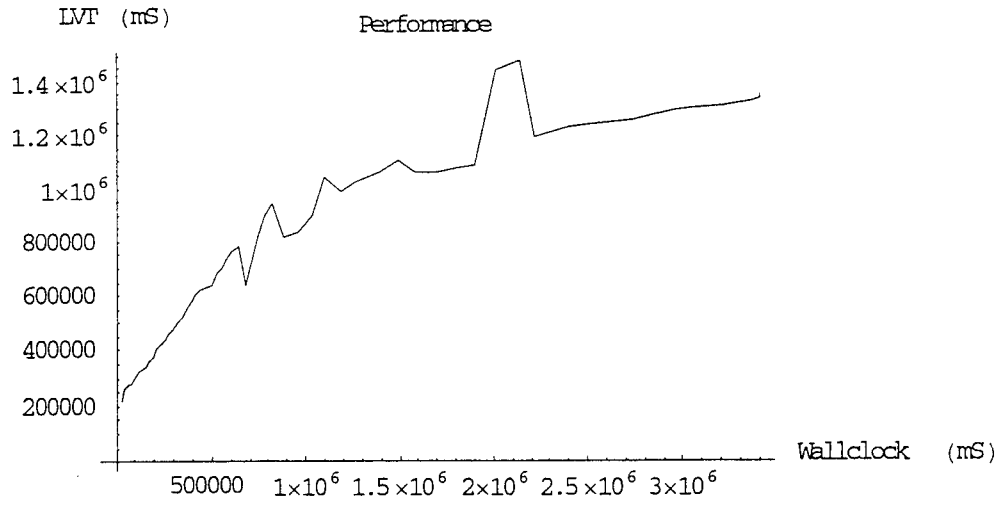


Figure 8.37. LVT as a Function of Wallclock.

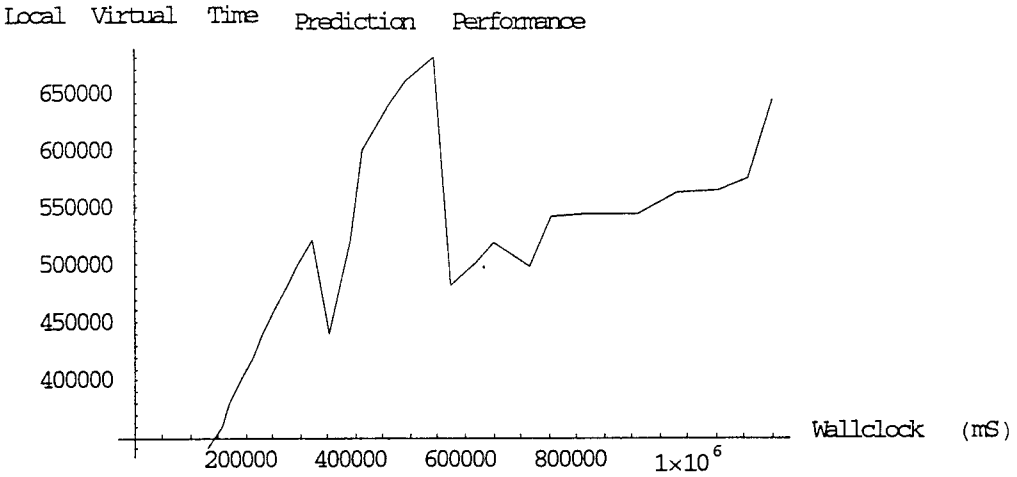


Figure 8.38. LVT as a Function of Wallclock with Multiple Driving Processes.

8.3.6 Virtual Message Rate (IPVmRate)

Figure 8.39, Figure 8.40, and Figure 8.41 show the expected virtual message-processing rate. Rollbacks and activity other than message processing cause the rate to decrease. It is expected that the rate will decrease as the number of rollback events increases. It is somewhat surprising that the rate increases initially. The initial increase could be because there are many rollbacks as the system starts and the predictor within the Driving Processes begins to make better predictions. These initial rollbacks make the virtual message processing rate appear low. Once

the initial "learning" process is over, the virtual message process continues unimpeded until the tolerance tightens enough to cause more rollbacks again.

```
makePlot[dir, "lPUptime.AN-1","lPVmRate.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Virtual Messages"}, PlotLabel-
>"Performance"}]
```

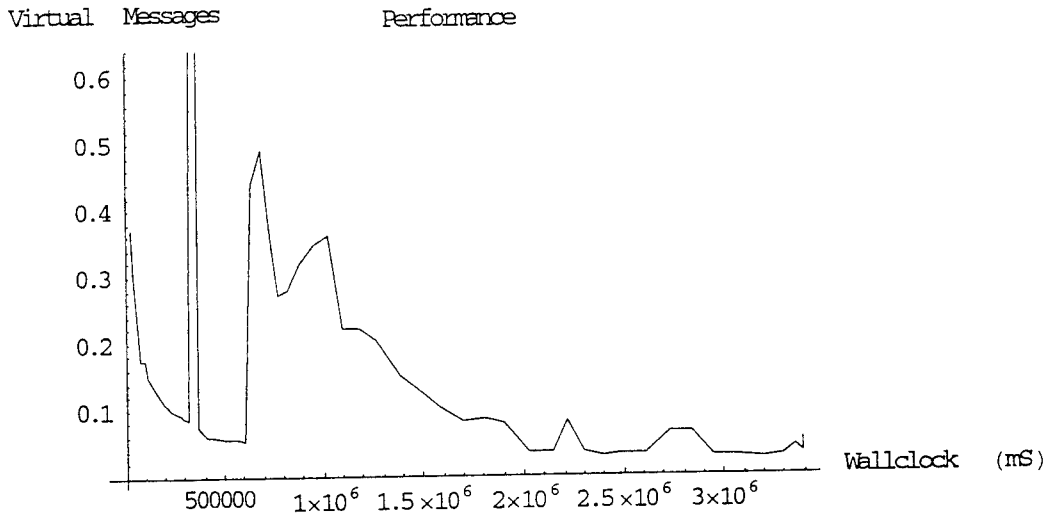


Figure 8.39. Virtual Message Rate as a Function of Wallclock Time.

```
makePlot[dir, "lPPropY.AN-1", "lPVmRate.AN-1", {PlotJoined->True,
AxesLabel->{"Proportion Out-of-Tolerance", "Virtual Messages"},
PlotLabel->"Performance"}]
```

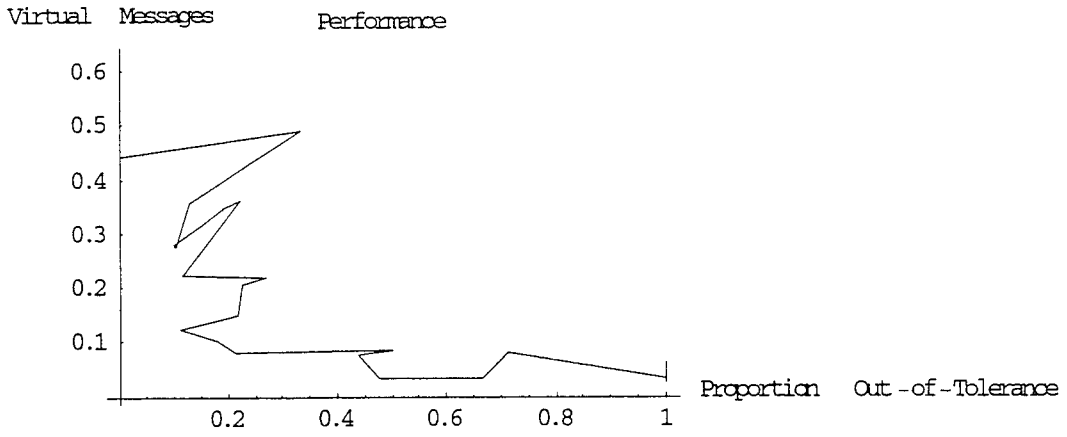


Figure 8.40. Virtual Message Rate as a Function of Proportion of Out-of-Tolerance Messages.

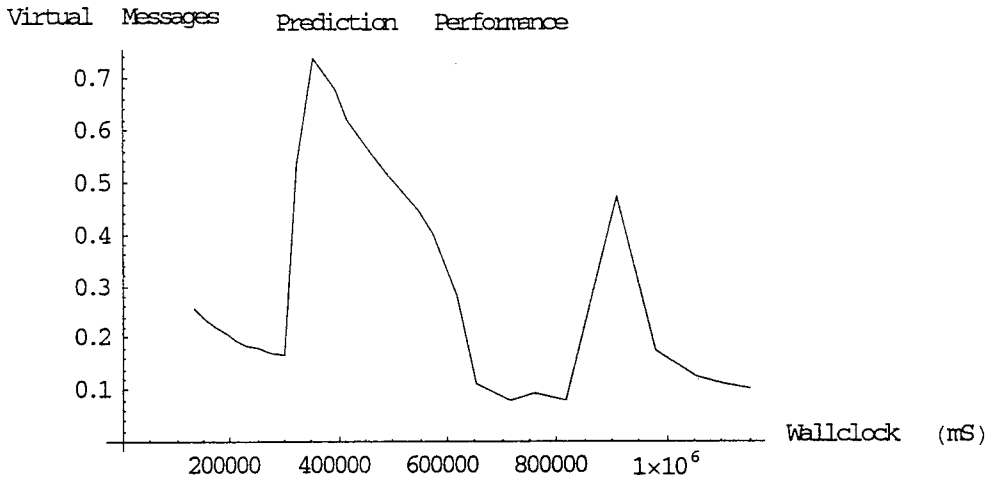


Figure 8.41. Virtual Message Rate as a Function Wallclock Time with Multiple Driving Processes.

8.3.7 Task Execution Time (lPETask)

The task execution time is the wallclock time the system spends executing a non-rollback message. It was expected that this value would be essentially constant; however, it increases in direct proportion to the number of rollbacks as shown in Figure 8.42 and Figure 8.43. This is believed to be because fossil collection is not being used. The increase in the number of values in the state queue is causing access of the state queue and MIB to slow in proportion to the queue size. Figure 8.44 and Figure 8.45 show expected task execution time as a function of the proportion of out-of-tolerance messages. It clearly increases as out-of-tolerance messages increase because these are causing the rollbacks.

```
makePlot[dir, "lPUptime.AN-1","lPETask.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Expected Task Time (mS)"}, PlotLabel-
>"Performance"}]
```

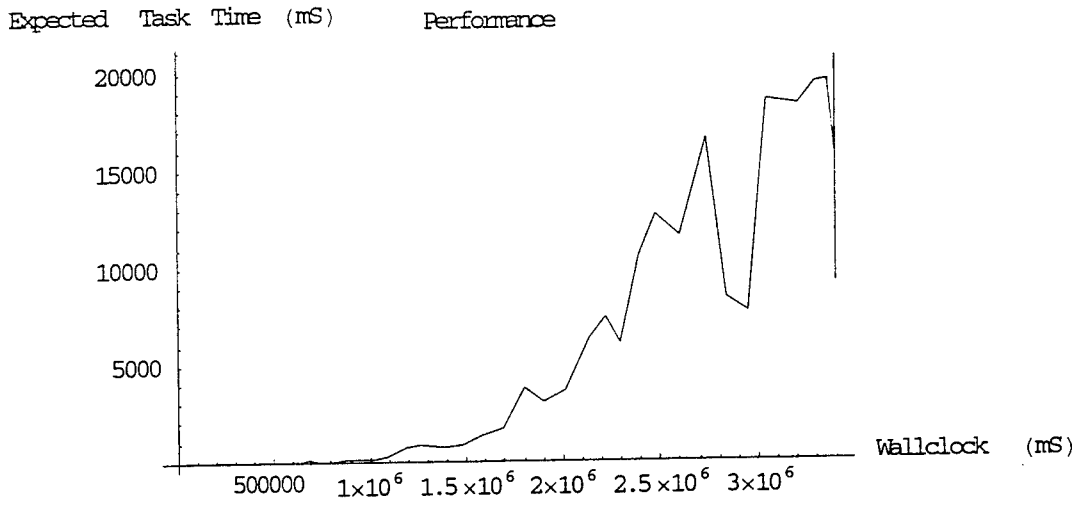


Figure 8.42. Expected Task Execution Time as a Function of Wallclock.

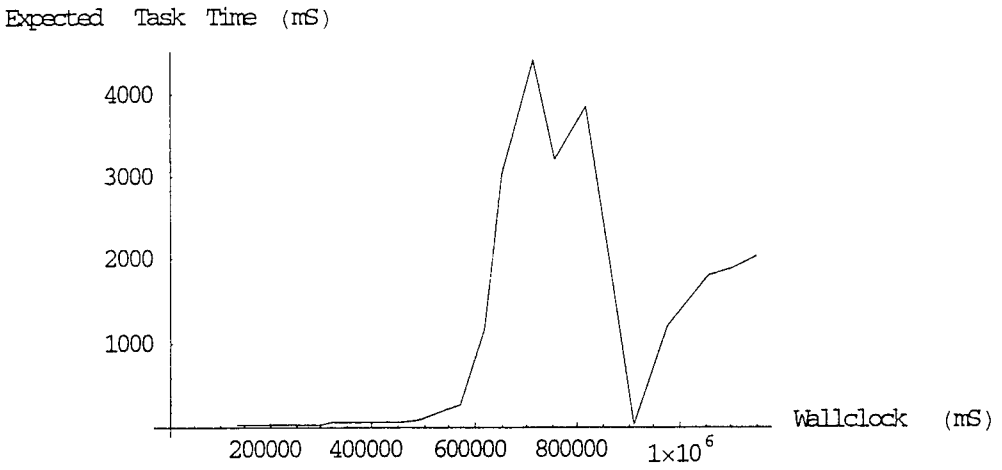


Figure 8.43. Expected Task Execution Time as a Function of Wallclock with Multiple Driving Processes.

```
makePlot[dir, "lPPropY.AN-1","lPETask.AN-1", {PlotJoined->True,
AxesLabel->{"Proportion Out-of-Tolerance", "Task Time (mS)"},
PlotLabel->"Performance"}]
```

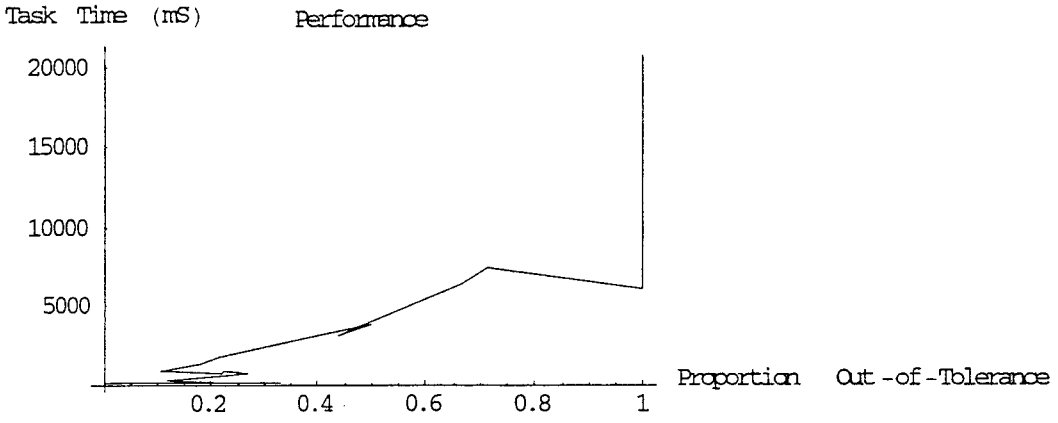


Figure 8.44. Expected Task Time as a Function of Out-of-Tolerance Message Proportion.

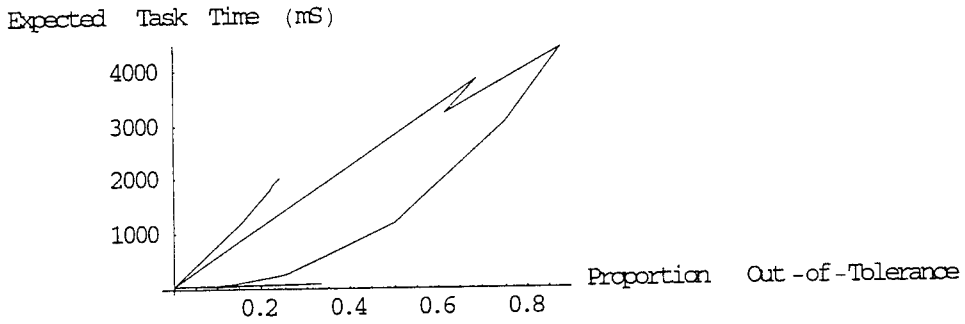


Figure 8.45. Expected Task Time as a Function of Out-of-Tolerance Message Proportion with Multiple Driving Processes.

8.3.8 Load Prediction (loadPredictionPredictedLoad)

Figure 8.46 and Figure 8.47 show a snapshot of the load prediction MIB showing the predicted load. The multiple Driving Process configuration results show approximately twice as much load. The oscillation in this case is believed to be due to the multiple Driving Process synchronization mechanism.

```
makePlot[dir, "loadPredictionPredictedTime.AN-
1.10","loadPredictionPredictedLoad.AN-1.10", {PlotJoined->True,
AxesLabel->{"Predicted Time (mS)", "Predicted Load"}, PlotLabel-
>"Accuracy"}]
```

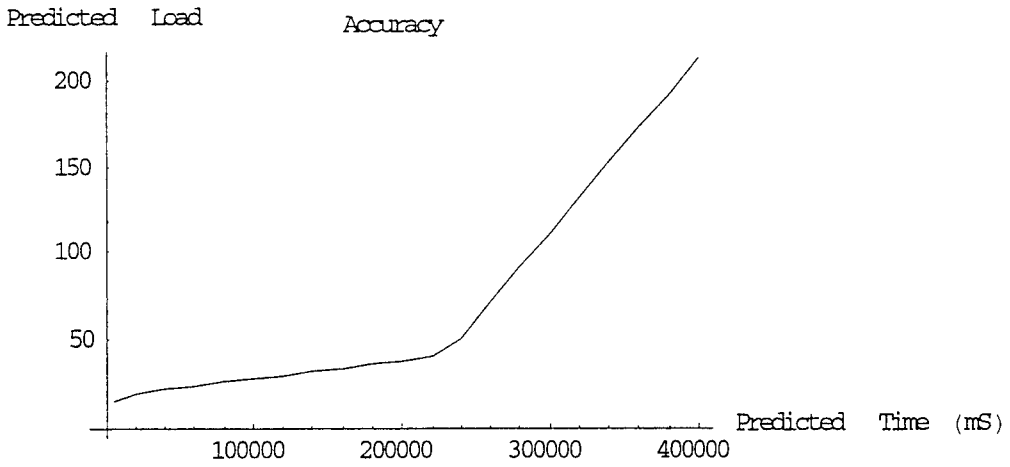


Figure 8.46. A Snapshot of Predicted Load versus Prediction Time of that Load.

