Objective-Driven Monitoring for Broadband Networks

Subrata Mazumdar and Aurel A. Lazar, Fellow, IEEE

Abstract—An approach to sensor configuration, installation, and activation for real-time monitoring of broadband networks for managing its performance is presented. An objective-driven measurement strategy for establishing the dynamic and statistical databases of the network is described. Objective driven monitoring allows the activation of sensors for data collection and abstraction based on a set of objectives. The objectives are derived from the quality of service requirements for real-time traffic control and operator submitted queries. The methodology of objective-driven monitoring for selective activation of sensors is implemented as a set of rules in the knowledge base of the monitor.

Index Terms—Network, quality of service, performance management, sensor, monitoring, knowledge-based systems.

1 INTRODUCTION

Broadband networks consist of many subsystems (switching nodes, multiplexers, links, etc.) that are geographically distributed, carry multiple classes of traffic and have access to different information patterns. Although these subsystems make their own local decisions, they work together for the achievement of the common system wide goal of information transport. The common goal is to guarantee the Quality of Service (QoS) negotiated during the call setup for each of the traffic classes [1]. The QoS is specified through a set of performance parameters.

Monitoring of these parameters and of all network resources, such as buffer space, switching and communication bandwidth, and call processing, is required in order to guarantee the QoS [1]. A network monitoring system should also be applicable to several representative networks. Therefore, a proposed set of measurement parameters must be network independent [2]. They must be declared in generic terms, such as throughput, time-delay, arrival rate, inter-arrival time, etc. Sensors (measurement points) for these parameters must be made available in all the networks to be monitored. A set of objective criteria or strategies are needed by which sensors can be selectively activated and deactivated among a large number of sensors in a distributed environment. One of the main objectives of the monitoring task is the real-time support of the network control and management system during the decision making process. A consistent view of the network is assumed to be available for monitoring [3].

The monitoring of networks can be viewed at different levels of abstraction. Monitoring takes place both at hardware and software level depending upon the hardware and software components that support the information transport. In [4], a network operation center to monitor, control, and manage ARPAnet-like packet-switching networks is presented. In [5], [6], [7], and [8], network monitoring is done for LANs or interconnected LANs carrying only single class (data) traffic. In the latter work, major emphasis was on the evaluation of usage of communication resources. In [9], monitoring of a metropolitan area network, called MAGNET II, is carried out by hardware observation units (HOU) connected to network access points. Real-time traffic measurements are reported. The quality of service of traffic classes in the network is evaluated by monitoring the buffer occupancy distribution, the packet time delay distribution, the packet loss, and the gap distribution of consecutively lost packets. In [10], the monitoring of switching resources was considered for managing AT&T’s dynamic nonhierarchical routing algorithm for automatic as well as operator oriented control of the network.

Since a network can be considered to be a distributed system, the approaches to monitoring of distributed systems can also be applied to monitoring broadband networks. The monitoring of distributed systems can be classified as event-driven monitoring and as a database approach to monitoring. Most of the work in event-driven monitoring of distributed systems was done on the application level. Debugging of distributed systems [11], [12], [13] and parallel programming environments [14] are typical examples. Here, major emphasis was given to the performance evaluation of processing resources. In [15], a relational approach to monitoring was presented. In the relational approach, monitoring is viewed as an information processing activity and the historical database, a class of relational databases that encode time, is considered an appropriate formalization of the information processed by the monitor.

In this paper, the steps required to configure, install and activate sensors for monitoring broadband networks are discussed and a knowledge-based approach is presented as a solution to the problem. In order to monitor object behavior, sensors need to be configured and installed in the
network. Sensor configuration specifies the characteristics of sensors declared in the knowledge database of the monitor. These characteristics are specified by a set of attributes and a set of procedures for operations. Sensor installation involves identification of the measurement points in the network.

The architecture of our objective-driven monitoring system is knowledge-based. It consists of a knowledge database and an inference engine for reasoning on the database [16]. The inference engine consists, in turn, of two parts: a deductive inference processor and a statistical inference processor. The role of the deductive inference processor is to process the queries about the network behavior and activate sensors in the network. The role of the statistical inference processor is to abstract the information obtained by the sensors.

The monitoring system processes queries on the system as well as on the conceptual level, and sets up sensors to collect information. The system level monitor supports queries only if precise knowledge about the system is available. On the conceptual level, the monitor allows general queries without the precise knowledge of the system architecture of the network.

An objective-driven monitoring scheme is presented that selectively activates and deactivates a subset from among a large number of sensors already installed in the network. The objective-driven monitoring scheme is closely related to the concept of experimental frame of [17] that characterizes modeling objectives by specifying the form of experimentation required for obtaining answers to the questions of interest. For the class of objective-driven monitoring tasks considered in this paper, the fundamental concepts are derived from the requirements of supporting Quality of Service and of operator submitted queries. The objective-driven monitoring scheme deals with the problem of complexity in monitoring broadband networks through the concept of observation frame that we have earlier proposed [18].

