On Generic Scalability Problems in Monitoring of Data Communication Networks

A dissertation for the degree "Doctor of Philosophy"
in Computer Science

David Breitgand

Submitted to the Senate of the Hebrew University in Jerusalem (2003)

This work was carried out under the supervision of
Prof. Danny Dolev and **Dr. Danny Raz**.

Abstract

Network monitoring is a fundamental building block of any network management system. Monitoring is needed to ensure that the network operates within the required specifications, and to account for user activities and resource consumption. The raw management information gathered from the individual physical and logical network elements is used to compute some higher level aggregate metrics that characterize operation of the network as a whole. These aggregate metrics serve as input to the management algorithms exercising the control actions in the functional areas of configuration, performance, fault, accounting, and security management as defined by the OSI Reference Model [OSI91]. The information about the behavior of the modified environment is once again acquired through monitoring, and further control actions are performed. These steps are being repeated continuously to facilitate the network management loop. Obviously, the scalability of the higher level management tools strongly depends on the scalability of the monitoring subsystem.

In this dissertation, we identify generic scalability problems involved with monitoring communication networks, define these problems rigorously, and provide algorithmic and architectural solutions that can be deployed *incrementally* to address these problems.

We identify two different types of network monitoring: *statistical*, and *on-line* ones. Statistical monitoring facilitates long-term management activities such as usage-based accounting and billing, history recording, capacity planning, *etc.* On-line monitoring enables short-term management activities such as proactive performance and fault management. These two types of monitoring are characterized by different timing constraints, bandwidth consumption, and availability levels. Management of data communication networks is, often, performed *in-band*. This means that the monitoring tools utilize the same network resources (specifically, bandwidth), as regular user applications. Therefore, bandwidth efficiency is one of the key factors determining the scalability of the monitoring subsystem. Specifically, our contributions belong to the following three areas.

- Reducing the total volume of the on-line monitoring traffic through hierarchical and event-driven monitoring schemes;
- Improving the network *goodput* through optimal scheduling of the in-band data transmissions of the monitoring data at times of network congestions for statistical monitoring;
- Improving availability and dependability of distributed monitoring through fault-tolerant and light-weight monitoring middleware being useful to both types of monitoring, and especially well-suited for the statistical one.

To address bandwidth efficiency of on-line monitoring, we suggest that in some commonly occurring important cases, computing high-level *additive* metrics by using hierarchical and event-driven distributed algorithms is preferable to other known techniques, such as *e.g.*, probabilistic polling. In particular, we study the problem of obtaining a sufficiently accurate estimation of a multicast group size (*i.e.*, the total number of currently active receivers which is an additive metric) in the context of accounting management. The need to compute additive metrics similar to this arises repeatedly in various contexts related to network management. However, the group size estimation at a level of the individual receivers is known to be particularly difficult.

We suggest novel network-level algorithms that combine a hierarchical control structure with an event-driven monitoring mechanism in order to conserve network traffic and cope with the problem of *feedback implosion*. We provide a rigorous definition of the problem, theoretically analyze our algorithms, and investigate their relative performance in terms of bandwidth using an extensive simulation study. We show, using real Internet traces, that using our algorithms reduces the bandwidth overhead caused by monitoring traffic by a factor of 2-10 (compared to other commonly used approaches) depending on the required accuracy and a specific behavior of the group size.

To address the issue of improving the network goodput when in-band monitoring is used, we proposed discriminating the monitoring traffic with soft timing constraints (*e.g.*, accounting records) in favor of the more sensitive user-level traffic at times of the transient network congestions. The key idea is allowing the network nodes to delay data by locally storing it. We refer to this as intermediate *parking*. The parking is possible, for example, in the *Active Network* paradigm, and in the *peer-to-peer* networks approach that recently received a great deal of interest.

In this dissertation, we show that we can improve the network's goodput dramatically, even if a very simple scheduling algorithm for intermediate parking is used. The parking imposes additional load on the hosting nodes. To obtain minimal cost schedules we define a generic optimization problem that we call the *Travelling Miser* problem.

