

# A Dynamic Group Management Framework for Large-scale Distributed Event Monitoring

*Ehab Al-Shaer*

*Multimedia Networking Research Laboratory*

*School of Computer Science, Telecommunications and Information Systems*

*DePaul University*

*Chicago, IL 60604*

*ehab@cs.depaul.edu*

## **Abstract**

Distributed event monitoring is an important service for fault, performance and security management. Next generation event monitoring services are highly distributed and involving a large number of monitoring agents. In order to support scalable event monitoring, the monitoring agents use IP multicasting as a group communication for exchanging events and control information. However, due to the dynamic nature of event detection and correlation during the monitoring process, agents groups organization and coordination such as membership and tasks assignments becomes a challenging issue. Thus, supporting an appropriate group management infrastructure is a key issue for developing scalable event monitoring services. This paper presents an efficient group management framework based on IP multicast that dynamically reconfigures the group structures and membership assignments at run-time according to the event correlation requirements which allows for optimal delivery of multicast messages between the management entities. This framework provides techniques for solving agents' state synchronization, free-collision group allocation and agents bootstrap problems in distributed event monitoring. The presented framework has been implemented within HiFi monitoring system which is a distributed hierarchical monitoring system.

## **Keywords**

distributed monitoring, event correlation, IP Multicast.

## **1 INTRODUCTION**

Event monitoring is the process of capturing and correlating events notifications from different sources in a distributed environment. In distributed

management environments, large number of events (notifications or alarms) are generated by network or system components during their execution and interaction with external objects (e.g. users or processes). These events are detected, classified, filtered and analyzed to accurately determine the actual cause of the problems such as faults, security threats or performance bottlenecks.

Next-generation event monitoring must be scalable and dynamic to handle large number of managed objects with minimal perturbation. In distributed event monitoring, a group of monitoring agents are used to exchange event notifications and perform correlation collaboratively. In other words, an agent may need to forward event notifications to a group of agents for further analysis or sometimes to a group of managers interested in this event. Similarly, managers also communicate with a group of agents in order to distribute the event correlation tasks. Using group communication is evidently necessary in this environment to improve the scalability and performance of distributed event monitoring. However, because agents' group membership must be dynamically changed based on the event notification and correlation process, a highly configurable and dynamic group management is required in order to provide efficient group communications.

In this paper, we present a dynamic group management framework based on IP multicast to support scalable distributed event monitoring. The proposed framework uses the event correlation information to dynamically re-configure the multicast group formation (i.e., join and leave). While multicasting minimizes the number of messages and processing delay compared with point-to-point communication, the proposed group management allows for the optimal formation of multicast groups in distributed event monitoring systems.

Although several distributed event correlation techniques were proposed (e.g., [5, 8, 9, 10, 17]), exploiting multicast group communication to improve the scalability and performance of event correlation is not sufficiently addressed. On the other hand, number of studies proposed integrating group communication in distributed management [7, 11, 12, 14]. However, as discussed in Section 3.3, none of such proposals have addressed the challenging issues of dynamic group management as described above which limits the advantage of group communication in these systems.

This paper is organized as follows: Section 2 gives an overview of HiFi monitoring system which includes the model, language and the architecture; Section 3 presents our dynamic group management and communication framework for distributed event monitoring, Section 3.3 presents the agents synchronization and state consistency protocol; Section 3.3 explains the automatic bootstrap mechanism for establishing the agents' hierarchy; Section 3.3 discusses related work; Section 3.3 presents the summary and concluding remarks.

## 2 HIFI MONITORING SYSTEM OVERVIEW

The presented group management and communication framework was implemented as part of HiFi monitoring system described in [1, 2]. Although we use HiFi throughout our discussion in this paper as a case study for this framework, the presented group management framework and techniques can be adopted in any multicast-based distribute mangement system.