An object-oriented definition of sensors is introduced and a method for specifying the configuration of the sensors in the network is given. This definition represents an alternative to “a collection of code” given in [15]. Through the specification of object-specific and variable-specific generic sensors, we can define the starting and stopping time for monitoring and also how frequently the observation samples are to be collected and recorded. Since the various measures for performance analysis are specified through a set of operators, we can easily add a new set of performance measures or selectively activate a subset of measures. Based on our approach, we can select any object, state variable, event or their performance parameter for monitoring.

This paper is organized as follows. Section 2 outlines the architecture of the experimental environment that represents a platform for knowledge-based monitoring of broadband networks. Section 3 describes the key ideas about sensor configuration, installation and query analysis for monitoring. Finally, in Section 4, the objective-driven measurement strategy and a query based activation of sensors for broadband networks are discussed.

2 THE SYSTEM ARCHITECTURE OF THE MONITOR

The architecture of the knowledge-based monitoring system was modeled as a real-time system (as shown in Fig. 1) where the monitor asynchronously interacts with the network through an interface [19]. Thus, the network can be viewed as the environment for the monitor. The network behavior is monitored through the interface and the collective information is sent to the monitoring system that maintains an image of the network. The interface is all the monitor sees of the network. The characteristics of the interface depend to large extent on the environment. What is and what is not part of the interface depends on the specific requirements of the management and control tasks.

![Knowledge-Based Monitoring System](image)

Fig. 1. Real-time system model for monitoring of integrated networks.

The interface between the network and the monitor consists of a set of state variables. A state variable is persistently present and throughout its existence it has a value that changes with progression of time. For the task of monitoring the network, state variables get their values from the processes operating in the environment. The semantic information about network objects and the interface are represented by the Entity-Relationship model [20]. The computational model consisting of a set of sample path and performance evaluation operators defined in [18], [19] is used to describe various processes that are associated with state variables.

Thus, in representing the environment and the interface, the concept of modularity was achieved through the object representation. The location and ownership of a state variable was declared through these objects. These objects are responsible for acquisition, manipulation, and dissemination of the information of their state variables. Note that, while implementing the network architecture, one has to explicitly declare a set of state variables that form the interface between the network and the monitor. These variables
characterize the observable behavior of the network. The exact specification of the interface depends on the monitor and the specific management tasks, such as performance, fault and configuration management, that the monitor is going to support.

The architecture of the knowledge-based monitoring system (as shown in Fig. 2) consists of the knowledge database and an inference engine for reasoning on the database for query processing, sensor activation, and interpretation of data collected by the sensors. The inference engine consists of two blocks:

1) the deductive inference processor and
2) the statistical inference processor.

The role of the deductive inference task is to set up a distributed observation frame, i.e., a data space in which a query may be answered.

Fig. 2. The system architecture of knowledge-based monitoring.

Fig. 3 shows the organization of the knowledge database of the network. The knowledge database is organized as follows. The system level knowledge about the network is represented in the configuration database. The configuration database contains the knowledge about the network entities, such as buffers, sources, and servers and their specific instances. Fig. 4a describes the attributes of those network entities. The dynamic database contains the information about the state and event variables of the objects in the configuration database. Fig. 4b describes the attributes of the state and event variables. The sensor database contains the generic description of sensor objects and also any specific class of state and event variables and all of the instances of the sensor object class. The objects in the sensor database indicate the specific sampling pattern for data collection and specific sensor instances indicate the activation of the sensors. The sensor database together with the configuration database forms the static database. These two databases change much less often than the dynamic database. The dynamic database only exists for those state and event variables that are measured by activating the sensors in the network. The statistical database is obtained by applying abstraction operators on both state and event variables and provides various performance measures for each state and event variable. Fig. 4c describes the attributes of the performance parameters.

Fig. 3. The organization of the knowledge database.

Fig. 4a. The attributes of basic network objects.

Typically, a query submitted by the query generator, i.e., a control task or a human operator, requires information about the performance of certain objects in the network. The query is then processed to find out the specific instances of the performance parameters of interest. Based upon these parameters, the deductive inference processor creates a derived object containing the identified performance parameters and their corresponding state and event variables. An associated derived sensor capable of monitoring the derived object is also created, which in turn cre-
ates the appropriate sensors in the sensor database for the selected state and event variables. Creation of the sensors for state variables in the knowledge database activates the sensors in the network and data is collected. The statistical inference processor then applies the statistical operators, passed during the query submission or implicit in the performance parameter specification, on the collected information and transmits the processed information to the query processor.