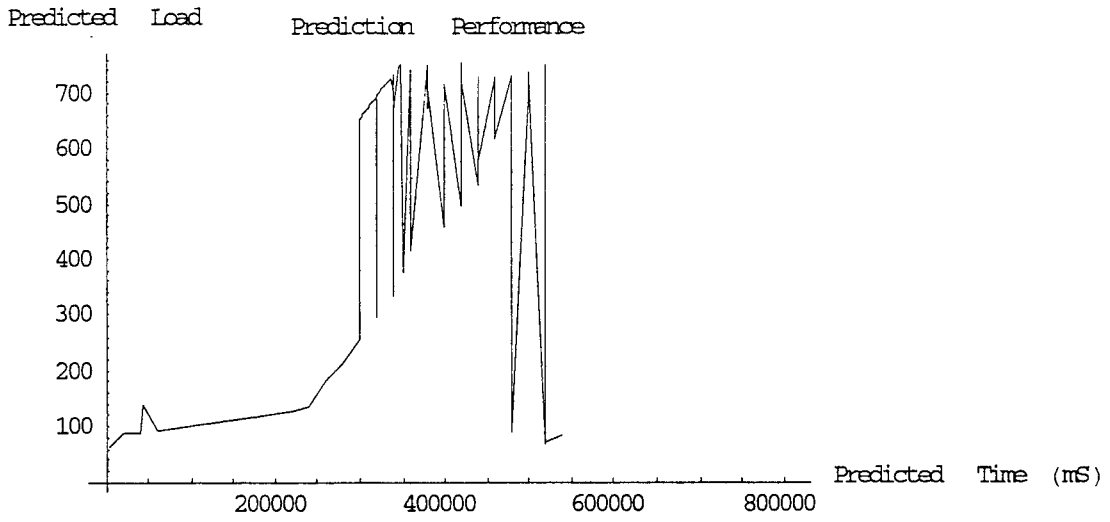


Figure 8.47. A Snapshot of Predicted Load versus Prediction Time of that Load with Multiple Driving Processes.

8.3.9 Rollback Execution Time (IPETrb)

Figure 8.48 and Figure 8.49 show the expected time taken to perform a rollback. It again appears that the expected time to perform a rollback increases with the size of the state queue.

```
makePlot[dir, "1PUptime.AN-1","1PETrb.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Expected Task Time (mS)"}, PlotLabel-
>"Performance"}]
```

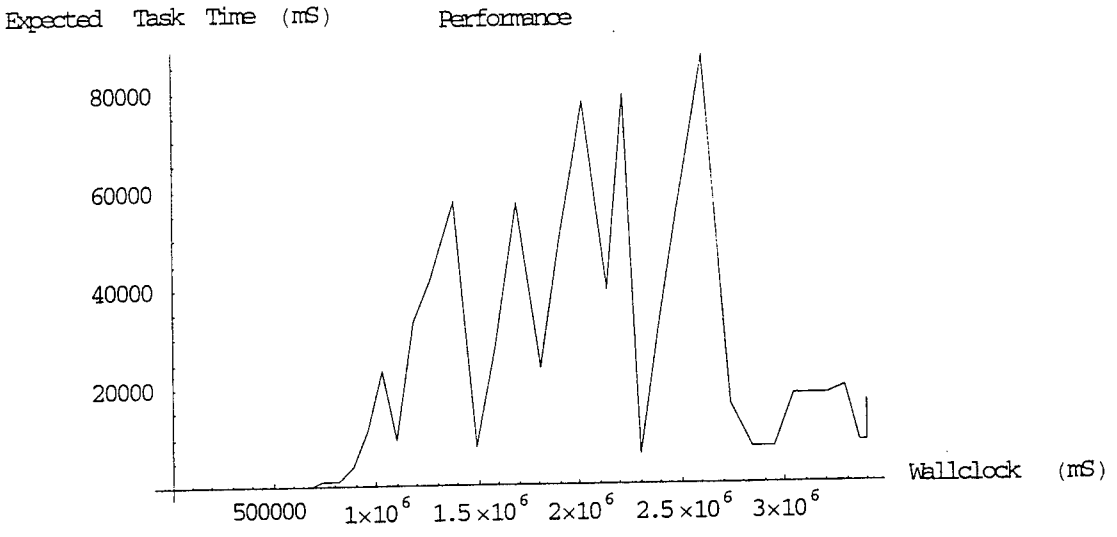


Figure 8.48. Expected Task Execution Time versus Wallclock.

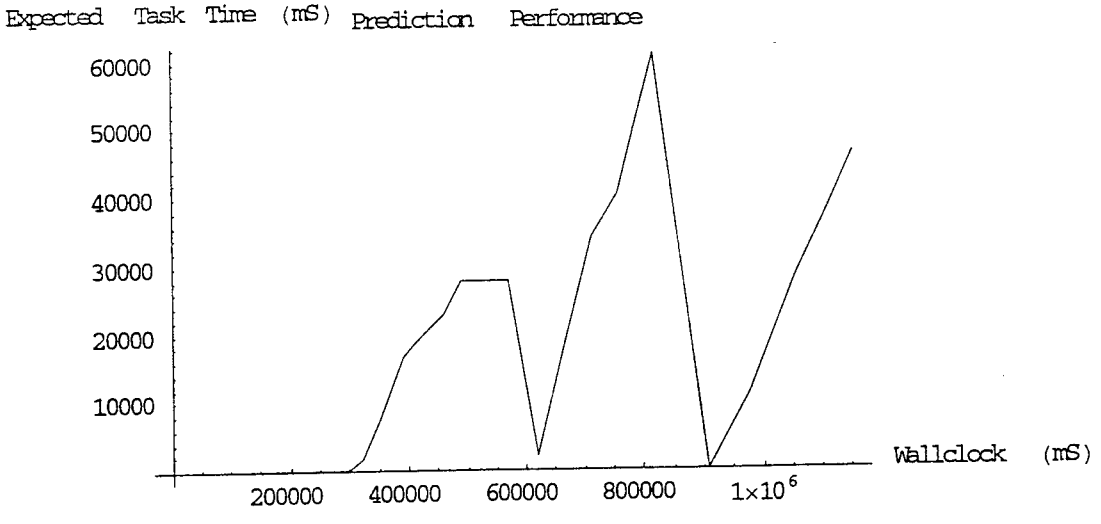


Figure 8.49. Expected Task Execution Time versus Wallclock with Multiple Driving Processes.

Equation 29 shows the combination of rollback statistics used to generate graphs as a function of the total number of rollbacks regardless of their type. Figure 8.50 and Figure 8.51 show the combined number of rollbacks as a function of time.

```
tXroll = Take[getData[dir, "lPPropX.AN-1"], 61];

tYroll = Take[getData[dir, "lPPropY.AN-1"], 61];

troll = tXroll+tYroll;

tm=Take[getData[dir, "lPUptime.AN-1"], 61];

makePlot[tm, troll, {PlotJoined->True, AxesLabel->{"Wallclock (mS)",
"Proportion Rollback Messages"}, PlotLabel->"Performance"}]
```

Equation 29 Combining Rollback Rates for Out-of-Tolerance and Out-of-Order Message Proportions.

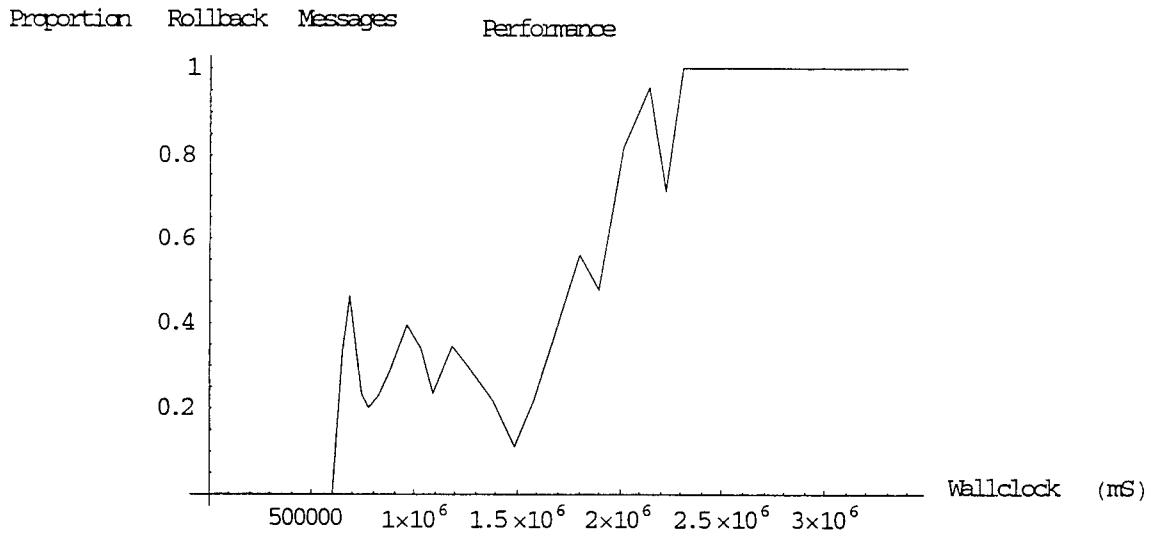


Figure 8.50. Combined Rollbacks versus Wallclock.

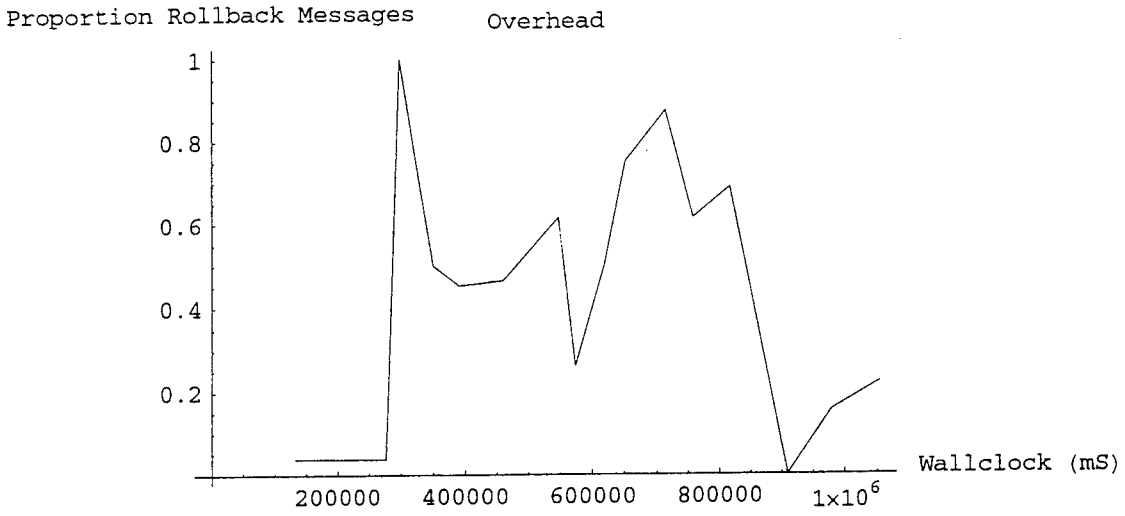


Figure 8.51. Combined Rollbacks versus Wallclock with Multiple Driving Processes.

8.3.10 Expected Task Rollback Time (lPETrb)

Figure 8.52 and Figure 8.53 show the expected task rollback time as a function of wallclock time. Figure 8.54 and Figure 8.55 confirm the suspicion that rollback time increases with State Queue size.

```
makePlot[getData[dir, "lPUptime.AN-1"], getData[dir, "lPETrb.AN-1"],
{PlotJoined->True, AxesLabel->{"Wallclock (mS)", "Rollback Time (mS)"},
PlotLabel->"Performance"}]
```

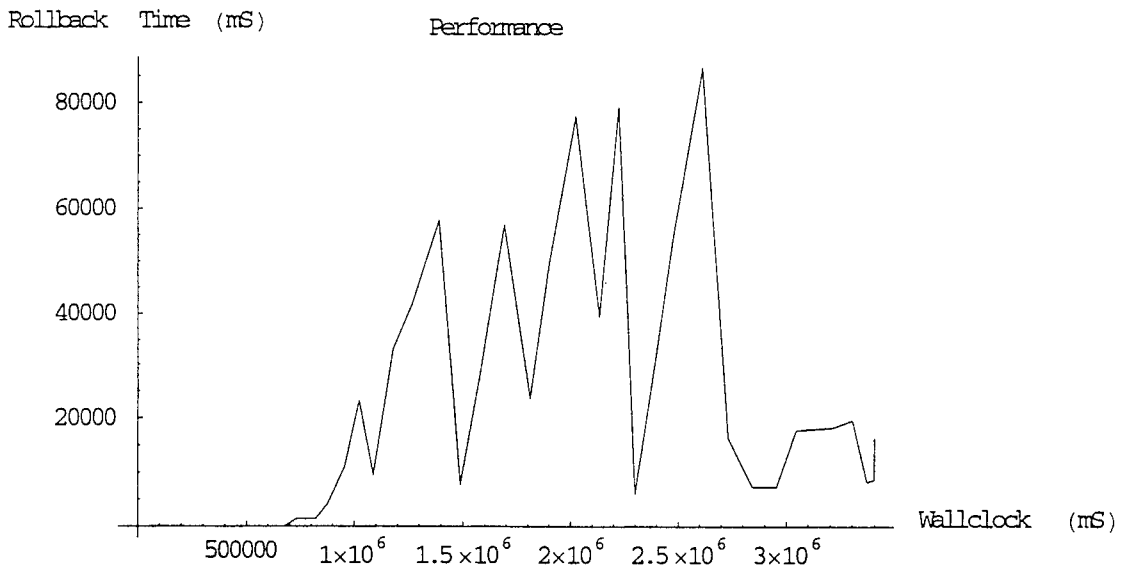


Figure 8.52. Mean Task Rollback Time versus Wallclock.

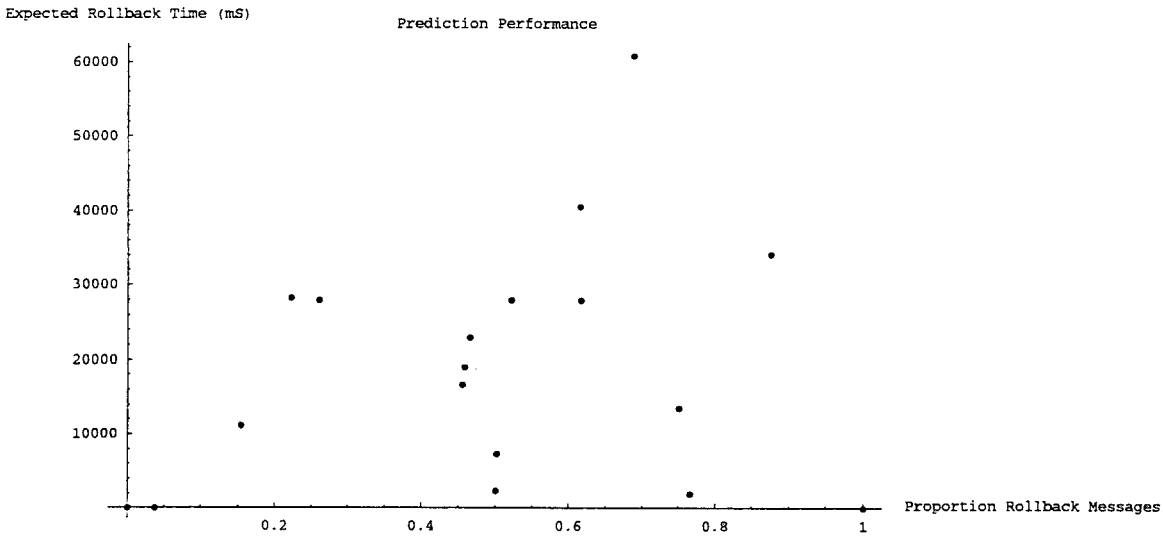


Figure 8.53. Mean Task Rollback Time versus Wallclock with Multiple Driving Processes.

```
makePlot[dir, "lPSQSize.AN-1", "lPETrb.AN-1", {PlotJoined->True,
AxesLabel->{"Rollback Time (mS)", "State Queue Size"}, PlotLabel-
>"Overhead"}]
```

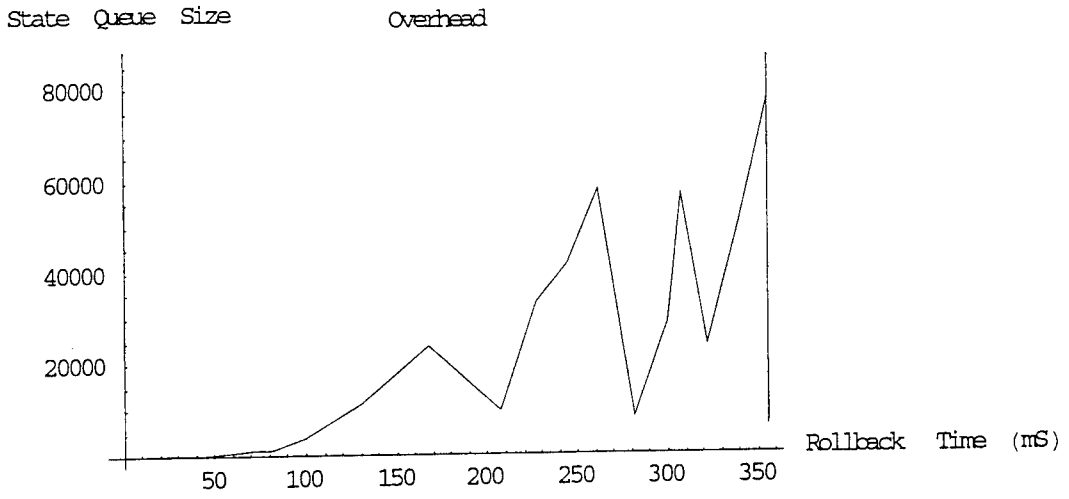


Figure 8.54. State Queue Size versus Wallclock.

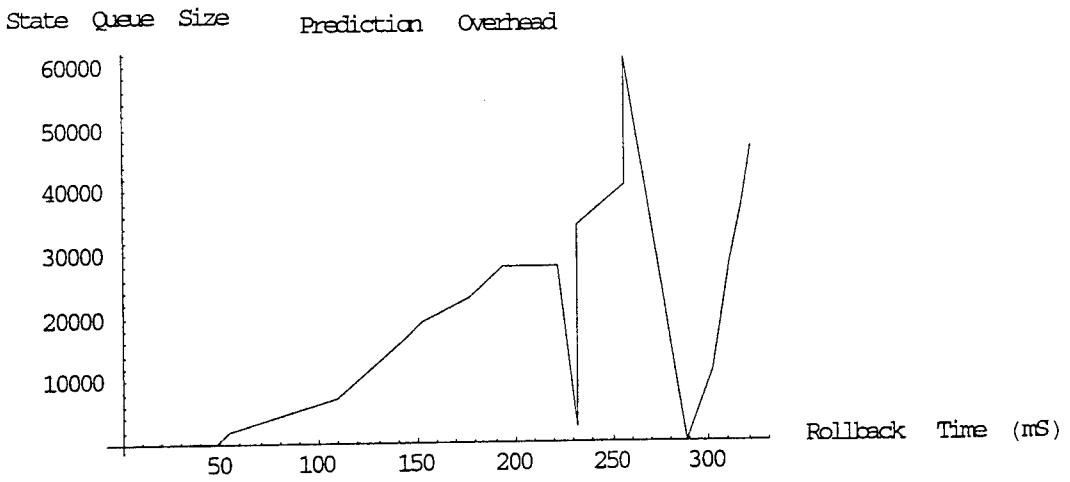


Figure 8.55. State Queue Size versus Wallclock with Multiple Driving Processes.