### Monitoring Architecture and Language:

In HiFi, managers (called *event consumers*) specify their monitoring demands by sending a *filter* program dynamically via the *subscription process* which configures the monitoring agents accordingly. A filter is a set of *predicates* where each predicate is defined as a boolean-valued expression that returns *true* or *false*. Predicates may be joined by *logical operators* (such as AND and OR) to form an expression. A typical HiFi filter consists of three major components: the *event expression* which specifies the relation between the interesting events, *filter expression* which specifies the event attributes value or the relation between the attributes of different events, and the *action* to be performed if both event and filters expressions are *true*. If an event is detected, the action specified in the filter is performed such as forwarding the monitoring information to the corresponding consumers. For example, assume a manager requests detecting events, `highCPULoad` and `PktLoss`, if they occurred in the receiver and sender machine respectively while the sender is transmitting to the receiver (i.e., *SrcMachine* is the same as the *DestMachine*) in a distributed environment. This is represented in the following filter:

```
FILTER= [(highCPULoad ^ pktLoss)];  
[(highCPULoad.SrcMachine=pktLoss.DestMachine)];  
[FORWARD];PktLoss.Reason_Filter.
```

HiFi also provides the Environment Specification Language (ESL) which the managers use to describe the application distribution in the network such as process and domain locations.

The task of detecting primitive and composite events is distributed among dedicated monitoring programs called *monitoring agents* (MA). HiFi has two types of MAs: *local monitoring agents* (LMA), and *domain monitoring agents* (DMA). The former is responsible of detecting primitive events generated by local applications in the same machine while the latter is responsible of detecting composite events which are beyond the LMA scope of knowledge. One or more event producers (i.e., processes) are connected to a local LMA in the same machine. Every group of LMAs related to one domain (geographical or logical domain) is attached to one DMA. These DMAs are also connected to higher DMAs to form a hierarchical structure for exchanging the monitoring information.

**Subscription Process:** The monitoring process starts by a consumer send-

ing a *filter program* that describes the monitoring request to the local MA. The filter is validated and decomposed into subfilters through the *decomposition process* in such a manner that each one represents a primitive event [2]. The filter expression is also decomposed into subexpressions where each one is contained within a domain. Then, each decomposed subfilter or subexpression is assigned to one or more LMAs or DMAs through the *allocation process* based on the event sources and application distribution. This fine grain decomposition and allocation provides for an optimal distribution of the monitoring tasks and minimal event propagation. In order to facilitate the event generation process, HiFi provides the Event Reporting Stub or ERS routine which is a library linked with the program during compilation. During the program execution, ERS constructs and generates the triggered events as they occurred and sends them to its LMA via UNIX sockets or pipes [15].

### 3 GROUP COMMUNICATION FOR DISTRIBUTED EVENT CORRELATION

We use IP multicast for group communications between HiFi monitoring agents. IP multicast is considered the de facto standard of multi-point group communication in the Internet. The monitoring agents and the managers use the Reliable Multicasting Server (RMS) described in [3] for this purpose. This section describes the dynamic group management framework and communication as implemented in HiFi.

#### 3.1 LMA-DMA Group Communication

This communication is used to forward detected primitive events from LMAs to DMAs. After the decomposition and allocation process, each LMA knows which primitive event to forward and each DMA knows which primitive event to request in order to evaluate the delegated subexpression. As a result, every DMA uses the names of the requested events to form/join multicast groups such that the event name is the prefix and “Grp” is the suffix ( $\langle \text{primEventName} \rangle \text{Grp}$ ). For example, if a DMA requires *EventX* notification in order to evaluate delegated subexpression, this DMA joins multicast group called *EventXGrp*. Similarly, LMAs use the names of detected events as the group names to which event notifications are multicast. For example, if *EventX* event is detected by an LMA, this LMA multicasts this event to the *EventXGrp* group. Consequently, this causes the *EventX* event to be forwarded to all DMAs interested in receiving this event. The event and filter information such as event names and filter subexpressions are distributed by the manager after the decomposition and allocation processes and during the agent hierarchy bootstrap as described in Section 3.3.

