ADAPTIVE SAMPLING FOR NETWORK MANAGEMENT

By

EDWIN A. HERNANDEZ

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2000

ACKNOWLEDGMENTS

I would like to thank God, my parents, and my beloved sisters for their help and support throughout the development of this thesis. Special thanks go to the Fulbright Commission and the Institute of International Education for the continuous monetary support required to complete this work. Thanks to Dr. George for his suggestions, guidance, patience, and support. My gratitude also goes to all the members of the HCS Lab (CG's and ITMSB), to my friends Jose, Miguel, Ryan, Damian, Christophe, Richard, Jesus, Stefan, Sri, Kim, and especially to Matt for all his help and suggestions correcting the manuscripts and providing useful ideas for this thesis. Finally, I would like to thank Dr. Arroyo and Dr. Lam for taking part of their time to review this thesis and form part of my committee.

TABLE OF CONTENTS

page

APPENDICES

LIST OF FIGURES

Figure page

LIST OF TABLES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ADAPTIVE SAMPLING FOR NETWORK MANAGEMENT

By

Edwin A. Hernandez

May 2000

Chairman: Alan D. George
Major Department: Electrical and Computer Engineering

High-performance networks require sophisticated management systems to identify sources of bottlenecks and detect faults. At the same time, the impact of network queries on the latency and bandwidth available to the applications must be minimized. Adaptive techniques can be used to optimize sampling of network information, reducing the amount of processed data and lessening the overhead on the network. Two adaptive sampling methods are proposed in this thesis based on linear prediction and fuzzy logic. The performance of these techniques is compared with conventional sampling methods by conducting simulative experiments using Internet and videoconference traffic patterns. The adaptive techniques are significantly more flexible in their ability to dynamically adjust with fluctuations in network behavior, and in some cases they are able to reduce the sample count by as much as a factor of two while maintaining the same accuracy as the best conventional sampling interval. The

results illustrate that adaptive sampling provides the potential for better monitoring, control, and management of high-performance networks with higher accuracy, lower overhead, or both.

CHAPTER 1
INTRODUCTION

In network management, accurate measures of network status are needed to aid in planning, troubleshooting, and monitoring. Network query rates must strike a balance between accurate performance characterization and low bandwidth consumption. This balance is often achieved through sampling. Sampling techniques are used to study the behavior of a population of elements based on a representative subset. In general, the samples are taken periodically at some fixed interval or in some random distribution. Such sampling reduces bandwidth and storage requirements for the monitored data. In a high-performance network, sampling overhead must have a minimal impact on the low-latency application transactions while high network throughputs require frequent sampling to capture transient behavior.

Under some traffic loads, simple periodic sampling may be poorly suited to the monitoring task. For example, during periods of idle activity or low network loads, a long sampling interval provides sufficient accuracy at a minimal overhead. However, bursts of high activity require shorter sample intervals to accurately measure network status at the expense of increased sample traffic overhead. To address this issue, adaptive sampling techniques can be employed to dynamically adjust the sampling interval and optimize accuracy and overhead.

Adaptive sampling monitors network behavior by dynamically adjusting the sampling time interval for each parameter monitored. When levels of high activity are

detected, a shorter sampling interval is employed to measure the behavior of the network with greater accuracy. When less activity is detected, the sampling interval is lengthened to reduce sampling overhead. The adaptive algorithms introduced in this thesis are used to select the most representative samples of the population by polling and monitoring key fluctuations in the variables being measured. Consequently, adaptive sampling techniques allow the management to take place in a less-intrusive fashion by avoiding unnecessary queries.

This thesis introduces two techniques for adaptive sampling. The first technique is based on Linear Prediction (LP) [1,2]. The LP sampler uses previous samples to estimate or predict a future measurement. The LP sampler can be used in conjunction with a set of rules defining the sampling rate adjustments to make when the prediction is inaccurate.

The second technique for adaptive sampling makes use of fuzzy logic. Fuzzy logic mimics the decision processes employed by the human brain [3]. For example, a network administrator may reason that when network load is low, the sample interval can be increased. The fuzzy logic model relies on previous experiences for defining the fuzzy set of parameters and boundaries for the fuzzy variables.

In order to gauge the performance of the adaptive techniques, they are compared with systematic sampling. This technique uses a deterministic interval of time to query the agents. Other non-adaptive sampling techniques include *random sampling* where samples are taken at intervals of time determined by a random distribution, and *stratified random sampling* where a sample is taken at a random point within a deterministic time interval [4,5].

This thesis introduces the concept of adaptive network sampling and provides experimental results with the various sampling techniques comparing them in terms of accuracy and performance using Internet and videoconference traffic. Chapter 2 resents the background information required to understand network management. In Chapter 3 the sampling techniques used in this study are defined. Chapter 4 describes the experiments and measurement techniques used to compare the performance of the sampling disciplines are described. The performance results are shown in Chapter 5. Related research is described in Chapter 6, while conclusions and directions for future research are drawn in Chapter 7.

CHAPTER 2
BACKGROUND

This chapter will introduce the general concepts of network management. First, the network management model and the Simple Network Management Protocol (SNMP) are explained. Finally, the management architecture and the management information base are presented at the end of the chapter.

Network Management

Network management comprises all the elements necessary to ensure the effective and efficient operation of a system and its resources. The objective of management is to provide a desired level of quality and assure availability for the services and applications of a networked environment. It is the key for efficient network utilization and control. In the 1980's, the Network Control Center (NCC) was in charge of all management functions [6]. The NCC was an entity formed by the Network Management Stations (NMS), computing element employed to monitor or configure the network, and the human operators. The process of management took place commenced by polling different network nodes and representing the information gathered in graphical terminals. As expected, all vendors proposed management architectures, which in general were not standardized or compatible with each other.

Because systems were inefficient and unscalable efforts were made to standardize procedures. The Internet Architecture Board (IAB) and International Standard Organization (ISO) were the institutions in charge of developing of standards for management. Those efforts lead to the development of the management protocols denoted by the Simple Network Management Protocol (SNMP) [7,8] and the Common Management Information Protocol (CMIP) [9].

The standards define a set of application level protocols used for management of network devices. For instance, SNMP works using the User Datagram Protocol (UDP) [10] while CMIP works in different transport protocols but was standardized to operate in conjunction with the Transmission Control Protocol (TCP) (also called CMIP over TCP/IP or CMOT).

The IAB proposed SNMP to be the solution for management of IP networks. ISO released the CMIP much later and by the time CMIP was available, SNMP was the standard utilized by the industry. Improved versions of the SNMP have been released. In 1993 the version 2 and recently the version 3 in 1998.

The Comite Consultatif International de Telegraphie et Telephonie (CCITT) defined the Open Systems Interconnection (OSI) model and other telecommunciation standards. In 1992 the CCITT became part of the International Telecommunications Union (ITU) which made use of a set of the recommendations made by the CCITT to propose the OSI-CMIP management architecture. CMIP is still being used in the management of public telephone networks. Besides CMIP and SNMP, the Desktop Management Task Force (DMTF) has proposed the Common Information Model (CIM)

[11]. The CIM proposed that management of systems, software, users, and networks could be structured and conceptualized upon the object-oriented paradigm.

The following section provides a description of the SNMP protocol.

Network Management Model for SNMP

Network management has four components [8]. The first component, *managed nodes*, encapsulate the management *agent*. The second corresponds to the *network management station* (NMS) where the network management applications reside, also called *managers*. The *network management protocol,* the third component, is used by all the management stations and the agents to exchange management information. Finally, the *Management information* represents the status of the agent.

In SNMP-based management, the "client" and "server" terms are not utilized. The terms "managers" and "agents" apply instead. In the SNMP architecture, a *managed node* stands for all the devices, such as switchers, routers, repeaters, mainframes, workstations, applications, and networks analyzers. In other words, the spectrum of managed nodes ranges from mainframes to modems. A management node must monitor activity as well as perform regular tasks. The agent should keep updated counters of transferred bytes, errors, throughput, and identification information.

The *network management station* (NMS) consists of a host system where software applications are executed. Both the agent and manager share the same management protocols; the primary difference relies on the capabilities found at the NMS. The

NMS showed to have better processing support and to be able to handle hundreds of agents. The network management protocol makes use of two operations:

- *Traversal operations* (Figure 1) which allows a management station to determine which variables a managed node supports; and,
- *Trap operations*, (Figure 2) which allows a managed node to report an extraordinary event to a management station.

If the information is represented in tables, the *traversal operation* provides the mechanism to retrieve the cells from the managed node. It could also allow the retrieval of large numbers of cells in a table, i.e., a routing table. The latter operation, *traps*, occurs when an event at the agent triggers a message and sends it to the management station.

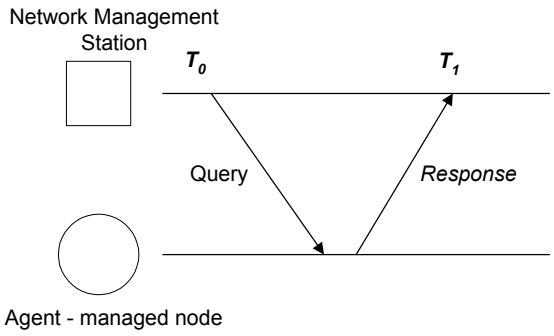


Figure 1. Polling-based management.

The network management protocol is based upon periodical polling of the agents and retrieving management information. Therefore, redundant information is polled since no selectivity is defined in the SNMP protocol.

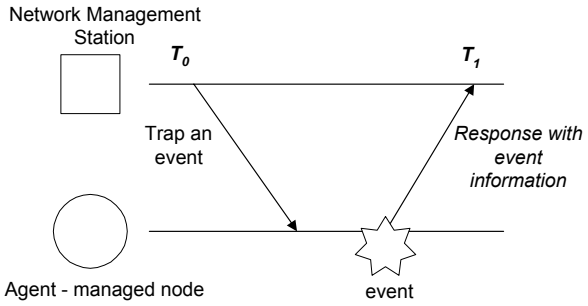


Figure 2. Trap-based management.

The *management information* termed *managed object* defines the data-types used by the management stations and the managed nodes. Each of the *managed nodes* or network elements maintains a database of information referred to as the *Management Information Base* (MIB).

Management Information Base

The language used to describe the database of information is the *Abstract Syntax Notation One* or ASN.1, which is a formal representation, developed and standardized by ITU (International Telecommunication Union formerly called CCIT) and ISO. ASN.1 is used to define the management information bases for SNMP and CMIP systems.

Each MIB uses a tree-structured architecture to organize all of the available information. The information in the tree is a labeled node that contains -- the Object Identifier (OID) and a short text description.

The Object Identifier (OID) is a series of integers separated by periods that name the node and denote the exact traversal of the ASN.1 tree. The short text description describes the labeled node.

Figure 3 depicts the MIB tree.

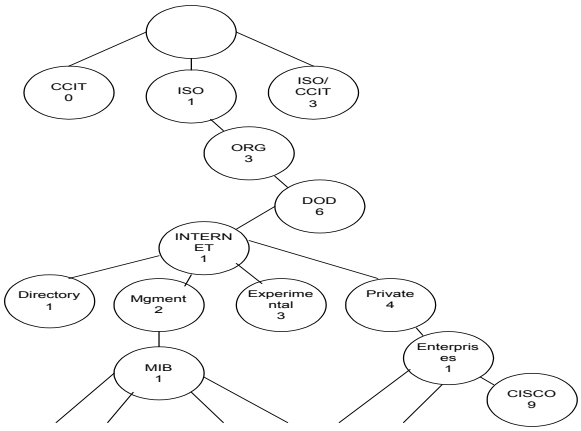


Figure 3. Management Information Base (MIB) tree structure.

The root node of the MIB tree does not have a name or number. This node is composed of three sub-trees: *ccitt(0)*, administered by the ITU, *iso(1)*, administered by the ISO and *iso-ccitt(2)*, administered by ISO and ITU.

In addition, several trees exist under the *iso(1)* node, including the ISO-defined sub-tree for other organizations, *org(3)*. Under *org(3)* sub-tree, a particular node of interest is the used by the US Department of Defense (DOD): *dod(6)*. This node is the parent of the Internet sub-tree.

In order to form the OID number, the sequence of nodes in the path to any node defines the number. For instance,

Figure 3 illustrates that the node for the company CISCO corresponds to an OID with the number 1.3.6.1.4.1.9.

```
sysUpTime OBJECT-TYPE
                SYNTAX TimeTicks
                ACCESS read-only
                STATUS mandatory
                DESCRIPTION
                        "The time (in hundredths of a second) since
                        the network management portion of the
                        system was last re-initialized."
                   ::= {system 3}
```

Figure 4. ASN.1 data-type example of the sysUpTime parameter.

Figure 4 illustrates the syntax used by the ASN.1 to define the parameter called *sysUpTime*. The SYNTAX line gives the type of the identifier; this parameter could be INTEGER, COUNTER or any ASN.1 data type. The ACCESS line gives the visibility of this identifier with the outside world; in other words, if this node can be updated or only read by a network manager without proper authorization, then the DESCRIPTION line is self-explanatory. The last line assigns another OBJECT IDENTIFIER to *sysUpTime*, which is a link to *{system 3}* or the *system node* of the MIB tree. The entire MIBs node must have a link to a parent node and its correspondent identification number.

Consequently, the *sysUpTime* is a node in the MIB tree located at: 1.6.3.1.2.1.1.3. or *iso.org.dod.internet.mgmt.mib2.system.sysUpTime*.

SNMP Architecture

Figure 5 illustrates the network management station and the interactions with agents. Observe that the Management Station has a direct access to an MIB. In order to manage the agents, the manager should have copies of the information bases present in the network. Unfortunately, an MIB discovery protocol is not specified in the standard [12].

The SNMP is composed of the basic commands (Figure 6):

- *GET,* which retrieves the information from the agent.
- *SET,* in charge of setting a value or values at the MIB.
- *TRAP,* which enables the agent to notify the management station for specific events, failures, and shutdowns.

The management application polls SNMP agent using User Datagram Protocol (UDP) at the port 161. Figure 6 illustrates a management scenario with a management station polling and receiving trap information from a router and two hosts.

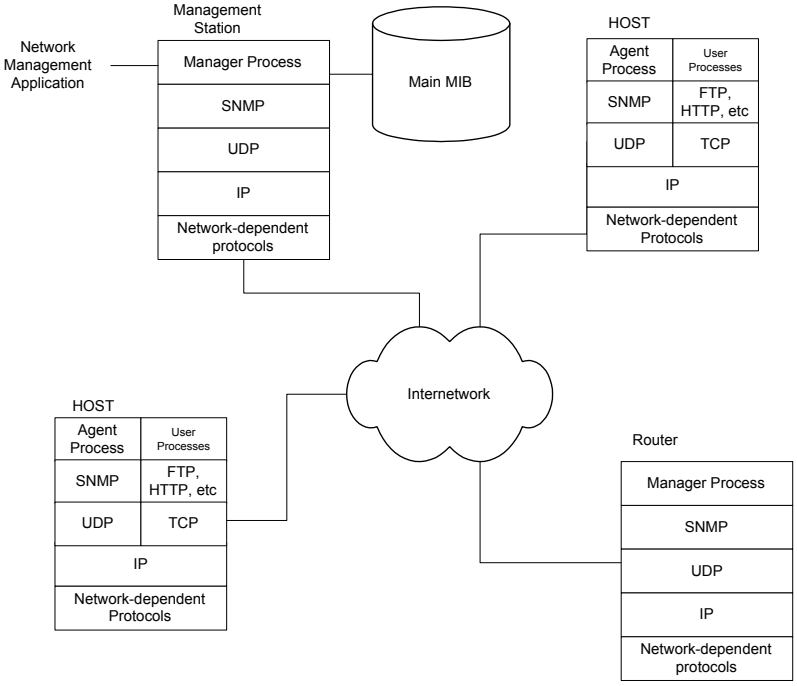


Figure 5. SNMP-based management framework..

