

Network Working Group
Internet-Draft
Expires: December 30, 2002

S. Bush
A. Kulkarni
N. Smith
GE GRC
July 2002

In-Line Network Management Prediction
draft-bush-inline-predictive-mgt-00

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC2026](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on December 30, 2002.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

In-line network management prediction exploits fine-grained models of network components, injected into the communication network, to enhance network performance. Accurate and fast prediction of local network state enables more intelligent network control resulting in greater performance and fault tolerance. Accurate and fast prediction requires algorithmic capability. Active and Programmable Networking have enabled algorithmic information to be dynamically injected into the network allowing enhanced capability and flexibility. One of the new capabilities is enhanced network management via in-line management code, that is, management

algorithms embedded within intermediate network devices. In-line network management prediction utilizes low-level algorithmic transport capability to implement low-overhead predictive management.

A secondary purpose of this document is to provide general interoperability information for the injection of general purpose algorithmic information into network devices. This document may help in some manner to serve as a temporary bridge between Internet Protocol and Active and Programmable Network applications. This may stimulate some thought as to the content and format of "standards" information potentially required for Active Networking. Management of the Internet Protocol and Active and Programmable Networking is vital. In particular, coexistence and interoperability of active networking and Internet Protocol management is specified in order to implement the injection of algorithmic information into a network.

Implementation Note

This document proposes a standard that assumes the capability of injecting algorithmic information, i.e. executable code, into the network. Active or programmable capability, as demonstrated by recent implementation results from the DARPA Active Network Program, Active Internet Protocol [8] or recent standards in Programmable Networking [9], help meet this requirement. While in-line predictive management could be standardized via a vehicle other than active packets, we choose to use active networking as a convenient implementation for algorithmic change within the network.

1. Introduction

This work in progress describes a mechanism that allows a distributed model, injected into a network, to predict the state of the network. The concept is illustrated in Figure 1. The state to be predicted is modeled within each actual network node. Thus, a distributed model, shown in the top plane, is formed within the actual network, shown in the bottom plane. The top plane slides ahead of wallclock time, although in an asynchronous manner. This means that each simulated node MAY have its own notion of simulation time.

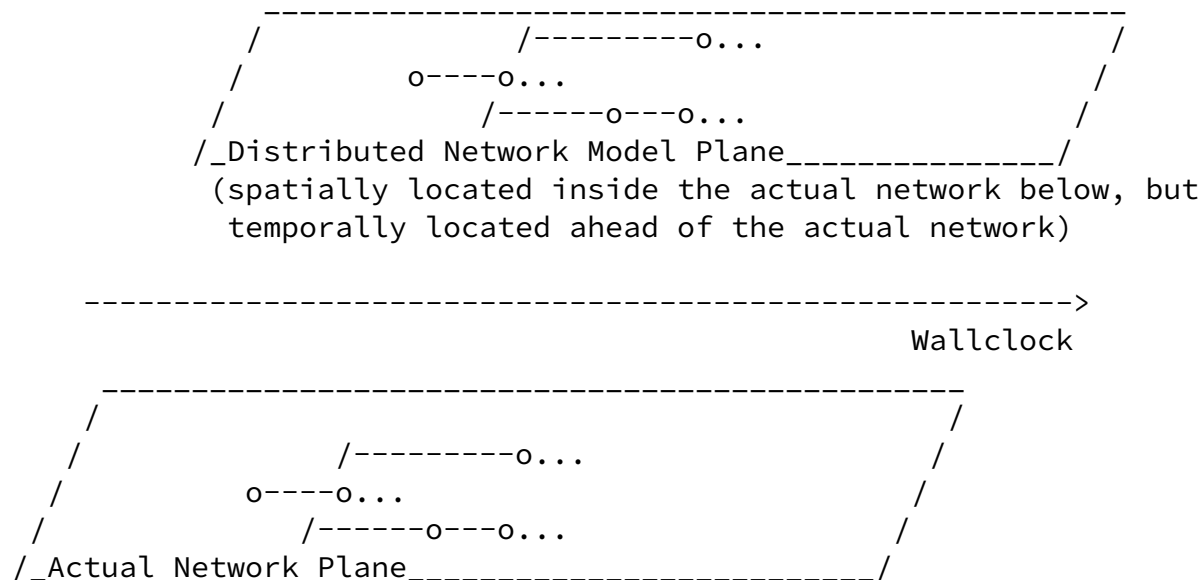


Figure 1: The Distributed Model Inside the Network.

This concept opens up a set of interoperability issues which do not appear to have been fully addressed. How can distributed model components be injected into an existing network? In-line models are

injected into the network assuming the overlay environment shown in Figure 2. In-line models in Figure 1 are designed to run as fast as possible in order to maintain a simulation time that is ahead of wallclock, communicating via virtual messages with future timestamps. What if messages are processed out-of-order because they arrive out-of-order at a node? How long do you wait (and slow your simulation down) to make sure they are not out-of-order? This specification provides a framework that allows synchronization to be handled in any manner; e.g. via a conservative (blocking) or optimistic (Time-Warp) manner within the network. Additionally, how can the models verify and maintain a reasonable amount of accuracy? A mechanism is provided in this document to allow local verification of prediction accuracy. Attempts to adjust accuracy are implementation dependent. How do

independent model developers allow their models to work coherently in this framework? Model operation is implementation dependent, however, this specification attempts to make certain that model messages will at least be transported in an inter-operable manner, both across and WITHIN, intermediate network devices. How does one publish their model descriptions? How are predicted values represented and accessed? Suggestion solutions for these questions are presented in this document as well.

[1.1](#) Overview

In-line predictive network management, which enables greater performance and fault tolerance, is based upon algorithmic information injected into a network allowing system state to be predicted and efficiently propagated throughout the network. This paradigm enables management of the network with continuous projection and refinement of future state in real time. In other words, the models injected into the network allow state to be predicted and propagated throughout the network enabling the network to operate simultaneously in real time and in the future. The state of traffic, security, mobility, health, and other network properties found in typical Simple Network Management Protocol (SNMP) [\[2\]](#) Management Information Bases (MIB) is available for use by the management system. To enable predictive management of applications, new MIBs will have to be defined that hold both current values as well as values expected to exist in the future.

The AgentX [\[5\]](#) protocol begins to address the issue of independent

SNMP agent developers dynamically and seamlessly interconnecting their agents into a single MIB under the control of a master agent. AgentX specifies the protocol between the master and sub-agents allowing the sub-agents to connect to the master agent. The AgentX specification complements this work-in-progress, namely, in-line network management prediction. The in-line network management prediction specification provides the necessary interface between agent functionality injected remotely via an Active Packet and dynamically 'linked' into a MIB. The agent code may enhance an existing MIB value by allowing it to return predicted values. Otherwise, coexistence with AgentX is SUGGESTED. The in-line network management prediction specification enables faster development of MIB modules with more dynamic algorithmic capability because Active and Programmable networks allow lower-level, secure, dynamic access to network devices. This has allowed injection of predictive capability into selected portions of existing MIBs and into selected portions of active or programmable network devices resulting in greater performance and fault tolerance.