Moreover, a DMA may occasionally need to multicast control or administrative information to other LMAs in the same domain for load adaptation or

accommodating new application entities [4]. In order to facilitate this domain-based communication, LMAs within the same domain (i.e., sharing the same DMA) join a multicast group designated by the domain name as a prefix and “Grp” as a suffix. The domain information such as domain names and managed objects distribution is specified by the manager using the Environment Specifications Language (ESL).

### 3.2 DMA-DMA Group Communication

After the decomposition and allocation, each DMA evaluates its delegated subexpression and forwards the evaluation result to one or more higher DMAs in the hierarchy which, in turn, use this result to complete the evaluation of its own filter (sub)expression. Therefore, in order to facilitate this group communication between the DMAs, every DMA joins a multicast group for each delegated subexpression. The names of the multicast groups are directly derived from the subexpression itself such that the expression is used as a prefix after replacing AND and OR operators by “A” and “O” respectively, and “Grp” is used as a suffix. For example, if the subexpression  $E_1 \wedge E_2$  is delegated to  $DMA_x$  and the evaluation result must be forwarded to  $DMA_y$ , then  $DMA_x$  joins  $E_1Grp$  and  $E_2Grp$  multicast groups as described in Section 3.1. to receive information about these events, and  $DMA_y$  joins the multicast group  $E_1AE_2Grp$  to receive information about the subexpression evaluation results. Consequently,  $DMA_x$  sends the evaluation result of this subexpression to  $E_1AE_2Grp$  group. The group communication between DMAs continues to deliver the partial evaluation results from one level to another in the hierarchy until the entire filter expression is evaluated which ends the the event correlation process.

### 3.3 Managers-Agents Group Communication

HiFi monitoring systems allows a group of managers to share the results of monitoring operations in a scalable manner. The monitoring agents forward event notifications to one or more managers according to their subscription requests. On the other hand, a manager may communicate with a group of agents in order to distribute monitoring demands. Therefore, many-to-many communication model is required between managers and agents. In our group management framework, every manager joins  $MgrGrp$  multicast group. LMAs and DMAs agents use this group to send control or administration information to managers. Moreover, each manager joins multicast groups based on its submitted filters such that the filter name is used as a prefix and “Grp” as a suffix in the group name ( $\langle FilterName \rangle Grp$ ). More than one manager can share the same filter (e.g., MyFilter) by simply joining the same multicast

*Input:* A filter specification (FX, and a filter name)  
*Output:* 1 or 0 for successful and unsuccessful submissions

```
SafeSubmit(FilterInfo)
  if (Check_MKB(FilterInfo.Fname)== NOTFOUND)
    SendMcastToMgrGrp(FilterInfo);
    Wait( $RTT * REXMT_{max} + \alpha$ ); /* Receive while waiting */
    if (Check_MKB(FilterInfo.Fname) == NOTFOUND)
      Submit(FilterInfo);
      return 1;
    else
      Wait( $RTT * REXMT_{max} + \beta$ ); /* Receive while waiting */
      if (Fname_was_Submitted)
        return 0;
      else
        HighestIP = GetHighestIP();
        if (HighestIP <= ThisIP)
          Submit(FilterInfo);
          return 1;
        else
          return 0;
  else
    return 0;
```

**Figure 1** Safe Filter Submission Algorithm

group associated with this filter name (*MyFilterGrp*). When the requested event correlation of a filter is detected, the monitoring agents send a notification to  $\langle FilterName \rangle Grp$  group of this filter (e.g., *MyFilterGrp*). Thus, managers joining this group receive this notification simultaneously. However, this process may cause two or more managers submitting different filter correlation to join the same multicast group because they randomly select the same filter name. To avoid this group conflict between managers, the name of a submitted filter must be unique in the monitoring environment. In other words, if a manager submits a filter successfully, no other manager is allowed to submit another filter with the same name. Only the manager who created the filter is permitted to modify or delete a filter.