<table>
<thead>
<tr>
<th>(defn</th>
<th>STATE-VARIABLE</th>
<th>NETWORK-OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(name</td>
<td>generic-name</td>
<td>state-variable</td>
</tr>
<tr>
<td>var-of-generic-object</td>
<td>range_of_var</td>
<td>state-variable</td>
</tr>
<tr>
<td>(value</td>
<td>(var-name, threshold)</td>
<td>threshold value of state variable</td>
</tr>
<tr>
<td>(dbs-time</td>
<td>(value</td>
<td>time location of the sample</td>
</tr>
</tbody>
</table>

The following attributes represent operations on state variables:

| (name | min-value | recorded minimum over a period |
| max-value | recorded maximum over a period |
| count | number of times threshold is crossed below |
| (count | count-threshold | number of times threshold is crossed above |
| (time | time-obs-time | the time of observation of last sample |
| (time | last-obs-time | time of sample |
| (time | (top, bound) | number of times upper bound is reached |
| (time | lower-bound | number of times lower bound is reached |

Fig. 4b. The attributes of state and event variables.

<table>
<thead>
<tr>
<th>(defn</th>
<th>PERF-PARAMETER</th>
<th>NETWORK-OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(name</td>
<td>generic-name</td>
<td>perf-parameter</td>
</tr>
<tr>
<td>var-of-generic-object</td>
<td>count</td>
<td>total number of samples collected for observation</td>
</tr>
<tr>
<td>(time</td>
<td>time-obs-interval</td>
<td>start time of the measurement interval</td>
</tr>
<tr>
<td>(value</td>
<td>last-obs-interval</td>
<td>end time of the measurement interval</td>
</tr>
<tr>
<td>(min-value</td>
<td>min-value</td>
<td>minimum value of collected samples</td>
</tr>
<tr>
<td>(max-value</td>
<td>max-value</td>
<td>maximum value of collected samples</td>
</tr>
<tr>
<td>(var-value</td>
<td>var-value</td>
<td>variance of the collected samples</td>
</tr>
<tr>
<td>(var-value</td>
<td>var-variance</td>
<td>variance of the collected samples</td>
</tr>
<tr>
<td>(time</td>
<td>time-wtd-avg-value</td>
<td>time-weighted average of samples</td>
</tr>
<tr>
<td>(time</td>
<td>time-wtd-variance</td>
<td>time-weighted variance of samples</td>
</tr>
<tr>
<td>(value</td>
<td>(min-value</td>
<td>minimum value over measurement interval</td>
</tr>
<tr>
<td>(value</td>
<td>max-value</td>
<td>maximum value over measurement interval</td>
</tr>
<tr>
<td>(var-value</td>
<td>(var-value</td>
<td>average of number of times upper bound is hit</td>
</tr>
<tr>
<td>(var-value</td>
<td>(var-value</td>
<td>average of number of times lower bound is hit</td>
</tr>
</tbody>
</table>

Fig. 4c. The attributes of performance parameters.

3 Monitoring

As mentioned in the previous section, the monitor is defined to be a real-time system that maintains an ongoing relationship with its environment, i.e., the network. The interface between the monitor and the network is defined by a set of state variables. The interface is all the monitor sees of the network. Thus, the characteristics of the interface depend to a large extent on the network.

Depending upon the information requirement, a network can be monitored in two ways: monitoring the states (status monitoring) and monitoring the events. During status monitoring the collection of values of any of the state variables obtained by sensor activation is recorded. The rate at which the information is generated by sensors depends on the speed of operation of the corresponding objects, such as buffers, servers, and sources. For example, on the network access level, the rate of generation of status information may be equal to the packet arrival and departure rate; and at the session layer, it may be equal to the rate of arrival of new calls. Events are abstractions of state variables obtained by applying the sample path operators of the computation model over time. An event derived from a state variable is recorded by a sensor as an event variable.

The design of the network monitor can be characterized by the following steps that are in part, adapted from [15]:

- **Step 1: Sensor Configuration.** This step involves the design of the sensor, i.e., specification of its attributes and the procedures of operation that handle the necessary interaction with the monitor, enabling and disabling the sensors and buffering of monitored data and requested tasks. The sensor attributes specify the starting and stopping times for monitoring, how frequently the monitored events, measures, or resources are to be recorded, and other related performance information to be collected simultaneously;

- **Step 2: Sensor Installation.** This step involves identifying the state and event variables that are to be monitored by sensors.

- **Step 3: Query Analysis Specification.** This step specifies how to decompose a query, activate sensors and create various dataspaces for information abstraction.

- **Step 4: Execution.** This step is comprised of activating the sensors, generating and abstracting the data collected from the network, transmitting the data from the network to the monitor, and finally presenting the data on a graphics terminal. (This step is discussed in Section 4.)

Even though we have adopted the steps in [15], there are differences between [15] and our approach. In [15], the approach was relational whereas our approach is object-oriented. Since a sensor monitors state variables, there exists only one type of sensors that need to be configured for installation in the network. Thus, we need to instrument an object only once in order to obtain various measures, such as average and variance of buffer occupancy from the buffer state variable associated with a buffer. In our approach the various performance measures are defined as operators for the sensor.