[1.2](#) Outline

This document proposes standards for the following aspects of in-line predictive management:

- o SNMP Object Time Series Representation and Manipulation
- o Common Algorithmic Description
- o Multi-Party In-line Predictive Model Access and Control
- o Common Framework for Injecting Models into the Network
- o Model Interface with the Framework

The high-level components of this proposed standard are shown in Figure 2. The Active Network Framework [[10](#)] is a work in progress. In-line Predictive Management is the subject of this document. The Internet Protocol and SNMP are well-known.

Figure 2 shows the various ways in which in-line predictive

management can be used in an active network given an implementation in a particular execution environment. The in-line predictive management application runs as an active application on an active node. The framework is independent of the underlying architecture of the active network, which can take one of two forms. The protocol stack on the left shows a fully active network in which the Node Operating System runs one or more Execution Environments. Multiple active applications may execute in any Execution Environment. The protocol stack on the right shows the architecture of an active network overlay over IP. Essentially, the overlay scheme uses the Active Network Encapsulation Protocol (ANEP) [7] as a conduit to use the underlying IP network. The predictive management application executes alongside the other active applications and interacts with any managed active applications to provide their future state. Since the predictive management application requires only the execution environment to run in, it is independent of whether the active network is implemented as an overlay or it is available as a fully active network.

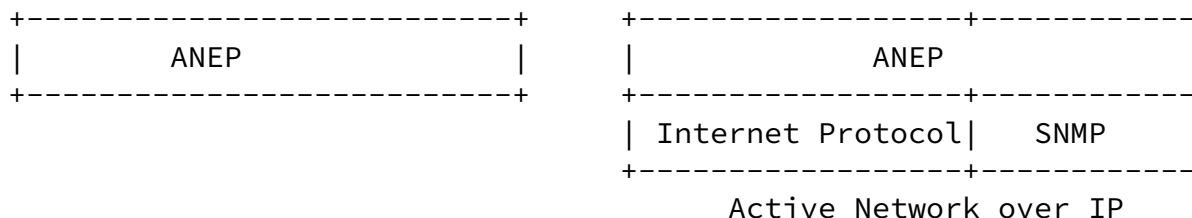
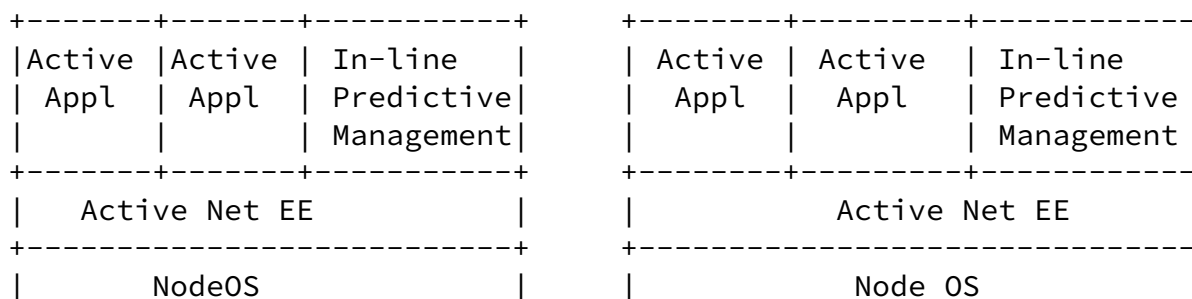


Figure 2: Relationship Among Underlying Assumptions about the Predictive Management Environment.

The next section provides basic definitions. Following that, the goals of this proposed standard are laid out. The remainder of the document develops into progressively more detail defining interoperability among algorithmic in-line network management

prediction components. Specifically, predictive capability requires careful handling of the time dimension. Rather than change the SNMP standard, a tabular technique is suggested. Then, in order to simplify design of predictive management objects, an extension to Case Diagrams is suggested for review and comment. This is followed by the specification of a distributed predictive framework. It is understood that multiple distributed predictive mechanisms exist, however, this framework is presented for comment and review because it contains all the necessary elements. Finally, the detailed interface between the active or programmable code and IP standard interfaces is presented.

[1.3](#) Definitions

The following acronyms and definitions are helpful in understanding the general concept of predictive network management.

- o In-line

Located within, or immediately adjacent to, the flow of network traffic.

- o Predictive Network Management

The capability of reliably predicting network events or the state of the network at a time greater than wall-clock time.

- o Fine-Grained Models

Small, light-weight, executable code modules that capture the behavior of a network or application component to enable predictive network management.

- o Algorithmic Information

Information, in the form of algorithms contained inside executable code, as opposed to static, non-executable data. Depending upon the complexity of the information to be transferred, an algorithmic form, or an optimal tradeoff between algorithmic and non-algorithmic form can be extremely flexible and efficient.

- o Non-Algorithmic Information

Information that cannot be executed. Generally requires a highly structured protocol to transfer with well-defined code pre-installed at all points in route including source and destination.

- o Small-State

Information caches that can be created at network nodes, intended for use by executable components of the same application.

- o Global-State

Information caches created at network nodes, intended to be used by executable components of different applications.

- o Multi-Party In-line Predictive Management Model

An in-line predictive management model comprised of multiple in-line algorithmic models that are developed, installed, utilized, and administered by multiple domains.

The following acronyms and definitions are useful in understanding the details of the specific predictive network management framework described in this document.

- o A (Anti-Toggle)

Used to indicate an anti-message. The anti-message is initiated by rollback and is used to keep the system within a specific range of prediction accuracy.

- o AA (Active Application)

An active network protocol or service that is injected into the

network in the form of active packets. The active packets are executed within the EE.

- o Active Network

A network that allows executable code to be injected into the nodes of the network and allows the code to be executed at the nodes.

- o Active Packet

The executable code that is injected into the nodes of an active network.

- o Anti-Message

An exact duplicate of a virtual message except for that the Anti-toggle bit is set. An Anti-message is used to annihilate an invalid virtual message. This is an implementation specific feature relevant to optimistic distributed simulation.

- o DP (Driving Process)

Generates virtual messages. Generally, the DP is implemented as an algorithm that samples network state and transforms the state into a prediction. The prediction is represented by a virtual message.

- o EE (Execution Environment)

The active network execution environment. The environment that resides on active network nodes that executes active packets.

- o Lookahead

The difference between Wallclock and LVT. This value is the distance into the future for which predictions are made.

- o LP (Logical Process)

An LP consists of the Physical Process and additional data

structures and instructions which maintain message order and correct operation as a system executes ahead of real time.

- o LVT (Local Virtual Time)

The LP contains a notion of time local to itself known as LVT. A node's LVT may differ from other nodes' LVT and Wallclock. LVT is a local, asynchronous notion of time.