One way to provide a collision-free multicast group allocation is to use distributed domain proxies as described in [13]. However, this technique will be inefficient in our environment because managers are usually located at different domains which equires allocating a proxy for each manager. This is obviously not scalable. This is in addition to the complexity of maintaining proxies agents in distributed environment. For this reason, We developed a simple fully distributed protocol to allocate filter names (i.e., multicast groups) exclusively by managers. The algorithm of this protocol is outlined in Figure 1 and described below.

Before a manager submits a filter, the manager first search for the filter

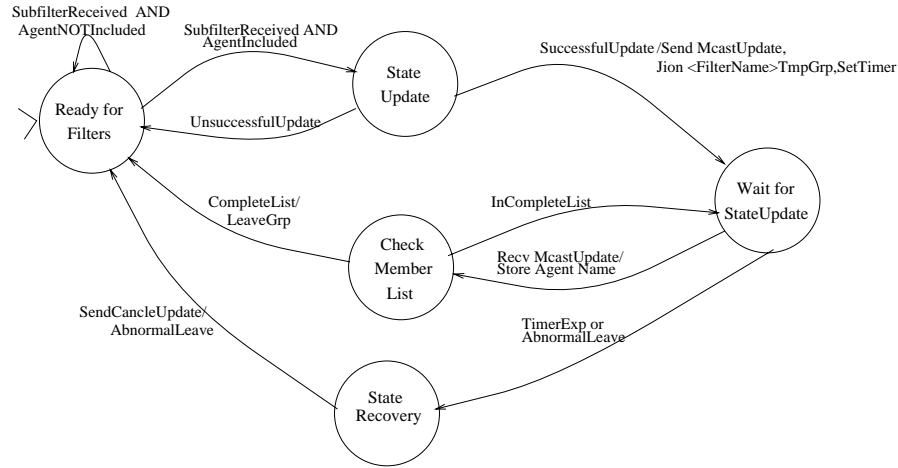
**Table 1** Monitoring Agents' Multicast Groups

Group Name	Member	Sender	Forwarded Information
<i>MgrGrp</i>	manager	manager	admin & group synchronization
	manager	<i>DMA_root</i>	agents' hierarchy confirmation
<i>DMAGrp</i>	DMA	manager	event,filter and environment
<i>LMAGrp</i>	LMA	manager	event,filter and environment
<i>&lt; EventName &gt; Grp</i>	DMA	LMA	primitive events notification
<i>&lt; FilterName &gt; Grp</i>	manager	DMA or LMA	event correlation
<i>&lt; DomainName &gt; Grp</i>	LMA	DMA	control requests
<i>&lt; SubExpr &gt; Grp</i>	DMA	DMA	FX subexpression evaluation

name in its local monitoring knowledge-base (*Check\_MKB()*). If it is not found, then the manager multicasts the filter name and specification such as filter correlation and sharing status to the *MgrGrp* group. When managers receive this information, they immediately add it to their local knowledge-base along with the IP address of the sending manager. Therefore, managers are always aware of the active filters exist in the monitoring environment. After multicasting the filter information, the manager waits for more than the maximum allowable re-transmission timeout period ( $RTT_{max} * REXMT_{max} + \alpha$  where  $\alpha$  is the maximum database update/check time). During the waiting period, managers do receive and process multicast messages. When timer expires, managers re-check the local database again in order to insure that no other manager is attempting to use the same filter name recently. If the second check passes successfully, then the manager submits the selected filter name which results in multicasting the filter information to *MgrGrp* group. Otherwise, another manager have submitted a filter with the same name simultaneously. In this this case, the manager waits again until it receives the submitted filter from *MgrGrp* for a timeout period ( $RTT_{max} * REXMT_{max} + \beta$  such that  $\beta = \alpha +$  submission time) or the manager that has the highest IP number, *HighestIP*, (and port number if they are from the same machine) is granted this filter name. In the latter case, managers update their MKB at the same time and they all find a match in the local KMB.

On the other hand, when a manager deactivates a filter (e.g., MyFilter), it then leaves *MyFilterGrp* and sends a multicast message to *MgrGrp* to indicate this deactivation which causes managers to delete this filter from its local knowledge-base.