In [15], there is no concept of object-specific or variable-specific generic sensors. Through the inheritance mechanism of the object-oriented approach we can specify object or variable specific generic sensors. In the case of sensor installation, we show how to select the measurement points based on performance management objectives. These measurement points are based on the actual variables that are responsible for generating information and on which the performance measures are applied. In the query analy-
sis specification step we show how the data transformation takes place during data collection and how the interpretation is done based on this data. We have a clear separation between the raw data that is collected and its abstractions. In this step, we show how the information provided by a simple query is used to identify sensors, collect data, detect events and how abstraction and interpretation is done on the collected data.

### 3.1 Sensor Configuration

A sensor is defined to be an object with a set of attributes and a set operators (algorithm or code), implemented either in hardware or software. The sensors installed in the network collect information about the state or event variables of an object and transfer it to the monitor. From an implementation point of view, every state and event variable includes its sensor as a component object. Sensor operations are executed by the set of sample path and statistical operators described in [18], [21].

Sensors installed in the network to monitor the state variables of an object are termed primitive sensors. The primitive sensor corresponds to the object class SENSOR in the sensor database; its attributes are the same as that of SENSOR (as shown in Fig. 5). The attribute sensor_code_id specifies the operator to be applied to abstract information from the history of a state variable. The attribute initiated_by indicates the initiator of the query based on which the sensor is activated. Primitive sensors contain the code for sample path and performance evaluation operators.

The attributes of a sensor are defined based on the requirements for both status and event monitoring and they are shown in Fig. 5. The abstraction operators of a primitive sensor operate on two time scales. The sample path operators abstract events on the time scale of the state variables. The performance evaluation operators operate on both state and event variables on a time scale based on the interval for statistics collection. The parameters for performance evaluation operators are provided by a set of sensor attributes. These attributes are sample_count, sample_on, sample_off, duration_of_activation, and sampling_interval.

The sensor attribute sample_count specifies the total number of state and event variables samples that are to be collected for statistical inference. The average on a fixed number of samples is computed based on the value of the sample_count and it is the default procedure for evaluating the average over a period. If the value of sample_count is not specified and the attribute sample_on is specified, then the later is used with sampling_interval to compute the total number of samples to be collected for statistical inference. The specific values of sample_count or sample_on for monitoring a state variable are determined by the rate at which the variables are changing their values. Their values are also determined by the control algorithm that is managing the object. In order to repeat the statistical inference process, the sensor attribute duration_of_activation specifies the duration of the monitoring process or the duration of time the sensor remains active. The attribute sample_off of sensor specifies the duration between two consecutive measurement intervals, i.e., sample_on period.

![Fig. 5. The schema description of sensor and its subclasses.](image-url)
Primitive sensors are activated by sending them a message. Conversely, a primitive sensor transmits information to the monitor by sending a message. Each message is time stamped with the time of creation of the information sent. If the transmitted message contains the value of a state variable then the time of creation indicates the last sampling time. If the message contains an event indication, then the time of creation indicates the event occurrence time. If the message contains the information about a performance parameter of a state or event variable, then the time of creation indicates when the value of the performance parameter was computed.

Primitive sensors are provided with the capability to queue up multiple requests for monitoring. The messages sent by the monitor to the sensor contain information about the specific sample path and performance evaluation operators to be applied to the collected values. In order to allow multiple users or control algorithms to query the state and event variables, the primitive sensors are provided with the ability of both one-to-one and one-to-many communications.

Two subclasses of primitive sensors, \texttt{STATUS SENSORS} and \texttt{EVENT SENSORS}, are defined to monitor the state and the event variables, respectively. The \texttt{STATUS SENSORS} inherit all the attributes of \texttt{SENSOR}. The relationship type \texttt{MONITORING-GENERIC-STATE-VAR} and \texttt{MONITORING-STATE-VAR} establishes the relationships between the sensor and the subclass of a state variable and the specific instance of the state variable being monitored, respectively. The \texttt{EVENT SENSORS} is declared as a subclass of the \texttt{STATUS SENSORS} and thus inherits all of its attributes. The \texttt{EVENT SENSORS} has an additional attribute, \texttt{event operator id}, that defines the operator for extracting the event. Since the behavior of all objects is represented by their state variables, only one type of primitive sensors needed to be configured. Thus, no matter how complex, an object can be monitored as long as its state variables are declared. Therefore, no object specific sensor needs to be configured.