The SNMP packet format is depicted in Figure 7. The packet contains the version, the community name (public or password), and the Protocol Data Unit (PDU). The PDU holds the information retrieved from the agents. The *ERROR ID* and *ERROR INDEX* fields indicate the error type in the returning frame. *Request ID* and *PDU Type* depend upon the five types of packets used in SNMPv1 (Figure 5.).

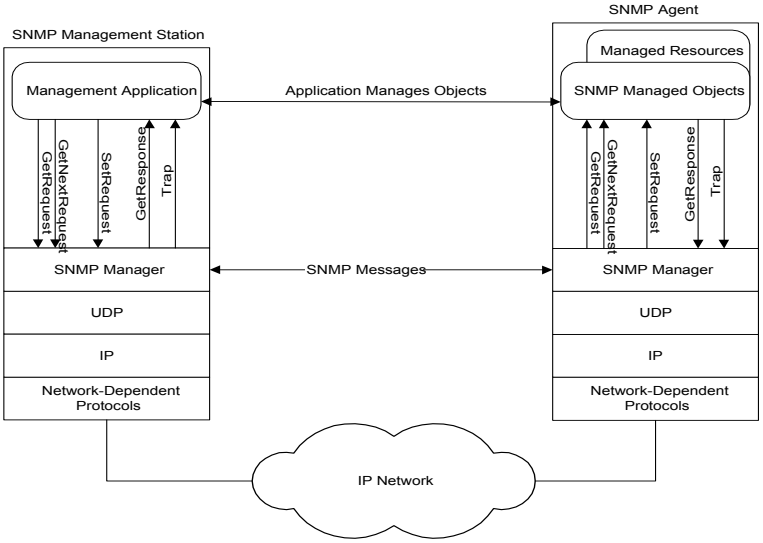


Figure 6. SNMP interaction between manger and agent.

The fields indicated by *Object ID*$_i$, called Object Identifiers (OID), are the ASN.1 data-types accessed by the SNMP packet. Consequently, the PDU has variable length depending upon the number of OIDs used. This variability permits the manager to review or modify several OIDs simultaneously.

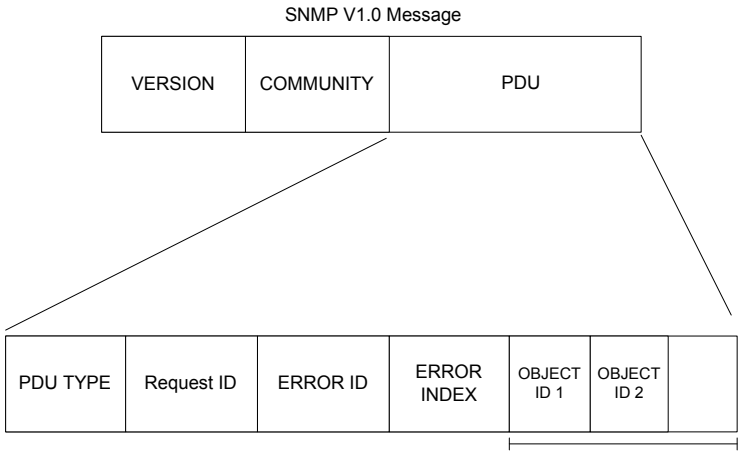


Figure 7. SNMP packet format.

CHAPTER 3
SAMPLING TECHNIQUES FOR NETWORK MANAGEMENT

In network management, status information regarding load, latency, queue occupancy, and other parameters is frequently available in devices such as routers, switches, and network interfaces. Such information is often accessed through the Simple Network Management Protocol (SNMP) [7,8,10]. In SNMP, a Network Management Station (NMS) queries the network devices, or agents, to periodically assess the status of the network devices or links.

The period of the sampling determines the accuracy of the measured data. Transient activity may not be accurately detected when the sampling interval is large, while small intervals consume more bandwidth on the network and require greater storage capacity at the NMS. In an effort to balance accuracy with sampling overhead, several sampling disciplines have been applied to network managers.

Conventional Sampling

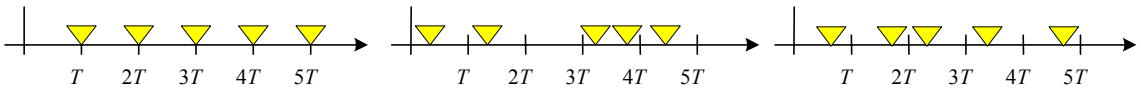
Traditionally, network management has made use of simple, non-adaptive sampling techniques. Such techniques use a fixed rule to determine when to sample data in each agent. The sampling rule can be deterministic, such as in periodic sampling, or it can involve a random component. Introducing randomness has been shown to improve accuracy in situations where the monitored data is uniform in nature

[4]. There are three conventional methods used by network management systems for sampling of agents:

Systematic sampling, or periodic sampling, deterministically samples data at a fixed time interval. Figure 8(a) shows systematic sampling with a period of *T* seconds.

Simple random sampling employs a random distribution function to determine when each sample should be taken. The distribution may be uniform, exponential, Poisson, etc. As shown in Figure 8(b), random sampling may take a varying number of samples in a given time interval.

Stratified random sampling combines the fixed-time interval used in systematic sampling with random sampling by taking a single sample at a random point during a given time interval. Figure 8(c) shows stratified random sampling with a time interval of *T*.



(a) Systematic sampling (b) Random sampling (c) Stratified random sampling

Figure 8. Conventional sampling methods.

The rate for any of these non-adaptive sampling techniques is typically set based on the expected average network load, the traffic distribution, or simply to a value that will yield an acceptably small amount of overhead. If the actual traffic differs from the

expected pattern, the measurements may prove inaccurate or may take an excessive number of samples.

Adaptive Sampling

Adaptive sampling dynamically adjusts the sampling rate based on the observed sampled data. A key element in adaptive sampling is the prediction of future behavior based on the observed samples. The two adaptive sampling techniques presented in this thesis differ by the technique used to predict the future behavior of the system. The Linear Prediction (LP) method attempts to predict the value of the next sample. If the prediction is accurate, the sampling rate can be reduced. Inaccurate predictions indicate a change in the network behavior and require an increased sampling rate to determine the new pattern. By contrast, the Fuzzy Logic Controller (FLC) [3,14,15] adjusts the sampling rate based on experiences in past situations with similar sample data. A rule structure is defined that the FLC can follow to determine the most appropriate action to take given a certain input condition.

Linear Prediction (LP) Method

The LP method proposed for adaptive sampling is based on the Linear Prediction Coefficient (LPC) technique used by Jacobson and Karels in the congestion control protocol for TCP to predict the Round Trip Time (RTT) of a packet [16]. LPC employs a low-pass filter to predict future data values. This technique filters out transient behavior, using the average value of the non-transient data as the prediction for the next

value. This predicted time value was used to adjust the transmission window base on the RTT of previous packets. The low-pass filter is of the form:

$$x_p = (1-\alpha)\times M + \alpha \times R \tag{1}$$

In this equation, x_p is the predicted value for the next sample, M is the most recently measured value, and R represents the average of the previous two samples. The coefficient α ranges between zero and one and is selected experimentally depending on the network load, where higher values of α reduce the impact of transient values on the average value.

A similar LP-based technique is used in this thesis to control the sampling time. The main difference between the approach described in this thesis and the one used by Jacobson and Karels is that here a variable number of previous samples are used to calculate the predicted value. Instead of using only the previous two samples, a window of N samples is used for the prediction. Moreover, the α coefficient for tuning to a specific network load is no longer employed.

Eq. (2) below defines the LP logic for a sampler of order N. As in Eq. (1), the value x_p represents the predicted value for the next sample. The vector x holds the value of the previous N samples where $x[N]$ is the most recent sample and $x[1]$ is the oldest sample. A second vector, t, records the time that each sample is taken. Since the period between samples is not necessarily constant, the next value is predicted based on the average rate of change in the previous N samples.

$$x_p = x[N] + \frac{\Delta T_{Current}}{N-1}\sum_{i=1}^{N-1}\left(\frac{x[i+1]-x[i]}{t[i+1]-t[i]}\right) \tag{2}$$

The prediction is then used as shown in Figure 9. The predicted output, x_p, is compared with the actual value of the next sample. A set of rules are applied to adjust the current sampling interval, $\Delta T_{Current}$, to a new value, ΔT_{Next}, which is used to schedule the next management query. The rules used to adjust the sampling interval compare the rate of change in the predicted sample value to the actual rate of change. This rate, m, is given by Eq. (3).

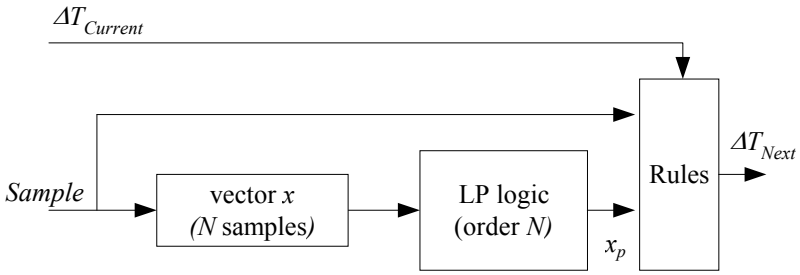


Figure 9. LP-based adaptive sampling.

$$m = \frac{\left| x_p - x[N] \right|}{\left| Sample - x[N] \right|} \tag{3}$$

The rate of change provided by Eq. (3) will take on a value near unity when the predicted behavior is close to the actual behavior. The range of values that satisfy this condition are defined as $m_{min} < 1 < m_{max}$. If m is below m_{min}, the measured parameter is changing faster than the prediction. Such behavior indicates more activity than predicted, so the sampling interval should be decreased to yield more accurate data values on which to base future predictions. Conversely, if m is above m_{max}, the measured value is changing more slowly than the prediction so the sampling interval can be increased.

Table 1. LP rules for adjusting sample interval.

Calculated m Value	Next Sample Interval
$m < m_{min}$	$\Delta T_{Next} = m \times \Delta T_{Current}$
$m_{min} \leq m \leq m_{max}$	$\Delta T_{Next} = \Delta T_{Current}$
$m_{max} < m$	$\Delta T_{Next} = \Delta T_{Current} + 1\ sec$

Table 1 lists the specific rules for generating ΔT_{Next} given the current value of m. In the experiments described later, m_{min} and m_{max} are set to 0.9 and 1.1 respectively. An additional constraint can be used to limit the range of possible values for ΔT_{Next}. For example, in this thesis, the sampling interval is restricted to be between 1 and 10 seconds regardless of the output of the LP logic. The lower bound is applied to limit the sampling to a rate the manager and agents can realistically service while the upper bound ensures there is some minimum set of samples on which to base future predictions.

Fuzzy Logic Controller (FLC) Method

Rather than attempting to predict the exact values of future samples, fuzzy logic applies "rules of thumb" to the observed behavior to adjust the future sampling rate in a reasonable fashion. Fuzzy logic has shown promising results in non-linear systems, especially those that are difficult to model or where an exact model is impossible. The

FLC mimics human reasoning by applying a set of rules based on one or more premises and a single implication. For example, if the premises "network load is very high" and "the sampling interval is somewhat low" are found to be true, the implication "reduce the sampling interval by a medium amount" might be taken. In this section, an FLC is used to adaptively control the sampling rate of the network monitor.

The proposed method for adaptive sampling using fuzzy logic is shown in Figure 3. The inputs of the FLC are the current sampling interval, $\Delta T_{Current}$, and the difference between the last two sample values, ΔX. The FLC output, F_{out}, corresponds to the amount of time to increment or decrement the current sampling interval, $\Delta T_{Current}$. This adjustment yields a new value for the next sampling interval, ΔT_{Next}. The following subsections describe the FLC, including the definition of the membership functions, the fuzzy rules, and the defuzzification process.

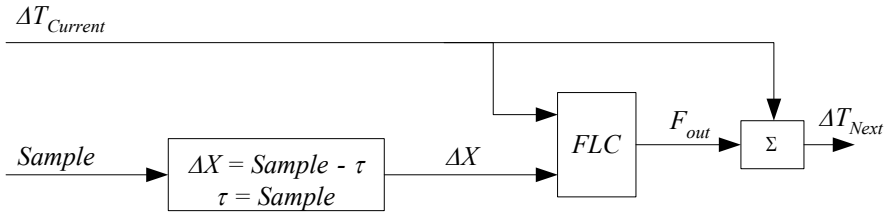
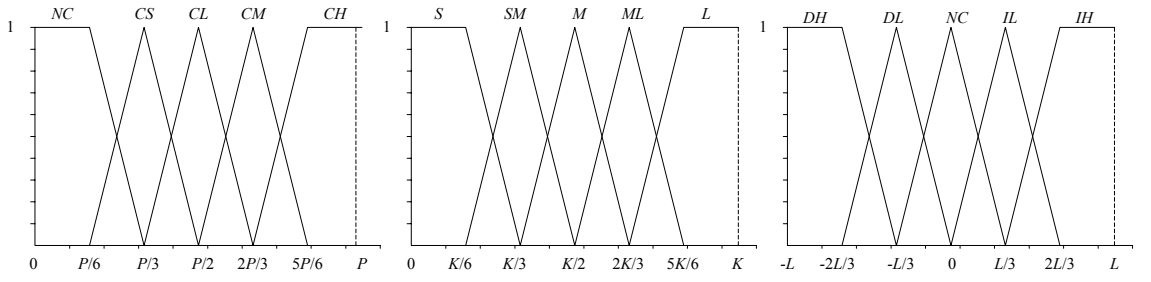


Figure 10. Adaptive sampling based on fuzzy logic.

FLC Membership Functions

Unlike traditional digital systems, fuzzy logic differentiates between several levels such as “high,” “medium,” and “low” when making decisions. The numerical ranges assigned to each of the levels are defined in a membership function. The FLC makes use of a set of membership functions such as those illustrated in Figure 11 to determine a suitable output value given the state of the input values. In this case, there are two membership functions for the inputs and one for the output. For the two input membership functions, ΔX and $\Delta T_{Current}$, the horizontal axis corresponds to the value of the input. A given value of the input is interpreted as being in one or more fuzzy states. For the ΔX input, the fuzzy states are *No-Change (NC), Change-Slight (CS), Change-Low (CL), Change-Medium (CM), and Change-High (CH)*. The $\Delta T_{Current}$ input falls into the categories of *Small (S), Small-Medium (SM), Medium (M), Medium-Large (ML),* and *Large (L)*.

The output of the FLC is also defined in terms of fuzzy variables. In Figure 11, the output of the FLC, F_{out}, falls into one of five fuzzy states: *Decrease-High (DH), Decrease-Low (DL), No-Change (NC), Increase-Low (IL),* and *Increase-High (IH).*



(a) ΔX (b) $\Delta T_{Current}$ (c) F_{out}

Figure 11. Membership functions used by the FLC.

The membership function for input ΔX describes the amount of change in measured throughput (in terms of SNMP byte count) between successive samples and, as shown in Figure 4, ranges from 0 to P bytes. Similarly, the membership function for input $\Delta T_{Current}$ denotes the current sampling interval and ranges from 0 to K seconds. Finally, the membership function for output F_{out} indicates the amount of increase or decrease to be applied to the sampling interval and ranges from -L to +L seconds.