To enable managers-to-agents group communication, both LMAs and DMAs use RMS to join *LMAGrp* and *DMAGrp* multicast groups respectively. Managers forward the monitoring information such as events, filters and application environment to the LMAs and DMAs via multicasting it to *LMAGrp* and *DMAGrp* respectively. However, when a manager multicasts the filter decomposition information to *LMAGrp* and *DMAGrp* groups, it includes



**Figure 2** Agents State Consistency Protocol State Diagram.

in the message the delegated tasks and the names of the agents assigned for these tasks. The agents use this information to extract their assigned tasks accordingly.

#### 4 AGENTS STATE CONSISTENCY PROTOCOL

Managers start the subscription process via adding, modifying or deleting filters on-the-fly. Because adding, modifying or deleting filter components can be performed on different agents concurrently, the agents' state might become inconsistent due to communication delay or failures. To solve this problem, the agents' state update must be *atomic* (i.e., an agent failure causes all other agents to abandon the process) and *timely synchronized* (i.e., agent waits until all participating agents commit the state update). We developed agents state consistency protocol as part of the subscription process to provide an atomic and synchronized subscription. The protocol is based on reliable multicasting (RMS) and group management described in Section and its state diagram is depicted in Figure 2. The *subscription component* in the manager program parses, decomposes filter programs, and multicasts the delegated tasks (subfilters) to the monitoring agents. The delegated subfilter messages contain the *filter name*, the *decomposed subfilters*, and a *list of agents IDs* needed for this monitoring task. When an agent receives a Subfilter message that contains its ID (*MachineName.DomainName*), it performs the *filter composition* to insert the delegated subfilters into its internal filtering representation (*StateUpdate* state). If the agent updates its state successfully, it uses the filter name included in the message to join  $\langle FilterName \rangle TmpGrp$  multicast

group and sends a join notification (or McastUpdate) to this group. When the other agents in the group receives McastUpdate message, they add the sender agent's ID to their local repositories (MemberList). Similarly, agents continue adding IDs into their local repositories until it contains all agents found in the subfilter message. This implies that all agents found in the subfilter message have completed their state update successful and thereby they can leave  $\langle FilterName \rangle TmpGrp$  multicast group and activate the delegated filters.

On the other hand, if an agent fails to update its state or join a group, one agent at least will time out (i.e., timer expires) and send a multicast message to the  $\langle FilterName \rangle TmpGrp$  group to cancel and recover the state update. Every agent sets up a timeout timer right after joining  $\langle FilterName \rangle TmpGrp$  group for a time period specified as:

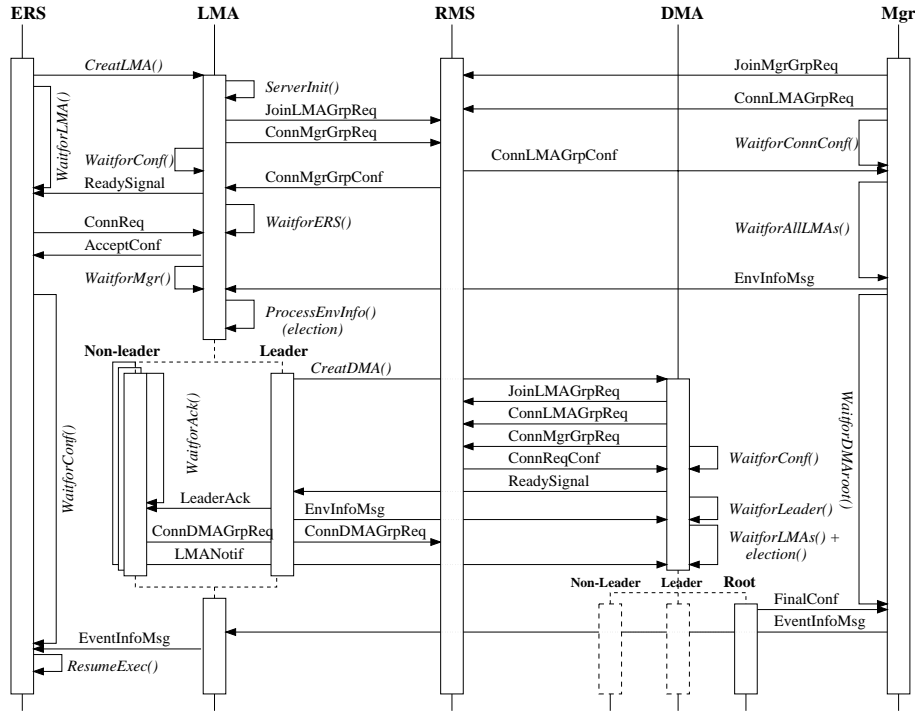
$Timeout = (2 * REXMT_{max} * RTT_{max}) + \alpha$  such that  $RTT_{max}$  is the maximum round trip delay in the network,  $REXMT_{max}$  is the maximum number of retransmissions in RMS. Therefore,  $2 * REXMT_{max} * RTT_{max}$  is the maximum aggregate retransmissions time in the network for both the Subfilter message and the McastUpdate. And  $\alpha$  is the maximum state update processing time. Notice that the  $Timeout$  expression represents the worst case so that agents never times out before the completion of the protocol operations successfully. In case of agents' crashes, RMS protocol detects this event and notifies all members in the group instantaneously which causes the agents to revoke the state update operation.