- o M (Message)

The message portion of a Virtual Message is implementation specific. This proposed standard SUGGESTS that the message contents be opaque, however, an SNMP varbind, intended to represent future state, MAY be transported. Executable code may also be transported within the message contents.

- o NodeOS (Node Operating System)

The active network Operating System. The supporting infrastructure on intermediate networks nodes that supports one or more execution environments.

- o PP (Physical Process)

A PP is an actual process. It usually refers the actual process being modeled, or whose state will be predicted.

- o QS (Send Queue)

A queue used to hold copies of messages that have been sent by an LP. The messages in the QS may be sent as anti-messages if a rollback occurs.

- o Rollback

The process of adjusting the accuracy of predictive components due to packets arriving out-of-order or out-of-tolerance. Rollback is specific to optimistic distributed simulation techniques and is thus an implementation specific feature.

- o RT (Receive Time)

The time message value is predicted to be valid.

- o RQ (Receive Queue)

A queue used in the algorithm to hold incoming messages to an LP. The messages are stored in the queue in order by receive time.

- o SQ (State Queue)

The SQ is used 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.

- o Tolerance

A user-specified limit on the amount of prediction error allowed by an LP's prediction.

- o TR (Real Time)

The current time as a time-stamp within a virtual message.

- o 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.

- o VM (Virtual Message)

A message, or state, expected to exist in the future.

- o Wallclock

The current time.

[1.4](#) Goals

The goals of this document are...

- o Simplicity

This document attempts to describe the minimum necessary elements for in-line management prediction. Model developers should be able to inject models into the network allowing SNMP Object value prediction. Such models should work seamlessly with other predictive models in the network. The goal is to minimize the burden on the model developer while also insuring model interoperability.

- o Conformance

This document attempts conformance with existing standards when and where it is possible to do so. The concept is to facilitate a gradual transition to the active and programmable networking paradigm.

- o In-line Algorithmically-Based Management

This document attempts to introduce the use of in-line algorithmic management information.

[2.](#) A Common Representation of SNMP Object Time Series for In-line Network Management Prediction

SNMP, as currently defined, has a very limited notion of time associated with state information. The temporal semantics are expected to be applied to the state by the applications reading the information. On the other hand, predictive management requires generation, handling and transport of information that understands the temporal characteristics of the state, i.e. whether the information is current, future, or perhaps past information. In other words, capability for handling the time dimension of management information needs to be extended and standardized in some manner. In this section, we propose a mechanism for handling time issues in predictive management that require minimal changes from the SNMP standard.

A proposed standard technique for handling the time dimension in predictive state systems is to build the SNMP Object as a Table Object indexed by time. This is shown in the following excerpt from a Load Prediction MIB...

.
.~
.
.

loadPrediction OBJECT IDENTIFIER ::= { loadPredMIB 1 }

loadPredictionTable OBJECT-TYPE
SYNTAX SEQUENCE OF LoadPredictionEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of load prediction information."
::= { loadPrediction 1 }

loadPredictionEntry OBJECT-TYPE
SYNTAX LoadPredictionEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of Atropos LP prediction information."
INDEX { loadPredictionPort }
::= { loadPredictionTable 1 }

LoadPredictionEntry ::= SEQUENCE {
loadPredictionID
DisplayString,

loadPredictionPredictedLoad
INTEGER,
loadPredictionPredictedTime
INTEGER
}

loadPredictionID OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The LP identifier."
::= { loadPredictionEntry 1 }

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

```

"This is the predicted load on the link."
 ::= { loadPredictionEntry 2 }

loadPredictionPredictedCPUTime OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the predicted processor time used by a packet
on this node."
 ::= { loadPredictionEntry 3 }

loadPredictionPredictedTime OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time at which the predicted event will be valid."
 ::= { loadPredictionEntry 4 }
.
.
~
.
.

```

Figure 3: MIB Structure for Handling Object Values with Predictive Capability.

In Figure 4, the result of an SNMP query of the relevant predictive MIB Object is displayed. Because the identifiers are suffixed by time, the object values are sorted temporally. If a client wishes to know the next predicted event on or before a given time, the the query can be formulated as a GET-NEXT with the next predicted event time to be determined as the suffix. The GET-NEXT-RESPONSE will contain the next predicted event along with its time of occurrence. Otherwise, a value outside the table will be returned if no such predicted value yet exists.

```

.
.

```

```

~
.
.
loadPredictionTable.loadPredictionEntry.loadPredictionID.1 -> OCTET STRING
loadPredictionTable.loadPredictionEntry.loadPredictionPort.1 -> INTEGER
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.1 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.2 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.3 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.4 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.5 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.6 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.7 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.8 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.9 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.10 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.1 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.2 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.3 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.4 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.5 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.6 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.7 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.8 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.9 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.10 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionCurrentLoad.1 -> REAL32
loadPredictionTable.loadPredictionEntry.loadPredictionCurrentTime.1 -> REAL32
.
.
~
.
.

```

Figure 4: Output from a Query of the MIB Structure for Handling Object Values with Predictive Capability.

This allows SNMP GET-NEXT operations from a client to locate an event nearest to the requested time as well as search in temporal order for next predicted events.

3. A Common Algorithmic Description

SNMP, as currently defined, assumes that non-algorithmic descriptive information will be generated, handled, or transported. Prediction requires model development and execution. This proposed standard SUGGESTS that models are to be small, low-overhead, and fine-grained. Fine-grained refers to the fact that the models are locally constrained in time and space. In this section, we propose algorithmic descriptions of management models designed to encourage the understanding and use of in-line predictive management techniques.

Case Diagrams[4] provide a well-known representation for the relation of management information to information flow as shown in Figure 5. The details of Case Diagrams will not be discussed here (see the previous reference for more information). The purpose of this section is to illustrate an enhancement to the diagram that allows algorithmic information to be specified, particularly for multi-party predictive model interaction.

An excerpt of an SNMP Case Diagram serves to provide a flavor of its current format. The diagram below shows packets arriving from a lower network layer. Some packets are determined to have encoding errors and are discarded. The remaining packets flow to the upper layer.

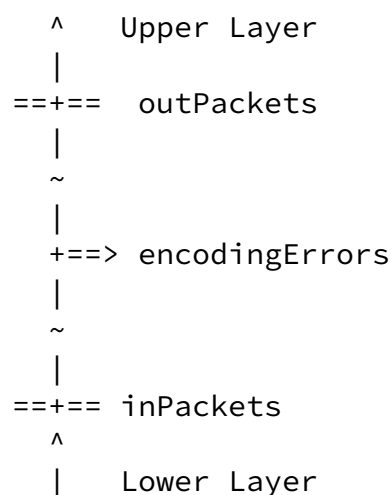


Figure 5: An Example Case Diagram.