We concentrate on the *on-line* variant of this problem, which is of greater practical interest in our context. For this variant, we developed a number of enhanced scheduling strategies and study their characteristics through a rigorous simulation study under different assumptions on the environment.

We prove that in a restricted yet practical case of a single transient blockage per path our strategies are logarithmically competitive. For the case when the blocked node is capable of signaling link recovery, we suggest a scheduling strategy that is 3-competitive.

One of the more difficult *meta-management* problems is highly available and reliable deployment of the monitoring processes. To deal with the issues of high availability and dependability of the monitoring tools, we developed a novel light-weight middleware architecture called HAMSA (Highly Available Monitoring System Architecture). We implemented a prototype of HAMSA, and demonstrated through this implementation that high availability of distributed monitoring can be achieved at low cost by utilizing Reliable Group Communication Paradigm. The fully functional prototype implementation is available on request.

In all the three areas (*i.e.*, high availability, bandwidth efficiency, and goodput improvement), our results extend beyond the network management domain, and are applicable in a wide variety of distributed computing and networking contexts. The main findings of this dissertation have been published in [BRS02b, BDR03, BDRS03].

Contents

List of Figures

Chapter 1

Introduction and Motivation

We start with a general discussion of monitoring in the context of network management. Section 1.2 describes the state of the art. Section 1.3 summarizes our methodology. We present an overview of our approach in Section 1.4. Section 1.5 explains the structure of the rest of this dissertation.

1.1 The Role of Monitoring in Network Management Systems

Network management is an integrated set of activities being continuously performed manually (by the trained personnel) and automatically (by the specialized software applications) to control and coordinate network resources in such a way that the business-level objectives would be met at all times, at a reasonable cost, and with optimum capacity.

A network management system provides tools for (a) monitoring of the network resources in the context of a specific administrative activity and (b) taking appropriate control actions to ensure that the business-level objectives related to this administrative task are satisfied. The former mechanism is termed *sensing*, the latter one is called *actuation*. This dissertation focuses on the generic problems that affect the scalability of the sensing mechanisms.

According to the OSI Reference Model [OSI91], network management activities are divided into the following five categories.

- **Fault management**: activities that enable detection, isolation, and correction of an abnormal operation of the network resources[1] with respect to a given set of the service-level objectives.

- **Accounting management**: activities that enable gathering information about the usage of network resources, identification of the usage costs, allocation of the usage costs to specific users, and eventual charging (not necessarily in the monetary form) of the users for resource consumption. Other accounting applications include capacity planning, trend analysis, auditing, customer relations management, *etc.*

- **Configuration management**: activities that identify network resources, and relationships between them, set attribute values of the resources (*e.g.*, queue lengths of a router, disk quota for users, transport port number of an application, security privileges of a user, *etc.*) initialize, allocate, and deactivate individual resources and collections of them, and continuously follow the status of the resources and their relationships.

- **Performance management**: activities that are aimed at evaluating the behavior of network resources and the effectiveness of network communication with respect to the given performance limits that ensure the required service-level objectives. Although it is naturally desirable that the network would perform at the highest possible level, this may not always be critical, and, as long the service-level objectives are met, less than optimal performance levels can be tolerated.

- **Security management**: activities that enable protecting of the resources from an unintended and/or unauthorized use and ensuring the required levels of confidentiality, privacy, integrity, and availability with respect to the different parties that may use the network resources.

In order to address these functional requirements efficiently, management tools from the different functional areas should interact and cooperate. This leads to the concept of *Integrated Management* being in the focus of the research efforts throughout the last decade.

All management facilities regardless of their respective functional areas involve sensing the network resources remotely to obtain the raw data about the relevant aspects of their functioning. This raw data is used to compute some higher-level metrics and predicates characterizing the operation of the network as a whole with respect to the business-level objectives. The computed high level metrics and predicates are communicated to the actuators (human and/or

[1] We use the term *resource* in a broad sense in order

software) to perform the appropriate analysis, coordination, correction, and control actions. This organization is common to all management tools.