At the end of this operation, managers get notified about the results of their subscription, (i.e., confirmed or aborted) by a monitoring agent which sends the result the to  $\langle FilterName \rangle Grp$  group. One of the main advantages of this protocol is simplicity and minimal overhead compared to other distributed algorithms such as two-phase commit protocol.

## 5 AUTOMATIC AGENTS BOOTSTRAP MECHANISM

To avoid the complexity of constructing and administrating the agents hierarchy and groups, we developed a protocol based on the group management framework described before that automates creating, allocating and setting up the agents' hierarchy dynamically without the users involvement. This service significantly facilitates administrating HiFi management system. Figure 3 shows the *interaction diagram* between HiFi entities performing this protocol. In the following, we give a brief description for this agents' hierarchy bootstrap protocol.

1. (*Manager Program starts.*) When a manager program starts it joins MgrGrp and connects to LMAGrp (as described before in Section , *connect* is used to send to a multicast group reliably but *join* is used for receiving and sending to multicast groups). Then, the manager waits for *all* LMAs to start and connect to MgrGrp. RMS sends a notification to the manager (MgrGrp) whenever a member joins or connects to a group.



**Figure 3** Automatic Agents' Hierarchy Bootstrap Protocol.

2. (*Instrumented Program starts.*) When the instrumented program starts, the ERS creates (`fork()`\*) an LMA and waits for *Ready* signal from this LMA.
3. (*LMA starts and LMA-ERS connection established.*) After the LMA starts it joins LMAGrp and sends a SIGUSR UNIX signal to ERS which consequently establishes a UNIX socket connection [15] with the LMA. When all LMAs are created and ready, the manager multicasts the environment information (EnvInfo) to LMAGrp. Notice that the managers know about the total number of LMAs from the environment specifications (ESL). Late-join LMAs can also get connected as described in [4].
  - (a) (*LMA election process starts.*) Upon receiving EnvInfo from a manager, LMAs go through an election process based on the position of LMA name/ID in the EnvInfo table. In particular, the first LMA name in the LMAs list of each domain is the LMA leader. As a result, LMAs are divided into two groups: a *leader group* that contains the LMA leaders

\*We assume that the agents binary exists in the same location or file server where the monitored programs exist.

for all domain, and a non-leader group that contains the other LMAs in each domain.