For the purposes of in-line predictive management, models SHOULD be specified and injected into the system. These models MAY coexist with the current SNMP management model supplementing the information with predictive values. This is denoted by adding algorithmic model information to the Case Diagram. A '+' sign after the name of an

Object Identifier identifies the object as one that can return future values. The model used to predict the future information is written within braces near the Object identifier and incorporates the name of the SNMP object identifiers. This document SUGGESTS using a common syntax for the notation such as that used for code blocks by the C Programming Language block constructs, Java Programming Language blocks, or the notation used by any number of other languages. Standardization of the model syntax is outside the scope of interest for this document. All functions MUST be defined. Operating system function calls MAY NOT be used. The salient point is that the algorithm must be clearly and concisely defined. The algorithm must also be a faithful representation of the actual predictive model injected into the system. As shown in Figure 6, 'encodingErrors' is predictively enhanced to be 10% of 'inPackets' for future values. The predictive algorithm MUST run on the network node and MUST be immediately available as input for other predictively enhanced objects. The predicted value MUST be available as a response to SNMP queries for future state information, or for transfer to other nodes via virtual messages, explained later in this document. SNMP Objects that are enhanced with predictive capability are assumed to always have the actual monitored value at Wallclock time.

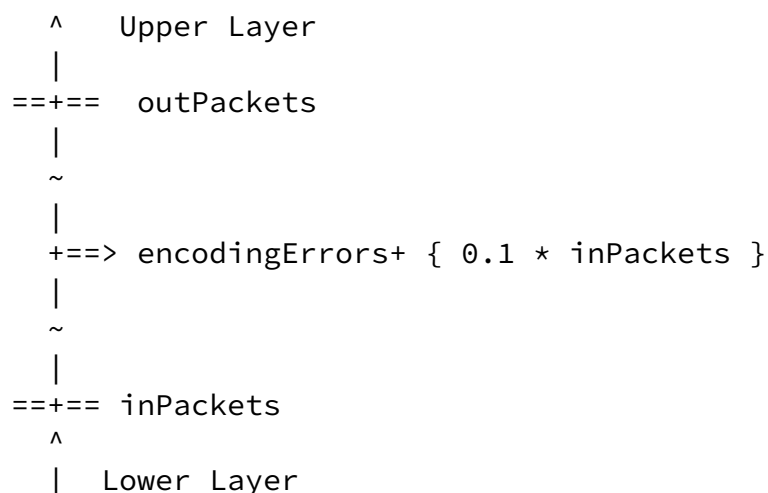


Figure 6: A Sample Algorithmic Description.

If this were a wireless network, a more realistic algorithmic model would likely incorporate channel quality SNMP Objects into the 'encodingErrors' prediction algorithm. In many cases, the

algorithmic portion of the Case Diagram will involve SNMP objects from other nodes. Syntax should include the ability to identify general topological information in the description of external objects. For example, 'inPackets[adj]' or 'inPackets[edge]' should indicate immediately adjacent nodes or nodes at the topological edge of the network.

In the example shown in Figure 7, a 'packetsForwarded' object has predictive capability denoted by the '+' symbol. The predictive capability comes from an algorithmic model specified within the braces next to the object name. In this case, the prediction will be the value of the 'driverForwarded' object from the node closest to the edge of the network.

```

      ^    Upper Layer
      |
==+==  outPackets
      |
      ~
      |
+==>  packetsForwarded+ { driverForwarded[edge] }
      |
      ~
      |
==+==  inPackets
      ^
      |    Lower Layer

```

Figure 7: An Algorithmic Description Using State Generated from Another Node Described in Figure 8.

In the following figure, which is an SNMP diagram of the edge node, the 'driverForwarded' object is predicted by executing the algorithm in braces. This algorithm predicts 'driverForwarded' packets to be a linear approximation of a sample of 'appPackets'. The sample is 'epsilon' time units apart and the prediction is 'delta' time units into the future.

```

      ^    Upper Layer
      |
==+==  driverPackets

```

```

|
~
|
+==> driverForwarded+
|    { delta * (appPackets(t-epsilon) - appPackets(t))/ epsilon
~
|
==+== inPackets
^
|  Lower Layer

```

Figure 8: A Node Generating State Information Used by the Node in Figure 7.

[4. Multi-Party Model Interaction](#)

Multiple developers and administrators of in-line predictive algorithmic models will require mechanisms to ensure correct understanding and operation of each others' models and intentions.

[4.1 Model Registration](#)

It may be necessary to register predictive models. Registration is often an IANA function [\[6\]](#). Algorithmic model registration needs to be handled more dynamically than AgentX models. Algorithmic models, while not necessary doing so, have the capability to install/de-install at rapid rates. The in-line model installation and de-installation proposed standard is described in [Section 7](#).

[4.2 Model Interaction](#)

Multiple models residing on a node need to inter-operate with one another. This document proposes to use SNMP Object Identifiers as much as possible for communication of state information among models. In addition, multiple Active Application models may choose to

communicate with one another via global state.

[4.3](#) Co-existence with Legacy SNMP

Querying an IP addressable node for SNMP objects that are predictively enhanced should appear transparent to the person polling the node. Multiple ports, etc.. should not be required. A program injected into a node that serves to extend an SNMP MIB MAY do so using global state. A global state cache holds the SNMP object values and responds via an internal port to connect with a master SNMP agent for the node.

[5.](#) A Common Predictive Framework

This section specifies an algorithmic predictive management framework. The framework allows details of distributed simulation, such as time management, state saving, and model development to be implementation dependent while ensuring in-line inter-operability both with, and within, the network. The general predictive network management architecture MUST contain at least one Driving Processes (DP), MAY contain Logical Processes (LP), and MUST use Virtual Messages (VM).

Figure 9 illustrates network nodes containing DPs and LPs. The annotation under nodes AH-1 and AN-1 are an SNMP Object Identifier. SNMP Object Identifier 'oid_1' represents state of node AH-1. The

predictively enhanced SNMP Object Identifier, 'oid+' on node AN-1 is a function of 'oid_1'. Note that 'f()' is shown as an arbitrary function in the figure, but MUST be well-defined in practice.

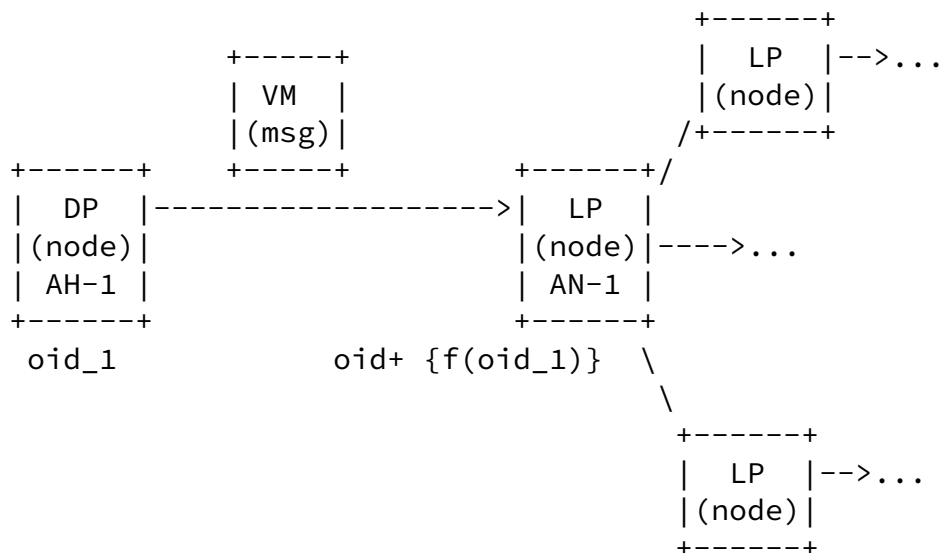


Figure 9: Framework Entity Types.