Table 2. Parameter values selected for the membership functions in the FLC.

Type of Traffic	P (bytes)	K (sec)	L (sec)
Internet	$0.4 \times S_{Internet}$	12	2.0
Videoconference	$4.0 \times S_{video}$	12	0.5

The appropriate selection of the membership parameters P, K, and L requires some understanding of the traffic behavior. When determining the values for these FLC parameters, it is desirable to select values that lead to a relatively uniform spread in frequency of occurrence of inputs across the states of the membership function. For instance, with parameter P, the goal is select the value that distributes the input data in a relatively uniform fashion across all five of the stages, from *NC* to *CH*. Based on several tuning experiments with the Internet and videoconference traffic models described in the next section, the values selected for this controller were determined as shown in Table 2. In this table, the variables $S_{Internet}$ and S_{video} refer to the peak throughputs (i.e. the maximum change in SNMP byte-count measured over a span of

one second) in the Internet and videoconference models, respectively. These values are multiplied by a window of time to produce the desired value for parameter *P*. For Internet traffic, a small window is selected to allow the FLC to respond to rapid changes in network behavior. The videoconference traffic employs a larger window since the periodic nature of the traffic requires less frequent adjustments in sample rate. Although beyond the scope of this thesis, the use of an FLC in a network with arbitrary traffic would require a training period during which the optimal spread of these three parameters across the fuzzy states in the membership functions could be ascertained.

FLC Rules

In addition to the membership functions for the inputs, the FLC needs a fuzzy set of rules to map the input values to an output response. The twenty-five statements shown in Table 3 represent a proposed set of fuzzy rules for the FLC. Each row in this table provides two *premises* along with a single *implication*. In the table, the first and second columns are correlated using the logic operator AND, which is analogous to the intersection in set theory. For example, the fuzzy logic expression shown in the first row of the Table 3 1ndicates that given no measured changes in the input, ΔX, and a small sampling interval, $\Delta T_{Current}$, then F_{out} should cause the sampling interval to be increased by a high amount.

Table 3. Rules for the fuzzy controller.

Rule	ΔX	$\Delta T_{Current}$	F_{out}
1	*No-Change (NC)*	*Small (S)*	*Increase-High (IH)*
2	*No-Change (NC)*	*Small-Medium (SM)*	*Increase-High (IH)*
3	*No-Change (NC)*	*Medium (M)*	*Increase-Low (IL)*
4	*No-Change (NC)*	*Medium-Large (ML)*	*Increase-Low (IL)*
5	*No-Change (NC)*	*Large (L)*	*No-Change (NC)*
6	*Change-Slight (CS)*	*Small (S)*	*Increase-High (IH)*
7	*Change-Slight (CS)*	*Small-Medium (SM)*	*Increase-Low (IL)*
8	*Change-Slight (CS)*	*Medium (M)*	*No-Change (NC)*
9	*Change-Slight (CS)*	*Medium-Large (ML)*	*Decrease-Low (DL)*
10	*Change-Slight (CS)*	*Large (L)*	*Decrease-Low (DL)*
11	*Change-Low (CL)*	*Small (S)*	*Increase-Low (IL)*
12	*Change-Low (CL)*	*Small-Medium (SM)*	*No-Change (NC)*
13	*Change-Low (CL)*	*Medium (M)*	*Decrease-Low (DL)*
14	*Change-Low (CL)*	*Medium-Large (ML)*	*Decrease-High (DH)*
15	*Change-Low (CL)*	*Large (L)*	*No-Change (NC)*
16	*Change-Medium (CM)*	*Small (S)*	*Decrease-Low (DL)*
17	*Change-Medium (CM)*	*Small-Medium (SM)*	*Decrease-Low (DL)*
18	*Change-Medium (CM)*	*Medium (M)*	*Decrease-High (DH)*
19	*Change-Medium (CM)*	*Medium-Large (ML)*	*Decrease-High (DH)*
20	*Change-Medium (CM)*	*Large (L)*	*Decrease-High (DH)*
21	*Change-High (CH)*	*Small (S)*	*Decrease-Low (DL)*
22	*Change-High (CH)*	*Small-Medium (SM)*	*Decrease-High (DH)*
23	*Change-High (CH)*	*Medium (M)*	*Decrease-High (DH)*
24	*Change-High (CH)*	*Medium-Large (ML)*	*Decrease-High (DH)*
25	*Change-High (CH)*	*Large (L)*	*Decrease-High (DH)*

The "fuzziness" of the FLC stems from the fact that a given input value may correspond to multiple fuzzy states. For example, referring to Figure 11(a), if ΔX has a value between $P/3$ and $P/2$ then it would be a partial member in the *CS* state and a partial member in the *CL* state. Such fuzziness makes it hard to determine a fixed expression for the output value from Table 3. In order to find the non-fuzzy output, the defuzzification process takes place. This process is composed of an inference method that makes use of the rules and membership functions to produce the control output. The correlation-product method is used here for defuzzification with the FLC [3].

FLC Defuzzification Process

Consider the example in Figure 12 using the correlation-product method for defuzzification. The figure assumes the input values of ΔX_0 and ΔT_0 for ΔX and $\Delta T_{Current}$, respectively. The membership functions indicate two degrees of membership for each input: a value of α in the *CS* state and β in the *NC* state for the input ΔX_0, and a value of δ in the *ML* state and ε in the *M* state for T_0. There are three steps in the defuzzification process using the correlation product method. The first step is to find the set of all rules that correspond to the input membership states by considering all permutations of the states. In this example, the rules 3, 4, 8, and 9 from Table 3 apply. The second step is to choose the minimum value of the degree of membership of the inputs among the matching rules. For instance, rule 3 involves the fuzzy states *NC* and *M* in the membership functions ΔX and $\Delta T_{Current}$, respectively. The degree of membership for ΔX_0 is β and for ΔT_0 is ε. Since $\varepsilon < \beta$, then ε is the minimum input value for rule 3. The third step scales the shape of the membership function for the F_{out} value of each rule by the minimum input value determined in the previous step. In the example below, rule 3 defines the *IL* state for F_{out}, so the *IL* shape is scaled down by ε. This process is repeated for all the rule graphs and the resulting scaled shapes are combined. The center of mass of the combined graph is calculated using Eq. (4) [3,17]. The resulting outcome, *z,* corresponds to the numerical value used for F_{out}.

$$z = \frac{\sum_{j=1}^{n} U_j \times V_j}{\sum_{j=1}^{n} V_j} \tag{4}$$

For each of the *n* scaled shapes in the combined graph, the center of mass equation uses U_j and V_j to represent the values for the peak of the shape on the x-axis and y-axis, respectively. For example, Figure 12 shows the combination of the scaled rule graphs where the U_j values are $L/3$, $L/3$, 0, and $-L/3$, and the V_j values are ε, β, ε, and α, from rules 3, 4, 8, and 9, respectively. The result of this equation represents the numerical (i.e. non-fuzzy) output for membership function F_{out}. Future sampling queries are rescheduled based upon this scalar result as shown in Figure 10.

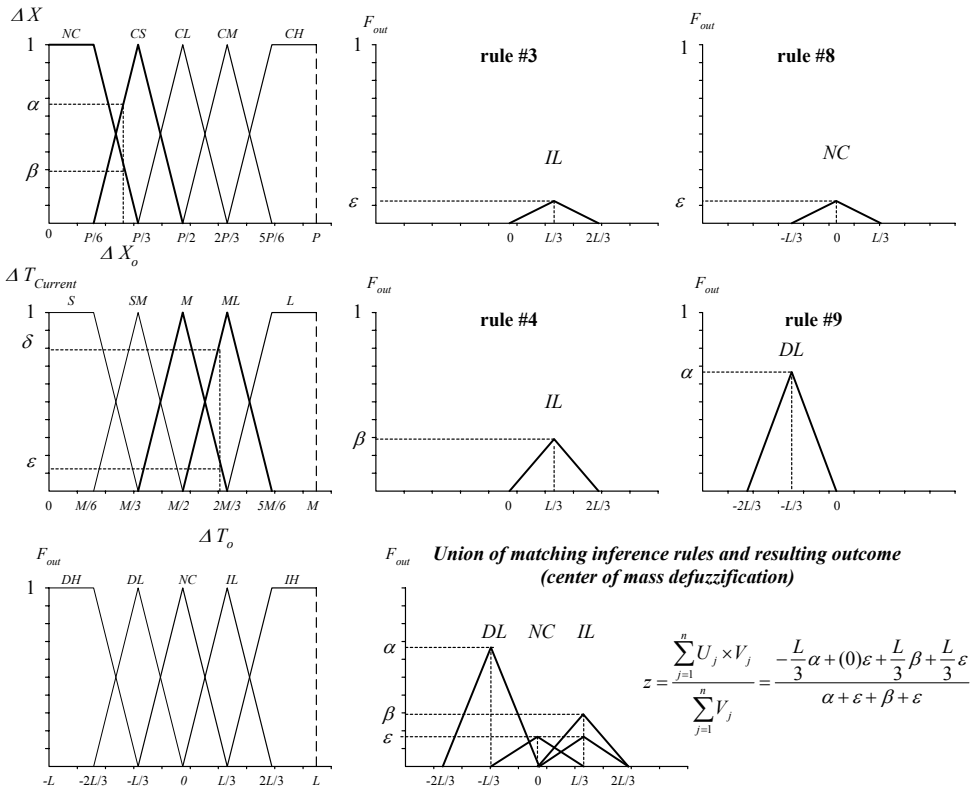


Figure 12. Correlation-product inference method and defuzzification process.

CHAPTER 4
EXPERIMENTS AND MEASUREMENTS TECHNIQUES

This section describes a set of simulative experiments to compare the adaptive and systematic sampling techniques for network management. The computing testbed that generated the traffic in the simulation is described in this section, as well as the metric used to compare the sampled data sets.

Traffic Models and Simulation

Two traffic traces of network activity are used as data models for the experiments. The two were selected to represent the opposite ends of the spectrum of potential traffic that might be encountered in a high-performance network, one characterized by bursty, aperiodic traffic and the other by streaming, periodic traffic.

The first trace was taken from the outgoing 100 Mb/s Fast Ethernet port on a Fore Systems PowerHub 7000 router providing Internet connectivity to a laboratory with dozens of computers. This *Internet* trace is representative of random, bursty network data. This type of data should be ideally suited to adaptive sampling because the sample rate can be adjusted to compensate for the level of loading on the network at any given time. Thus, the goal with adaptive sampling for this type of traffic is to reduce the sample count while maintaining the accuracy achieved by the equivalent

systematic sampler. Or, conversely, the goal is to increase the accuracy while maintaining the same sample count as the equivalent systematic sampler.

A second trace was captured from a port of a Fore Systems ASX-200BX ATM switch that was connected directly to a Fore Systems AVA-300 ATM audio/video decoder. This connection hosts a continuously running videoconference feed transferring MJPEG video and uncompressed audio at 30 frames/second over a 155 Mb/s ATM link. The resulting *videoconference* trace is representative of a highly periodic pattern of network traffic. The periodic nature of this data may not be better suited to adaptive sampling techniques than the systematic sampler. However, the adaptive sampling algorithm can dynamically select a sampling interval that accurately measures the data. By contrast, a systematic technique requires the proper interval to be correctly selected *a priori* by the network administrator, which is not easily realized.

Each trace was taken by measuring the cumulative number of bytes received over a two-hour period using SNMP queries. The resolution of the measurements was 0.1 seconds, yielding a total of 72,000 samples. The adaptive and systematic sampling methods were then simulated and evaluated by resampling the data in each trace. For example, a systematic sampling with $T_s = 1$ sec would select every tenth element from the original data, yielding a new trace with 7,200 samples. During the simulation, all sampling intervals were rounded to the nearest 0.1 seconds so that they correspond to a single measurement from the original trace.

For simplicity, the throughput measurements gathered by the systematic and adaptive samplers with these traces are all normalized to the peak value stored in the baseline data model (i.e. either Internet or videoconference) used in the experiment.

Metric for Comparison

In order to compare the performance of the adaptive sampling techniques with the systematic baseline, a measure of accuracy is needed. The mean-squared error metric for comparing two *N*-sample sets was used as shown below in Eq. (5) [2]. This expression makes a point-by-point comparison between the reference and sampled signals using the normalized magnitude of the instantaneous throughput. The reference signal, $f_R(n)$, is a systematic sampling of the original trace of Internet or videoconference data with a period of $T_s = 1$ sec. The comparison signal, $f_C(n)$, is the output from one of the adaptive or systematic sampling methods applied to the original trace. Since Eq. (5) requires sets with an equal number of samples, both the reference and the comparison traces are resampled to 72,000 samples. Linear interpolation is used to produce the added points.

$$Error = \sum_{n=1}^{N} (f_R(n) - f_C(n))^2 \tag{5}$$

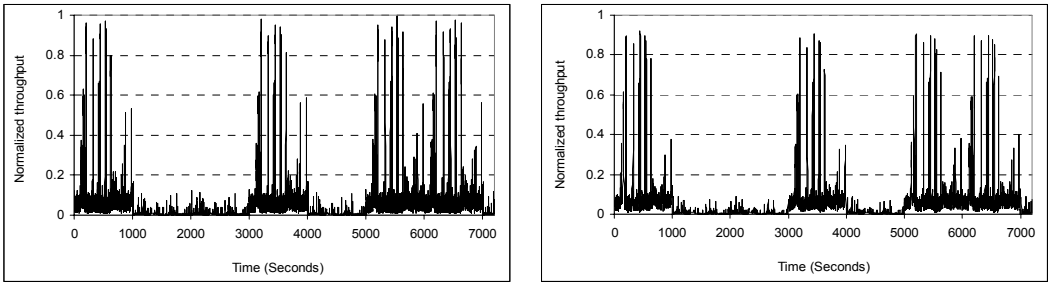
CHAPTER 5
RESULTS AND ANALYSIS

This section presents results and analyses from experiments conducted to ascertain the performance of the systematic and adaptive sampling algorithms as applied to the Internet and videoconference traces. In the next two subsections, a qualitative comparison of the signals produced by several systematic and adaptive sampling measurements is provided. Then, a quantitative assessment of the adaptive sampling techniques is conducted by comparing the adaptive techniques to an equivalent systematic baseline in terms of sample count and mean-squared error relative to the reference trace.

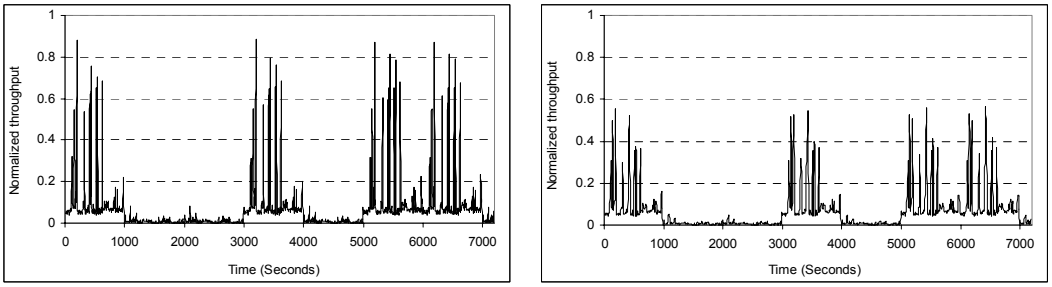
Systematic Sampling Measurements

In this section, the effect of sampling rate is qualitatively illustrated using traditional, systematic sampling techniques.