In order to monitor an object whose behavior is defined by a set of state or event variables, \texttt{DERIVED SENSORS} are defined. The \texttt{DERIVED SENSORS} are an aggregation of a set of primitive and derived sensors. Derived sensors belong to the object class \texttt{DERIVED SENSORS}, which is a subclass of \texttt{SENSOR}. The \texttt{DERIVED SENSORS} is obtained based on the primitive sensors associated with the state and event variables of an object to be monitored. An instance of \texttt{DERIVED SENSORS} created for monitoring an object will contain the corresponding instance of \texttt{STATUS SENSORS} and/or \texttt{EVENT SENSORS}. The \texttt{DERIVED SENSORS} maintains a list of primitive or derived sensors by the relationship attribute \texttt{DERIVED FROM}. Since the behavior of an object is always expressed by its state and event variables, the sensor that monitors an object is always a member of the subclass \texttt{DERIVED SENSORS} and is an aggregate object containing the status and event sensors. Thus, in order to monitor the state of a \texttt{BUFFER}, a \texttt{DERIVED SENSORS} will be created for the \texttt{BUFFER} and it is composed of a \texttt{STATUS SENSORS} that monitors the state variable representing the \texttt{BUFFER}’s state. If an object is an aggregation of a set of objects, the \texttt{DERIVED SENSORS} for the aggregate object will consist of the \texttt{DERIVED SENSORS} of the component object.

The \texttt{DERIVED SENSORS} may be specialized to represent object specific monitoring information such as the sampling pattern. Thus, in order to obtain the behavior of an object \texttt{NETWORK_SERVICE, SENSOR NETWORK SERVICE}, a subclass of \texttt{DERIVED SENSORS}, is defined. The relationship between \texttt{SENSOR NETWORK SERVICE} and the corresponding object class, is established by \texttt{MONITORING-GENERIC-OBJECT} and the relationship between the specific instances of the \texttt{SENSOR NETWORK SERVICE} and the specific instance of \texttt{NETWORK SERVICE}, which is being monitored, is established by \texttt{MONITORING-OBJECT-INSTANCE}.

Whenever a specific object in the network is to be monitored, an instance of the \texttt{DERIVED SENSORS} is created in the sensor database. The value of the \texttt{MONITORING-GENERIC-OBJECT} of \texttt{DERIVED SENSORS} specifies the class name of the object that the derived sensor is monitoring. The instance of the \texttt{DERIVED SENSORS} and its association with the object in the configuration database is deleted when the sensor is deactivated at the end of the monitoring period. The derived sensors only exist in the sensor database. Unlike primitive sensors, no counterpart of derived sensor exists in the network. Both primitive and derived sensor instances are stored in a database called \texttt{sensor database}, as shown in Fig. 3.

### 3.2 Sensor Installation

Sensor installation allows the selection of the measurement points in the network, i.e., the state variables of the network objects that define the interface between the monitor and the network. Since network object and state variables can be uniquely identified in the system, the events associated with state variables and their performance parameters can be selected. In the modeling process of the monitor in [21], state and event variables were identified based on the performance management requirements. The sensor location was determined by the location of the identified state variables in the network.

Along with sensor configuration, the installation of primitive sensors is the only manual process associated with our monitoring scheme. Since the specification of state variables is to be decided based on the performance objectives of the control tasks, these two steps will now be required to be carried out during the specification and design step of the network. This is by itself a manual process. Thus, the design of the network monitor has been shifted to the network design phase. This design process results in a robust network design that requires very little tuning during operations. The identification of the sensor location during the specification phase helps to alleviate the reliability and the correctness problem of sensor operations.

### 3.3 Query Analysis Specification

Query analysis specification derives levels of abstraction of the collected information for performance analysis. It also selects the performance analysis criteria and algorithms for various performance measures. As shown in Fig. 6a, a transaction for a query has three parts: an identification function \( f \) that selects the state or event variables of a specific object to be monitored, a data transformation function
(F) for performance evaluation through statistical inference, and an inference rule (R) to be applied to the abstracted data. The data space generated by monitoring a state or an event variable is denoted by the circle \( F_i \) in Fig. 6a. It is created after activating the sensor associated with a state or an event variable. The data transformation function (F) (derived from the set of statistical operators) is applied on the data space \( F_i \) to abstract information from the history of a state or an event variable. Application of F generates a data space \( G_i \) that consists of only statistical information. If the statistical abstraction is not needed then F is reduced to the identity operator. The inference rule R operates on the data space of \( G_i \) to further evaluate the statistical information, e.g., for event detection through threshold crossing. If no such operation is needed, then the rule R is reduced to the identity operator.

\[
\text{QUERY} = \begin{bmatrix}
F \\
R
\end{bmatrix}
\]

\[
\text{SENSOR} \rightarrow F_i \rightarrow \begin{bmatrix}
G_i \\
R
\end{bmatrix} \rightarrow D_i
\]

Fig. 6a. Decomposition of a simple query about a state variable.