The framework makes a distinction between a Physical Process and a Logical Process. A Physical Process is nothing more than an executable task defined by program code i.e. it is the implementation of a particular model or a hardware component or a direct connection to a hardware component representing a device. An example of a Physical Process is the packet forwarding process on a router. Each Physical Process MUST be encapsulated within a Logical Process, labeled LP in Figure 9. A Logical Process consists of a Physical Process, or a model of the Physical Process and additional implementation specific data structures and instructions to maintain message order and correct operation as the system executes ahead of

current (or Wallclock) time as illustrated in greater detail in Figure 10. The details of the DP and LP structure and operation are implementation specific, while the inter-operation of the DP/LP system must be specified. The LP architecture is abstracted in Figure 10. The flow of messages through the LP is shown by the arrows entering from the left side of the figure. The in-line predictive framework components are shown in Figure 9, where AH-1 and

AN-1 are Active Host 1 and Active Node 1 respectively. In this context, active hosts are nodes that can inject new packets into the network while active nodes are nodes that behave as intermediate hops in a network.

The Logical Process MUST handle time management for the model. The Logical Process and the model that it implements MAY be implemented in any manner, however, they must be capable of inter-operating. The framework MUST be capable of supporting both conservative and optimistic time management within the network. Conservative time management REQUIRES that the model block when messages MAY be received out-of-order while optimistic time management MAY allow model processing to continue, even when messages are received out-of-order. However, additional implementation specific mechanisms MAY be used to account for out-of-order messages. Such mechanisms MAY be embedded within the Logical Process and this specification does not attempt to standardize them.

Virtual input messages directed to a Logical Process MUST be received by the Logical Process, passed to the model, and processed. Virtual output messages MAY be generated as a result.

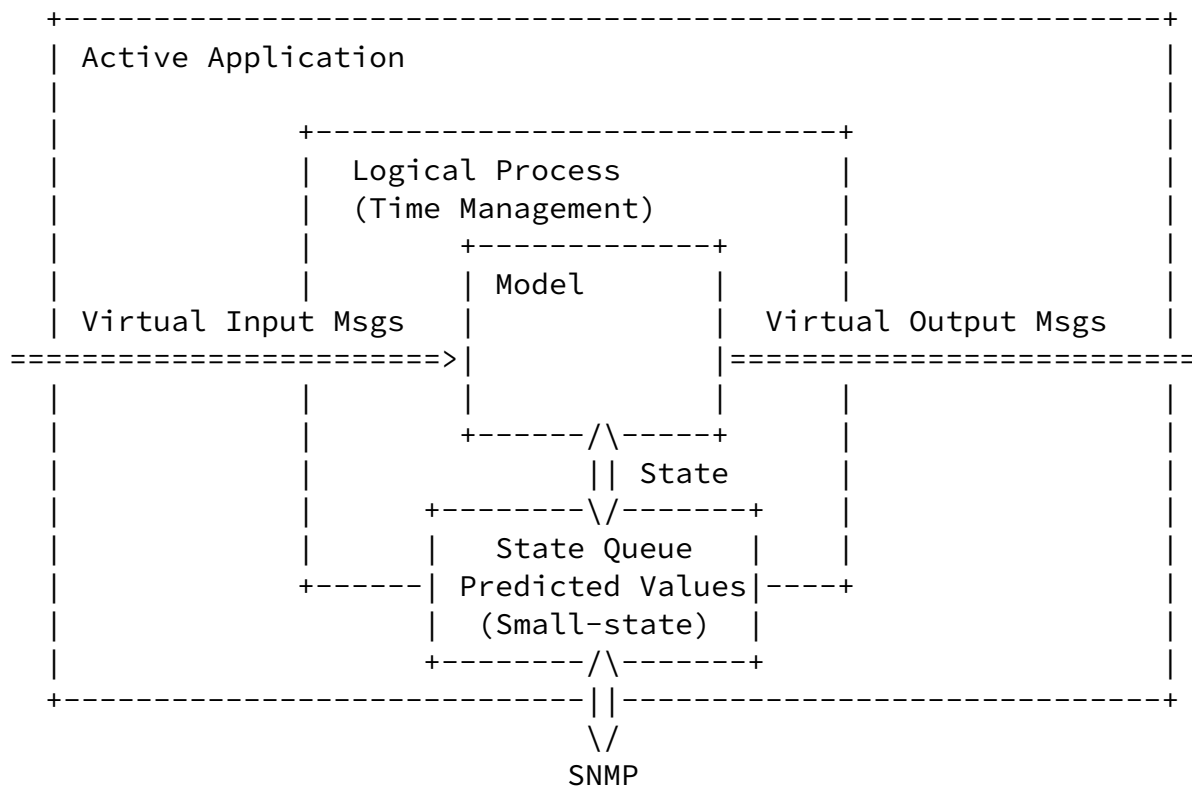


Figure 10: A High-Level View of the Logical Process Framework Component within an Active Application.

Virtual messages contain the following fields:

- o Send Time (TS) which MUST contain the LVT (local simulation time) at which the message was sent
- o Receive Time (TR) which MUST denote the time the message is expected to exist in the future
- o MAY contain an (optional) Anti-toggle (A) bit for out-of-order message handling purposes such as message cancellation and rollback
- o MUST contain the message content itself (M) which is model specific

Thus, a Virtual Message (VM) MUST have the following structure...