Figure 13 and Figure 14 show the normalized throughput sampled using fixed sampling intervals (T_s) from one to ten seconds on the Internet and videoconference traces, respectively. The shapes for Internet and videoconference traffic at T_s = 1 sec shown in Figure 13(a) and Figure 14(a) are used as the reference value in the comparisons of the different sampling disciplines.

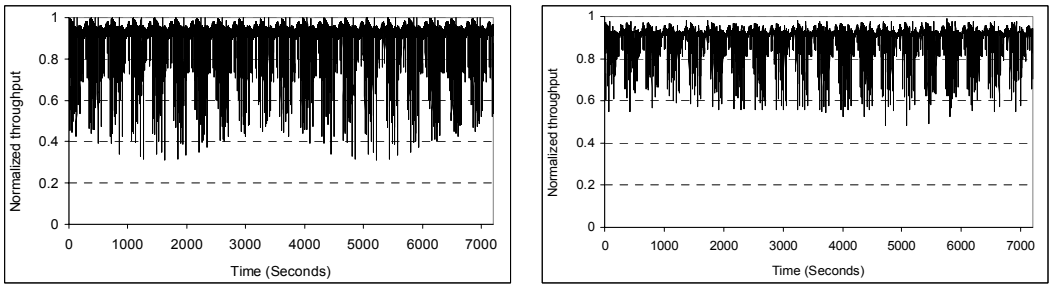


(a) T_s = 1 sec (7200 samples) (b) T_s = 2 sec (3600 samples)

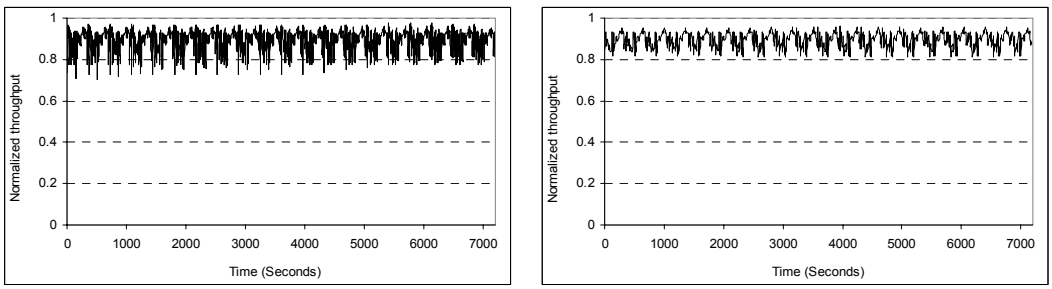


(c) T_s = 5 sec (1440 samples) (d) T_s = 10 sec (720 samples)

Figure 13. Measured traffic patterns of Internet data using systematic sampling



(a) T_s = 1 sec (7200 samples) (b) T_s = 2 sec (3600 samples)



(c) T_s = 5 sec (1440 samples) (d) T_s = 10 sec (720 samples)

Figure 14. Measured traffic patterns of videoconference data using systematic sampling.

Both Figures demonstrate a "smoothing" effect at larger sampling intervals. This effect results from the fact that the samples actually measure a cumulative SNMP byte-count rather than a throughput. Therefore, at higher values of T_s, the measured byte-count is averaged over a longer interval of time. The result is that large throughput spikes are averaged with periods of low activity. This effect both reduces the magnitude of the throughput spikes as well as filters much of the variation from the actual traffic pattern.

The measurements conducted at T_s = 10 sec for the Internet data in Figure 13 indicate a maximum throughput of only about 55% of the peak value at the reference. Figure 14 illustrates a similar situation with videoconference traffic. For example, in the trace sampled at T_s = 5, the highly fluctuating behavior found at the reference is completely lost. In fact, the variation observed in Figure 14(c) is only approximately 25%, whereas the reference indicates a variation of 50-70% of the maximum peak throughput.

Since high-performance management services require precise throughput estimations, an inappropriate selection of the sampling time would lead to inaccurate observations. The following section will show how adaptive sampling can be used to maintain accurate measurements for Internet and videoconference traffic.

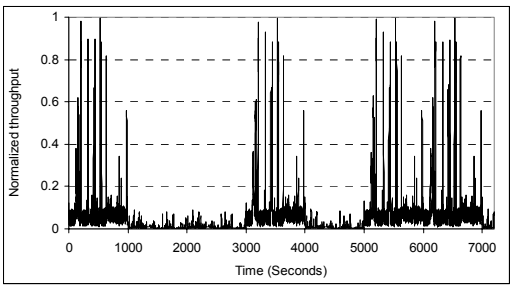
Adaptive Sampling Measurements

The results of applying the adaptive sampling techniques to the Internet trace using second-, third-, and fourth-order LP samplers and the FLC sampler are shown in Figure 15. The second-order LP sampler was able to reduce the sample count to about

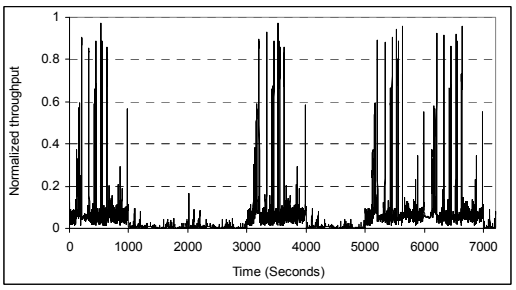
51% of the reference with almost no discernable change in the peak magnitude or variation of throughput behavior. As the order of the LP sampler is increased, the number of samples are further reduced with only a small visible loss in peak magnitude and variation. The FLC approach yields the fewest number of samples at 29% of the reference. The FLC shows more variation than the higher-order LP samplers but with marginally lower peaks.

Figure 16 shows the results of applying the adaptive sampling techniques to the periodic videoconference trace. The LP techniques reduce the number of samples to approximately 65% of the reference but at a slight reduction in signal variation. The FLC achieves shows slightly less variation at 43% of the reference sample count. However, the accuracy of the adaptive sampling techniques on the videoconference data is not nearly as good as it is with the Internet data.

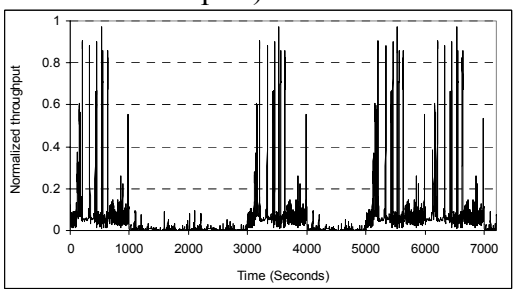
Although a qualitative comparison of the adaptive sampling techniques shows the ability of the adaptive sampling techniques to reduce sample count while retaining a certain degree of accuracy, an accurate picture of the performance is difficult to ascertain. For example, a simple systematic sampler with $T_s = 2$ sec is also able to reduce the number of samples by 50% while still retaining much of the characteristics in the reference signal. Therefore, it is necessary to quantify the accuracy of the adaptive and systematic sampling techniques.



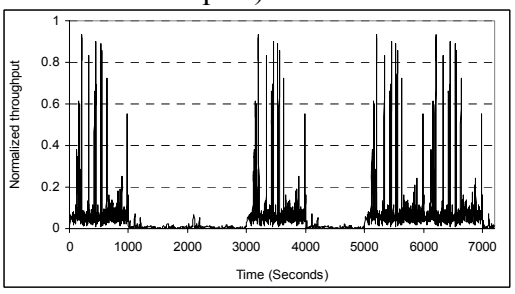
(a) Second-order LP (3728 samples)



(b) Third-order LP (3028 samples)

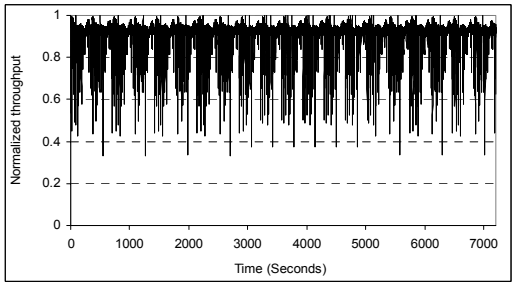


(c) Fourth-order LP (2641 samples)

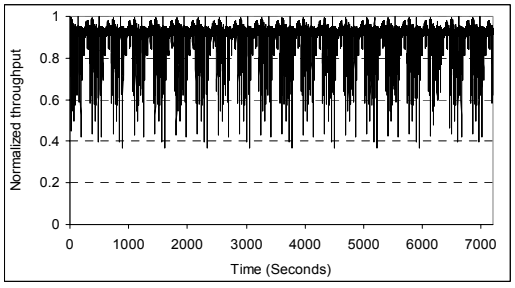


(d) FLC (2095 samples)

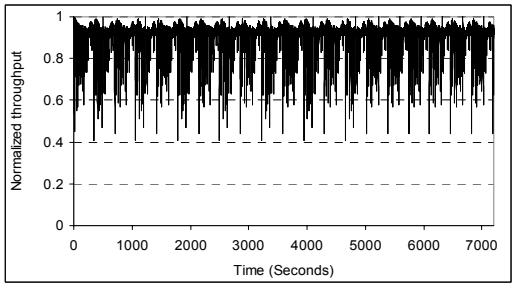
Figure 15. Measured traffic patterns of Internet data using adaptive sampling.



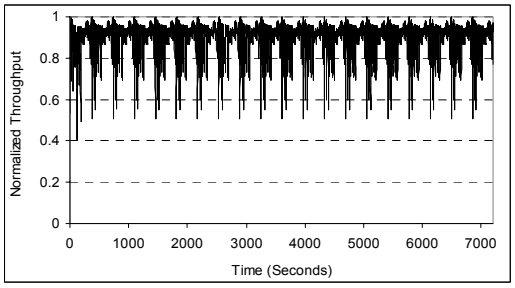
(a) Second-order LP (4769 samples)



(b) Third-order LP (4651 samples)



(c) Fourth-order LP (4621 samples)



(d) FLC (3095 samples)

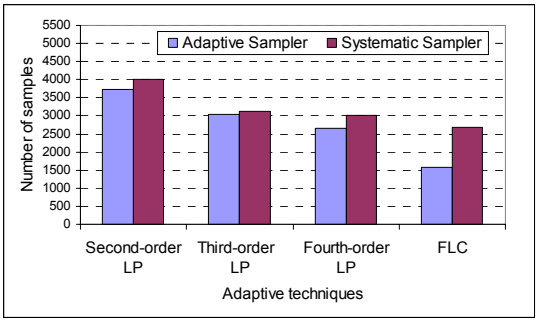
Figure 16. Measured traffic patterns of videoconference data using adaptive sampling

Quantitative Analysis of Adaptive Samplers

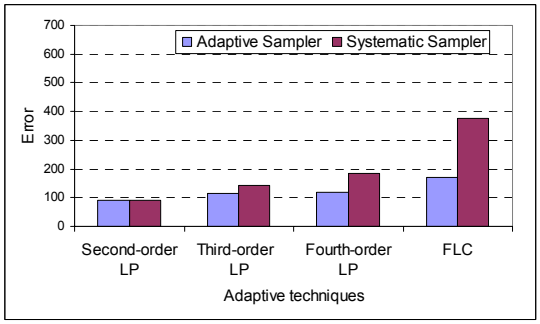
It is difficult to quantitatively compare the performance of the adaptive sampling techniques to an equivalent systematic approach when the measurements differ in both the number of samples and the error relative to the reference. In general, traces with more samples show a lower error. Therefore, two forms of experiments were conducted to compare the adaptive and systematic sampling methods, first with a fixed number of samples and then with a fixed error. In both cases, a single run of the adaptive sampling techniques was used. As demonstrated in the previous section, each technique yielded a different number of samples with a different relative error when compared to the reference. In the fixed sample-count comparison, a systematic sampling interval was selected for each adaptive sampling result such that it would contain the same number of samples. The relative error from each adaptive sampler

was then calculated and compared to its equivalent systematic sampler. For the constant-error comparison, a chart of sample count versus relative error was made for a range of systematic sample intervals. Using this chart, the minimum number of systematically distributed samples necessary to achieve a certain relative error was determined.

Figure 17 shows the results of the experiments conducted with Internet traffic. In Figure 17(a), the amount of sampling error is held constant and then the sample count is measured for each of the adaptive samplers and compared to a systematic sampler with the same number of samples.



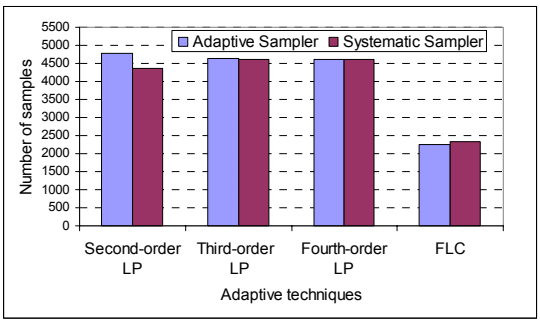
(a) Sample count with constant error



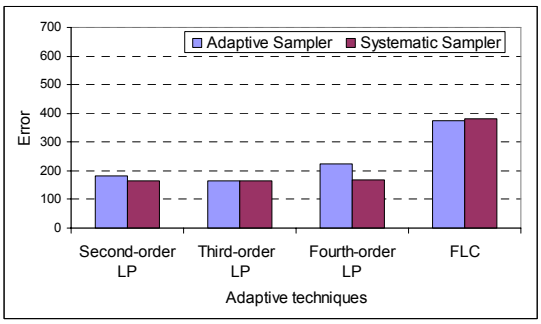
(b) Error with constant sample count

Figure 17. Sample count and error variation with Internet traffic.

In Figure 17(b), the sample count is held constant and the error is measured for each case. As the results indicate, each of the four adaptive samplers outperforms its corresponding systematic sampler. The greatest improvement is seen with the FLC sampler. The FLC sampler achieves nearly twice the performance of its equivalent systematic sampler, which can be interpreted either as decreasing the sample count by a factor of two for a given level of accuracy or as increasing the accuracy by a factor of two for a given number of samples. In contrast, the second-order LP sampler is only marginally better in performance than its systematic counterpart. As previously hypothesized, traffic of a bursty, aperiodic nature is found to be well suited for the capabilities of adaptive sampling, since these samplers are able to dynamically adjust their sampling rate with periods of increased and decreased activity on the network.



(a) Sample count with constant error



(b) Error with constant sample count

Figure 18. Sample count and error variation with videoconference traffic.

Figure 18 shows the results of the same experiments with the videoconference traffic. As before, Figure 18(a) displays the measurements of sample count for a fixed level of accuracy and Figure 18(b) displays the measurements of accuracy for a fixed number of samples. The results in these experiments with videoconference traffic are markedly different from those with the Internet traffic. The LP samplers show slightly higher error or slightly larger sample counts than their systematic counterparts. The FLC sampler achieves slightly lower sample counts and slightly smaller error rates. Given the nature of the traffic being sampled, these results are also promising. When dealing with traffic of a periodic nature, as is the case with the videoconference, systematic sampling is effective provided that the appropriate sampling interval can be identified and employed. Thus, the primary goal of using an adaptive sampler for periodic traffic is to match the performance of the best systematic sampling rate. This performance is achieved and even surpassed in results from the experiments with videoconference data by using the FLC sampler, and the LP samplers show only a marginal degradation in performance. An advantage of adaptive sampling is that this

optimal rate can be dynamically adjusted if the frequency or type of periodic traffic changes.

Taken together, the results of the experiments with Internet and videoconference traffic confirm that adaptive sampling can play an important role in decreasing the load on the network and the network manager, increasing the accuracy of the measurements, or both. While these two traffic models do not of course encompass the entire universe of possible traffic patterns, they are reasonably representative of the boundaries of that universe in terms of bursty, aperiodic traffic versus streaming, periodic traffic. Thus, these results illustrate that adaptive sampling provides the potential for better monitoring, control, and management of high-performance networks.