- (b) Each LMA leader creates the designated DMA for this domain, forwards the environment information to it, and sends an acknowledgment to non-leader LMAs to announce DMA creation. Upon receiving this acknowledgment, non-leader LMAs connect to the DMAGrp group.
4. (*DMA starts and the election process.*) After all LMAs in the domain are connected to the DMAGrp, the DMAs go through the same election process used by the LMA which classifies DMAs into: DMA leader, DMA non-leader and DMA root. The first two groups (DMA leader and non-leader) follow the same steps described for LMA leader and non-leader. This implies that every DMA leader creates its containing or higher DMA (called superDMA) and this hierarchical construction continues until DMA root is created. When the DMA root starts, it immediately sends a final confirmation to the manager (MgrGrp) confirming the completion of the agents' hierarchy.
5. The Manager then multicasts the event information (EventInfoMsg) to the LMAGrp and ERS. ERS uses the events information to construct and send events notifications and LMAs use the event information to join primitive event multicast groups as described in Section .
6. ERS resumes the program execution and the event reporting process. The LMAs and DMAs are completely set up in their groups and ready for filter delegations.

The protocol scales well with the number of agents because all DMAs at the same level in the hierarchy and all LMAs operate concurrently, and the effect of the hierarchy height is minimal. It is important to mention that process crashes or abnormal leaves from the multicast groups (ERS, LMAGrp, DMAGrp and MgrGrp) are immediately propagated to the rest of the agents and causes the agents to quit and abandon this process. This guarantees that the final confirmation is sent only if the agent hierarchy is constructed successfully.

## 6 RELATED WORK

Our related work study focuses on the systems that attempt to develop or exploit group communication for distributed network management applications.

A framework for using IP multicast group communication with SNMP is proposed in [14]. This framework provides a primitive group membership structure. It also uses an election algorithm to choose a master agent that facilitates the group communication between other SNMP agents. The SNMP

trap messages over IP multicast are utilized for exchanging control information. This framework is an example of developing autonomous SNMP agents using IP multicasting. However, it does not present a general framework for integrating IP multicasting in standard SNMP agents because (1) it lacks the flexibility of re-configuring the multicast group structure and the communication model dynamically based on the application needs, (2) it requires a major changes in the SNMP agent in order to use this framework, and (3) due to the periodic trap messages and the election process, this approach may cause a considerable overhead on the SNMP agent.

In [11], a reliable group communication protocol for distributed management that preserves message ordering and atomicity was described. This protocol uses a hierarchy of servers and logical timestamps to ensure reliability and causal ordering of group delivery. It seems that this architecture was developed over unicast connections, instead of employing IP multicasting, to emulate group communication. This significantly limits the performance and scalability of the proposed architecture. Furthermore, this work does not address the issue of group management of monitoring agents which is the main theme of this paper.

Another group communication infrastructure based on Transis [7] group communication system was proposed in [6]. It supports an efficient solution for some distributed system management applications such as software installation, simultaneous remote execution and configuration management on a cluster of servers. A monitor program uses Transis to communicate with a group of management servers that perform system management tasks reliably and consistently. A similar framework is proposed in [12] to use IP multicasting for control and management of distributed applications. Although these systems show the advantage of using group communication for some targeted management applications, they do not provide a general group management and communication framework for distributed event filtering applications. In addition, using a distributed systems toolkit such as Transis as a core element limits the usability of the system particularly for Internet-based network management.

In conclusion, as we explored, employing an efficient group management and communication for distributed event correlation was not sufficiently addressed by previous related work.

## 7 CONCLUSION AND FUTURE WORK

Employing efficient group management and communication is significantly important for supporting distributed event correlation services. This paper presents a new group management framework based on the IP multicast standard for distributed event correlation that exhibits the following key advantages:

- It supports a dynamic and scalable monitoring information dissemination to managers and agents efficiently compared with unicast communication.
- It provides a fine-grain group communication that enables agents disseminate monitoring information based on the event correlation tasks and delegations. This provides for an optimal multicast delivery among agents and managers.
- The presented multicast management framework facilitates distributing the correlation tasks among a group of agents which minimizes the monitoring latency and eliminates performance bottleneck in the event correlation process.
- The presented framework also supports agents' synchronization protocol that ensures agents' state consistency during group communication, a distributed algorithm that enables managers to share monitoring views (results) without multicast group conflict, and a bootstrap mechanism that facilitates the agents' creation and administration.
- IP multicasting improves the robustness and survivability of the system significantly in the presence of agents or network failures. In case of network partitioning or agents malfunctioning, other agents can communicate and negotiate recovery procedures. This is unlike point-to-point TCP connections where a large number of back up connections is required.