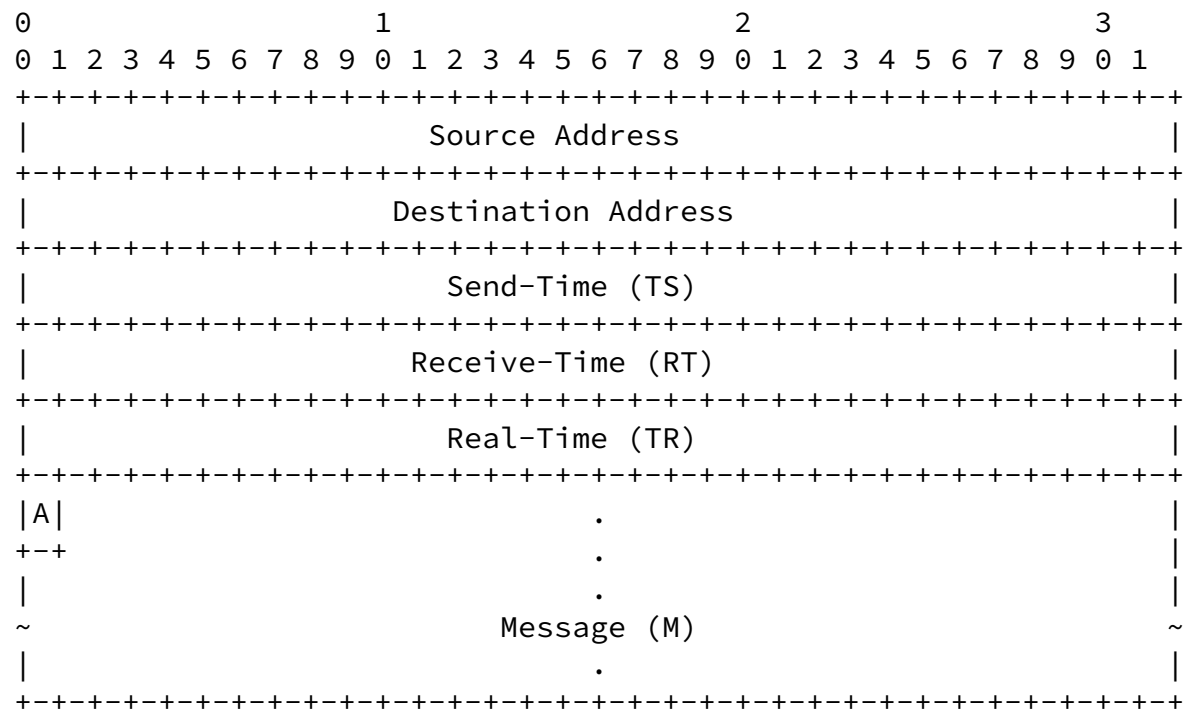


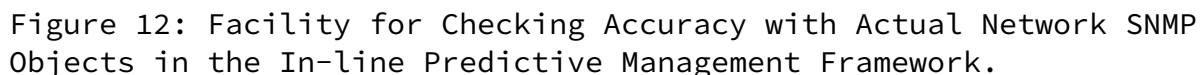
Figure 11: An In-line Management Prediction Virtual Message.

These in-line predictive messages, or virtual messages, that contain invalid fields because the transmitting Logical Processes used an incompatible time management technique MUST be dropped. However, it

[Page 24]

July 2002

The Driving and Logical Processes MUST communicate via virtual messages as shown in Figure 12. The Driving Process MAY generate predictions based upon SNMP queries of other layers on the local node. The Logical Process MAY check its prediction accuracy via SNMP queries of other layers on its local node.



The in-line predictive framework MAY allow for prediction refinement and correction by communicating with the actual component whose state is to be predicted via an SNMP query. The asynchronous prediction mechanism has the following architecture for Logical Process...

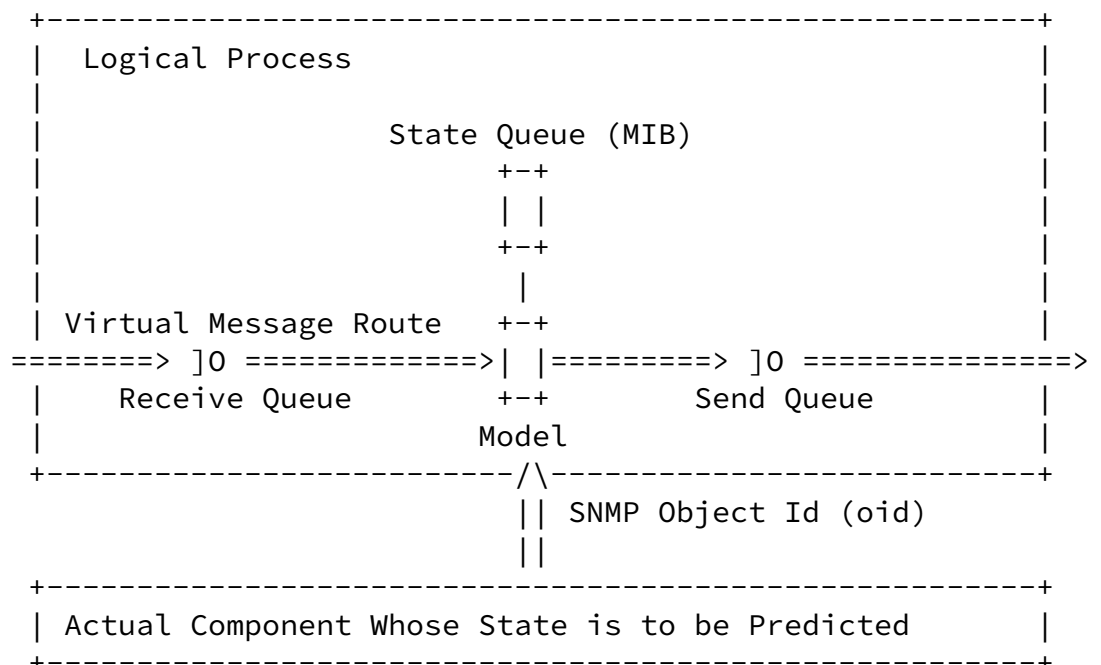


Figure 13: A Logical Process Implementation and Interface.

All of the Logical Process queues and caches MAY reside in an active node's Small-State. Small-State is a persistent memory cache left

behind by an active packet that is available to trailing active packets that have the proper access rights. Typically, any type of information can be stored in Small-State.

The Receive Queue MAY maintain active virtual message ordering and scheduling. All active packets MUST be encapsulated inside Active Packets following the Active Network Encapsulation Protocol [\[7\]](#) format. Once a virtual message leaves the Receive Queue, the virtual time of the Logical Process, known as Local Virtual Time, MUST be updated to the value of the Receive Time from the departing virtual message. Virtual messages MUST originate from Driving Processes, shown in Figure 9 that predict future events and inject them into the system as virtual messages. The development of a Driving Process and Logical Process are dependent upon the model used to enhance the desired state of the system with predictive capability. Logical Processes MUST only operate upon the arrival of virtual input messages and MUST NEVER spontaneously generate virtual messages.

Following the arrows across Figure 13, virtual messages enter either

the Physical Process. The state of the Logical Process is periodically saved in the State Queue (SQ) shown as the State Cache in Figure 13. State Queue values are used to restore the Logical Process to a known safe state when false messages are received. State values are continuously compared with actual values from the Physical Process to check for prediction accuracy, which in the case of load prediction is the number and arrival times of predicted and actual packets received. If the prediction error exceeds a specified tolerance, a rollback MAY occur.