CHAPTER 6
RELATED WORK

Several researchers have studied sampling techniques in networks. Claffy et al. studied three non-adaptive sampling methodologies: systematic (i.e. periodic), random, and stratified random sampling [4]. Their results indicated that stratified random sampling has better accuracy when the three methods are used to capture the same number of samples. Cozzani and Giordano made use of conventional sampling methods and studied their effects in quality of service measurements for ATM networks [5]. Drobisz proposed a network capture device for Gigabit Ethernet (GE) adapting CPU utilization based on traffic burst anticipation by increasing CPU usage when a bursty period of traffic was expected [18].

Fuzzy and LP-filter adaptations have been used in network applications for congestion, admission, and flow control. Jacobson and Karels studied flow-control protocols based on filters for the congestion control algorithm for the Transport Control Protocol (TCP) [16]. Kalampoukas et al. applied filter-based techniques for window adaptation in TCP [19]. These methods control protocol parameters based upon network traffic behavior. The adaptive sampling techniques presented in this thesis provide a similar approach but in an effort to control the sampling time based upon the monitored traffic information.

Fuzzy-logic controllers have also been studied for ATM networks. Bonde and Ghosh [20] and Catania et al. [15] proposed queue managing and congestion control

based on fuzzy sets. Cheng and Chang suggested a fuzzy architecture for both congestion and call admission control in ATM networks [17], and discovered that congestion control, using fuzzy logic, performed better than the leaky bucket algorithm inherent to ATM. The proposed FLC-based sampling technique uses an adaptation of the methods applied in these studies on control of ATM networks. Givan and Chong applied Markov and Bernoulli processes to model network traffic behavior and to create learning algorithms to predict future traffic measurements in a switch-based application [21]. In contrast, the adaptive techniques presented in this thesis assume minimal knowledge of the traffic model.

CHAPTER 7
CONCLUSIONS AND FUTURE RESEARCH

This thesis has presented two techniques to adaptively monitor network behavior. One approach is based on using linear prediction to dynamically alter the sample rate based on the accuracy of the predictions, where inaccurate predictions indicate a change in the network's behavior and result in a smaller sampling interval. The second approach models the cognitive process of a human network manager by using fuzzy logic. When certain pre-determined conditions are met, such as an increase in network traffic, corresponding actions are taken such as a decrease in sampling interval.

These adaptive techniques are shown to perform well on random, bursty data such as conventional Internet traffic. All approaches are able to reduce the sample count while maintaining the same degree of accuracy as the best systematic sampling interval. Equivalently, all approaches are able to increase accuracy while maintaining the same sample count. The higher-order LP samplers perform better than the lower-order ones, while the FLC sampler shows the greatest reduction in sample count. The reduced sample count is an important factor for network management in high-performance networks. Accurate measurements are required to find bottlenecks while, at the same time, the impact on the network must be minimized to allow applications to take advantage of the low latency and high throughputs such networks provide.

For periodic data such as that found in a multimedia or videoconference environment, the adaptive sampling approaches perform comparably to the best systematic approach in terms of accuracy and sample count. However, the adaptive techniques are able to adjust to changing traffic loads and therefore would outperform any static sampling interval in a dynamic environment. For periodic traffic, the LP samplers are marginally out performed by systematic sampling while FLC approach shows a slight improvement over systematic sampling.

In general, the evidence suggests that the fuzzy-logic adaptive technique provides more flexibility and better performance than the LP methods. The main disadvantage of the FLC is the selection of the boundaries of the membership functions. Future work is needed to study methods to optimally tune the parameters of the adaptive samplers. One possible area of concentration is the development of an autonomous adaptive manager using fuzzy logic that dynamically adapts the parameters for its membership functions while the monitoring process is underway. In addition, other approaches for the design of adaptive samplers and management systems are worthy of investigation, such as samplers based on neural-network or neuro-fuzzy controllers.

APPENDIX A
SIMULATION OF AN ADAPTIVE NETWORK MANAGEMENT STATION

The simulation program presented in this section requires GNU C/C++ version 2.7.2. It was compiled in both Linux Red Hat 6.0 and Solaris 2.5. The files required for the simulator are *Simula_f.cc*, *correlation.h*, and *FLC.h*.

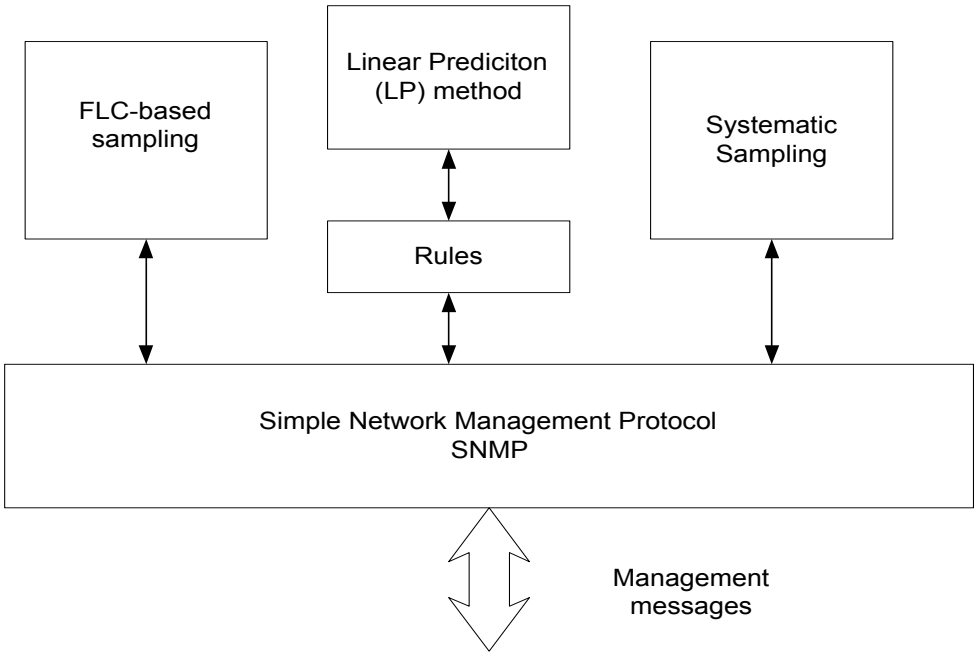


Figure 19. Adaptive network management station

The first program illustrates the simulation routine used to test the adaptive sampling techniques. Figure 19 illustrates the simulated NMS and the interaction with

the network elements. The remainder source codes are the libraries associated with the mean square error and the fuzzy rules.

Simula_f.cc

The simulation program takes an input file of throughputs and generates sampled versions depending upon the sampling technique employed. The program iterates once assuming negligible latency of the monitored element response.