One limitation of the described approach is the potential of creating too many groups that may cause high resource consumption (i.e., socket descriptors). However, this is not generally anticipated unless thousands of *different* primitive events (or alarms) are defined in each managed object which is very unlikely the case in typical enterprise network or system management systems. Examples of future issues to be explored include supporting soft real-time monitoring, integrating event ordering, using customized reliability and providing group fault recovery.

## REFERENCES

- [1] Ehab Al-Shaer. Active Management Framework for Distributed Multimedia Systems. *Journal of Network and Systems Management (JNSM)*, March 2000.
- [2] Ehab Al-Shaer, Hussein Abdel-Wahab, and Kurt Maly. HiFi: A New Monitoring Architecture for Distributed System Management. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS'99)*, pages 171–178, Austin, TX, May 1990.
- [3] Ehab Al-Shaer, Hussein Abdel-Wahab, and Kurt Maly. Application-Layer Group Communication Server for Extending Reliable Multicast Protocols Services. In *IEEE Int. Conference on Network Protocols*, pages 267–274, Atlanta, GA, October 1997.
- [4] Ehab Al-Shaer, Hussein Abdel-Wahab, and Kurt Maly. Dynamic Monitoring Approach for Multi-point Multimedia Systems. *International Journal of*

- Networking and Information Systems*, pages 75–88, June 1999.
- [5] S. Alexander, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High Speed and Robust Event Correlation. *IEEE Communication Magazine*, pages 433–450, May 1996.
  - [6] E. Amir, D. Breitgand, G.V. Chockler, and D. Dolev. Group communication as an infrastructure for distributed system management. In *Proceedings of Third International Workshop on Services in Distributed and Networked Environments*, pages 84–91, July 1996.
  - [7] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a communication subsystem for high availability. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 76–84, 1992.
  - [8] R.D. Gardner and D.A. Harle. Methods and systems for alarm correlation. In *Global Telecommunications Conference (GLOBECOM '96)*, volume 1, pages 136–140, May 1996.
  - [9] R.D. Gardner and D.A. Harle. Pattern discovery and specification techniques for alarm correlation. In *Network Operations and Management Symposium (NOMS'98)*, volume 3, pages 713–722, March 1998.
  - [10] J. F. Jordann and M. E. Paterok. Event Correlation in Heterogeneous Networks Using the OSI Management Framework. In *Integrated Network Management I*, pages 365–379. IFIP, 1989.
  - [11] Kwang-Hui Lee, Jong-Kun Lee, and Han-Soo Kim. A multicast protocol for network management system. In *Proceedings of IEEE International Conference on Information Engineering*, pages 364–368, June 1995.
  - [12] P. Parnes and D. Schefstorm. Real-Time Control and Management of Distributed Application using IP-Multicast. In *Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management (IM)*, pages 901–914, 1999.
  - [13] S. Pejhan, A. Eleftheriadis, and D. Anastassiou. Distributed multicast address management in the global internet. *IEEE Journal on selected areas in communications*, pages 1445–1456, Oct. 1995.
  - [14] J. Schonwalder. Using multicast-SNMP to coordinate distributed management agents. In *Proceedings of Second IEEE International Workshop on Systems Management*, pages 136–141, March 1996.
  - [15] W. Richard Stevens. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*. Addison-Wesley, Reading, Massachusetts, 1996.
  - [16] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Englewood Cliffs, NJ, 1993.
  - [17] P. Wu, R. Bhatnagar, L. Epshtein, and M. Bhandaru Z. Shi. Alarm correlation engine (ACE). In *Network Operations and Management Symposium (NOMS'98)*, volume 3, pages 733–742, March 1998.