8.3.11 Out-of-Order Frequency (IPPropX)

Figure 8.56 and Figure 8.57 show the frequency of out-of-order messages. This is expected to be relatively small since in a feed-forward network configuration and larger in a multiple Driving Process network configuration. However, the protocol chosen from the Magician execution environment does not guarantee message order. In addition, rollbacks can cause out-of-order message arrival. This is the proportion of out-of-order messages as a function of tolerance. It is much lower than the proportion of out-of-tolerance messages expected since this is a feed-forward network.

```
makePlot[dir, "lPUptime.AN-1", "lPPropX.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Proportion  Out-of-Order"}, PlotLabel-
>"Overhead"}]
```

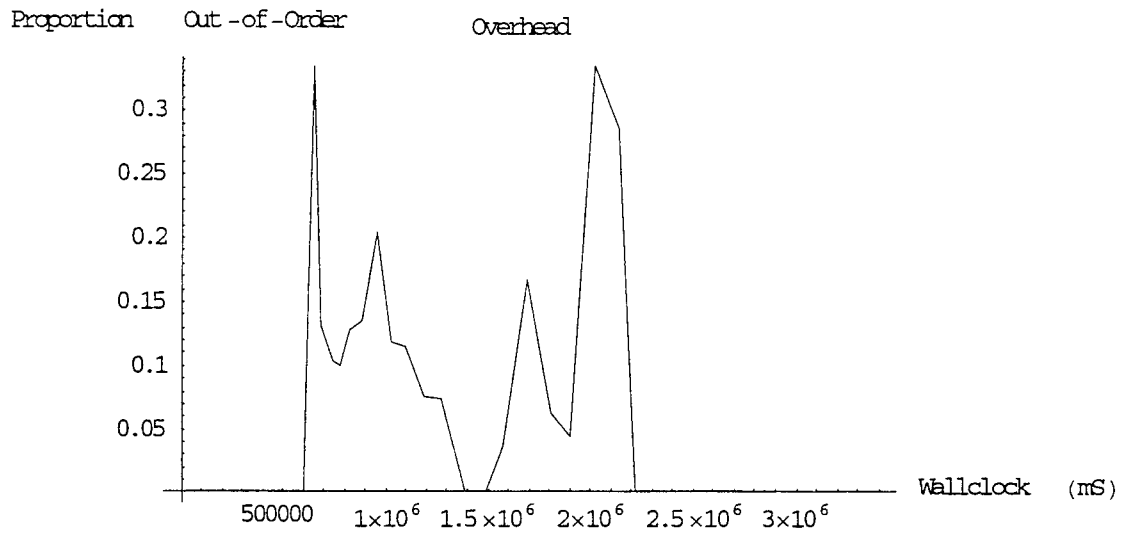


Figure 8.56. Proportion Out-of-Order versus Wallclock.

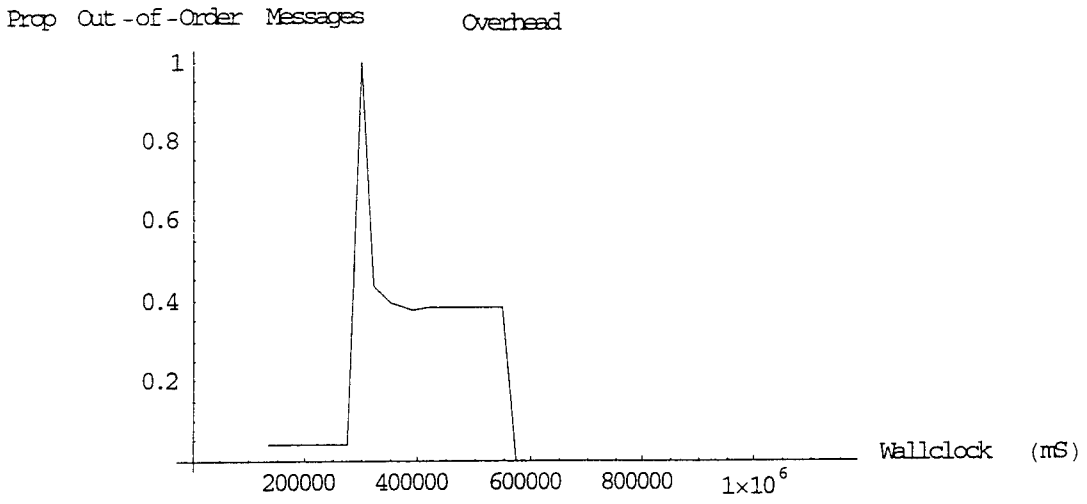


Figure 8.57. Proportion Out-of-Order versus Wallclock with Multiple Driving Processes.

8.3.12 Out-of-Tolerance Frequency (lPPropY)

Figure 8.58 and Figure 8.59 show the proportion of out-of-tolerance messages. Clearly this should increase as tolerance decreases and will thus increase over time as tolerance is programmed to decrease during execution.

```
makePlot[dir, "lPActTolerance.AN-1", "lPPropY.AN-1", {PlotJoined->True,
AxesLabel->{"Tolerance", "Proportion Out-of-Tolerance"}, PlotLabel-
>"Overhead"}]
```

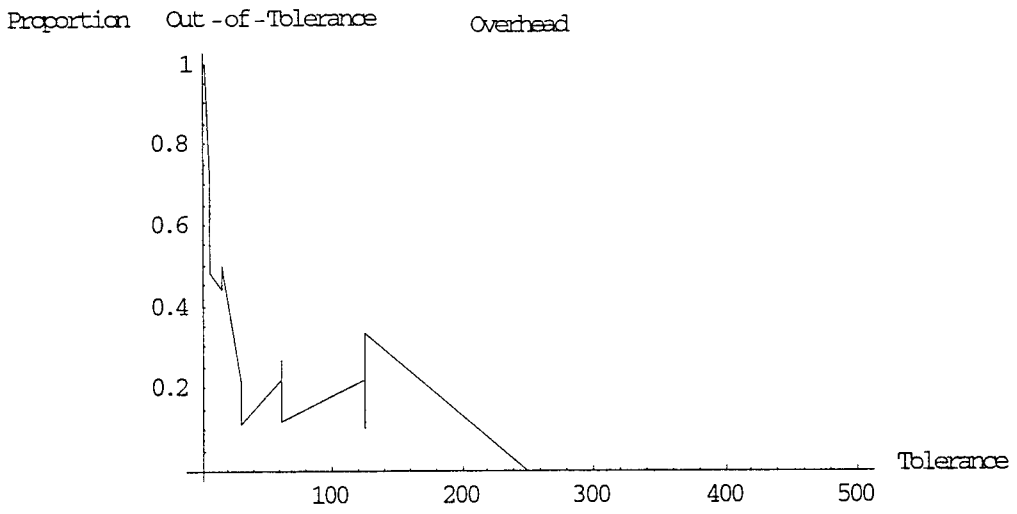


Figure 8.58. Proportion of Out-of-Tolerance Messages versus Tolerance.

```
makePlot[dir, "1PUptime.AN-1", "1PPropY.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Proportion Out-of-Tolerance"},
PlotLabel->"Overhead"}]
```

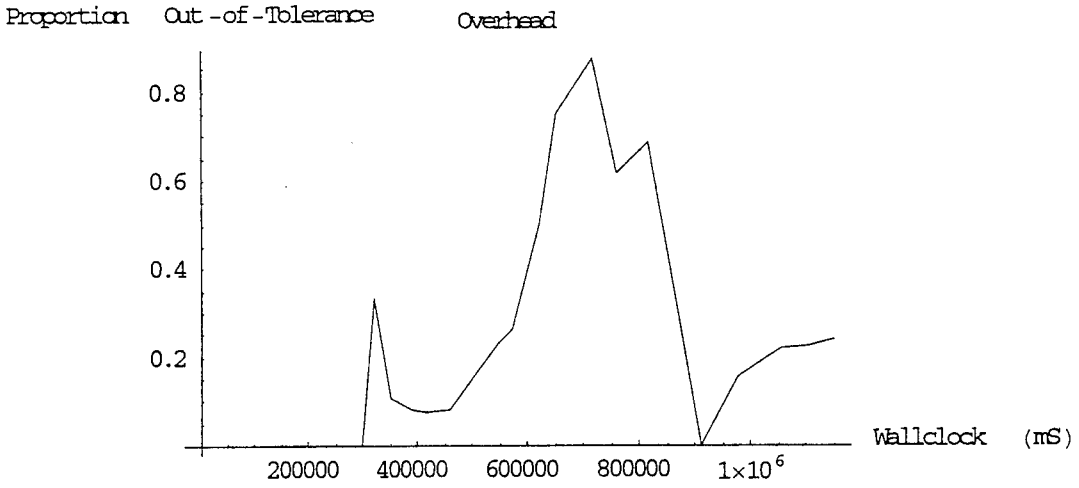


Figure 8.59. Proportion of Out-of-Tolerance Messages versus Wallclock Time with Multiple Processes.

8.3.13 Queue Sizes (lPSQSize, lPQSSize, lPQRSize)

This section examines the queue sizes of the various queues in AVNMP. Figure 8.60, Figure 8.61, Figure 8.62, Figure 8.63, Figure 8.64 and Figure 8.65 show the rate of queue size increases versus Wallclock time. As stress increases, the rate of addition of values to the state queue decreases because most of the time is used to accomplish rollback. However, during this time of stress, the Send Queue and Receive Queue continue to increase slightly as anti-messages are transmitted.

```
makePlot[dir, "lPUptime.AN-1","lPSQSize.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "State Queue Size"}, PlotLabel-
>"Overhead"}]
```

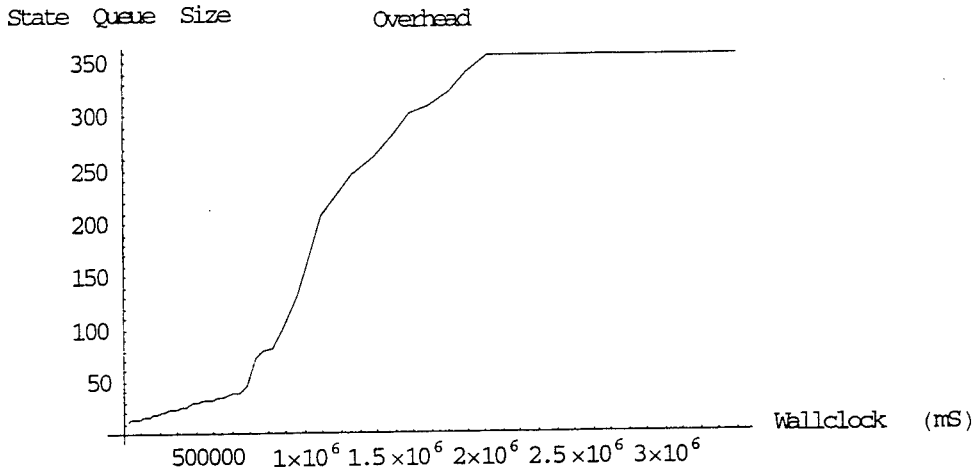


Figure 8.60. State Queue Size versus Wallclock.

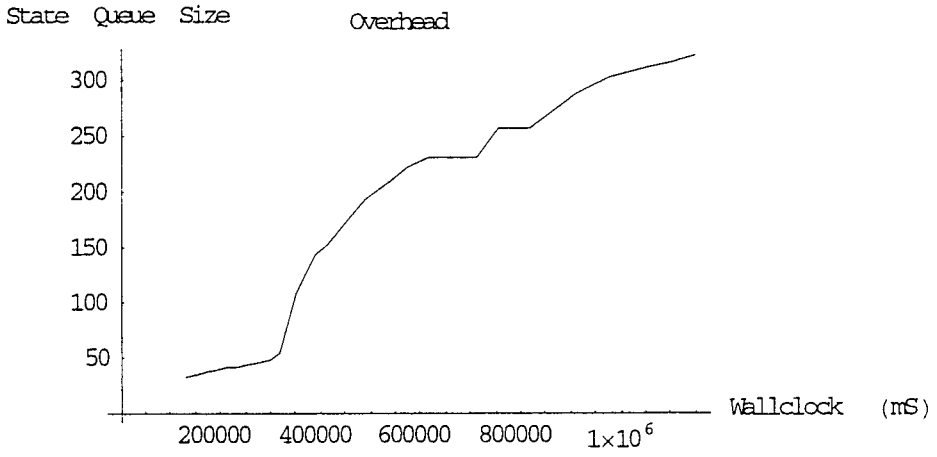


Figure 8.61. State Queue Size versus Wallclock with Multiple Driving Processes.

```
makePlot[dir, "lPUptime.AN-1","lPQSSize.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Send Queue Size"}, PlotLabel-
>"Overhead"}]
```

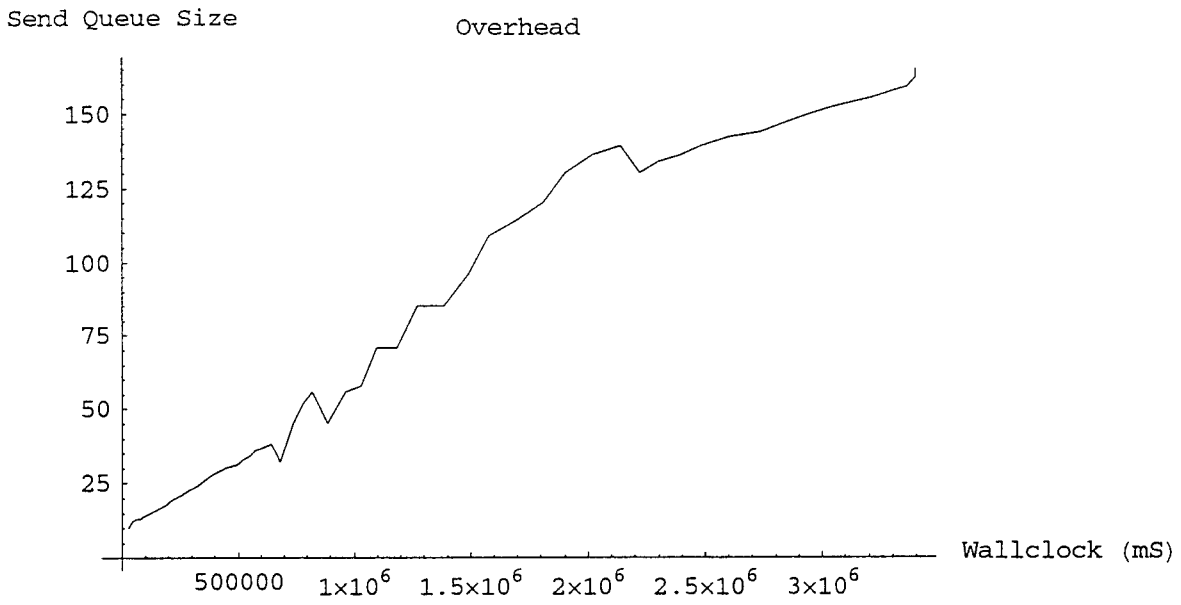


Figure 8.62. Send Queue Size versus Wallclock.

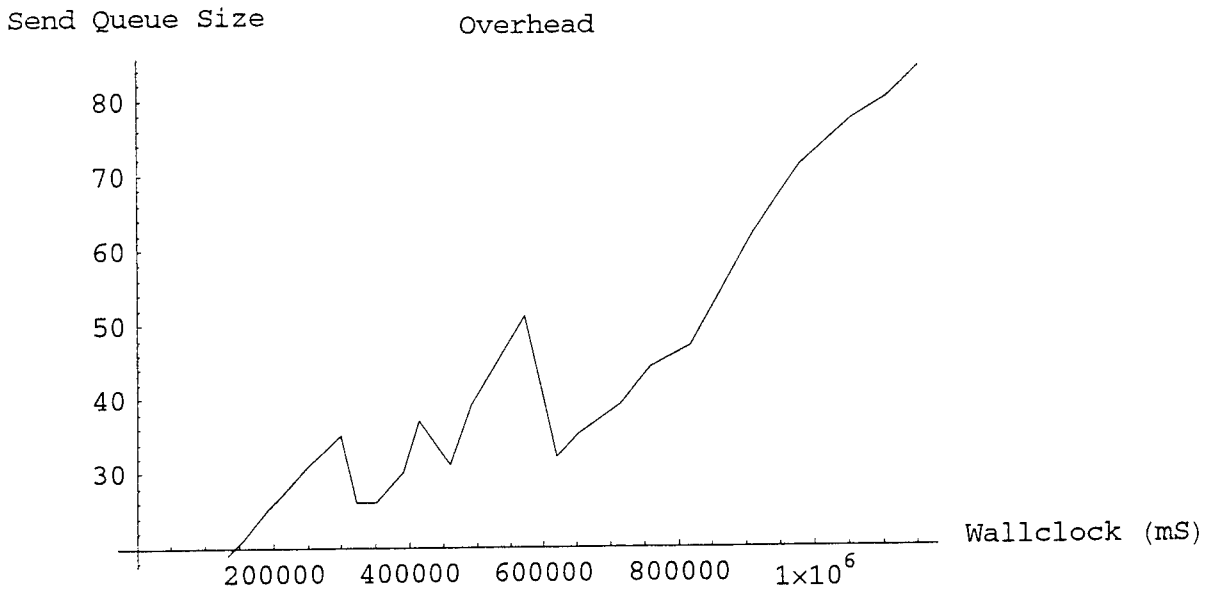


Figure 8.63. Send Queue Size versus Wallclock with Multiple Processes.

```
makePlot[dir, "1PUptime.AN-1","1PQRSize.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Receive Queue Size"}, PlotLabel-
>"Overhead"}]
```

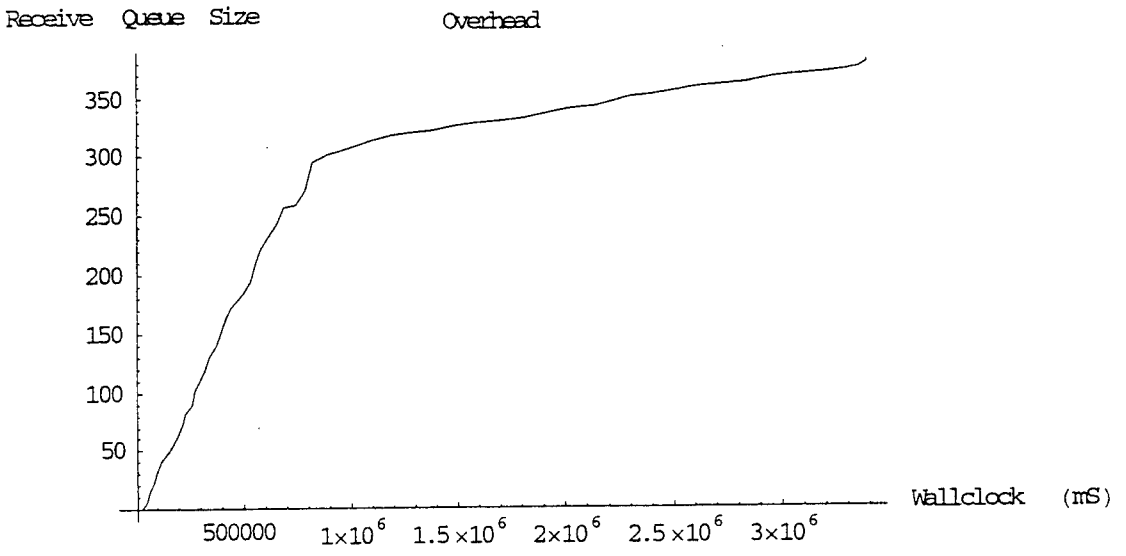


Figure 8.64. Receive Queue Size versus Wallclock.