An important part of the architecture for network management is the fact that the State Queue within the in-line management prediction architecture is the node's Management Information Base. The State Queue values are the SNMP Management Information Base Object values; but unlike legacy SNMP values, these values are expected to occur in the future. The State Queue operation is implementation dependent, however, it holds the predicted SNMP Objects, is SUGGESTED to be implemented in small-state, and MUST use the interface specified in [Section 7.2](#) to respond to SNMP queries. The current version of SNMP has no mechanism to indicate that a managed object is reporting its future state; currently all results are reported with a timestamp that contains the current time. In working on predictive active

network management prediction 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 Management Information Base table indexed by predicted time as discussed in [Section 2](#). This can be seen in the loadPredictionTable shown in Figure 3. Thus a Simple Network Management Protocol client, who 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. The figure illustrates an SNMP request and the corresponding response.

Future times are the LVT of the Logical Process running on a particular node. As Wallclock approaches a particular future time, predicted values MAY be adjusted, allowing the prediction to become more accurate. The table of future values MAY be maintained within a sliding Lookahead window, so that old values are removed and the prediction does exceed a given future time. Continuing along the arrows in Figure 13, any virtual messages that are generated as a result of the Physical Process or model computation proceed to the Send Queue (QS).

The Send Queue is implementation dependent, however, it MAY maintain

copies of virtual messages to be transmitted in order of their send times. The Send Queue is required for the generation of anti-messages during rollback. Anti-Messages annihilate corresponding virtual messages when they meet to correct for previously sent false messages. Annihilation is simply the removal of both the actual and the anti-message. Where the annihilation occurs is implementation specific and left to the implementor. After leaving the Send Queue, virtual messages travel to their destination Logical Process. Further details on the optimistic synchronization mechanism are implementation dependent and outside the scope of this work in progress.

6. Summary of In-line Prediction Requirements

An in-line management prediction model developer **MUST** implement at least one Driving Processing and **MAY** implement a Logical Process using the same time management technique. The model developer **MAY** include an SNMP client within the model in order to query the modeled component in order to improve prediction accuracy. The model developer's Driving Process **MUST** generate virtual messages. The

Logical Process MUST receive and process those messages. The Logical Process MAY respond to virtual messages by generating virtual message(s). The Logical Process MAY use active network node Small-state to hold a time series of the SNMP Object Id whose value is being continuously predicted. The interface to the SNMP MIB small-state is specified in the following section.

The general active network architectural framework, without any specific network management paradigm implementation, is shown in Figure 14.

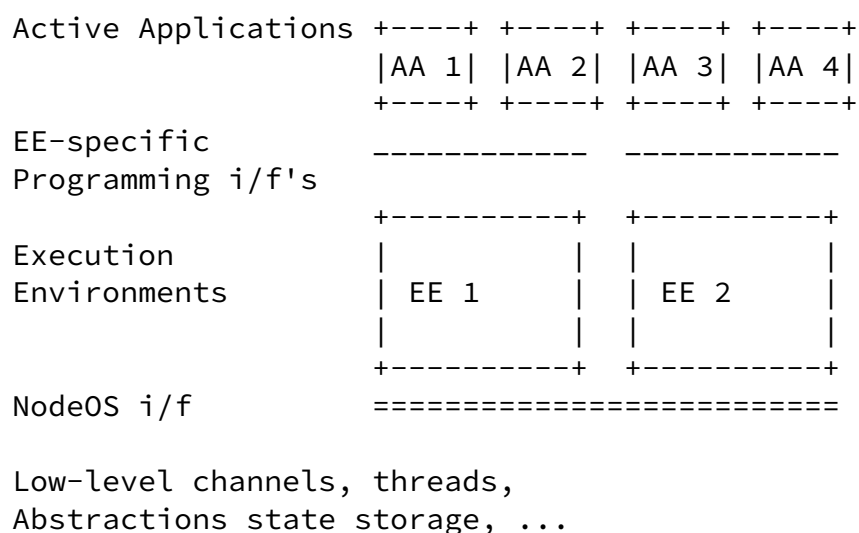


Figure 14: The Active Network Framework.

In-line network management prediction requires a general active network framework that supports active applications to be injected into the proper execution environments. The in-line management prediction framework enforces certain minimal requirements on the execution environment, which are listed below.

[7.1](#) Information Caches

The execution environment **MUST** provide an information cache called 'Small State' as defined in [Section 1.3](#) to enable information exchange between active packets, defined in [Section 1.3](#). The execution environment **MAY** also provide an information cache called 'Global State', defined in [Section 1.3](#), to enable the in-line management prediction framework to communicate with a predictively managed active application to query its current state. The EE **MUST** provide an API to be able to store and query both 'Small State' and also to 'Global State', if it is implemented. The EE **SHOULD** provide appropriate access control mechanisms to both 'Small State' and also to 'Global State', if it is implemented.

[7.2](#) Interface to SNMP

The execution environment **MUST** provide an interface that enables both the in-line management prediction values and the values of the actual

component being managed to publish their state to an SNMP MIB. This enables the in-line management prediction framework to store the predicted state in a well-known format and also enables legacy SNMP tools to query the predicted state using SNMP operations. Additionally, the managed application is also able to update its current state using SNMP, which the Logical Process will be able to query. In a particular implementation of such an interface, a generic SNMP agent coded as an active application MAY be injected into the active nodes. The agent creates a 'Global State' on the active node with a well-known name. The agent reads information coded in a known format that has been written to the 'Global State' and publishes it to the MIB. Any active application that wishes to advertise its state uses an interface that enables it to store its information in the well-known 'Global State' in the given format.

The format of the messages that are posted between the SNMP agent and an active application are shown below,

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Message Type           |           Object ID           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     .                                     |
~                                     Value                                |
|                                     .                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 15: Message Packet.

The SNMP Agent and the active application MAY use special interfaces to implement messaging between them. A Message Packet, whose format is shown in Figure 15, is the basic unit of inter-application communication. Each message consists of a message type. The type SHOULD assume one of the following values:

- o MSG_ADDINT: to add a new MIB Object of type SNMP INTEGER
- o MSG_UPDATEINT: to update the value of an MIB Object of type SNMP INTEGER
- o MSG_GETINT: to get the value of an MIB Object of type SNMP INTEGER
- o MSG_ADDLONG: to add a new MIB Object of type SNMP LONG
- o MSG_UPDATELONG: to update the value of an MIB Object of type SNMP LONG

- o MSG_GETLONG: to get the value of an MIB Object of type SNMP LONG

- o MSG_ADDSTRING: to add a new MIB Object of type SNMP STRING
- o MSG_UPDATESTRING: to update the value of an MIB Object of type SNMP STRING
- o MSG_GETSTRING: to get the value of an MIB Object of type SNMP STRING

The active application SHOULD send a message of the valid message type to the SNMP agent to perform the required operation. On receipt of a message, the SNMP agent SHOULD attempt to perform the requested operation. It MUST then respond with an acknowledgment message in a format shown in Figure 16.