The deductive inference rule is used to decompose a query into a set of simple queries and then aggregate the information received by servicing the simple queries. For example, in order to find the total average time delay of a call, a set of sensors at the nodes along the route of the call is to be activated to measure the time delay at each node. Once the average time delay from every node is available, they are aggregated to compute the total average time delay. Therefore, a query for the average time delay of a call will be divided into multiple simple queries and appropriate sensors will be activated to measure the average time delay at each node along the route of the call. The function \( f(D_1^N, ..., D_N^N) \) in Fig. 6b represents the deductive part of the query for data aggregation and it is applied after the data is collected from the appropriate sensors.

\[
\text{SENSOR} \rightarrow \begin{bmatrix}
F_1 \\
F_N
\end{bmatrix} \rightarrow \begin{bmatrix}
G_1 \\
G_N
\end{bmatrix} \rightarrow \begin{bmatrix}
R_1 \\
R_N
\end{bmatrix} \rightarrow \begin{bmatrix}
D_1^N \\
D_N^N
\end{bmatrix}
\]

\[
f(D_1^N, ..., D_N^N) = \text{inference rule based on } (D_1^N, ..., D_N^N)
\]

Fig. 6b. Decomposition of a compound query about state variables of multiple object.

We may also want to find out whether a certain average throughput-time delay condition, at a buffer of a node, is satisfied. In this case, we need to activate sensors for both the throughput and the average time delay and send an event indication if the average throughput-time delay condition is not met. In order to do that, the original query will be divided into two simple queries based on the throughput and the time-delay to be computed. Fig. 6b represents such cases where the original query is divided into multiple simple queries identifying each of the state variables to be measured. \( f(D_1^N, ..., D_N^N) \) indicates the function that generates an event if the throughput-time-delay condition is not met.

Fig. 6c represents the case when a state variable is monitored for status monitoring and event reporting or multiple event reporting. \( f(D_1^N, ..., D_N^N) \) represents any deduction to be done after the data is collected. One such deduction scheme is the correlation between two events generated from the same variable. This scheme can also be used to define higher level events based on the history of event variables.

\[
\text{SENSOR} \rightarrow \begin{bmatrix}
F_1 \\
F_N
\end{bmatrix} \rightarrow \begin{bmatrix}
G_1 \\
G_N
\end{bmatrix} \rightarrow \begin{bmatrix}
R_1 \\
R_N
\end{bmatrix} \rightarrow \begin{bmatrix}
D_1^N \\
D_N^N
\end{bmatrix}
\]

\[
f(D_1^N, ..., D_N^N) = \text{inference rule based on } (D_1^N, ..., D_N^N)
\]

Fig. 6c. Decomposition of a compound query about state variables of same object.

### 4 THE OBJECTIVE-DRIVEN MEASUREMENT STRATEGY

In the process of performance management, a set of objectives often lead to asking a specific set of questions about the network. The questions of interest could be: Does the network support a specified performance or can the network provide enhanced performance? They can be answered after appropriate monitoring functions are incorporated into the system and the observed data is processed.

Monitoring as a process determined by objectives is called objective-driven monitoring. Objective-driven monitoring is closely related to the concept of experimental frame of [17] that characterizes modeling objectives by specifying the form of experimentation that is required to obtain answers to the questions of interest.

A query to collect information from the network can be submitted to the monitor either by an user from a terminal or by the various knowledge specialists responsible for network control and management. Such agents are called Query Generators (as shown in Fig. 2). The submitted query can be of two types: real-time data query and nonreal-time data query. The real-time data query represents the query to those objects whose attributes are updated using the sensors located in the various subsystems of the network. The nonreal time data query represents the query to those objects whose attributes' value do not depend
upon the sensory information. The nonreal-time queries are handled based on the information available in the knowledge base of the monitor.

The real-time data queries are handled by using the deductive query processing technique, where the inference and retrieval phases of the query have been separated [22], [15]. The schematic of such query processing is shown in Fig. 2. Based on the relationships established in the knowledge database between the various objects, the monitor decomposes the requested query into a set of simple queries and analyzes them to determine specific state and event variables that need to be monitored. Once the state and event variable is identified, corresponding sensors in the network are activated. Activated sensors then collect information about the state and event variables and thus update the dynamic database shown in Fig. 3. If the query requires statistical abstraction of the collected information, then the statistical inference processor applies the corresponding operators on the state and event variables and updates the statistical database.

As described before, a plausible way to find the total average time delay of a call is to activate sensors to measure the time delay at the nodes along the route of the call and then aggregate the average time delays. A more elegant solution, however, is to first obtain a derived object that contains all the state variables that exhibit the average time delay of the intermediate nodes along the route of the call. A derived sensor can be attached to this object and finally adding up the time delays leads to the required result. Thus, to answer a query only a restricted data space, called an observation frame, is needed. The observation frame contains only the state and event variables, the performance parameters, and the derived sensor and its components. In the next section a general methodology for an objective-driven measurement strategy is described.

4.1 Deductive Inference

Real-time control algorithms for resource allocation operate based on a set of cost functions and a set of constraints on the behavior of the variables of a system. These system variables could be either describing the state of the system or a statistical abstraction of its state. Thus, a control task first leads to monitoring the system variables. In the case of integrated networks, QOS parameters define the target operating points and maintain these QOS parameters near the operating point becomes the control objective of network operations.