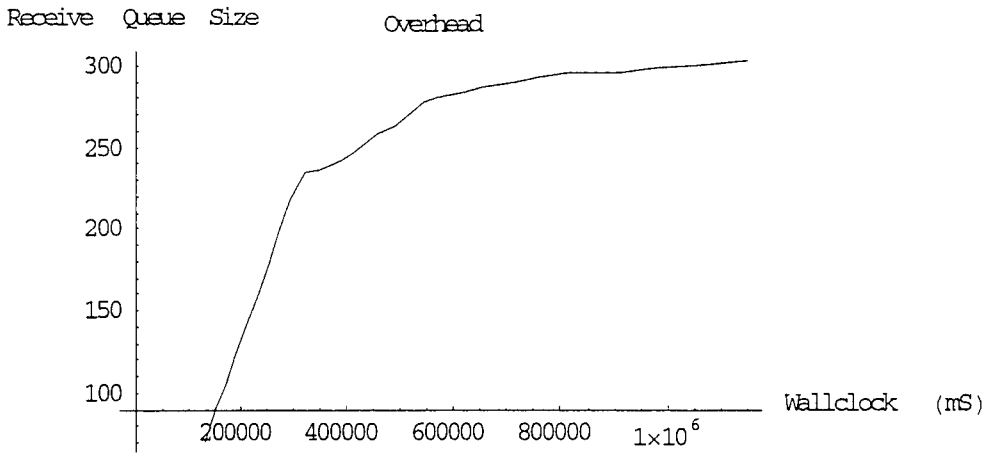


Figure 8.65. Receive Queue Size versus Wallclock with Multiple Processes.

8.3.14 Total Number of All Message Types Processed (lPNumPkts)

Figure 8.66 and Figure 8.67 show the total number of all message types that are processed by the Logical Process. Note that this is reset after runMinutes, which in this case is 5 minutes or 300,000 milliseconds.

```
makePlot[dir, "lPUptime.AN-1","lPNumPkts.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Packets"}, PlotLabel->"Overhead"}]
```

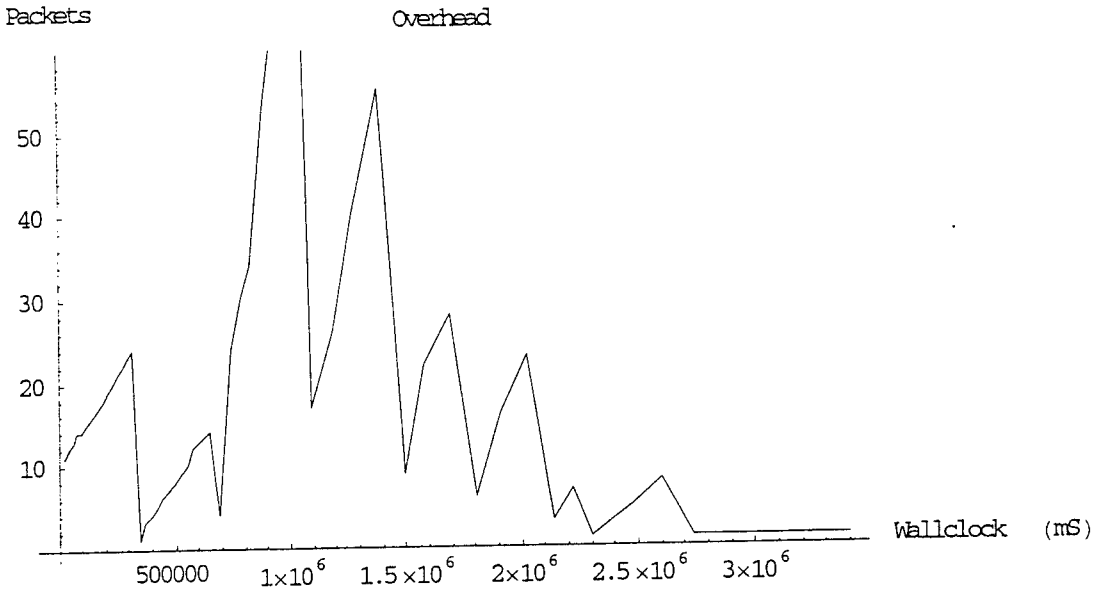


Figure 8.66. Total Number of Messages Processed versus Wallclock.

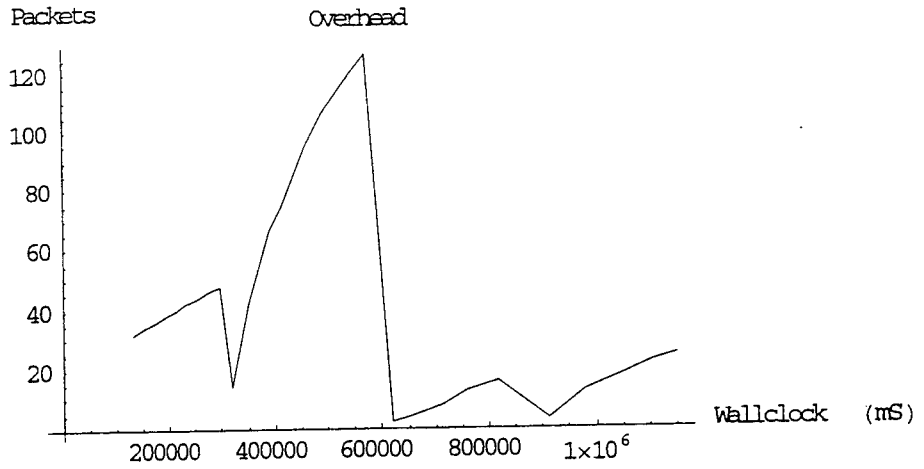


Figure 8.67. Total Number of Messages Processed versus Wallclock with Multiple Processes.

8.3.15 Number of Virtual Messages (IPVirtual)

Figure 8.68 and Figure 8.69 show the total number of virtual messages processed. The ability to process virtual messages decreases as the system becomes stressed with rollback and increasing queue sizes.

```
makePlot[dir, "1PUptime.AN-1","1PVirtual.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Virtual Messages"}, PlotLabel-
>"Overhead"}]
```

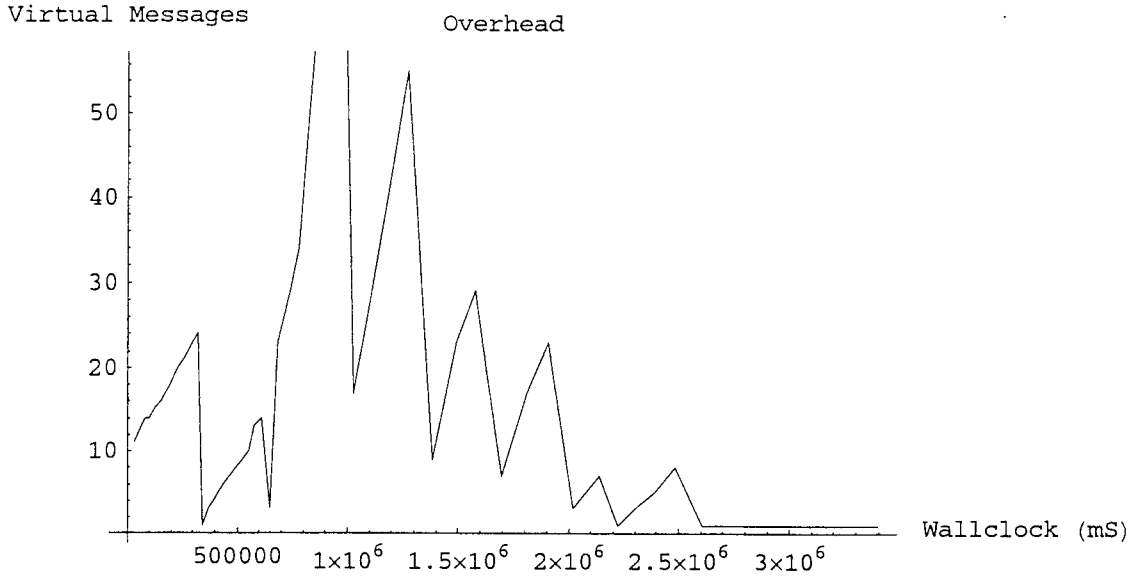


Figure 8.68. Number of Virtual Messages versus Wallclock.

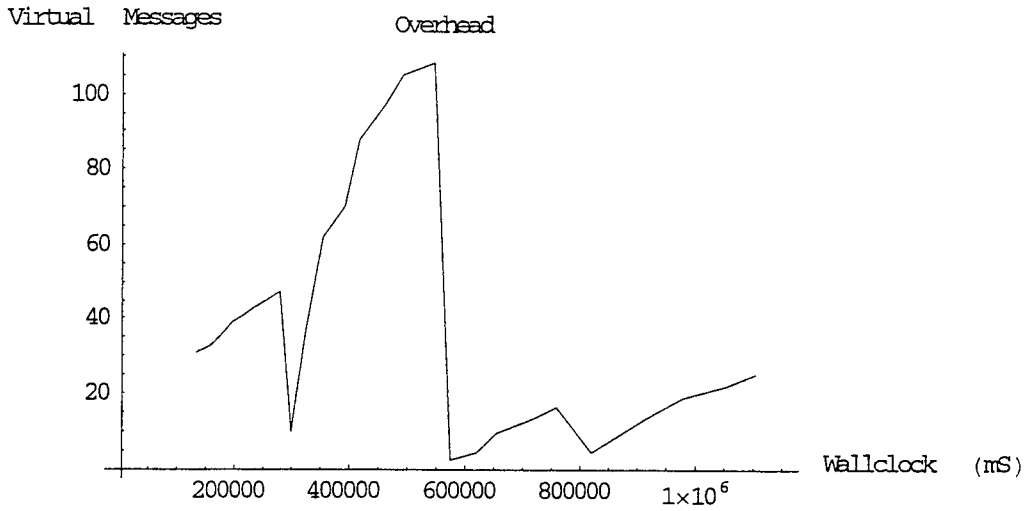


Figure 8.69. Number of Virtual Messages versus Wallclock with Multiple Driving Processes.

8.3.16 Number of Anti-Messages (lPNumAnti)

Figure 8.70 and Figure 8.71 display the total number of anti-messages. This is expected to increase over time. This value is reset every runMinutes, which in this case is 300,000 milliseconds. This is the total number of anti-messages produced over wallclock time.

```
makePlot[dir, "lPUptime.AN-1","lPNumAnti.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Anti-Messages"}, PlotLabel->"Overhead"}]
```

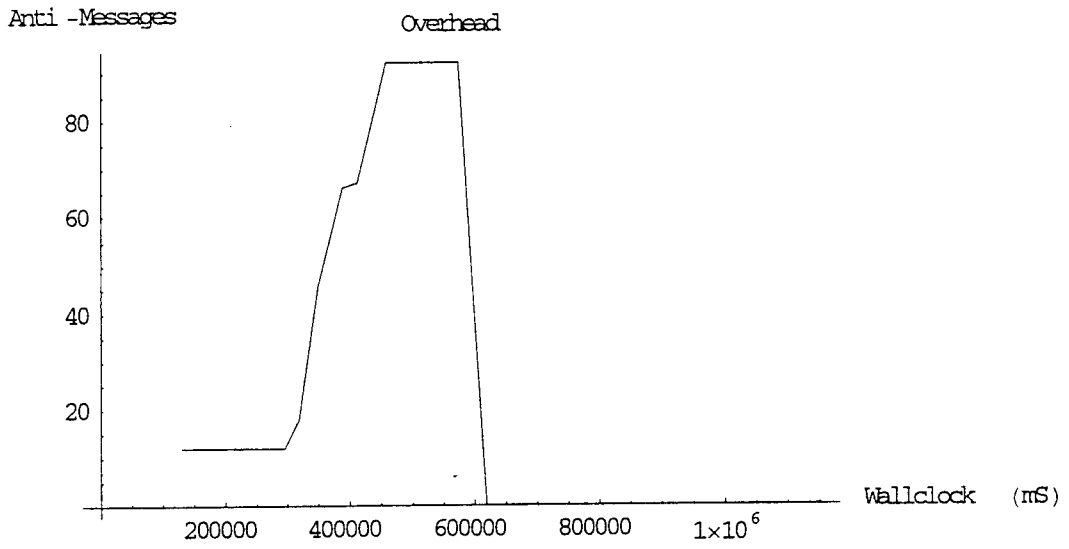


Figure 8.70. Number of Anti-Messages versus Wallclock with Multiple Driving Processes.

```
makePlot[dir, "lPUptime.AN-1", "lPNumAnti.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "AntiMessages"}, PlotLabel->"Overhead"}]
```

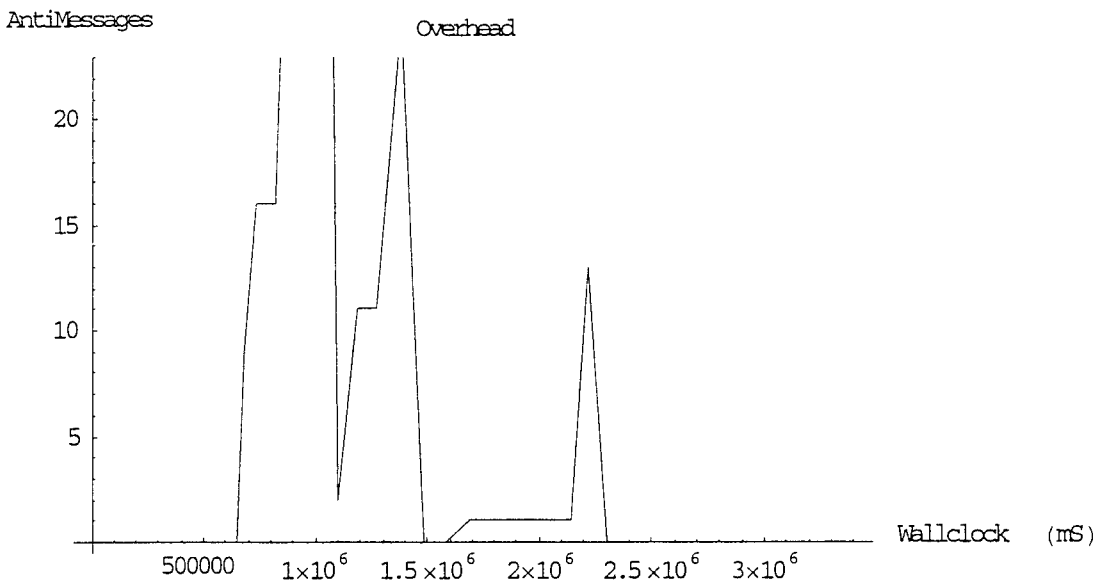


Figure 8.71. Number of Anti-Messages versus Wallclock.

8.3.17 Difference between actual value and closest Send Queue packet value (IPStateError)

Figure 8.72 and Figure 8.73 show the difference between the application value and the closest in time send queue message value. This is the difference between the send queue value and actual application value over wallclock time. Clearly, the prediction error decreases in order to meet the tighter tolerance.

```
makePlot[dir, "lPUptime.AN-1","lPStateError.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Prediction Error"}, PlotLabel-
>"Accuracy"}]
```

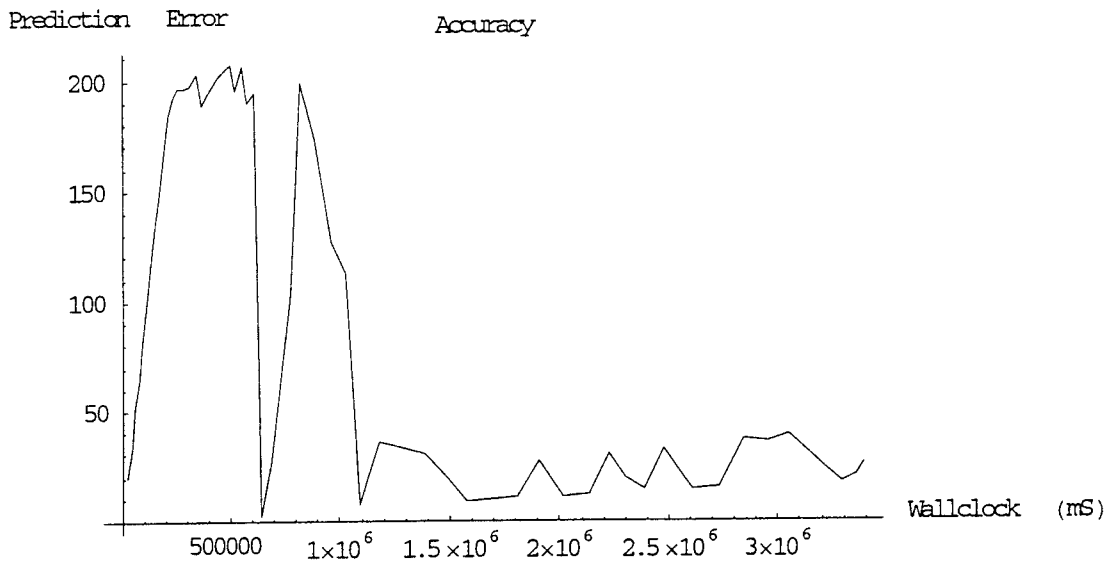


Figure 8.72. Prediction versus Wallclock.