The LP routines are found in the functions *Predict_value(float sample, int order, OID& oid)* and *Predict_Timing(float sample, OID& oid, int order, float j, float Peak)*. These routines apply the rules and linearly predict the expected value for the parameter under study suing the *N* previous elements at the filter. The *OID* is a class representing the object identifier being measured. This class stores a vector of the *N* previous values sampled and the scheduled time for the next sample. The FLC employs the function *Predict_Timing_FLC (float sample, OID& oid, int order, float j, float Peak)* to reschedule the queries based upon the fuzzy rules.

```
// ------------------------------------------------------------
// High-performance Computing and Simulation Research Lab
// University of Florida
// October 25th, 1999
// ------------------------------------------------------------
//
// Edwin Hernandez    hernande@hcs.ufl.edu
// v1.0
// Program created to simulate the behavior of SNMP queries. The
// input file is traffic data. The file should have values
// of rate (bytes/sec) and one value per line.
// ------------------------------------------------------------
#include <stdlib.h>
#include <math.h>

#define MAX_Samples  5
#define MAX_Expected 5
```

```
#define MAX_Parameters 64
#define MAX_Bins 300

#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <fstream.h>
#include <stream.h>

//Libraries added for performance measurements
#include <sys/time.h>
#include <time.h>
#include "FLC.h"         // Fuzzy logic library
#include "correlation.h" // Mean-square error and correlation function

#define T_g    10        // 10 is equivalente to 1 second Granularity
                         // 1 is equivalent to 0.1 seconds

/* getMaxTime():
**
** This function returns the maximum sampling time value used
** for the network managers.
*/
long getMaxTime(){
  return 10*T_g;     // equivalent to 10 seconds
  return ((long) (Max_time*0.1));
}


/* Padding()
**
** Linearly add N samples to the file F
*/
void padding(ofstream &f, float N, float what){
  for (int i =1; i<=(int) N; i++)
     f<<what<<endl;
}

/* Class OID:
**
** This class represents the object identifier to be monitored
** it contains, previous samples, current sampling time, and the
** expected value calculated by the filter.
*/
class OID {
  float timing;
public:
  float etiming;
  float rate[MAX_Samples];
  char* name;
  float Lastvalues[MAX_Samples];
  float LastTiming[MAX_Samples];
  float expectedvalues[MAX_Expected];
  float nextsampleTime;
```

```
  OID();
  ~OID(){;}
  void seteTiming(float t){etiming=t;}
  void setTiming(float t){timing=t;}
  void setTiming(int t){timing=t;}
};

/* OID::OID()
**
** Constructor definition of the class OID
*/
OID::OID(){
  nextsampleTime = T_g;  // equivalent to 1 second
  etiming = T_g;
  timing = T_g;
  name = NULL;
  for (int i=1; i<=MAX_Samples; i++){
        Lastvalues[i]=0;
        LastTiming[i]=1;
        expectedvalues[i]=0;
        rate[i]=0;
  }
}

/* TotalSamples(long arr[])
**
** This function returns the total number of samples in the array
** of bins
*/
long TotalSamples(long Arr[]){
  long  s=0;
  for (int i=0; i<MAX_Bins; i++)
     s = s + Arr[i];
  return s;
}

/* CalculateSpearman(char*, char*)
**
** given two files of equal size, this function calculates the
** correlation of the two files, using Spearman's method of
** correlation
*/
double CalculateSpearman(char* ref, char* comp){
   ifstream r1;
   ifstream r2;
   double a, b,s, s1, s2, s3, s4, s5;
   long n=0;
   s=0; s1=0; s3=0; s2=0; s4=0; s5=0;
   r1.open(ref);
   r2.open(comp);
   r1.seekg(0); r2.seekg(0);
   r1.clear();
   r2.clear();
   while ((!r1.eof()) && (!r2.eof())){
       r1>>a;
       r2>>b;
```

```
        s=a*b+s;
        s1=a*a+s1;
        s2=b*b+s2;
        s3=a+s3;
        s4=b+s4;
        n=n+1;
    }
    r1.close();
    r2.close();
    return (1-(s-1/n*s3*s4)/sqrt((s1-1/n*s3*s3)*(s2-1/n*s4*s4)));
}


/* CalculatePhi(long*, long*)
**
** This function calculates the phi-coefficient between
** two arrays of bins. It is used to compare statistical
** distributions
*/
float CalculatePhi(long refArr[], long ComArr[]){
  float chi_sqrt=0;
  long cumRef[MAX_Bins];
  long cumComArr[MAX_Bins];
  float sumall=0;
  int j;

  for (int i=0; i<MAX_Bins; i++){
    cumRef[i]=0; cumComArr[i]=0;
    for (j=0; j<=i;j++){
       cumRef[i]=refArr[j]+cumRef[i];
       cumComArr[i]=ComArr[j]+cumComArr[i];
    }
  }
  float tref=TotalSamples(refArr);
  float tcom=TotalSamples(ComArr);
  for (int i=0; i<MAX_Bins;i++){
    if (refArr[i]>=5) {
     chi_sqrt=(refArr[i]-ComArr[i])*(refArr[i]-\
                          ComArr[i])/refArr[i]+chi_sqrt;
   }
      sumall=(float)(refArr[i]+ComArr[i])+sumall;
  }
  if (sumall>0)
     return sqrt(chi_sqrt/sumall);
  return 0;
}

/* Predict_Value(float, int, OID&)
**
** Linearly predicts a expected value for a COUNTER data
** type in SNMP based management. By defining the sequence order
** we indicate how many previous samples are used to predict the
** future or expected value
*/
float Predict_Value(float sample, int Order, OID &oid){
   float eValue=0;
```

```
   float alpha=0.9;
   for (int i=Order-1; i>0; i--){
       eValue = (oid.Lastvalues[i+1]-
oid.Lastvalues[i])/(oid.LastTiming[i+1] - oid.LastTiming[i])+eValue;
    }
   eValue = eValue*(j/T_g-oid.LastTiming[Order]);
   for (int i=1; i<Order;i++){
       oid.Lastvalues[i]=oid.Lastvalues[i+1];
       oid.LastTiming[i]=oid.LastTiming[i+1];
   }
   oid.Lastvalues[Order]=sample;
   oid.LastTiming[Order]=j/T_g;
   if (Order>1) {
      eValue = sample+eValue/(Order-1);
   } else eValue = sample*(1-alpha)+oid.Lastvalues[1]*alpha;
    // Low Pass filter just Van Jacobson for order-1.
    // This can be useful for NON-COUNTER data types
   oid.expectedvalues[1] =  eValue;
   return(eValue);
}

/* Predict_Timing(float, OID&, int, float, float)
**
** Based on some rules and the rate "m" involving the error
** between expected and current sample. The sampling time is
** modified to be increased or decreased.
**
*/
float Predict_Timing(float sample, OID &oid, int order, float j, float
Peak ){
    float timing=oid.etiming;
    float m;
    float eValue = oid.expectedvalues[1];
    if (sample-oid.Lastvalues[order]!=0) {
    if (sample != oid.Lastvalues[order]) {
        m = (eValue-oid.Lastvalues[order])/(sample-\
                                oid.Lastvalues[order]);
    } else m==200;
    if ((m>1.10)) {
        timing = timing + T_g;
        if (timing > getMaxTime())
             timing = getMaxTime();
      }
      if ((m<=1.10) && (m>=0.90))
         timing = timing;  // NO CHANGE

      if ((m<0.90) && (m>0)) {
        timing = timing*m;
        if (timing<T_g) timing = T_g;
      }
      if (m<0)  timing=T_g;
     } else
     {  if (sample==0)
          timing=timing*2;
        else
          timing=timing+T_g;
```

```
    }
  oid.nextsampleTime=(int)(timing+j);  // Re-schedules the next sample
  Predict_Value((float) sample, order, oid); // makes prediction
  oid.seteTiming((int) timing);        // Stores current sampling time
  return timing;
};

/* Predict_Timing_FLC(float sample, OID&, int)
**
** Based on the sample, and the rate of change of the values
** measures. The FLC determines the amount of seconds to increase
** or decrease the sampling time. See FLC.h file for more details
** on implementation of the FLC.
*/
float Predict_Timing_FLC(float sample, OID &oid, int j){
  float Itiming;
  float timing =oid.etiming;
  Itiming = GetTimming((float)(sample-oid.Lastvalues[1]), timing);
  timing = timing + Itiming;
  if (timing<T_g) timing = T_g;
  if (timing>getMaxTime()) timing=getMaxTime();
  oid.nextsampleTime=(int) (timing+j);  // Re-schedules sample
  oid.seteTiming((int) timing);
  oid.Lastvalues[1]=sample;             // Keeps previous values
  return timing;
};

/* getMAX_peak(char*)
**
** Based on the granularity chosen for the simulation
** this function determines the maximum peak value possible
** in the file of SNMP-simulated data
*/
float getMAX_peak(char* name){
 ifstream f;
 f.open(name);
 int j=0;
 float m, m1, lastV;
 m=0;
 f>>lastV;
 cout <<" Getting peak value from :"<< name << endl;
 while (!f.eof()){
    f>>m1;
    j++;
    if (j%T_g==0) {
        if ((m1-lastV)>m)
             m=m1-lastV;
        lastV=m1;
    }
 }
 if ((m1-lastV)>m)
             m=m1-lastV;
 f.close();
 return m;
}
```

```
/* interpolate(float, float, float, float)
**
** This procedure executes linear interpolation between two
** points on the graph
*/
void interpolate(float dt, float last, float current, float
interpolated[]){
   float m=(current - last)/dt;
   float b=last;
   for (int i=1; i<=(int)(dt-2);i++){
      interpolated[i]=m*i+b;
   }
}


/* OptimizeFLC(char*, long*, long*, float)
**
** This procedure, determines the behavior of the FLC by
** modifying its membership functions used to control
** the sampling time. It varies Delta_V and Output, and
** stores in a resulting file, the behavior and performance
** of the adaptive sampling algorithms using different
** combinations. We can latter plot the results and determine
** the optimal behavior for the FLC, given an input pattern.
*/
void OptimizeFLC(char* name, long Bin_ref[], long Bin_flc[], float
Peak){
  ifstream iFile;
  ofstream flc_opt, fout, raw;
  int i, k;
  float avg=0; float stdev=0;
  float j, L; long t; float square;
  float delta_t;
  flc_opt.open("FLC_optimization.log");
  float readSNMP, lastreadSNMP;
  long padded=0;
  OID oid_flc;
  L=5;
  while (L>0){
     padded = 0;
     Output[0]=-L*T_g; Output[1]=-2*L*T_g/3; Output[2]=-L*T_g/3;
     Output[3]=0;
     Output[4]=L*T_g/3; Output[5]=2*L*T_g/3; Output[6]=L*T_g;
     L=L-1;
     avg =0; stdev =0;
     j=Peak/2;
     while (j>=Peak*0.1){
       for (i=0;i<MAX_Bins;i++)
         Bin_flc[i]=0;
      for (i=0; i<N_ELEM; i++) DeltaV[i]=j/6*i;
      j=j-Peak*0.1;
      iFile.open(name);
      iFile.seekg(0);
      iFile.clear();
```

```
 oid_flc.nextsampleTime = T_g;
 iFile>>oid_flc.Lastvalues[1];
 lastreadSNMP=oid_flc.Lastvalues[1];  t=0;
 fout.open("flc_data");
 raw.open("rawdata.txt");
 while ((!iFile.eof())&& (t<T_g*7200)){
     t=t+1;
     iFile>>readSNMP;
     if (readSNMP<0) readSNMP = 0;
     if (t%T_g==0){
       raw <<t/10<<"\t  "<<(readSNMP - lastreadSNMP)/Peak<<endl;
       lastreadSNMP=readSNMP;
     }
     if (oid_flc.nextsampleTime == t) {
        delta_t= oid_flc.etiming/T_g;
      k=(int)((readSNMP-\
           oid_flc.Lastvalues[1])/(delta_t*Peak)*MAX_Bins);
      fout<<(float) t/10<<"\t "<<(readSNMP-\
          oid_flc.Lastvalues[1])/(Peak*delta_t)<<endl;
      square = square + pow((readSNMP-\
         oid_flc.Lastvalues[1])/delta_t,2);
      if (k>=MAX_Bins) k=MAX_Bins-1;
      if (k>=0) Bin_flc[k]=Bin_flc[k]+1; //+(long)delta_t;
      Predict_Timing_FLC(readSNMP, oid_flc,t);
    }
 }
fout.close();
iFile.close();
raw.close();
cout <<" Trying from -"<<L<<" to "<<L<<endl;
cout <<"Phi-coefficient \
           FLC:\t"<<CalculatePhi(Bin_ref,Bin_flc)<<endl;
cout <<"Samples FLC   :\t "<<TotalSamples(Bin_flc)<<endl;
flc_opt<<" Average   =\t "<<avg/TotalSamples(Bin_flc)<<"\t";
flc_opt<<" VARIANCE  = \t"<<(square-\
      1/TotalSamples(Bin_flc)*pow(avg,2))/\
               (TotalSamples(Bin_flc)-1)<< endl;
cout <<"out of :\t"<<TotalSamples(Bin_ref)<<"\t Padded \
      \t"<<padded<<endl;
flc_opt<<" from -"<<L<<" to "<<L<<endl;
flc_opt <<"MAX\t "<<j<<"\t"<<L<<"\tPhi-coefficient \
      FLC:\t"<<CalculatePhi(Bin_ref,Bin_flc)<<"\t";
flc_opt <<" "<<TotalSamples(Bin_flc)<<"\t  padded \t " << \
      padded<<endl;
flc_opt <<"out of :\t"<<TotalSamples(Bin_ref)<<" \t \
      "<<CalculateSpearman("rawdata.txt","flc_traffic") << endl;
cout<<" r "<<CalculateSpearman("rawdata.txt", \
      "flc_traffic")<<endl;
cout <<"MAX="<<j<<" and L="<<L<<endl;
// Used to determine the square-mean error of the data
// with the interpolated file.
CorrIndex Index;
Index.fileCorrelate("flc_data", "i-flc_data");
Index.fileCorrelate("rawdata.txt","i-rawdata.txt");
Index.simpleCorrelation("i-rawdata.txt", "i-flc_data");
}
```

```
  }
}

/* Convert(char*, char*)
**
** Convert Throughput file and converts it to a
** Agent-like or SNMP captured data
*/
void convert(char* input, char* output){
  float f, g;
  f=0;
  g=0;
  long k=0;
  ifstream fin(input);
  ofstream fout(output);
  while ((!fin.eof()) && (k<=T_g*7200*10)){
    k++;
    fin >> g;
    if (g<0) g=0;
    f=f+g;
    fout << (long long)f << endl;
  }
  fin.close();
  fout.close();
}


/*
** MAIN program
**
*/
int main(int argc, char** argv){
   ifstream iFile;
   long t=0, po2, po3, po4, pflc;
   int k;
   long Bin_flc[MAX_Bins], Bin_o2[MAX_Bins], Bin_ref[MAX_Bins];
   long Bin_o3[MAX_Bins], Bin_o4[MAX_Bins];
   float rates[20];
   float readSNMP, lastreadSNMP, lrate,  dt, l150;
   OID oid_2, oid_3, oid_4, oid_flc;
   ofstream f_o2("monitor_2"), f_o3("monitor_3");
   ofstream f_o4("monitor_4"), f_flc("flc_data");
   ofstream fsys("monitor_ts_1");
   ofstream raw("rawdata.txt");
   ofstream ts_3("ts_3");
   if (argc<2) {
       cout << " Usage : \n";
       cout << argv[0] <<"[Stimulation File]";
       cout << " example : "<<argv[0]<<" Hurst0.5 \n";
       cout << " Note: "<< endl;
       cout << " Assume each point in the file is separated by 1/10s";
       cout << " \n outputs: " ;
       cout << "          monitor_i, order-i filter \n";
       cout << "          monitor_flc, FLC controller\n";
       cout << "  Mean-square error and interpolated files I-*\n";
       exit(-1);
```

```
   }
   cout<<"Converting file to Agent-like output"<<endl;
   convert(argv[1], "temp.log");
   // File of throughput to agent-like/SNMP data
   cout<<" -- converted! now processing"<<endl;
   float Peak = getMAX_peak("temp.log");  // Using agent-like file
   for (int i=0; i<N_ELEM; i++){
         Output[i]=(2*i/3-2)*T_g;
         DeltaT[i] = i*T_g*10/6; //T_g=10
         DeltaV[i]=3*Peak/6*i;    // for internet
         cout <<" "<<DeltaV[i]<<"\t ";
     }
   cout <<"Maximum Peak "<< Peak<< " At 1/"<<T_g<<"s\
                  granularity"<<endl;
iFile.open("temp.log");
for (int j=0;j<=MAX_Bins;j++){
        Bin_o2[j]=0;  Bin_o3[j]=0; Bin_o4[j]=0;
        Bin_flc[j]=0; Bin_ref[j]=0;
   }
   oid_2.nextsampleTime = T_g;
   oid_3.nextsampleTime = T_g;
   oid_4.nextsampleTime = T_g;
   oid_flc.nextsampleTime = T_g;
   iFile >> lastreadSNMP;
   oid_2.Lastvalues[2] =lastreadSNMP;
   oid_3.Lastvalues[3] =lastreadSNMP;
   oid_4.Lastvalues[4] =lastreadSNMP;
   oid_flc.Lastvalues[1] =lastreadSNMP;
   l150 = lastreadSNMP;
   po2=0; po3=0; po4=0; pflc=0;
   while ((!iFile.eof()) && (t<T_g*7200)){
       t = t + 1;                    // timing of 10 ms
       iFile >> readSNMP;
       if (readSNMP<0)    readSNMP = 0;
       if (t%T_g==0){
            k=(int)((readSNMP-lastreadSNMP)/Peak*MAX_Bins);
            if (k>=MAX_Bins) k=MAX_Bins-1;
            Bin_ref[k]=Bin_ref[k]+1;
            raw <<t/10<<"   "<<(readSNMP - lastreadSNMP)/Peak<<endl;
            lastreadSNMP=readSNMP;
       }

       if (oid_2.nextsampleTime==t) {
            dt = oid_2.etiming/T_g;
            k=(int)((readSNMP-\
                   oid_2.Lastvalues[2])/(dt*Peak)*MAX_Bins);
            if (k>=MAX_Bins) k=MAX_Bins-1;
            Bin_o2[k]=Bin_o2[k]+1;
            f_o2<<(float) t/10<<" "<<(readSNMP-\
                   oid_2.Lastvalues[2])/(dt*Peak)<<endl;
            Predict_Timing(readSNMP, oid_2, 2, t, Peak);
       }
       if (oid_3.nextsampleTime==t) {
            dt =oid_3.etiming/T_g;
            k=(int)((readSNMP-\
                  oid_3.Lastvalues[3])/(dt*Peak)*MAX_Bins);
```

```
            if (k>=MAX_Bins) k=MAX_Bins-1;
            Bin_o3[k]=Bin_o3[k]+1;
            f_o3 <<(float) t/10<<" "<<(readSNMP-\
                  oid_3.Lastvalues[3])/(dt*Peak)<<endl;
            Predict_Timing(readSNMP, oid_3, 3, t, Peak);
        }
        if (oid_4.nextsampleTime==t)  {
            dt = oid_4.etiming/T_g;
            k=(int)((readSNMP-\
                  oid_4.Lastvalues[4])/(dt*Peak)*MAX_Bins);
            if (k>=MAX_Bins) k=MAX_Bins-1;
            Bin_o4[k]=Bin_o4[k]+1;
            f_o4 <<(float) t/10<<" "<<(readSNMP-\
                 oid_4.Lastvalues[4])/(dt*Peak)<<endl;
            Predict_Timing(readSNMP, oid_4, 4, t, Peak);
        }
        if (oid_flc.nextsampleTime == t) {
            dt = oid_flc.etiming/T_g;
            k=(int)((readSNMP-\
               oid_flc.Lastvalues[2])/(dt*Peak)*MAX_Bins);
            if (k>=MAX_Bins) k=MAX_Bins-1;
            Bin_flc[k]=Bin_flc[k]+1;//(int)dt;
            f_flc<<(float) t/10<<"  "<<(readSNMP-\
               oid_flc.Lastvalues[2])/(dt*Peak)<<endl;
            Predict_Timing_FLC(readSNMP, oid_flc,t);
        }
    }
    cout <<" About to end... writting last stuff "<< endl;
    CorrIndex* Index = new CorrIndex;
    Index->fileCorrelate("rawdata.txt", "i-rawdata.txt");
    Index->fileCorrelate("monitor_2", "i-monitor_2");
    cout <<"Sample count order-2: "<<TotalSamples(Bin_o2)<<endl;
    Index->fileCorrelate("monitor_3", "i-monitor_3");
    cout <<"Sample count order-3: "<<TotalSamples(Bin_o3)<<endl;
    Index->fileCorrelate("monitor_4", "i-monitor_4");
    cout <<"Sample count order-4: "<<TotalSamples(Bin_o4)<<endl;
    Index->fileCorrelate("flc_data", "i-flc_data");
    cout <<"Sample count FLC: "<<TotalSamples(Bin_flc)<<endl;
    cout<<"Order-2"<<endl;
    Index->simpleCorrelation("i-rawdata.txt", "i-monitor_2");
    cout<<"Order-3"<<endl;
    Index->simpleCorrelation("i-rawdata.txt", "i-monitor_3");
    cout<<"Order-4"<<endl;
    Index->simpleCorrelation("i-rawdata.txt", "i-monitor_4");
    cout<<"FLC"<<endl;
    Index->simpleCorrelation("i-rawdata.txt", "i-flc_data");
    fsys  <<"Binary\t distribution  "<< endl;
    delete Index;

    iFile.close();
    fsys << "Bins\tTs=1\torder-2\torder-3\torder-4\t FLC"<<endl;
    for (int j=0;j<MAX_Bins;j++){
       fsys  <<j<<"\t"<<Bin_ref[j]<<"\t"<<Bin_o2[j];
       fsys  <<"\t"<< Bin_o3[j];
       fsys  <<"\t"<< Bin_o4[j];
       fsys  <<"\t"<< Bin_flc[j]<< endl;
```

```
   }

   fsys <<"Phi-coefficient Order-\
              2:\t"<<CalculatePhi(Bin_ref,Bin_o2)<<endl;
   fsys <<"Samples Order-2 :\t "<<TotalSamples(Bin_o2)<<endl;
   fsys <<"Phi-coefficient Order-\
              3:\t"<<CalculatePhi(Bin_ref,Bin_o3)<<endl;
   fsys <<"Samples order-3 :\t "<<TotalSamples(Bin_o3)<<endl;
   fsys <<"Phi-coefficient Order-4 \
            :\t"<<CalculatePhi(Bin_ref,Bin_o4)<<endl;
   fsys <<"Samples Order-4 :\t "<<TotalSamples(Bin_o4)<<endl;
   fsys <<"Phi-coefficient\
           FLC:\t"<<CalculatePhi(Bin_ref,Bin_flc)<<endl;
   fsys <<"Samples FLC   :\t "<<TotalSamples(Bin_flc)<<endl;
   fsys <<"Total Number of Samples "<<TotalSamples(Bin_ref)<<endl;
   fsys <<"Maximum value found in the ref is :\t"<<Peak<<endl;
   f_o2.close();
   f_o3.close();
   f_o4.close();
   f_flc.close();
   fsys.close();
   OptimizeFLC("temp.log", Bin_ref, Bin_flc, Peak);
}
```