Based on the specified QOS parameters, a set of performance parameters are identified. The difference between the QOS parameters and the performance parameters is that the latter depend on the systems architecture of the network. From the specification of the QOS parameters, the corresponding performance parameters are derived by the deductive processor based on the knowledge about the system architecture of the network. The latter resides in the configuration database. As an example, the maximum average end-to-end time delay might be a QOS parameter. The average time delay experienced by a call in a given network is the performance parameter associated with it. It is the aggregation of all the average time delays at nodes and links along the route of the call.

Thus, the request for monitoring a QOS parameter is a query consisting of the class name of an object and the corresponding performance parameter. This general query can be made specific by providing values for one or more key attributes of the object class. Based on the submitted query, the deductive inference processor identifies specific objects and the performance parameters that need to be monitored. First, all the objects in the knowledge base that contain the appropriate performance parameter are identified. Second, the instances of the performance parameters associated with the selected objects are identified. Third, the instances of state and event variables associated with the selected instances of the performance parameters are identified.

For each of the selected objects a component object is created and relationships are established between the new object and the selected performance parameters and the corresponding state and event variables. In order to monitor the selected objects, a derived sensor is associated with each of the component objects. The creation of the derived sensor generates the instances STATUS_SENSOR and/or EVENT_SENSOR for each of the state and event variables associated with the component object. Creation of the STATUS_SENSOR or EVENT_SENSOR activates the corresponding primitive sensors installed in the network. The component object, the associated derived sensor, and the collected information represent a data space called observation frame. Thus, the observation frame forms a restricted data space. The answer to queries is obtained by examining, processing, and aggregating monitored information in this space. The statistical inference process takes place only after data has been collected by the sensors.

4.2 An Algorithm for Objective-Driven Monitoring

The operation of the deductive inference processor described above can be formalized into an algorithm consisting of the following steps:

1) Identify the instances of the object class specified in the query;
2) Identify the instances of the performance parameter specified in the query associated with the selected objects;
3) For each of the selected objects, identify the instances of state and event variables associated with the selected performance parameters;
4) Create a new object and associate with it the selected performance parameters and the state and event variables of all the selected objects;
5) Associate a derived sensor with the new object and create sensors that monitor the state and/or event variables;
6) Activate the sensors in the network;
7) Apply statistical inference procedures to evaluate the performance parameter.

The above steps are in part adopted from the "objectives-driven" methodology for modeling of systems [17].
The main goal of the above algorithm is to automatically identify the network sensors to be activated in order to compute a generic performance parameter associated with a given object class. Fig. 7, describes the semantic network associated with an instance SWITCH_BUFFER_0.1.1. In step 1, for a given object class name and and key-values, we identify the instances of the object class that matches the description. Then in step 2, we identify the instance of the performance parameters that matches the generic performance parameter specified in the query. For example, if we are interested in the THROUGHPUT of a selected instance SWITCH_BUFFER_0.1.1, then THROUGHPUT_SWITCH_BUFFER_0.1.1 will be selected. Given a specific instance of a performance parameter, in step 3 we identify the corresponding instance of variable associated with the object. In Fig. 7, the variable PACKET_OUTOF_BUFFER_0.1.1 is associated with THROUGHPUT_SWITCH_BUFFER_0.1.1. In step 4, we create an instance of an object class OBJVIEW, as shown in Fig. 8 and associate the selected performance parameters and the corresponding variables with the new object. In step 5, we create an instance of DERIVED_SENSOR and associate it with the instance of OBJVIEW. A side-effect of association of a derived sensor with an object is that it will create instances of primitive sensors (STATUS_SENSOR or EVENT_SENSOR) to monitor the variables associated with the object. The instances of OBJVIEW and DERIVED_SENSOR together form the OBSERVATION_FRAME. In step 6, instances of primitive sensor will send message to the network to activate sensors installed in the network. In step 7, statistical inferences are applied to compute the value of the performance parameter from the network measurement and sent to the observation frame in the knowledge base.

4.3 Examples

Two examples are given for objective-driven monitoring. The first shows the evaluation of the throughput of a buffer while the second gives the evaluation of the time delay of a call.

4.3.1 Monitoring a Traffic Buffer

In the following example, it is shown how to monitor the average throughput of a buffer at a network station of MAGNET II [23] network. Let us assume that the query requests the THROUGHPUT of SWITCH_BUFFER with following key attribute-value pair: buffer_id = 1 at node_no = 0 and st_no = 1.
and the activation of sensor causes start of the measurement of the variable. Once the measurement is completed and the statistical operators are applied, the value of the throughput performance parameter is sent back to the knowledge base.

### 4.3.2 Monitoring the Time Delay of a Call

The maximum time delay of a call appears as a QoS constraint for the Class-I traffic of MAGNET II. In order to guarantee that this requirement is met, the time-delays of Class I calls are requested.