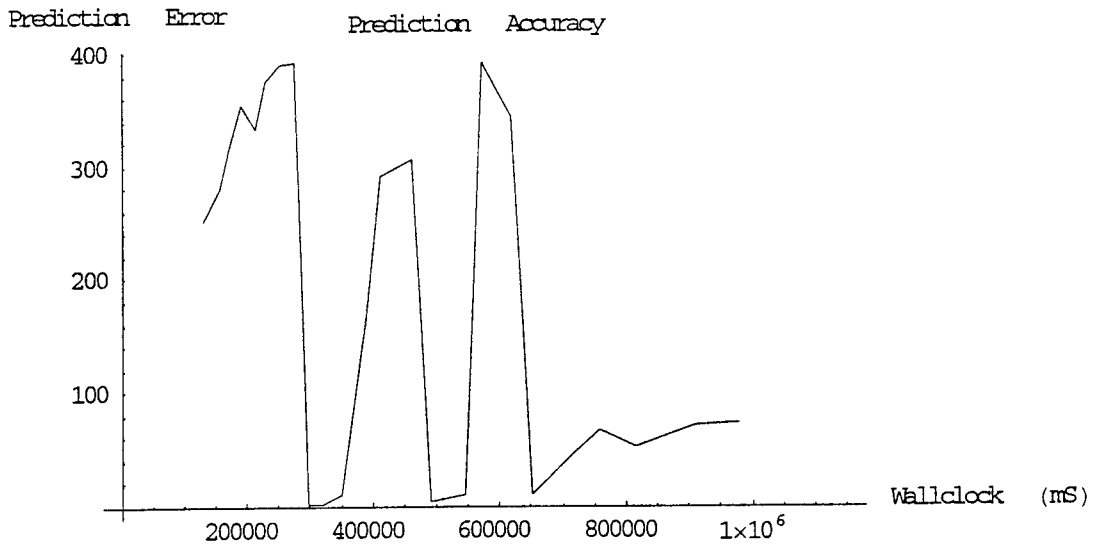


Figure 8.73. Prediction versus Wallclock with Multiple Driving Processes.

8.3.18 Time Difference (lPTdiff)

Figure 8.74 and Figure 8.75 display the difference between the time of the actual value and the predicted time with which it is compared in order to determine out-of-tolerance conditions. Clearly, these values should be as close in possible in time so that a fair comparison can be made. As the system is stressed, it becomes harder to find predicted values that are close to actual values in time. This is likely to be due to the fact that fewer predictions are being made and the predictions are farther apart, making an exact time match with actual harder to obtain.

```
makePlot[dir, "lPUptime.AN-1","lPTdiff.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Time Difference"}, PlotLabel-
>"Accuracy"}]
```

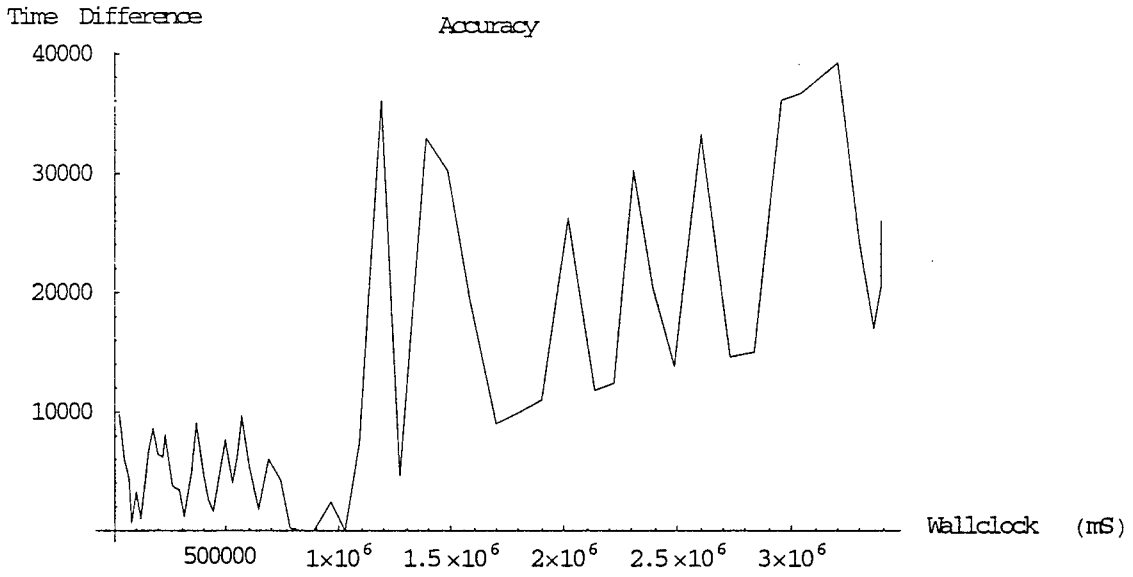


Figure 8.74. Time Difference in Prediction Check versus Wallclock.

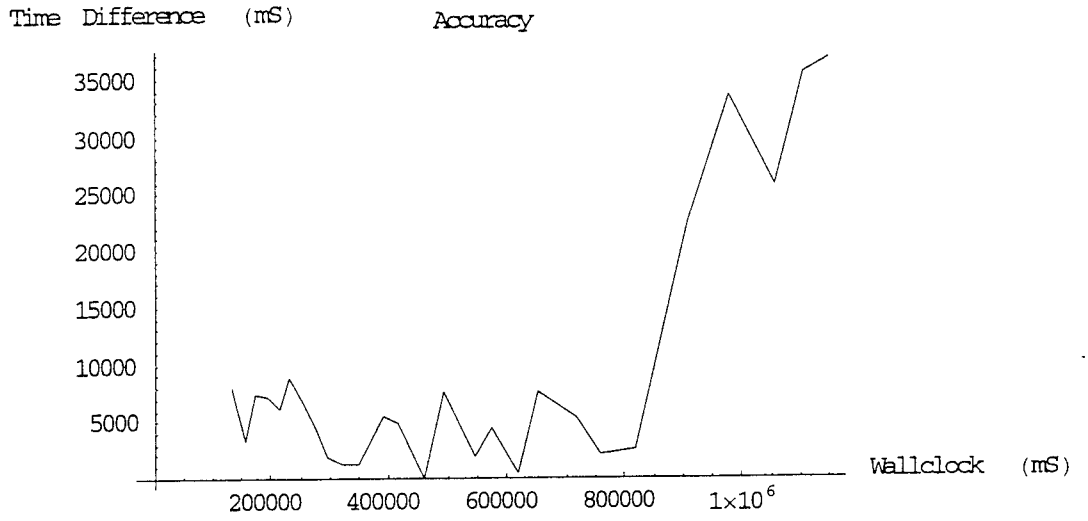


Figure 8.75. Time Difference in Prediction Check versus Wallclock with Multiple Driving Processes.

8.3.19 Number of Causality Rollbacks (lPCausalityRollbacks)

Figure 8.76 and Figure 8.77 display the total number of causality rollbacks. This is anticipated to occur early for the multiple Driving Process configurations as the Driving Processes synchronize among themselves. In order to further stress the system, Magician best-effort packet delivery is being used. This means that packets are not guaranteed to arrive in order, or at all. However, the large number of causality rollbacks in the multiple Driving Process scenario is due to the synchronization among the Driving Processes.

```
makePlot[dir, "lPUptime.AN-1","lPCausalityRollbacks.AN-1", {PlotJoined-
>True, AxesLabel->{"Wallclock (mS)", "Causality Rollbacks"}, PlotLabel-
>"Overhead"}]
```

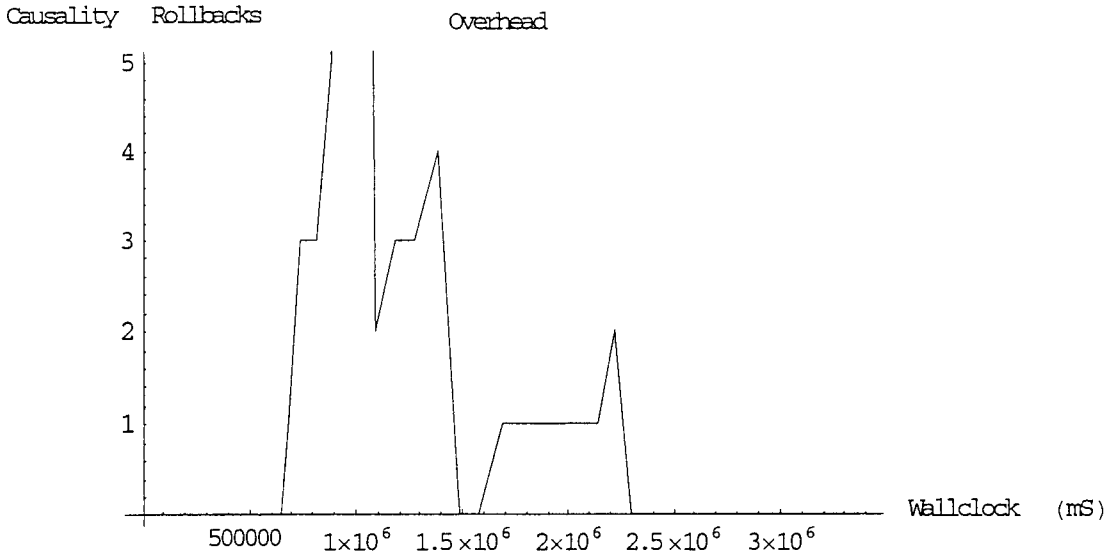


Figure 8.76. Number of Causality Rollbacks versus Wallclock.

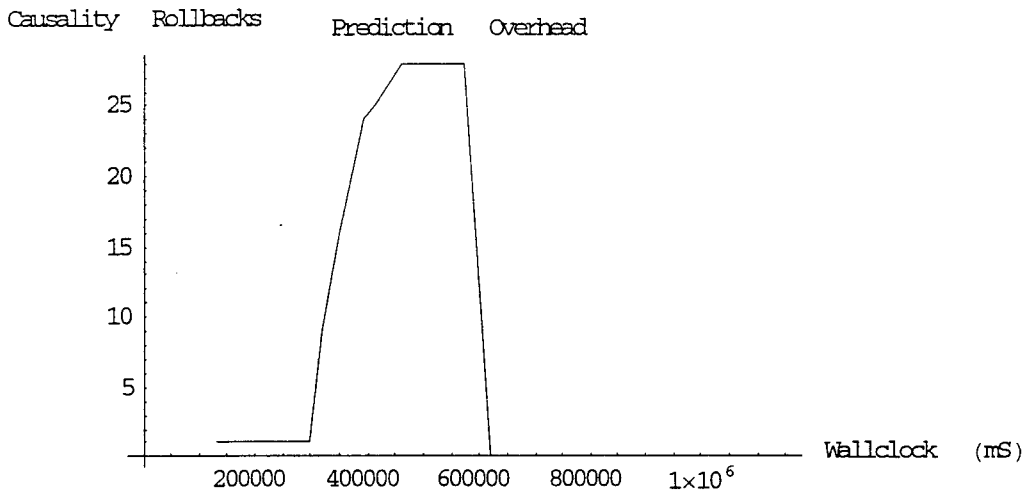


Figure 8.77. Number of Causality Rollbacks versus Wallclock with Multiple Driving Processes.

8.3.20 Number of Tolerance Rollbacks (lPToleranceRollbacks)

Figure 8.78 and Figure 8.79 display the total number of tolerance based rollbacks. These appear to decrease. However, in proportion to the total number of packets processed, tolerance-based rollbacks are actually an increasing proportion because the total number of packets is decreasing over time due to exploding queue sizes and the increasing number of rollbacks.

```
makePlot[dir, "lPUptime.AN-1","lPToleranceRollbacks.AN-1", {PlotJoined-
>True, AxesLabel->{"Wallclock (mS)", "Tolerance Rollbacks"}, PlotLabel-
>"Overhead"}]
```

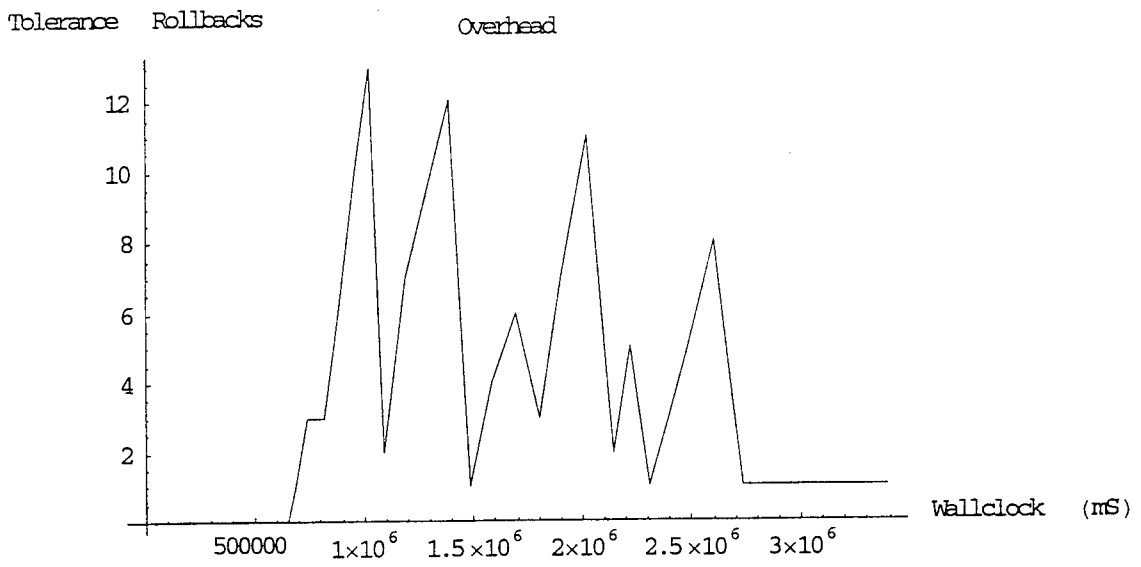


Figure 8.78. Number of Tolerance Rollbacks versus Wallclock.

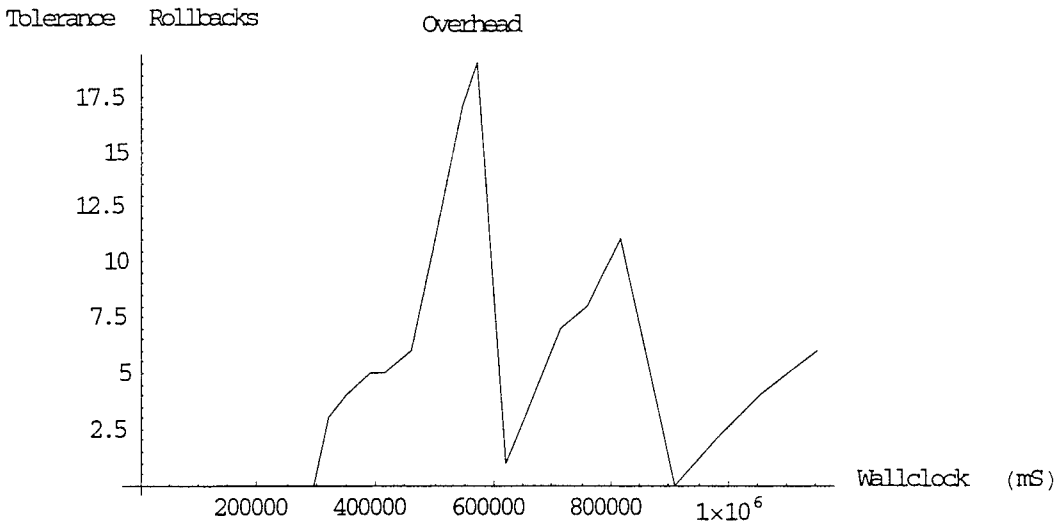


Figure 8.79. Number of Tolerance Rollbacks versus Wallclock with Multiple Driving Processes.

8.3.21 State Error (lPStateError)

Figure 8.80 and Figure 8.81 display the difference between the predicted and actual application values. Clearly in both the feed-forward and multiple Driving Process scenarios, the error is within the required tolerance and decreases appropriately.

```
makePlot[dir, "lPUptime.AN-1","lPStateError.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "State Error"}, PlotLabel->"Overhead"}]
```

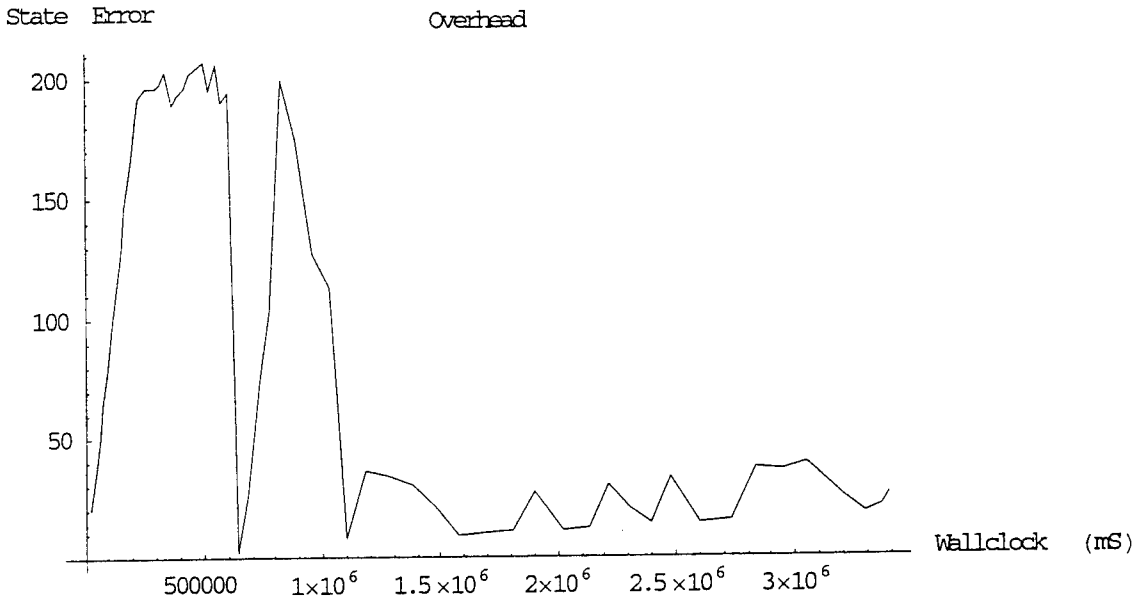


Figure 8.80. Prediction Error versus Wallclock.

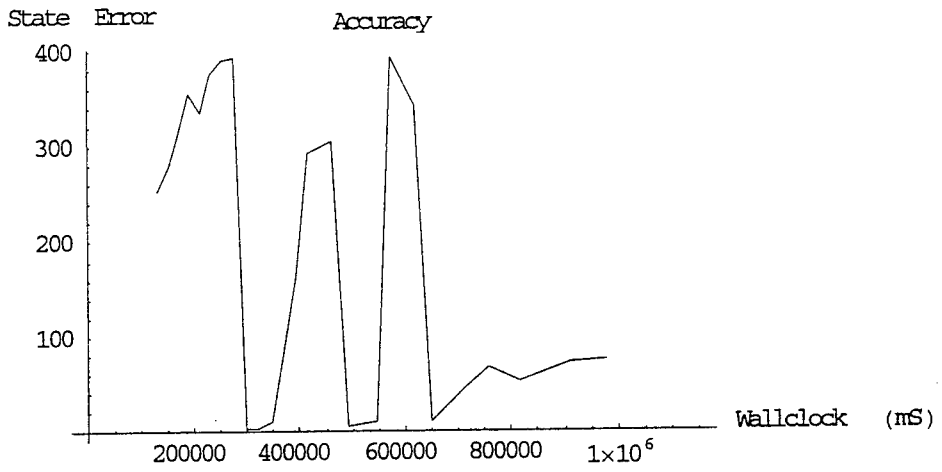


Figure 8.81. Prediction Error versus Wallclock with Multiple Driving Processes.