The acknowledgment message has the following format.

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Status Code                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     .                                               |
~                                     Status Message                               ~
|                                     .                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 16: Acknowledgment Message Packet.

The status code MUST have one of the following values:

- o OK: to indicate successful operation
- o ERR_DUPENTRY: if for a MSG_ADD operation, an Object identifier of given name already exists
- o ERR_NOSUCHID: if for a MSG_UPDATE operation, an Object identifier of given name does not exist.

The Status message MAY be any descriptive string explaining the nature of the failure or SHOULD be "Success" for a successful

operation.

[8](#). Implementation

Models injected into the network allow network state to be predicted and efficiently propagated throughout the active network enabling the network to operate simultaneously in real time as well as project the future state of the network. Network state information, such as load, capacity, security, mobility, faults, and other state information with supporting models, is automatically available for use by the management system with current values and with values expected to exist in the future. In the current version, sample load and processor usage prediction applications have been experimentally validated using the Atropos Toolkit [[11](#)]. The toolkit's distributed simulation infrastructure takes advantage of parallel processing within the network, because computation occurs concurrently at all participating active nodes. The network being emulated can be queried in real time to verify the prediction accuracy. Measures such as rollbacks are taken to keep the simulation in line with actual performance.

[8.1](#) Predictive In-line Management Information Base

Further details on the in-line network management prediction concept can be found in Active Networks and Active Network Management [[1](#)]. The SNMP MIB for the in-line predictive management system described in this proposed standard follows in the next section.

```
ATROPOS-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
MODULE-IDENTITY, OBJECT-TYPE, experimental,  
Counter32, TimeTicks
```

```
FROM SNMPv2-SMI
```

```
DisplayString  
FROM SNMPv2-TC;
```

```
atroposMIB MODULE-IDENTITY  
LAST-UPDATED "9801010000Z"  
ORGANIZATION "GE CRD"  
CONTACT-INFO  
"Stephen F. Bush bushsf@crd.ge.com"  
DESCRIPTION  
"Experimental MIB modules for the Active Virtual Network  
Management Prediction (Atropos) system."  
::= { experimental active(75) 4 }  
  
--  
-- Logical Process Table  
--
```

Bush, et al.

Expires December 30, 2002

[Page 33]

Internet-Draft

In-Line Network Management Prediction

July 2002

```
lP OBJECT IDENTIFIER ::= { atroposMIB 1 }
```

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

```
LPEntry OBJECT-TYPE  
SYNTAX LPEntry  
MAX-ACCESS not-accessible  
STATUS current  
DESCRIPTION  
"Table of Atropos 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 }

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
 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
 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

"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 }

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 }

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 Atropos 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 }

lPETrb 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

Bush, et al.

Expires December 30, 2002

[Page 39]

Internet-Draft

In-Line Network Management Prediction

July 2002

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
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
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 INTEGER (0..2147483647)
--SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time in milliseconds that Atropos has been
running on this node."
::= { lPEntry 32 }

END
```

Figure 17: The Atropos MIB.

[9](#). Security Considerations

Clearly, the power and flexibility to increase performance via the ability to inject algorithmic information also has security implications. Fundamental active network framework security implications will be discussed in [\[10\]](#).

References

- [1] Bush, S. and A. Kulkarni, "Active Networks and Active Network Management (ISBN 0-306-46560-4)", March 2001.
- [2] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", [RFC 2570](#), April 1999.
- [3] Wijnen, B., Harrington, D. and R. Presuhn, "An Architecture for Describing SNMP Management Frameworks", [RFC 2571](#), May 1999.
- [4] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2)", [RFC 1450](#), April 1993.
- [5] Daniele, M., Wijnen, B., Ellison, M. and D. Francisco, "Agent Extensibility (AgentX) Protocol Version 1", [RFC 2741](#), January 2000.

- [6] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 2434](#), October 1998.
- [7] University of Pennsylvania, USC/Information Sciences Institute, University of Pennsylvania, BBN Technologies, University of Pennsylvania, University of Kansas and MIT, "Active Networks Encapsulation Protocol", July 1997.
- [8] MIT and MIT, "The Active IP Option", September 1996.
- [9] IETF, "Proposed IEEE Standard for Application Programming Interfaces for Networks", October 2000.
- [10] Princeton University, "Active Network Framework", July 2002.
- [11] <<http://avnmp.sourceforge.net/download.html>>

Bush, et al.

Expires December 30, 2002

[Page 42]

Internet-Draft

In-Line Network Management Prediction

July 2002

Authors' Addresses

Stephen F. Bush
GE Global Research Center
1 Research Circle
Niskayuna, NY 12309
US

Phone: +1 518 387 6827
EMail: bushsf@crd.ge.com
URI: <http://www.crd.ge.com/~bushsf/>

Amit B. Kulkarni
GE Global Research Center
1 Research Circle
Niskayuna, NY 12309
US

Phone: +1 518 387 4291
EMail: kulkarni@crd.ge.com

Nathan J. Smith
GE Global Research Center
1 Research Circle
Niskayuna, NY 12309
US

Phone: +1 518 387 6285
EMail: smithna@crd.ge.com

Bush, et al.

Expires December 30, 2002

[Page 43]

Internet-Draft

In-Line Network Management Prediction

July 2002

Index

A

AA 5, 5, 7

Active IP 2

Active Network 8
Active Networking 2, 2
Active Packet 8, 26
AgentX 4
Algorithmic Management 11
Algorithmic Information 1
Algorithmic Information 7
Algorithmic
 Change 2
 Information 2
Anti-Message 8, 25
Anti-Toggle 7
Atropos Toolkit 33

C

Case Diagram 16
Common Predictive Framework 21
Conformance 11

D

DARPA Active Network Program 2
Descriptions
 Algorithmic 16
Driving Process 21
Driving Process 8

E

EE 5, 8
Executable Code 2

F

Fine-Grained Models 6

G

Global-State 7

I

In-line Predictive Network Management 4
In-line Management Code 1
In-line 6

L

Legacy SNMP 20

Local Virtual Time 9
Logical Process 21
Logical Process 8, 21
Lookahead 8, 27

M

Management Algorithms 2
Model Interaction 20
Model Registration 20
Multi-Party In-line Predictive Management Model 7
Multi-Party Model Interaction 20

N

NodeOS 5, 9
Non-Algorithmic Information 7

P

Physical Process 9, 21
Predictive Network Management 6
Predictive Network Management 2
Programmable Networking 2
Programmable Networking 2

R

Real Time 10
Receive Queue 10, 26
Receive Time 10
Rollback 9, 27

S

Send Queue 9, 27, 27
Send Time 10
Simplicity 11
Small-State 7, 26
State Queue 10, 27

T

Tolerance 10

V

Virtual Message 9, 10, 17, 21

W

Wallclock 10

Internet-Draft In-Line Network Management Prediction

July 2002

Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Bush, et al.

Expires December 30, 2002

[Page 46]