The enabling sensing mechanisms differ from each other considerably, depending on the specific requirements of the actuators. These differences are best illustrated by means of example. Let us look at the two intentionally simplified management facilities: one being taken from performance management area, and another one being taken for the accounting management domain.

First, consider an example of a performance management facility. Central to the definition of performance management is the definition of what constitutes a "problem", and by which visible symptoms the problems are manifested in the network. As opposed to errors (*e.g.*, CRC errors on communication links), performance problems do not have absolute meaning. The behavior of a resource is defined as faulty with respect to the expected normative specification (objective) for this resource. Performance problems, thus, constitute abnormal conditions that can be corrected by management actions such as resource reconfiguration, reallocation of available resources, temporary rerouting of the traffic, lowering the number of simultaneous transport connections to the resource, *etc.* Thus, the performance management facility is required to monitor the network resources for detecting an existence or an imminent occurrence of the performance problems, diagnosing the root cause of the problem, and launching corrective procedures to deal with it.

Let us consider a server that accepts user requests over the TCP/IP connections. When the number of the simultaneously opened connections exceeds certain *threshold*, the average response time may increase dramatically, and the high-level objectives would not be met with respect to this service parameter[2]. Thus, when the symptom of the performance fault (in this case, slow response time) is detected by the the performance management system, it has to inspect the counter of connections, and, if needed, close some of them. This approach to performance management is called the *root cause analysis*. On the other hand, when the performance management system detects that the number of connections is dangerously close to the threshold, it may conclude that the performance degradation is imminent, and take a preemptive corrective action. This approach is termed *trend analysis*. Both these models can be deployed simultaneously to complement each other.

What about the sensing mechanism enabling either of these two performance management

[2]Of course, the average response time also may depend on many other factors, but when the average response time falls down, say, by the factor of two, the number of simultaneously opened connections may be the primary cause of this faulty behavior.

paradigms? One conceivable approach is to *poll* the network resources continuously for the relevant metrics[3]. Another possibility is to enable the managed resources to send *event reports* to the involved actuator(s), when the resource locally detects that the relevant metrics violate their prescribed thresholds.

Regardless of whether the polling or event-driven sensing mechanism is deployed for our example performance management facility, we can observe the following generic properties.

- **Strict timing requirements**: the monitoring mechanism should communicate the relevant symptom in real, or almost real time.

- **Low-volume traffic**: the amount of the management information transferred over the network in a single interaction is relatively small: a few bytes of payload to represent the counter value[4].

- **Moderate Reliability**: the symptom monitoring should be reliable in the sense that eventually a report about the problematic behavior should reach the relevant actuator in spite of possible network and site errors. However, if performance problems persist, then a loss of a single report or even of a several reports can be alleviated by correlating reports from other sources that may be affected, and by retransmission of the reports from the same source.

As an example of a management facility with different requirements on the sensing mechanism, consider an accounting management tool that keeps track of the usage of our server. Among other things, we may be interested in the usage attributes such as the number of open connections, and the total number of bytes sent and received over these connections by each user for the given accounting period. To obtain these metrics, the raw data about the connections should be recorded at the server side. Then, this information has to be communicated to a mediation point that would correlate the users with their connections and prepare the per-user resource consumption metrics of interest. Next, these metrics are communicated to the price

[3]In case of the root cause analysis in our example, the relevant metric is the average response time, while for the trend analysis, this is the number of open connections.

[4]In case of continuous polling, the total bandwidth consumption over the period of time is considerably higher than in case of the threshold-driven monitoring [DR02]. Also, as was shown in [BACW90], polling scales poorly in terms of the number of resources that can be supported by a single actuator. However, for the sake of the example (single server), it is important to notice that in both cases the total traffic is relatively low when compared to the accounting management facility that we discuss next.

setting, billing, planning, and other facilities. As before, there are two operation modes for retrieving the monitoring information: either through continuous polling, or through a threshold and/or timer-based event reporting[5].

With respect to the requirements on the monitoring mechanism posed by such an accounting management tool, we can observe the following.