8.3.22 Lookahead Analysis versus Actual

Figure 8.82 and Figure 8.83 show the Lookahead as function of the proportion of out-of-tolerance messages. In the feed-forward network configuration, Lookahead reduces as out-of-tolerance messages increase. However, this is not so clearly the case in the multiple Driving

Process configuration. This is likely to be due to the driver synchronization mechanism and its causality rollbacks. Equation 30 shows the combined list of values being generated for the analytical versus actual plot of Lookahead. These plots are shown in Figure 8.84 and Figure 8.85.

```
makePlot[dir, "lPPropY.AN-1","lPELkAhead.AN-1", {PlotJoined->True,
AxesLabel->{ "Proportion Out-of-Tolerance", "Expected Lookahead (mS)"},
PlotLabel->"Overhead"}]
```

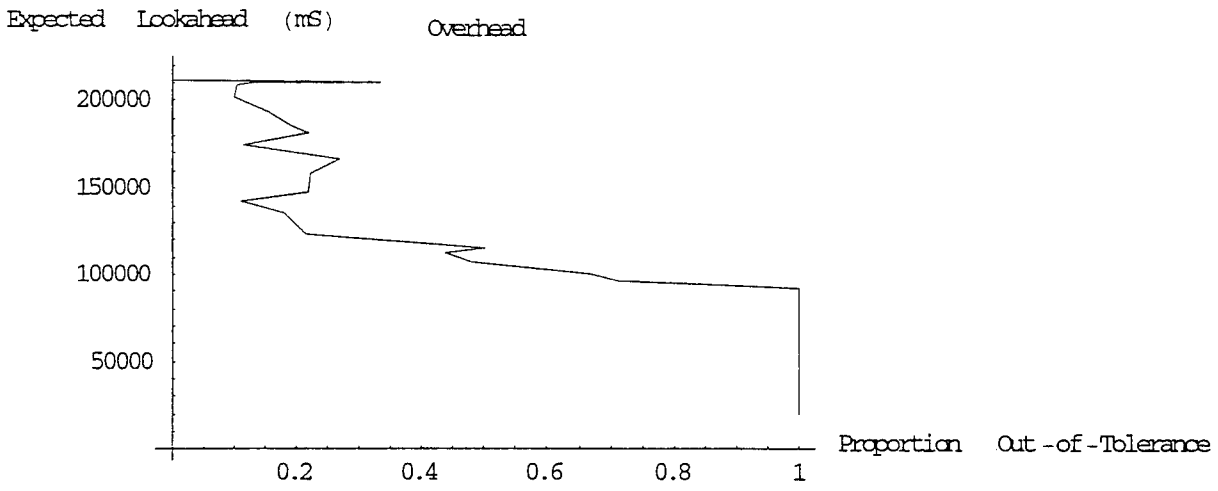


Figure 8.82. Lookahead versus Proportion Out-of-Tolerance.

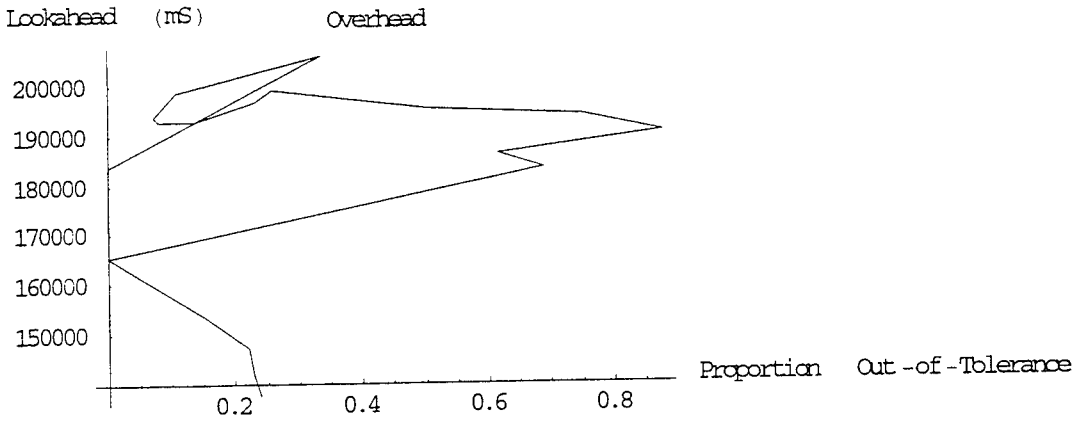


Figure 8.83. Lookahead versus Proportion Out-of-Tolerance with Multiple Driving Processes.

```
l=getList[dir, "lPPropY.AN-1","lPELkAhead.AN-1"];
al=Table[{l[[i]][[1]],l[[i]][[2]]/1000.}, {i,2,Length[l]}];
m=MultipleListPlot[
   al,
   Table[{Y,ESLa[\[Lambda]vm,\[CapitalDelta]vm,1.0,task\[Tau],\[Tau]rb,
x,Y,0.,200.]},{Y,.0,.5,.1}],
   PlotJoined->{True,True},
   AxesLabel->{"Proportion out-of-tolerance","Expected Lookahead
(Seconds)"}
   ]
        ]
```

Equation 30 Generate Lists of Actual and Analytical Values for Plot.

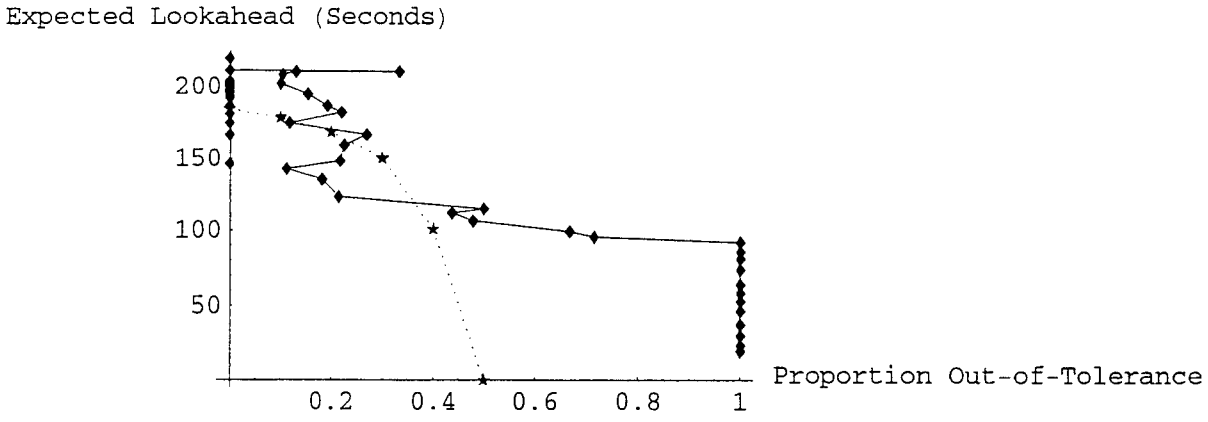


Figure 8.84. Analytical (Dashed Line) versus Actual (Solid Line) Lookahead as a Function of Proportion Out-of-Tolerance Messages.

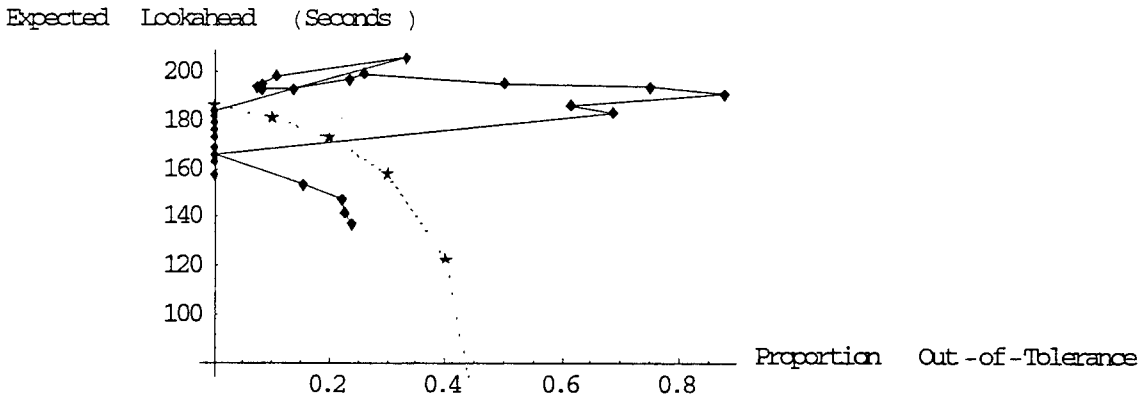


Figure 8.85. Analytical (Dashed Line) versus Actual (Solid Line) Lookahead as a Function of Proportion Out-of-Tolerance Messages with Multiple Driving Processes.

8.3.23 Speedup Analysis versus Actual

Figure 8.86 and Figure 8.87 show AVNMP speedup as a function of the proportion of out-of-tolerance messages. Equation 31 shows the generation of the data for the plot of analytical versus

actual speedup. The plots of analytical versus actual speedup are shown in Figure 8.88 and Figure 8.89.

```
makePlot[dir, "1PPropY.AN-1", "1PSpeedup.AN-1", {PlotJoined->True,
AxesLabel->{"Speedup", "Proportion Out-of-Tolerance"}, PlotLabel-
>"Overhead"}]
```

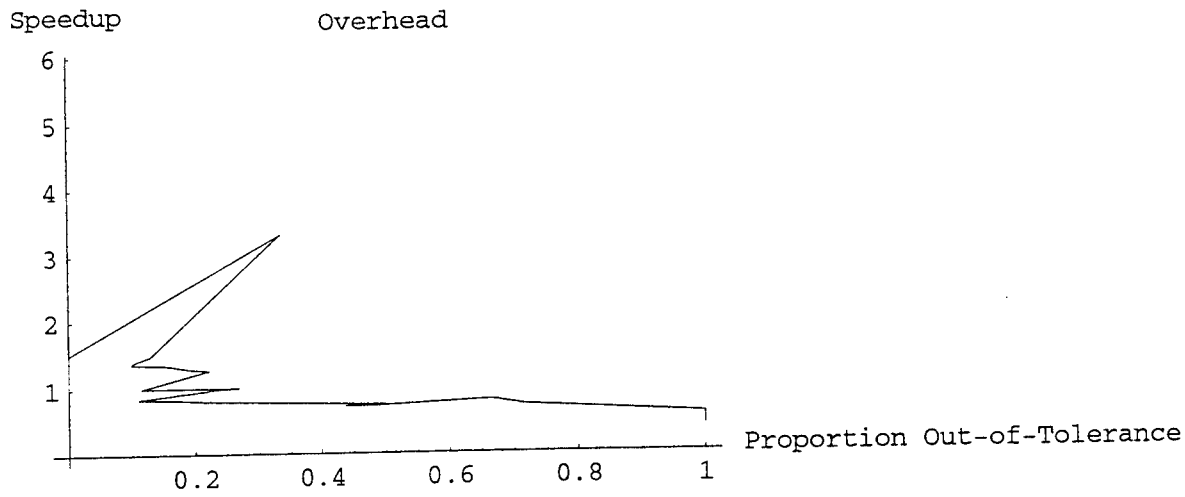


Figure 8.86. Proportion Out-of-Tolerance Messages versus Speedup.

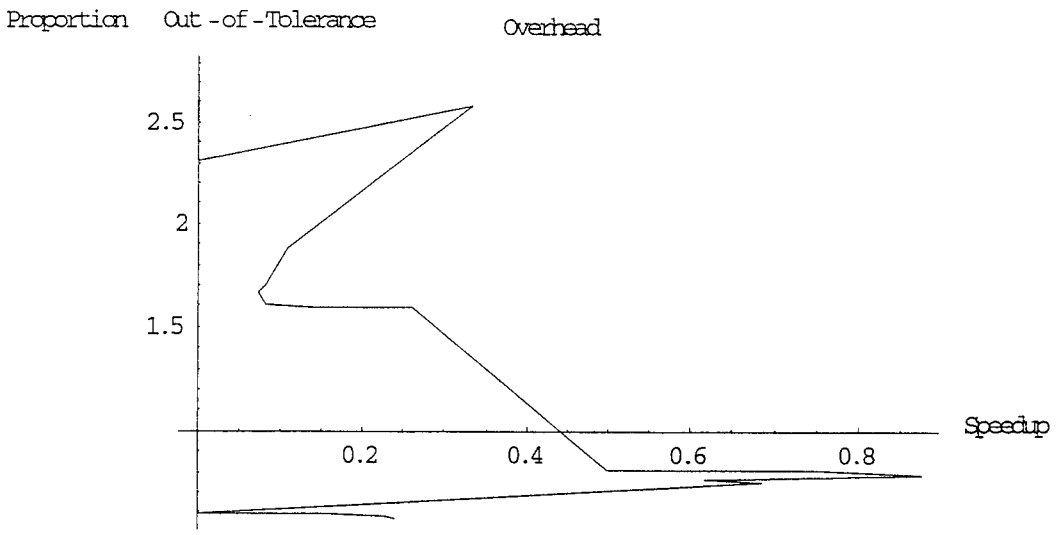


Figure 8.87. Proportion Out-of-Tolerance Messages versus Speedup with Multiple Driving Processes.

```
su=getList[dir, "1PPropY.AN-1","1PSpeedup.AN-1"];

suMod= Table[{su[[i]][[1]], su[[i]][[2]]}, {i,1, Length[su]}];

MultipleListPlot[suMod,
Table[{Y,Speedup[\[Lambda]vm,\[CapitalDelta]vm,1.0,task\[Tau],\[Tau]rb,
x,Y,0.,0.,1.]},{Y,.1,.9,.1}],
PlotJoined->True,AxesLabel->{"Proportion out-of-tolerance","Speedup"}
]
```

Equation 31 Generation of Analytical versus Actual Data.

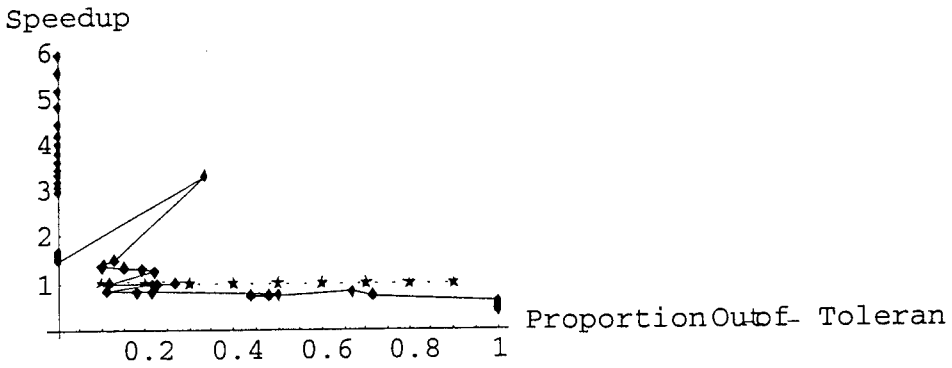


Figure 8.88. Analytical (Dashed Line) versus Actual (Solid Line) Speed as a Function of Proportion Out-of-Tolerance.

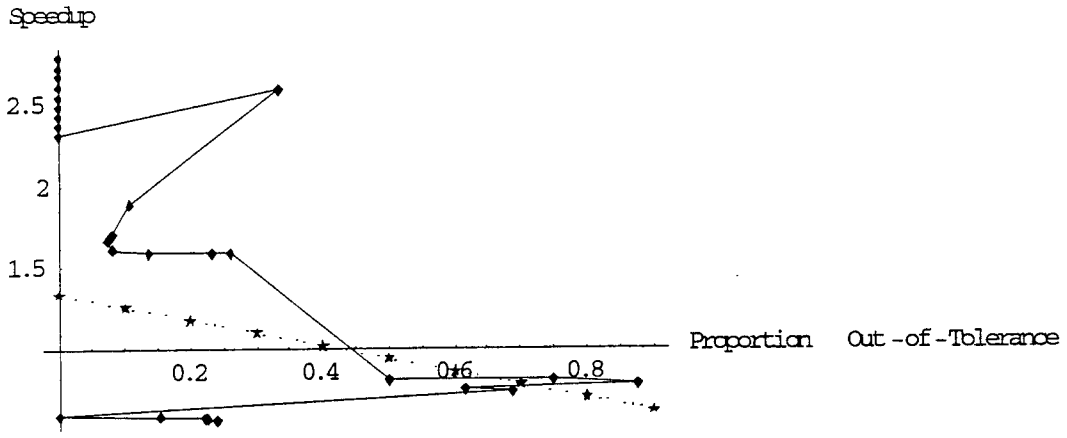


Figure 8.89. Analytical (Dashed Line) versus Actual (Solid Line) Speed as a Function of Proportion Out-of-Tolerance with Multiple Driving Processes.

8.3.24 Accuracy Analysis

Equation 32 shows the actual and analytical data being prepared for the plots in Figure 8.90 and Figure 8.91. The predicted values are shown as a function of wallclock time and *LVT*. This data was collected by SNMP polling an active execution environment that was enhanced with AVNMP. The valleys between the peaks are caused by the polling delay. A diagonal line on the *LVT/t* plane from the front right corner to the back left corner separates *LVT* in the past from *LVT* in the future; future *LVT* is towards the back of the graph, past *LVT* is in the front of the graph. Starting from the front, right hand corner, examine slices of fixed wallclock time over *LVT*; this shows both the past values and the predicted value for that fixed wallclock time.

```
d1 = readSnmp3DPlot[dir, "lPUptime.AN-1",
{"loadPredictionPredictedTime.AN-1",30},
{"loadPredictionPredictedLoad.AN-1",30}, 1, 1000 60];

plot3DSnmp[d1, AxesLabel->{"Wallclock (minutes)", "LVT (minutes)",
"Actual Packets"}, ViewPoint->{2.383, -1.410, 1.945}]
```

Equation 32 Generation of Actual versus Predicted Values.