Correlation.h

This is a C++ library used by the simulator to apply the mean square error equation and linear interpolation. The class, *CorrIndex*, utilizes the *method SimpleCorrelation(char* Reference, char* Comparison)* to calculate the mean-square error, and the error function of the file *Reference* and the *Comparison*. It requires of time-stamped files of equal size to provide an accurate result. The method *fileCorrelate(char* input, char* output)* takes the *input* file and returns an interpolated set (output file) with 72000 samples.

```
// -------------------------------------------------------------
// High-performance Computing and Simulation Research Lab
// University of Florida
// October 25th, 1999
// -------------------------------------------------------------
//
// Edwin Hernandez    hernande@hcs.ufl.edu
// v1.0
```

```
//
// This library is in charge of calculation the correlation index
// and the difference between two signals
// Both signals are sampled at differente or similar time intervals
// but they must be stored in a file like:
//
//     TimeValue     Value
//              i         i
//
// Timevalues have to be increasing to avoid malfunctioning
// ------------------------------------------------------------
#include <fstream.h>
#include <stream.h>
#include <math.h>


/* class POINT
**
** used in the interpolation process, stores time and value data
*/
class Point{
public:
   float time; double value;
   Point(){time=0; value=0;}
   ~Point(){;}
};

/*
** class CorrIndex
** calculates the correlation index and the Mean-square error
** between two files, uses linear interpolation to make sizes
** the same.
*/
class CorrIndex {
    ifstream ref, com;
    double diff, square;
public:
    CorrIndex(){;}
    ~CorrIndex(){;}
    void L_Interpol(Point* p1, Point* p2, float t, Point* c);
    float CalcCorrelation(Point* a, Point* b);
    void fileCorrelate(char*, char* );
    void correlation(char* r, char *c);
    void simpleCorrelation(char* r, char* c);
};

/*
** L_Interpol(Point*, Point*, float t, Point* r)
** given two points, interpolates the given point in time
** t. t has to be between, p1->time and p2->time, otherwise
** is extrapolation.
*/
void CorrIndex::L_Interpol(Point* p1, Point* p2, float t, Point*
rpoint){
    float m, b;
    m = (p2->value - p1->value)/(p2->time - p1->time);
```

```
    b = p2->value - m*p2->time;
    rpoint->value = m*t+b;
    rpoint->time = t;
};


/* Filecorrelate(char*, char*)
**
** This program Grabs a file input from "name" and generates a file
** output on "OutName", using steps of 0.1 seconds for each iteration.
** When it reads
** each points, interpolates them at the 0.1 second level.
**   if the file input is:
**       T    value
**       1      1
**       2      0
**    the outcome would be:
**         0    0
**       0.1   0.1
**       0.2   0.2
**       0.3   0.3
**       ...
** the mean-square error and any correlation equation is
** calculated, based on the Interpolated file. Interpolated files
** have the same amount of data.
*/
void CorrIndex::fileCorrelate(char* name, char* OutName){
   ofstream oRef;
   ifstream iRef;
   iRef.open(name);
   oRef.open(OutName);
   Point* Ref_Data = new Point;
   Point* Ref_Data1 = new Point;  // Parameters to read from file
   Point* P = new Point;          // Used to pass parameters
   Point* LastEntry = new Point;  // read file and process data
   char str1[6], str2[20];
   long g=0;
   oRef<<"0\t0\tM"<<endl;     // Syntax of output, time, value, FLAG
                              // Flag='M' measured, I='interpolated'
   while (!iRef.eof()){
     iRef>>str1>>str2;        // more reliable have strings to read
     Ref_Data->time = atof(str1);
                              // the file and later convert the data
     Ref_Data->value = atof(str2);
      for (float m=Ref_Data1->time+0.1;m<Ref_Data->time;m=m+0.1){
          L_Interpol(Ref_Data1, Ref_Data, m, P);
          // Linear interpolation
          if (Ref_Data->time-P->time>=0.05)
             oRef<<m<<" \t "<<P->value<<"\tI"<<endl;
               // At the time interval m
             LastEntry->time = P->time;
               // Store the last value
             LastEntry->value = P->value;
               // sometimes write same
        }
                                 // data twice, so I had to
```

```
      oRef<<Ref_Data->time<<" \t "<<Ref_Data->value<<"\tM"<<endl;
      Ref_Data1->time= Ref_Data->time;
      Ref_Data1->value=Ref_Data->value;
    }
    delete P;
    delete Ref_Data;
    delete Ref_Data1;
    delete LastEntry;
    oRef.close();
    iRef.close();
}


/*  SimpleCorrelation()
**
** it is assumed that the interpolation was made at the
** File being analized. In other words, it has to be in the format
** Generated by fileCorrelate
** This procedure calculates the sum of squared differences,
** and the mean squared differences, having as a reference of
** comparasion, the file pointed by char* reference
** returns the index on the screeen
*/
void CorrIndex::simpleCorrelation(char* reference, char* compare){
    ref.open(reference);
    com.open(compare);
    ref.seekg(0);
    com.seekg(0);
    ref.clear(); com.clear();
    square = 0;
    long i = 0;
    float time;
    float sqr2=0; float sqr3=0;
    double vRef, vCom;
    cout <<"comparing...";
    char str1[6], str2[20], flagR[5], flagC[5];
    long k=0;
    while ((!ref.eof())&&(!com.eof())){
       i++;
       ref >>str1>>str2>>flagR;
       vRef=atof(str2);
       com >>str1 >>str2>>flagC;
       vCom=atof(str2);
       sqr3 = pow((vCom - vRef),2) + sqr3;
       if ((strcmp(flagR,"M")==0)||(strcmp(flagC,"M")==0)){
         if (vRef>0)
            square = (vRef - vCom)*(vRef-vCom)/vRef + square;
            k=k+1;
            sqr2 = (vRef - vCom)*(vRef-vCom)+sqr2;   // Sum of squared
      }
    }
    cout << "  Coefficient ="<< square <<" N compared ="<<k<< endl;
    cout << "  other coeff ="<< sqr2   << " ave= "<<sqr2/k<<endl;
    cout << " using "<< i <<"Points is Sum squared of \
                          errors:"<<sqr3<<endl;
    ref.close();
```

```
   com.close();
}
```

FLC.h

This C++ library is used by the simulator to calculate the F_{out} parameter of the fuzzy logic controller. The functions provided here use twenty-five fuzzy rules, the definition of the membership functions and the correlation product method of defuzzyfication. This file has to be included in the main program to call the function defined by *GetTiming(float* Sample, float* Timing)* which processes the inference rules to generate the required outcome.

```
// Fuzzy logic controller -  adaptive sampling
// based on Available bandwidth and network utilization
// ---------------------------------------------------
// Edwin Hernandez  - HCS Lab
// University of Florida
//
// Based on Henry Hurdon - Fuzzy Fan Controller
// on CMU AI - repository
// ***************************************************

#include <math.h>
#include <stream.h>
#include <math.h>

/* Function exported to the world */
float GetTiming(float, float);

// ---- Definitions of the membership function associated to  ---
// ---- FLC and the adaptive sampling technique               ---
// ------------------------------------------------------------
// DeltaV[] =  definition of the change between previous and present
// value
// NoChange[]        = NO Change at all, PresentV == Previous
// Change-slightly[] = not much of change but slight change
// Change-Low[]      = change is minimun (threshold)
// Change-med[]      = change is more than minimun
// Change-High[]     = Change High PresentV >> Previous
//
// DeltaT[] =  Sampling
// Small[]     = Sampling is done at very low intervals (high freq)
// slightMed[]  = Defined as med-low sampling
// medium[]   = "average" or intermediate sampling
// SlightLong[] = medHigh intervals of sampling
```

```
// Long[]    = High intervals of sampling, MAX = 0.1 Tobservation (low
// freq)

// ------------ ONLY TWO INPUTS ------------------------ version 1.0
// Output[]    = Increase/Dec level for the Sampling Time
// Dec-High[]  = decrease it a lot
// Dec-mid[]   = decrease it but midly
// No-Change[] = no change in output
// Inc-mid[]   = Increast it midly
// Inc-high[]  = increase it a lot.

// Fuzzy membership functions
//
//               DeltaV
//
//   |
//   |------\    /\     /\    /\     /--------
//   |       \  /  \   /  \  /  \   /
//   |        \/    \ /    \/    \ /
//   |    NC  /\ CS \  CL  /\ CM  \  CH
//   |       /  \   / \   /  \   / \
//   |      /    \ /   \ /    \ /   \
//   ------------------------------------------
//   0    P/6    P/3   P/2   2*P/3   5*P/6   P
//
// P, is the maximum peak value that can be registered in the
// maximum DeltaT
//
//               DeltaT
//
//   |
//   |------\    /\     /\    /\     /--------
//   |       \  /  \   /  \  /  \   /
//   |        \/    \ /    \/    \ /
//   |  Short /\ Slig\ medi \slig \ Long
//   |       /  \Med/ \ um/  \Long \
//   |      /    \ /   \ /    \ /   \
//   ------------------------------------------
//   0    M/6     M/3   M/2  2*M/3  5*M/6  M
//
//
//        Output (interval to increment/decrement)
//  |
//  |------\   /\     /\    /\     /----------
//  |       \ /  \   /  \  /  \   /
//  |        \    \ /    \/    \ /
//  |    DH / \DM  \  NC /\ IM  \  IH
//  |      /   \  / \   /  \   / \
//  |     /     \/   \ /    \ /   \
//  ------------------------------------------------
// -L -2*L/3     -L/3   0      L/3    L*2/3     L

// ************************************************************
// RULES
// ************************************************************
//   remember: DeltaT small  = high frequ  Ts min = 1sec
```

```
//             DeltaT Long   = low Freque  Ts max = 10 sec
//    ____________________________________________________
//     DeltaV        DeltaT              Output
//    ----------------------------------------------------
//     NChange          Low                IH
//     NChange          Med-Low            IH
//     NChange          Med                IM
//     NChange          Med-High           IM
//     NChange          High               NC
//     Change-Slight    Low                IH
//     Change-Slight    Med-Low            IM
//     Change-Slight    Med                NC
//     Change-Slight    Med-High           DM
//     Change-Slight    High               DM
//     Change-Low       Low                IM
//     Change-Low       Med-Low            NC
//     Change-Low       Med                DM
//     Change-Low       Med-High           DM
//     Change-Low       High               DH
//     Change-medium    Low                NC
//     Change-medium    Med-Low            DM
//     Change-medium    Med                DH
//     Change-medium    Med-High           DH
//     Change-medium    High               DH
//     Change-High      Low                DM
//     Change-High      Med-Low            DH
//     Change-High      Med                DH
//     Change-high      Med-High           DH
//     Change-High      High               DH
//     ----------------------------------------------
// We must analyze this rules and maybe use G.A. to get
// better results.


#define N_ELEM 7

// fuzzy set values for DeltaV
// Assuming a BW = 155 Mbits/sec = 20 Mbytes/sec
// DeltaV is in Bytes, asumming a maximum of 10% of BW utilization
// per node, the maximum DetaV would be only 2 Mbytes in the
// smallest interval, 1sec.
// For FE works
//float  DeltaV[N_ELEM] ={0,15000,30000,45000,600000,75000,1000000};
// For simulation they change
float DeltaV[N_ELEM] ={0,100000,200000,300000,400000, 500000,700000};
float NChange[N_ELEM] = {1.0,   1.0,      0,     0,     0,  0,   0};
float CSlight[N_ELEM] = {0,       0,    1.0,     0,     0,  0,   0};
float    CLow[N_ELEM] = {0,       0,      0,   1.0,     0,  0,   0};
float CMedium[N_ELEM] = {0,       0,      0,     0,   1.0,  0,   0};
float   CHigh[N_ELEM] = {0,       0,      0,     0,     0,1.0, 1.0};

// For CELLS/sec 53Bytes cells/sec then the Count is 37Kcells/sec
// ATM parameters.
//float  DeltaV[N_ELEM] = {0, 2500, 5000, 10000, 20000, 25000, 40000};
// Vconf
//float  DeltaV[N_ELEM] = {0, 1250, 2500, 5000, 10000, 15000, 50000};
```

```
// in general

// Fuzzy sets values for DeltaT
// Asumming a interval of a Maximum of 20 seconds
// in other words, designed for a population > 3600 values
// H=0.5
// Real-world situation
//float DeltaT[N_ELEM] =  {0,     2,      4,     6,      8, 10,12};
// Simulation Times are different... scaled by 100
float   DeltaT[N_ELEM] =  {0,  20, 40, 60, 80, 100, 120};
float    Small[N_ELEM] =  {1.0, 1.0,       0,     0,     0,  0,   0};
float  SligMed[N_ELEM] =  {0,     0,     1.0,     0,     0,  0,   0};
float   Medium[N_ELEM] =  {0,     0,       0,   1.0,     0,  0,   0};
float SligLong[N_ELEM] =  {0,     0,       0,     0,   1.0,  0,   0};
float     Long[N_ELEM] =  {0,     0,       0,     0,     0,1.0, 1.0};

// Fuzzy set values for Output
float Output[N_ELEM] ={-6,   -4,   -2,     0,   1.33,    2.66,    4};
float     DH[N_ELEM] ={1.0,  1.0,     0,   0,     0,    0,    0};
float     DM[N_ELEM] ={  0,    0,   1.0,   0,     0,    0,    0};
float     NC[N_ELEM] ={  0,    0,     0, 1.0,     0,    0,    0};
float     IM[N_ELEM] ={  0,    0,     0,   0,   1.0,    0,    0};
float     IH[N_ELEM] ={  0,    0,     0,   0,     0,  1.0,  1.0};
float    ZERO[N_ELEM] = {0,    0,     0,   0,     0,    0,    0};

/* standard max() definition, used by a number of routines.. */
// #ifndef max(a,b)
#define max(a,b)    (((a) > (b)) ? (a) : (b))
// #endif

/* GetMembership()
**
** GetMembership() is used to calculate the membership value Ao
** with respect to a given Xo value based on the universe X[]
** and the fuzzy set A[]. The calculated value is returned.
**
*/
float GetMembership(float xo, float X[], float A[])
{
  float ao;
  int si;
  /* error check bounds on xo..
   * if xo outside of universe X[N_ELEM] then
   * return closest membership value. */
  if (xo < X[0]) {
     return(A[0]);
  } else if (xo > X[N_ELEM -1]) {
    return(A[N_ELEM -1]);
  }

  /* Ok, calculate the membership value */
  for (si = 0; si < N_ELEM-1; si++) {
    /* calculate membership value */
    if ((xo >= X[si])&&(xo <= X[si+1])) {
      if (X[si] == X[si+1]) {
         /* membership value based on maximum value */
```

```
         ao = max(A[si], A[si+1]);
       } else {
         /* membership value based on linear interpolation */
         ao = (xo-X[si]) * ((A[si+1]-A[si])/(X[si+1]-X[si])) + A[si];
       }
     }
   }

   return(ao);
}


/* GetMaximum()
**
** GetMaximum() is used to perform the MAX operation on the fuzzy sets
** A[] and B[]. The results are stored in C[].
**
*/
void GetMaximum(float A[N_ELEM], float B[N_ELEM], float C[N_ELEM])
{
   int   si;
   for (si = 0; si < N_ELEM; si++) {
     C[si] = max(A[si], B[si]);
   }
}


/* DoInferEngine()
**
** DoInferEngine() is used to perform the inference engine calculation
** based on Ao and Bo and C[]. The results are stored in O[].
**
*/
void DoInferEngine(float Wo, float C[], float O[])
{
   int   si;
   for (si = 0; si < N_ELEM; si++) {
     O[si] = Wo*C[si];
   }
}


/* DeFuzzyOutput()
**
** DeFuzzyOutput() is used to de-fuzzify the universe Y[] based on
** the fuzzy set B[], using the center of gravity theorem.
** The result is stored in *Yo.
**
*/
void DeFuzzyOutput(float Y[N_ELEM], float B[N_ELEM], float *Yo)
{
   float yB_sum, B_sum;
   int   si;

   yB_sum = 0.0;
   B_sum  = 0.0;
```

```
  /* get summations of Y[]B[] and B[] */
  for (si = 0; si < (N_ELEM -1); si++) {
    yB_sum += ((Y[si] + Y[si+1])/2)*((B[si] + B[si+1])/2);
     B_sum += (B[si] + B[si+1])/2;
  }

  /* check for divide by zero */
  if (B_sum != 0.0) {
    *Yo = yB_sum/B_sum;
  }
}

float R[N_ELEM];
/* R[] - interm inference result        */
float O[N_ELEM];
/* O[] - accumulative inference result  */

/*********************************************************************
    The above linguistic matrix can be reduced to the following fuzzy

    IF DeltaV = NChange       AND DeltaT = Small     THEN Output = IH
    IF DeltaV = NChange       AND DeltaT = Slig-Med  THEN Output = IH
    IF DeltaV = NChange       AND DeltaT = Med       THEN Output = IM
    IF DeltaV = NChange       AND DeltaT = Slig-long THEN Output = IM
    IF DeltaV = NChange       AND DeltaT = Long      THEN Output = NC

    IF DeltaV = Change-Slight AND DeltaT = Small     THEN Output = IH
    IF DeltaV = Change-Slight AND DeltaT = Slig-Med  THEN Output = IM
    IF DeltaV = Change-Slight AND DeltaT = Med       THEN Output = NC
    IF DeltaV = Change-Slight AND DeltaT = Slig-Long THEN Output = DM
    IF DeltaV = Change-Slight AND DeltaT = Long      THEN Output = DM

    IF DeltaV = Change-Low    AND  DeltaT = Small     THEN Output = IM
    IF DeltaV = Change-Low    AND  DeltaT = Slig-Med  THEN Output = NC
    IF DeltaV = Change-Low    AND  DeltaT = Med       THEN Output = DM
    IF DeltaV = Change-Low    AND  DeltaT = Slig-Long THEN Output = DH
    IF DeltaV = Change-Low    AND  DeltaT = Long      THEN Output = DH

    IF DeltaV = Change-medium AND  DeltaT = Small     THEN Output = NC
    IF DeltaV = Change-medium AND  DeltaT = Slig-Med  THEN Output = DM
    IF DeltaV = Change-medium AND  DeltaT = Med       THEN Output = DH
    IF DeltaV = Change-medium AND  DeltaT = slig-Long THEN Output = DH
    IF DeltaV = Change-medium AND  DeltaT = Long      THEN Output = DH

    IF DeltaV = Change-High   AND  DeltaT = Small     THEN Output = DM
    IF DeltaV = Change-High   AND  DeltaT = Slig-Med  THEN Output = DH
    IF DeltaV = Change-High   AND  DeltaT = Med       THEN Output = DH
    IF DeltaV = Change-High   AND  DeltaT = Slig-Long THEN Output = DH
    IF DeltaV = Change-High   NAD  DeltaT = High      THEN Output = DH
*/


float  GetTimming(float Value, float Timming )
{
   float Oput;
```