Let **call** be the object class that represents a call with key attributes **call_id**, **user_id**, and **traffic_class**. Let the two users of a call **call** be A and B and the values of attributes **call_id**, **user_id**, **traffic_class** be A, B, and I respectively. The association between a call and the corresponding nodes and links along the route of the call is needed. Since a node has multiple access points, the description of the route needs to include the name of the input-output buffers at all the nodes. The relationship type **has-vcct-route** associates the buffers and servers (links and switches) along the route with the call. Thus, the route for the call between the users A and B (shown in Fig. 9) will contain the following objects: \( B_{h,j,k}, B_{h,j,k}, B_{j,h,k}, \) and \( B_{j,h,k} \) as buffers, \( N_h \) and \( N_j \) as switch fabrics, and \( L_{j,h} \) as links, where the indices \( j \) and \( k \) indicate the node number, \( t \) indicates the access points at the node, and \( k \) indicates the traffic class. The relationship type **has-vcct-route** introduces another relationship **has-component-object**, to reflect the fact that buffers, switches, and links form part of the call. Both relationship types **has-vcct-route** and **has-component-object** are included as multi-valued attributes in the list of attributes of the object class **call**.

![Fig. 9. The description of the route of a call between two users.](image)

Since the maximum time delay of a call will be the aggregation of the maximum time delays of the buffers and servers along the route of the call, an aggregate performance parameter is defined to represent the maximum time delay of the call. The new object class is called **aggr-perf-parameter**. It is defined as a subclass of **perf-parameter**, but has an additional procedure associated with it for aggregating the values of its component performance parameters. The time delay of the object class **call** is defined as **time-delay-call** which is a subclass of **aggr-perf-parameter**. The value of the **max-value** of **time-delay-call** is the sum of the values of the **max-value** of its component performance parameters.

In Fig. 10, the description of the **call** and its time delay performance parameter, **time-delay-call**, are shown. The object **time-delay-call** is declared as a subclass of **aggr-perf-parameter** and associated with the generic object **call**.
In order to monitor the maximum time delay of the call, the following objective is defined:

Find the maximum time delay of a call between the pair of users A and B.

Based on the algorithm for objective driven monitoring, the steps for computation of the average time delay of a call can described as follow:

1) Create an observation frame OBJ_VIEW_CALL_A_B_1 for CALL_A_B_1 with:
   - HAS-PERF-PARAMETER
   - HAS-VCKT-ROUTE
   - TIME_DELAY_CALL_A_B_1
     - TIME_DELAY_SWITCH_BUFFER_h,s,k
     - TIME_DELAY_SWITCH_BUFFER_h,s,t
     - TIME_DELAY_SWITCH_BUFFER_h,t,s
     - TIME_DELAY_SWITCH_BUFFER_h,t,k
     - TIME_DELAY_SWITCH_BUFFER_h,s,t,k
     - TIME_DELAY_SWITCH_BUFFER_h,s,k,s
     - TIME_DELAY_SWITCH_BUFFER_h,t,s,s
     - TIME_DELAY_SWITCH_BUFFER_h,t,k,k
   - HAS-OBJ-VIEW-PERF-PARAMETER

2) For each TIME_DELAY_VAR_SWITCH_BUFFER_h,s,t,k send a message to the corresponding sensor at the buffer to compute the maximum time delay. The location of the variable is given by the values of h, s, t, and k and the buffer is identified based on the buffer_id (equal to 1).

3) Once the value of TIME_DELAY_SWITCH_BUFFER_h,s,t,k is computed, all the values are sent to OBJ_VIEW_CALL_A_B_1.

4) When all the TIME_DELAY_VAR_SWITCH_BUFFER_h,s,t,k are available, TIME_DELAY_CALL_A_B_1 is computed based on (4.1).

\[
\begin{align*}
&\text{TIME_DELAY_CALL_A_B_1.max\_value} \\
&= \sum_{k=1}^{R} \text{TIME_DELAY_SWITCH_BUFFER_h,s,t,k}_k \cdot \text{max\_value} \\
&+ \sum_{k=1}^{R} \text{TIME_DELAY_SWITCH_BUFFER_h,s,t}_k \cdot \text{max\_value} \\
&+ \sum_{k=1}^{R} \text{TIME_DELAY_SWITCH_BUFFER_h,s,t}_k \cdot \text{time\_delay} \\
&+ \sum_{k=1}^{R} \text{TIME_DELAY_SWITCH_BUFFER_h,s,t}_k \cdot \text{time\_delay,}
\end{align*}
\]

where \( R \) is the total number of nodes in the route of CALL_A_B and LINK_h,s \_time\_delay is the fixed transmission time delay of the link LINK_h,s.
ACKNOWLEDGMENTS

The research reported here was supported in part by the National Science Foundation under Grant CDR-84-21402 and in part by the New York State Center for Advanced Technology under Project NYSTF CAT (84)-15 005.

REFERENCES