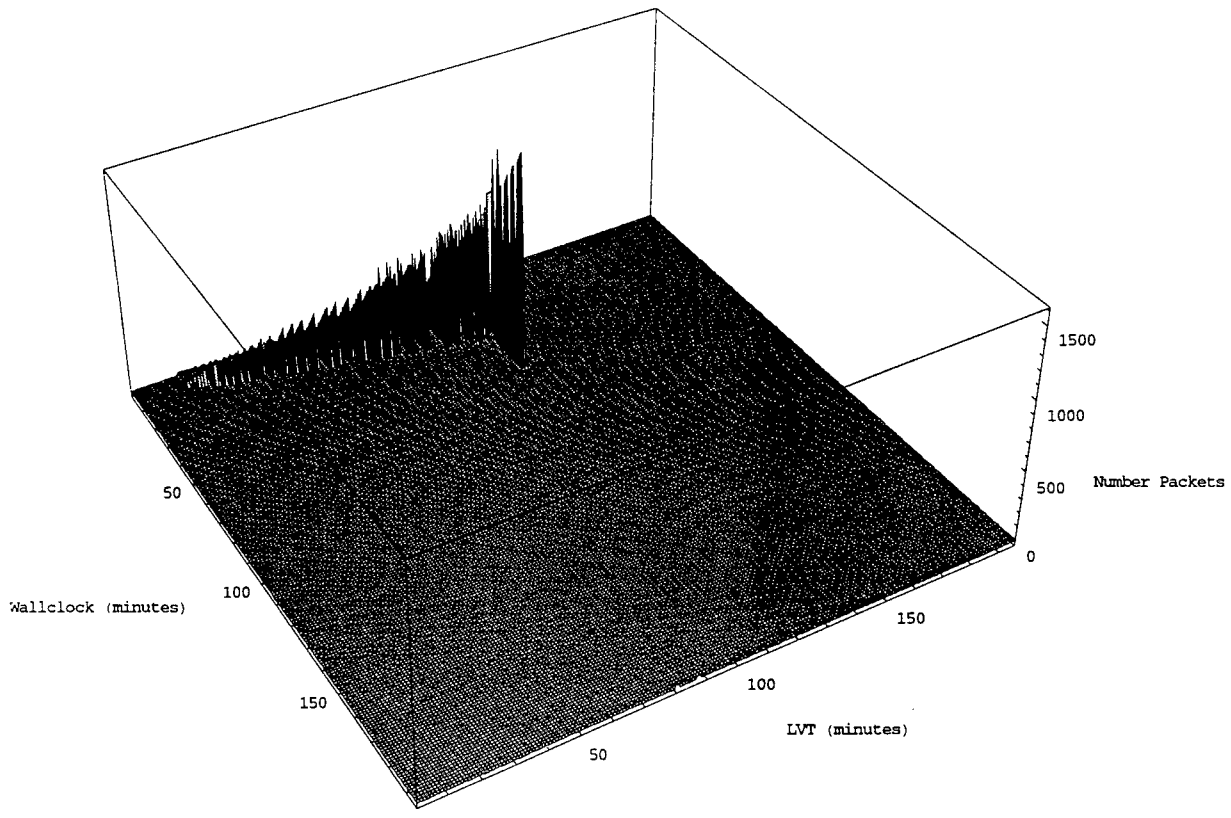


Figure 8.90. Number of Packets versus LVT and Wallclock.

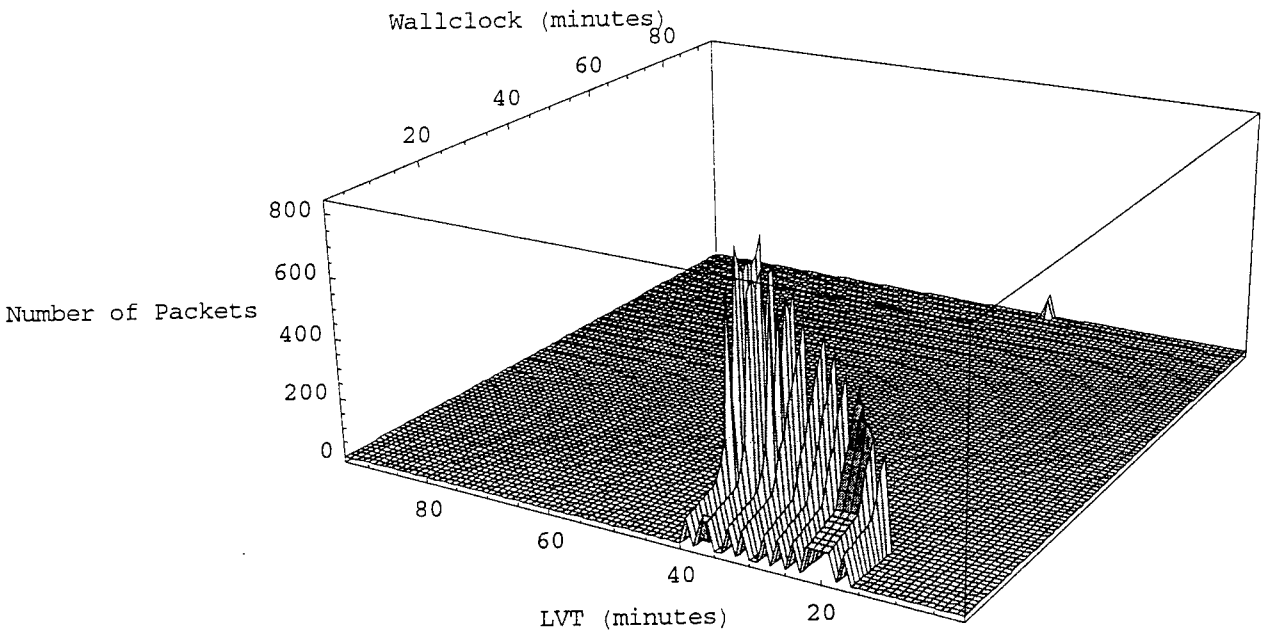


Figure 8.91. Number of Packets versus LVT and Wallclock with Multiple Driving Processes.

8.3.25 Time Difference (lPTdiff)

The graphs in Figure 8.92 and Figure 8.93 show the difference in the predicted event time versus the actual event time. As stress increases, fewer predictions are made and they are farther apart in time. Thus, it less likely that a predicted event is in close temporal proximity to a given actual event. In this version of AVNMP, the temporally closest predicted event is compared with an actual event and the difference is computed. There is no attempt to compensate for the potential time difference. Thus, this appears as prediction error even though it is possible that the prediction is correct; there is simply no predicted value existing near the time of the actual event.

```
makePlot[dir, "lPUptime.AN-1", "lPTdiff.AN-1", {PlotJoined->True,
AxesLabel->{"Wallclock (mS)", "Check Time Difference (mS)"}, PlotLabel-
>"Prediction Accuracy"}]
```

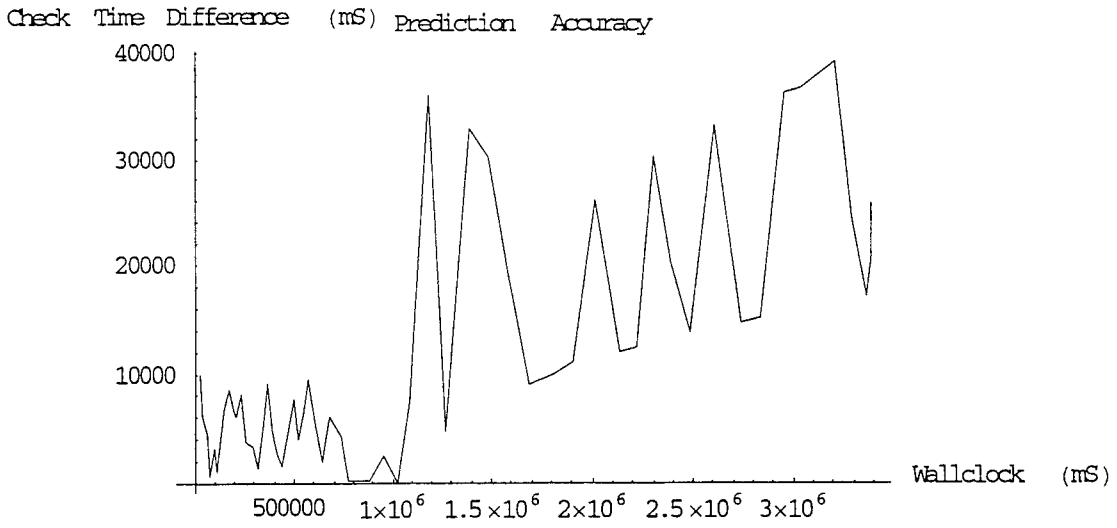


Figure 8.92. Time Difference between Actual and Predicted Value when Tolerance Checked.

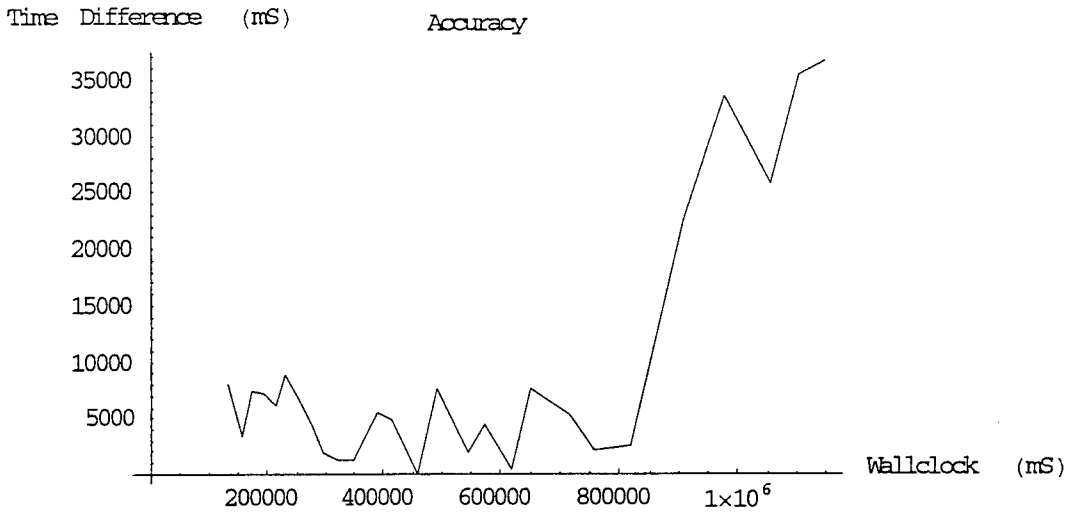


Figure 8.93. Time Difference between Actual and Predicted Value when Tolerance Checked with Multiple Driving Processes.

8.4 Summary

This chapter has presented an experimental validation of AVNMP running in the Magician Execution Environment. While many detailed results were presented in this section, the salient points are the following. The AVNMP system, injected into the network as an active application, is able to model the system and predict state information in a manner that meets the demand for accuracy at a particular active node. Greater demand is met at the cost of AVNMP performance, that is, the ability of AVNMP to predict farther into the future. Two experimental configurations were presented; a feed-forward network configuration and a configuration in which two Driving Processes feed into the same Logical Process. The latter configuration is of interest because Driving Processes had been considered as independent processes that "drive" the Logical Processes forward in time. However, the Driving Processes require feedback in order to prevent the possibility of each injecting a virtual message out of order with regard to Receive Time. This is prevented by a message from the common Logical Process to the slower Driving Processing asking it to jump forward in Local Virtual Time by a small increment. This mechanism appears to work; however, the synchronization of Driving Processes adds additional overhead to the common Logical Process and could use further refinement. For example, the common Logical Process appears to be rolling back in this case, which is not necessary. However, the concept of AVNMP is shown in this chapter to be a feasible one. This chapter has focused on network traffic and load prediction; however, as this chapter is being written AVNMP is also being applied to CPU utilization prediction in collaboration with NIST.

9

SUMMARY AND CONCLUDING REMARKS

This project has challenged itself to consider the benefits of Active Networking and to apply those benefits towards the management of Active Networks. The inherently distributed nature of communication networks and the computational power unleashed by the Active Networking paradigm have been used to mutual benefit in the development of the Active Virtual Network Management Prediction mechanism. Both load and CPU prediction capability have been explored using AVNMP. Active Networks benefit from AVNMP by continuously providing information about potential problems before they occur. AVNMP benefits from Active Networks in many ways. The first and most practical is the ease of development and deployment of this novel protocol. This could not have been accomplished so quickly or easily given today's closed, proprietary network device processing. Another benefit is the fact that network packets now have the unprecedented ability to control their own processing. Great advantage is taken of this new capability in AVNMP. Virtual messages, varying widely in content and processing, can adjust their predicted values as they travel through the network. Finally, Active Networks add a level of robustness that cannot be found in today's networks. This robustness is due to the ability of the AVNMP system components, which are themselves active packets, to easily migrate from one node to another in the event of failure -- or the prediction of failure provided by AVNMP!

There are two readily apparent directions in which this work can be carried forward. The first is the practical development and integration of prediction into an active network management framework. AVNMP can provide early warning of potential problems; however, the identification of a solution and marshaling of automated solution entities within an active network has not yet been fully addressed. This project has begun to lay the groundwork for such automated composition of management solutions within an active network (Kulkarni et al., 1998).

The second direction in which this work should be carried forward is the exploration of a relatively unexplored area –understanding the benefits of active networking Algorithmic Information Theory and its close companion, Complexity Theory. To our knowledge, this work is the first to propose and begin investigation into using the newly available processing power of Active Networks through the concept of Algorithmic Information (our "streptichrons"). Complexity Theory has been receiving more attention lately and is making significant theoretical progress. Active Networking is the ideal place to be taking advantage of this progress.

Reference

A.B. Kulkarni and S.F. Bush. Active Network Management, Kolmogorov, Complexity, and Streptichrons. GE CRD Class I Technical Report 2000CRD107 (http://www.crd.ge.com/crd_reports).

GLOSSARY

AA Active Application. An Active Application is supported by the Execution Environment on an active network device. The Active Application consists of active packets that support a particular application.

Autoanaplasis Autoanaplasis is the self-adjusting characteristic of streptichrons. One of the virtues of the Active Virtual Network Management Prediction Algorithm is the ability for the predictive system to adjust itself as it operates. This is accomplished in two ways. When real time reaches the time at which a predicted value had been cached, a comparison is made between the real value and the predicted value. If the values differ beyond a given tolerance, then the logical process rolls backward in time. Also, active packets which implement virtual messages adjust, or refine, their predicted values as they travel through the network.

AVNMP Active Virtual Network Management Prediction. An algorithm that allows a communications network to advance beyond the current time in order to determine events before they occur.

C/E Condition Event Network. A C/E network consists of state and transition elements which contain tokens. Tokens reside in state elements. When all state elements leading to a transition element contain a token, several changes take place in the C/E network. First, the tokens are removed from the conditions which triggered the event, the event occurs, and finally tokens are placed in all state outputs from the transition which was triggered. Multiple tokens in a condition and the uniqueness of the tokens are irrelevant in a C/E Net.

CE Clustered Environment. One of the contributions of (Avril and Tropper, 1995) in CTW is an attempt to efficiently control a cluster of LPs on a processor by means of the CE. The CE allows multiple LPs to behave as individual LPs as in the basic time warp algorithm or as a single collective LP.

Channel Channel. An active network channel is a communications link upon which active packets are received. The channel determines the type of active packet and forwards the packet to the proper Execution Environment. Principals use anchored channels to send packets between the execution environment and the underlying communication substrate. Other channels are cut through, meaning that they forward packets through the active node–from an input device to an output device–without being intercepted and processed by an Execution Environment. Channels are in general full-duplex, although a given principal might only send or receive packets on a particular channel.

CMB Chandy-Misra-Bryant. A conservative distributed simulation synchronization technique.

CMIP Common Management Information Protocol. A protocol used by an application process to exchange information and commands for the purpose of managing remote computer and communications resources. Described in (ISO, 1995).

CS Current State. The current value of all information concerning a PP encapsulated by an LP and all the structures associated with the LP.

CTW Clustered Time Warp. CTW is an optimistic distributed simulation mechanism described in (Avril and Tropper, 1995).

EE Execution Environment. The Execution Environment is supported by the Node Operating System on an active network device. The Execution Environment receives active packets and executes any code associated with the active packet.

Fossil In an AVNMP Logical Process, as the Local Virtual Time advances, the state queue is filled with predicted values. As the wallclock advances, the predicted values become actual values. When the wallclock advances beyond the time a predicted value was to occur, the value becomes a fossil because it is no longer a prediction, but an actual event that has happened in the past. Fossils should be removed periodically to avoid excessive use of memory.

FSM Finite State Machine. A five-tuple consisting of a set of states, an input alphabet, an output alphabet, a next-state transition function, and an output function. Used to formally describe the operation of a protocol.

GPS Global Positioning System. A satellite-based positioning service developed and operated by the Department of Defense.

GSV Global Synchronic Distance. The maximum Synchronic Distance in a Petri-Net model of a system.

GVT Global Virtual Time. The largest time beyond which a rollback based system will never rollback.

IETF Internet Engineering Task Force. The main standards organization for the Internet. The IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual.

IPC Inter-Processor Communication. Communication among Unix processes. This may take place via sockets, shared memory, or semaphores.

LP Logical Proces. An LP consists of the PP and additional data structures and instructions which maintain message order and correct operation as a system executes ahead of real time.

LVT Local Virtual Time. The Logical Process contains its notion of time known as Local Virtual Time.

NodeOS Node Operating System. The Node Operating System is the base level operating system for an active network device. The Node Operating System supports the Execution Environments.

MIB Management Information Base. A collection of objects which can be accessed by a network management protocol.

MTW Moving Time Windows. MTW is a distributed simulation algorithm that controls the amount of aggressiveness in the system by means of a moving time window. The trade-off in having no roll-backs in this algorithm is loss of fidelity in the simulation results.

NFT No False Time-stamps. NFT Time Warp assumes that if an incorrect computation produces an incorrect event ($E_{i,T}$), then it must be the case that the correct computation also produces an event ($E_{i,T}$) with the same time-stamp. This simplification makes the analysis in (Ghosh et al., 1993) tractable.

NPSI Near Perfect State Information. The NPSI Adaptive Synchronization Algorithms for PDES are discussed in (Srinivisan and Paul F. Reynolds, 1995b) and (Srinivisan and Paul F. Reynolds, 1995a). The adaptive algorithms use feedback from the simulation itself in order to adapt. The NPSI system requires an overlay system to return feedback information to the LPs. The NPSI Adaptive Synchronization Algorithm examines the system state (or an approximation of the state) calculates an error potential for future error, then translates the error potential into a value which controls the amount of optimism.

NTP Network Time Protocol. A TCP/IP time synchronization mechanism. NTP (Mills, 1985) is not required in VNC on the RDRN because each host in the RDRN network has its own GPS receiver.

PA Perturbation Analysis. The technique of PA allows a great deal more information to be obtained from a single simulation execution than explicitly collected statistics. It is particularly useful for finding the sensitivity information of simulation parameters from the sample path of a single simulation run. It may be an ideal way for VNC to automatically adjust tolerances and provide feedback to driving process(es).Briefly, assume a sample path, (Θ,ξ) from a simulation. Θ is vector of all parameters and ξ is a vector of all random occurrences. $L(\Theta,\xi)$ is the sample performance. $J(\Theta,\xi)$ is the average performance, $E[L(\Theta,\xi)]$. Parameter changes cause perturbations in event timing. Perturbations in event timing propagate to other events. This induces perturbations in L. If perturbations into (Θ,ξ) are small, assume event trace $(\Theta + d\Theta,\xi)$ remains unchanged. Then $dL(\Theta,\xi)/d\Theta$ can be calculated. From this, the gradient of $J(\Theta)$ can be obtained, which provides the sensitivity of performance to parameter changes. PA can be used to adjust tolerances while VNC is executing because event times are readily available in the SQ.