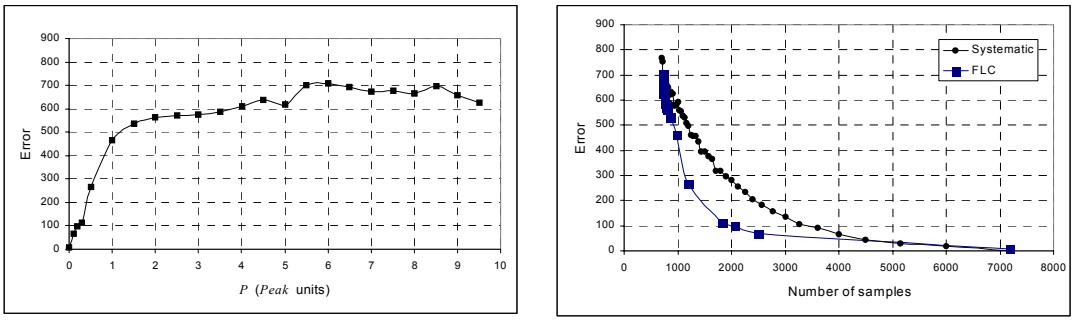
```
   float V_NChange, V_ChSlight, V_ChLow, V_ChMed, V_ChHigh;
   float T_Low, T_MedLow, T_Med, T_MedHigh, T_High;
   float O_DH, O_DM, O_NC, O_IM, O_IH;

   O_DH = 0;
   O_DM = 0;
   O_NC = 0;
   O_IM = 0;
   O_IH = 0;
   V_NChange   = GetMembership(Value,   DeltaV,     NChange);
   V_ChSlight  = GetMembership(Value,   DeltaV,     CSlight);
   V_ChLow     = GetMembership(Value,   DeltaV,        CLow);
   V_ChMed     = GetMembership(Value,   DeltaV,     CMedium);
   V_ChHigh    = GetMembership(Value,   DeltaV,       CHigh);
   T_Low       = GetMembership(Timming, DeltaT,       small);
   T_MedLow    = GetMembership(Timming, DeltaT,   slightMed);
   T_Med       = GetMembership(Timming, DeltaT,      Medium);
   T_MedHigh   = GetMembership(Timming, DeltaT,  slightLong);
   T_High      = GetMembership(Timming, DeltaT,        Long);

//  IF DeltaV = NChange   AND DeltaT = Small      THEN Output = IH
// and all of the fuzzy rules are here.
   O_IH =  max(V_NChange,            O_IH);
   O_IH =  max(V_NChange*T_MedLow,   O_IH);
   O_IM = max(V_NChange*T_Med,       O_IM);
   O_IM = max(V_NChange*T_MedHigh,   O_IM);
   O_NC = max(V_NChange*T_High,      O_NC);
   O_IH = max(V_ChSlight*T_Low,      O_IH);
   O_IM = max(V_ChSlight*T_MedLow,   O_IM);
   O_NC = max(V_ChSlight*T_MedLow,   O_NC);
   O_DM = max(V_ChSlight*T_MedHigh,  O_DM);
   O_DM = max(V_ChSlight*T_High,     O_DM);
   O_IM = max(V_ChLow*T_Low,         O_IM);
   O_NC = max(V_ChLow*T_MedLow,      O_NC);
   O_DM = max(V_ChLow*T_Med,         O_DM);
   O_DH = max(V_ChLow*T_MedHigh,     O_DH);
   O_DH = max(V_ChLow*T_High,        O_DH);
   O_NC = max(V_ChMed*T_Low,         O_NC);
   O_DM = max(V_ChMed*T_MedLow,      O_DM);
   O_DH = max(V_ChMed*T_Med,         O_DH);
   O_DH = max(V_ChMed*T_MedHigh,     O_DH);
   O_DH = max(V_ChMed*T_High,        O_DH);
   O_DM = max(V_ChHigh*T_Low,        O_DM);
   O_DH = max(V_ChHigh*T_MedLow,     O_DH);
   O_DH = max(V_ChHigh*T_Med,        O_DH);
   O_DH = max(V_ChHigh*T_MedHigh,    O_DH);
   O_DH = max(V_ChHigh*T_High,       O_DH);
  /* Applying all the inference rules to all the values entered */
   DoInferEngine(O_DH, DH, R);
   GetMaximum(R, ZERO,  O);
   DoInferEngine(O_DM, DM, R);
   GetMaximum(R,    O,  O);
   DoInferEngine(O_NC, NC, R);
   GetMaximum(R,    O,  O);
   DoInferEngine(O_IM, IM, R);
   GetMaximum(R,    O,  O);
```

```
  DoInferEngine(O_IH, IH, R);
  GetMaximum(R,    O,  O);
  DeFuzzyOutput(Output, O, &Oput);
  return Oput;  /* Return the Defuzzified Output */
}
```

APPENDIX B
FUZZY LOGIC OPTIMIZATION

In order to configure the fuzzy logic controller, the definition of membership functions, rules, and their conformant parameter values need to be determined. The FLC technique used for adaptive sampling was adjusted by comparing it with systematic sampling in terms of the error measurement and the number of samples obtained with both methods. Once the behavior of both techniques is drawn, optimization is achieved by locating the region of the plot where sampling adaptively requires fewer samples at the same amount of error.

The variations of P, which represents the boundaries of the membership function ΔX, are show in Figure 20. The figure depicts the value of error and the number of samples returned by the FLC and systematic sampling in the same graph. For the plot illustrated in Figure 20(a), the values of P (see Table 2) ranged between $S_{Internet}$ /10 to $S_{Internet}$. The increment of P has an exponential effect in the growth of the magnitude of error with a limit at 700. The parameter L was as well varied (from 2 to 5 sec) providing a negligible impact on the performance of the FLC.



(a) Error and variation of P at the FLC

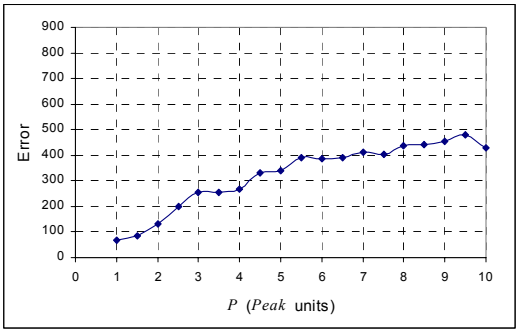
(b) Error and number of samples with systematic and FLC-based sampling

Figure 20. FLC tuning for Internet traffic

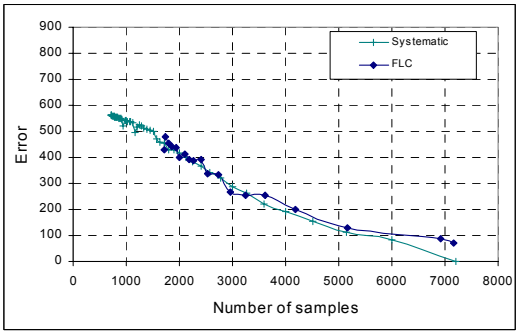
Figure 20(b) indicates that systematic sampling of Internet data yields to a larger error than the FLC-based adaptation for a fixed number of samples. The FLC situated with $L = 2$ sec and $P = 0.4 \times S_{Internet.}$ bytes presented the best performance. This conFigureuration provides the minimum error value ($error \approx 100$) and 2095 samples, which if compared with the systematic sampling present a difference of approximately 1300 elements (40% of the amount systematically). This difference indicates that the FLC with 2095 samples presents the same magnitude of error than a systematic sampling measurement with approximately 3300 samples.

Similarly to the results presented for Internet traffic, Figure 21 depicts the results of the same procedure but used for sampling of videoconference traffic. As illustrated in Figure 21(b) the FLC matches the behavior of systematic sampling. The figure also indicates that the FLC set with $P = 4.0 \times S_{video}$ showed the best improvements. In addition, this setting required that the output membership function, F_{out}, be defined with $L = 0.5$ *sec.* Increased values of L presented a decrement on the performance of the FLC. This combination of L and P are the points in the plot where the FLC showed a smaller magnitude of error with fewer samples. In spite of the improvement, the

difference of the number of samples between adaptive and the systematic technique was only of a few hundreds of samples (≈2% of the total amount of samples at the reference). Therefore the proposed FLC used to sample videoconference data only matches the quality of the same measurements performed systematically.



(a) Error and variation of *P* at the FLC



(b) Error and number of samples with systematic and FLC-based sampling

Figure 21. FLC tuning for videoconference traffic

REFERENCES

1. S. Kay. *Modern spectral estimation–Theory and application.* New York: Prentice-Hall, 1988.

2. B. Widrow and S. Stears. *Adaptive signal processing.* New York: Prentice-Hall, 1985.

3. E. Cox. *The fuzzy systems handbook*, 2nd ed., New York: AP Professional, 1999.

4. K. Claffy, H. Braun, and G. Polyzos. Application of sampling methodologies to network Traffic Characterization. *Proceedings of ACM SIGCOMM '93*, San Francisco, CA:ACM, 194-203, September 1993.

5. I. Cozzani and S. Giordano. Traffic sampling methods for end-to-end QoS evaluation in large heterogeneous networks. *Computer Networks and ISDN Systems*, 30 (16-18): 1697-1706, 1998.

6. S. Feit. *SNMP–A guide to network management.* New York: McGraw-Hill, 1995.

7. J. Case, M. Fedor, M. Schoffstall, and J. Davin. *A simple network management protocol.* RFC1157. VA: IETF/CNRI, May 1990.

8. M. Rose. *The simple book: An introduction to internet management*. 2nd ed., New York: Prentice-Hall, 1994.

9. V. Cerf, *IAB recommendations for the development of internet network management standards.* RFC 1052. VA:IETF/CNRI, April 1988.

10. W. Stallings. *SNMP, SNMPv2, and CMIP. The practical guide to network management standards.* MA: Addison-Wesley, 1993.

11. Desktop Management Task Force (DMTF). *Common information model*. http://www.dmtf.org/spec/cim_core/ October 1998.

12. J. Martin-Flatin. Push vs. pull in web-based network management. *Technical-report EPFL-ICA,* Submitted to International Symposium on Integrated Network Management. Boston. MA:IFIP/IEEE, May 1999.

13. C. Therrien. *Discrete random signals and statistical signal processing.* River, NJ: Prentice-Hall, 1992.

14. H. Li and V. Yen. *Fuzzy sets and fuzzy decision making.* Boca Raton, FL: CRC-Press, USA, 1995.

15. V. Catania, G. Ficili, S. Palazzo, and D. Panno. A comparative analysis of fuzzy versus conventional policing mechanisms for ATM networks. *IEEE/ACM Transactions on Networking*. 4 (3): 449-459, June 1996.

16. V. Jacobson and M. Karels. Congestion Avoidance and Control, *Proceedings of the ACM SIGCOMM '88*, Stanford, CA: ACM, 273-288, August 1988.

17. R. Cheng and C. Chang. Design of a fuzzy traffic controller for ATM networks. *IEEE/ACM Transactions on Networking.* 4 (3): 460-469, June 1996.

18. J. Drobisz. Adaptive sampling methods for high speed networks to determine traffic statistics including the Hurst parameter. Master's Thesis, University of South Florida, Tampa, 1998.

19. L. Kalampoukas, A. Varma, and K. Ramakrishman. Explicit window adaptation: A method to enhance TCP performance. *Technical Report.* University of California Santa Cruz: UCSC-CRL-97-21, August 1991.

20. A. Bonde and S. Ghosh. A comparative study of fuzzy versus 'fixed' threshold for robust queue management in cell-switching networks. *IEEE/ACM Transactions on Networking*, 2 (4): 337-344, August 1994.

21. R. Givan and E. Chong. Intelligent and adaptive management of multi-class networks. *Technical Report.* West Lafayette, IN: Purdue University, ARPA No. G371, April 1999.

BIOGRAPHICAL SKETCH

Edwin Hernandez was born in Tegucigalpa, Honduras. In 1995, he graduated as Electronics Engineer from the Instituto Tecnologico de Costa Rica in Cartago, Costa Rica. His undergraduate work was conducted in the design of a throughput and PBX monitoring system for the switched packet network of Costa Rica. He went back to his country in 1995 and joined the telecommunications industry. From 1995 through 1997, he worked for the Central American Telecommunication Commission (COMTELCA), where he was in charge of Internet services, the video-conference system, and software development. At the same time, he belonged to the executive and technical boards of COMPUNET, an entrepreneur Internet Service Provider (ISP), in Tegucigalpa. In 1997, he was awarded a Fulbright scholarship and decided to enroll in the graduate program at the University of Florida. In September of 1997, Edwin decided to join the High-performance Computing and Simulation (HCS) Research Laboratory where he has pursued research in network management and distributed computing. His plans include finishing a Ph.D. in computer engineering and continuing his career as an entrepreneur in Internet technologies with part-time involvement in research and academia.