PDES Parallel Discrete Event Simulation. PDES is a class of simulation algorithms which partition a simulation into individual events and synchronizes the time the events are executed on multiple processors such that the real time to execute the simulation is as fast as possible.

PDU Protocol Data Unit. 1. Information that is delivered as a unit among peer entities of a network and that may contain control information, address information, or data. 2. In layered systems, a unit of data that is specified in a protocol of a given layer and that consists of protocol-control information of the given layer and possibly user data of that layer.

P/T Place Transition Net. A P/T Network is exactly like a C/E Net except that a P/T Net allows multiple tokens in a place and multiple tokens may be required to cause a transition to fire.

PIPS Partially Implemented Performance Specification. PIPS is a hybrid simulation and real-time system which is described in (Bagrodia and Shen, 1991). Components of a performance specification for a distributed system are implemented while the remainder of the system is simulated. More components are implemented and tested with the simulated system in an iterative manner until the entire distributed system is implemented.

PP Physical Process. A Physical Process is nothing more than an executing task defined by program code. An example of a PP is the RDRN beam table creation task. The beam table creation task generates a table of complex weights which controls the angle of the radio beams based on position input.

Principal The primary abstraction for accounting and security purposes is the principal. All resource allocation and security decisions are made on a per-principal basis. In other words, a principal is admitted to an active node once it has authenticated itself to the node, and it is allowed to request and use resources.

QR Receive Queue. A queue used in the VNC algorithm to hold incoming messages to a LP. The messages are stored in the queue in order by receive time.

QS Send Queue. A queue used in the VNC algorithm to hold copies of messages which have been sent by a LP. The messages in the QS may be sent as anti-messages if a rollback occurs.

QoS Quality of Service. Quality of Service is defined on an end-to-end basis in terms of the following attributes of the end-to-end ATM connection: Cell Loss Ratio, Cell Transfer Delay, Cell Delay Variation.

RT Real Time. The current wall clock time.

SLP Single Processor Logical Process. Multiple LPs executing on a single processor.

SLW Sliding Lookahead Window. The SLW is used in VNC to limit or throttle the prediction rate of the VNC system. The SLW is defined as the maximum time into the future for which the VNC system may predict events.

SmallState SmallState is a named cache within an active network node's execution environment that allows active packets to store information. This allows packets to leave information behind for other packets to use.

SNMP Simple Network Management Protocol. The Transmission Control Protocol/Internet Protocol (TCP/IP) standard protocol that (a) is used to manage and control IP gateways and the networks to which they are attached,(b) uses IP directly, bypassing the masking effects of TCP error correction,(c) has direct access to IP datagrams on a network that may be operating abnormally, thus requiring management, (d) defines a set of variables that the gateway must store, and (e) specifies that all control operations on the gateway are a side-effect of fetching or storing those data variables, i.e., operations that are analogous to writing commands and reading status. SNMP is described in (Rose, 1991).

SQ State Queue. The SQ is used in VNC as a LP structure to hold saved state information for use in case of a rollback.The SQ is the cache into which pre-computed results are stored.

Streptichron A Streptichron is an active packet facilitating prediction. It is a superset of the virtual message. It can contain self-adjusting model parameters, an executable model, or simple state values.

TR Receive Time. The time a VNC message value is predicted to be valid.

TS Send Time. The LVT that a virtual message has been sent. This value is carried within the header of the message. The TS is used for canceling the effects of false messages.

REFERENCES

Alexander, D.S., Arbaugh, W.A., Hicks, M.W., Kakkar, P., Keromytis, A.D., Moore, J.T., Gunter, C.A., Nettles, S.M., and Smith, J.M. (1998). The SwitchWare Active Network Architecture. *IEEE Network*, 12(3):27--36.

Alexander, D.S., Braden, B., Gunter, C., Jackson, A., Keromytis, A., Minden, G., and Wetherall, D., editors (1997). *Active Network Encapsulation Protocol (ANEP).* Active Networks Group. Request for Comments: draft

Andre, C., Armand, P., and Boeri, F. (1979). Synchronic Relations and Applications in Parallel Computation. *Digital Processes,* pages 99,113.

Aronson, I., Levine, J., and Tsimring, L. (1994). Controlling Spatio-Temporal Chaos. *Phys. Rev. Lett.*, 72.

Avril, H. (1996). *Clustered Time Warp and Logic Simulation.* PhD thesis, McGill University.

Avril, H. and Tropper, C. (1995). Clustered Time Warp and Logic Simulation.

Bagrodia, R. and Shen, C.-C. (1991). MIDAS: Integrated Design and Simulation of Distributed Systems. *IEEE Transactions on Software Engineering.*

Ball, D. and Hoyt, S. (1990). The Adaptive Time-Warp Concurrency Control Algorithm. In *Proceeding of SCS'90.*

Berry, O. and Jefferson, D. (1985). Critical Path Analysis of Distributed Simulation. In *SCS Multi Conference on Distributed Simulation.*

Boukerche, A. and Tropper, C. (1994). A Static Partitioning and Mapping Algorithm for Conservative Parallel Simulations. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation,* pages 164,172.

Braden, B., Cerpa, A., Fischer, T., Lindell, B., Kaum, J., and Phillips, G. (2000). Introduction to the ASP Execution Environment. Technical report, USC/ISI. url: http://www.isi.edu/active-signal/ARP/index.html.

Bush, S.F., , Kulkarni, A., Evans, S., and Galup, L. (2000). Active Jitter Control. In *7th International IS\&N Conference, Intelligence in Services and Networks (ISN) '00, Kavouri, Athens, Greece.*

Bush, S.F. (1997). *The Design and Analysis of Virtual Network Configuration for a Wireless Mobile ATM Network.* PhD thesis, University of Kansas.

Bush, S.F. (1999). Active Virtual Network Management Prediction. In *Parallel and Discrete Event Simulation Conference (PADS) '99.*

Bush, S.F. (2000). Islands of Near-Perfect Self-Prediction. In *Proceedings of Vwsim'00: Virtual Worlds and Simulation Conference, WMC'00: 2000 SCS Western Multi-Conference, San Diego, SCS (2000).*

Bush, S.F. and Barnett, B. (1998). A Security Vulnerability Assessment Technique and Model. Technical Report 98CRD028, General Electric Corporate Research and Development Center.

Bush, S.F., Frost, V.S., and Evans, J.B. (1999). Network Management of Predictive Mobile Networks. *Journal of Network and Systems Management,* 7(2).

Calvert, K. (1998). Architectural Framework for Active Networks (Version 0.9). Active Networks Working Group.

Chandy, K.M. and Misra, J. (1979). Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering.*

Christopher Landauer and Kirstie L. Bellman (1996). Semiotics of Constructed Complex Systems. In *Intelligent Systems: A Semiotic Perspective Proceedings of the 1996 International Multidisciplinary Conference Volume I: Theoretical Semiotics, Workshop on Intelligence in Constructed Complex Systems.*

daSilva, S., Florissi, D., and Yemini, Y. (1998). Composing Active Services in NetScript. In *Proceedings of the DARPA Active Networks Workshop (Tucson, Arizona).*

DiBernardo, M. (1996). An Adaptive Approach to the Control and Synchronization of Continuous-Time Chaotic Systems. *Int. J. of Bifurcation and Chaos*, 6(3).

Felderman, R.E. and Kleinrock, L. (1990). An Upper Bound on the Improvement of Asynchronous versus Synchronous Distributed Processing. In *SCS '90.*

Fujimoto, R.M. (1990). Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30--53.

Ghosh, K., Fujimoto, R.M., and Schwan, K. (1993). Time Warp Simulation in Time Constrained Systems. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pages 163,166.

Glazer, D.M. (1993). *Load Balancing Parallel Discrete Event Simulations.* PhD thesis, McGill University.

Glazer, D.M. and Tropper, C. (1993). A Dynamic Load Balancing Algorithm for Time Warp. *Parallel and Distributed Systems*, 4(3):318,327.

Goltz, U. (1987). Synchronic Distance. In *Petri Nets: Central Model and Their Properties. Advances in Petri Nets 1986, Proceedings of an Advanced Course, Bad Honnef. Lecture Notes on Computer Science 254*, pages 338,358. Springer-Verlag.

Goltz, U. and Reisig, W. (1982). Weighted Synchronic Distances. In *Applications and Theory of Petri Nets, Informatik-Fachberichte*, pages 289,300. Springer-Verlag.

Gupta, A., Akyldiz, I.F., and Fujimoto, R.M. (1991). Performance Analysis of Time Warp with Multiple Homogeneous Processors. *IEEE Transactions on Software Engineering.*

Hershey, J. and Bush, S.F. (1999). On Respecting Interdependence Between Queuing Policy and Message Value. Technical Report 99CRD151, General Electric Corporate Research and Development.

Hicks, M., Kakkar, P., Moore, J.T., Gunter, C.A., and Nettles, S. (1999). PLAN: A programming language for active networks. *ACM SIGPLAN Notices*, 34(1):86--93.

Ho, Y.-C. (1992). Perturbation Analysis: Concepts and Algorithms. In *Proceedings of the 1992 Winter Simulation Conference.*

Hoare, C.A.R. (1981). Communicating Sequential Processes. *Communications of the ACM.*

Hofstadter, D.R. (1980). Gödel, Escher, Bach: An Eternal Golden Braid. Vintage Books. ISBN 0-394-74502-7.

Hong, D. and Rappaport, S.S. (1986). Traffic Model and Performance Analysis for Cellular Mobile Radio Telephone Systems with Prioritized and Non prioritized Handoff Procedures. *IEEE Transactions on Vehicular Technology.*

Huber, O.J. and Toutain, L. (1997). Mobile agents in active networks. In *3rd ECOOP Workshop on Mobile Object Systems,* Jyväskylä, Finland.

ISO (1995). Open Systems Interconnection - Management Protocol Specification - Part 2: Common Management Information Protocol.

Jefferson, D.R. and Sowizral, H.A. (1982). Fast Concurrent Simulation Using The Time Warp Mechanism, Part I: Local Control. Technical Report TR-83-204, The Rand Corporation.

Jha, V. and Bagrodia, R.L. (1994). A Unified Framework for Conservative and Optimistic Distributed Simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pages 12,19.

J. Martinez and Silva, M. (1982). A Simple and Fast Algorithm to Obtain all Invariants of a Generalized Petri Net. In *Proceedings of the Second European Workshop on Application and Theory of Petri Nets.*

Kleinrock, L. (1975). *Queuing Systems Volume I: Theory.* John Wiley and Sons.

Konas, P. and Yew, P.C. (1995). Partitioning for Synchronous Parallel Simulation. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pages 181,184.

Kulkarni, A.B., Minden, G.J.,.Hill, R., Wijata, Y., Sheth, S., Pindi, H., Wahhab, F., Gopinath, A., and Nagarajan, A. (1998). Implementation of a Prototype Active Network. In *OPENARCH '98.*

Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM.*

Landauer, C. and Bellman, K.L. (1996). Integration Systems and Interaction Spaces. In *Proceedings of FroCoS'96: The First International Workshop on Frontiers of Combining Systems.*

Lazowaska, E. and Lin, Y.-B. (1990). Determining the Global Virtual Time in a Distributed Simulation. Technical Report 90-01-02, University of Washington.

Legedza, U., Wetherall, D.J., and Guttag, J. (1998). Improving the Performance of Distributed Applications Using Active Networks. Submitted to *IEEE INFOCOM.*

Leong, H.V. and Agrawal, D. (1994). Semantics-based Time Warp Protocols. In *Proceedings of the 7th International Workshop on Distributed Simulation.*

Lin, Y.-B. (1990). Understanding the Limits of Optimistic and Conservative Parallel Simulation. Technical Report UWASH-90-08-02, University of Washington.

Lin, Y.-B. and Lazowska, E.D. (1990). Optimality Considerations of "Time Warp" Parallel Simulation. Technical Report UWASH-89-07-05, University of Washington.

Lipton, R.J. and Mizell, D.W. (1990). Time Warp vs. Chandy-Misra: A Worst-Case Comparison. In *SCS '90.*

Liu, G., Marlevi, A., and Jr., G. Q.M. (1995). A Mobile Virtual-Distributed System Architecture for Supporting Wireless Mobile Computing and Communications. In *Mobicom '95.*

Liu, G.Y. (1996). *The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications.* PhD thesis, Royal Institute of Technology, Stockholm, Sweden.

Liu, G.Y. and Jr., G. Q.M. (1995). A Predictive Mobility Management Algorithm for Wireless Mobile Computing and Communications In *International Conference on Universal Personal Communications (ICUPC)*, pages 268,272.

Lubachevsky, B., Schwatz, A., and Weiss, A. (1989). Rollback Sometimes Works ... if Filtered. In *Proceedings of the 1989 Winter Simulation Conference*, pages 630--639.

Lubachevsky, B.D. (1989). Efficient Distributed Event Driven Simulations of Multiple-Loop Networks. *Communications of the ACM,* 32(1):111--131.

Luenberger, D.G. (1989). *Linear and Nonlinear Programming.* Addison-Wesley.

Ma, S. and Ji, C. (1998). Wavelet models for video time-series. In Jordan, M.I., Kearns, M.J., and Solla, S.A., editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

Madisetti, V., Walrand, J., and Messerschmitt, D. (1987). MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm. In *Proc. 1987 Winter Simulation Conference.*

Massechusetts Institute of Technology (1999). http://ana.lcs.mit.edu/darpa/. http://ana.lcs.mit.edu/darpa/.

McAffer, J. (1990). A Unified Distributed Simulation System. In *Proceedings of the 1990 Winter Simulation Conference*, pages 415,422.

Mikami, K., Tamura, H., Sengoku, M., and Yamaguchi, Y. (1993). On a Sufficient Condition for a Matrix to be the Synchronic Distance Matric of a Marked Graph. *IEICE Transactions Fundamentals*, E76-A(10).

Mills, D.L., editor (1985). *Network Time Protocol.* M/A-COM Linkabit.

Murphy, S. (1998). Secure Active Network Prototypes. Active Networks Working Group.

Noble, B.L. and Chamberlain, R.D. (1995). Predicting the Future: Resource Requirements and Predictive Optimism. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pages 157,164.

Pandit, S.M. and Wu, S.-M. (1983). *Time Series and System Analysis with Applications.* John Wiley and Sons.

Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes.* McGraw Hill.

Peterson, J.L. (1981). *Petri Net Theory and the Modeling of Systems.* Prentice-Hall.

Peterson, L. (1998). Node OS and API for Active Networks. Active Networks Working Group.

Rajaei, H., Ayani, R., and Thorelli, L.E. (1993a). The Local Time Warp Approach to Parallel Simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation,* pages 37--43.

Rajaei, H., Ayani, R., and Thorelli, L.-E. (1993b). The Local Time Warp Approach to Parallel Simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation,* pages 119,126.

Reisig, W. (1985). *Petri Nets.* Springer-Verlag.

Rose, M.T. (1991). *The Simple Book, An Introduction to the Management of TCP/IP Based Internets.* Prentice Hall.

Samrat Bhattacharjee, Kenneth L. Calvert and Ellen W. Zegura (1998). Reasoning About Active Network Protocols. In *Proceedings of ICNP '98.* Austin, TX.

Seshan, S., Balakrishnan, H., and Katz, R.H. (1996). Handoffs in Cellular Wireless Networks: The Daedalus Implementation Experience. *Kluwer International Journal on Wireless Personal Communications.*

Silva, M. and Colom, J.M. (1988). On the Computation of Structural Synchronic Invariants in P/T Nets. In *Lecture Notes on Computer Science*, volume 340, pages 386,417. Springer-Verlag.

Silva, M. and Murata, T. (1992). B-Fairness and Structural B-Fairness in Petri Net Models of Concurrent Systems. *Journal of Computer and System Sciences*, 44:447--477.

Sokol, L.M., Briscoe, D.P., and Wieland, A.P. (1988). MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. In *Proceedings of the 2nd Workshop on Parallel and Distributed Simulation,* pages 34--42.

Sokol, L.M. and Stucky, B.K. (1990). WOLF: A Rollback Algorithm for Optimistic Distributed Simulation Systems. In *Proc. 1990 SCS Multi conference on Distributed Simulation*, pages 169--173.

Srinivisan, S. and Paul F. Reynolds, J. (1995a). Adaptive Algorithms vs. Time Warp: An Analytical Comparison. Technical Report CS-95-20, University of Virginia.

Srinivisan, S. and Paul F. Reynolds, J. (1995b). NPSI Adaptive Synchronization Algorithms for PDES. Technical Report CS-94-44, University of Virginia.

Steinman, J.S. (1992). SPEEDES: A Unified Approach to Parallel Simulation. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, pages 75,84.

Steinman, J.S. (1993). Breathing Time Warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pages 109,118.

Steven Berson and Bob Braden and Livio Riciulli (2000). Introduction to the ABone. http://www.csl.sri.com/activate/.

Tamura, H. and Abe, T. (1996). Obtaining a Marked Graph from a Synchronic Distance Matrix. *Electronics and Communications in Japan*, 79(3).

Tennenhouse, D.L. and Bose, V.G. (1995). SpectrumWare: A Software-Oriented Approach to Wireless Signal Processing. In *Mobicom '95.*

Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., and Minden, G.J. (1997). A survey of active network research. *IEEE Communications Magazine,* 35(1):80-86.

Thomas, R., Gilbert, H., and Mazziotto, G. (1988). Influence of the Movement of Mobile Stations on the Performance of the Radio Cellular Network. *Proceedings of the 3rd Nordic Seminar.*

Tinker, P. and Agra, J. (1990). Adaptive Model Prediction Using Time Warp. In *SCS '90.*

Turnbull, J. (1992). A Performance Study of Jefferson's Time Warp Scheme for Distributed Simulation. Master's thesis, Case Western Reserve University.

Voss, K., Genrich, H.J., and Rozenberg, G. (1987). *Concurrency and Nets.* Springer-Verlag. ISBN 0-387-18057-5.

Wetherall, D., Guttag, J., and Tennenhouse, D. (1999). ANTS: Network services without the red tape. *Computer*, 32(4):42-48.

Yemini, Y., Konstantinou, A.V., and Florissi, D. (2000). NESTOR: An architecture for self-management and organization. *IEEE Journal on Selected Areas in Communications.* To appear in the IEEE JSAC special issue on network management (publication 2nd quarter 2000).

Zegura, E. (1998). Composable Services for Active Networks. AN Composable Services Working Group.

*U.S. GOVERNMENT PRINTING OFFICE: 2001-610-055-10097

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.