- **Relaxed timing requirements**: the monitoring mechanism should communicate accounting metric to the relevant actuator by some deadline (*e.g.*, the predefined time by which the billing facility has to receive the data about the usage to prepare the bills for the users). The time scale ranges from tens of seconds at the low end, to hours at the high end.

- **High-volume traffic**: the bandwidth consumption is high. The actual amount of data transferred in a single interaction between the sensor and the actuator depends on the nature of the usage metrics that are collected, and the timing requirements of the actuator. However, since there are potentially many users that perform multiple simultaneous activities, the accounting logs may become very large even for a single server, and even when the amount of the data about the usage associated with a single user is relatively small.

- **High Reliability**: accounting information monitoring should be reliable since an irrecoverable loss of the sensitive accounting information due to the possible network and site errors may result in severe financial penalties.

Abstracting the low-level details of the specific facilities, we can define the following two generic categories of monitoring.

- **On-line Monitoring**: small volumes of the monitoring information should be communicated from the sensors to the actuators in almost real time, *i.e.*, the deadline for communicating the monitoring traffic is on the order of the round-trip time supported by the network. This type of monitoring enables the short-term management actuators.

- **Statistical Monitoring**: large volumes of the monitoring information should be communicated from the sensors to the actuators by some predefined deadline that is much larger than the round-trip time supported by the managed network. This type of monitoring enables the long-term management actuators.

[5] In fact, the timer-based reporting can be viewed as a special case of event reporting.

Reliability is a requirement that is desirable in both kinds of monitoring. However in some applications this requirement may be relaxed (*e.g.*, when the sensed data can be easily regenerated).

Note that categorizing the monitoring mechanisms into the on-line ones and statistical ones is independent from the functional areas in which these mechanisms are to be used. Different management facilities belonging to the same functional area may need different monitoring mechanisms. To illustrate this, let us again consider performance management tools. It is conceivable that we have the short-term performance management actuators (*e.g.*, an emergency traffic re-routing policy) and the long-term ones (*e.g.*, the capacity planning facility). While the former actuator requires on-line monitoring, the latter one is better suited by the statistical monitoring mechanism.

Following [DR02], we define the *Generic Monitoring Problem (GMP)* informally as follows. Given a set of atomic variables maintained locally by a set of *agents* residing at the network elements, and the node where the actuator executes, called *manager*, one has to devise a mechanism for computing an actuator-specific function f defined on the set of the local variables belonging to the agents, by time $t < D$ where D being the predefined *deadline* characterizing the timing requirements of the actuator that uses the value of the function f as its input.

All monitoring facilities are designed to solve this problem in the context of a specific actuator. The details of a concrete solution, its efficiency, and scalability depend on the overall architectural framework in which the specific solution should operate. Different management architectures admit different solutions. To realize the Integrated Management approach, the monitoring tools should fit into a specific architecture. In our opinion, the following three factors characterizing the overall management architecture are more important for understanding technical feasibility and scalability of different monitoring solutions.

- **In-band or out-of-band communication**: this is the fundamental characteristic of the management framework that defines whether the monitoring data is communicated from the agents to the manager using the *same* network that is being monitored, or through the dedicated one. In the former case, the monitoring mechanism is the *in-band* one. In the latter case, it is the *out-of-band* one. In contrast to telecommunication networks [Int], management of IP networks is in-band.

- **Reliability level of the resource access protocol**: this parameter defines the level of resiliency to network failures available in the application-level management protocol defined by the architecture and enabling the interactions between the agents and the manager. In some management frameworks for IP networks, *e.g.*, SNMP [Sta99], being developed by IETF [Forc], communication between the agents and the manager is connectionless and unreliable, while in the WEB-based management architecture, CIM [Fora] being developed by DMTF [Forb], this communication is connection-oriented and reliable.

- **Local instrumentation versatility**: this dimension is involved with the versatility of the local mechanisms used by the agents at the network elements to sense local variables. In particular, the local instrumentation may support actuator-specific definition of the monitoring events of interest such as counter-based thresholds, predicates, and timers, defined on the local variables. When an agent detects a specific event locally, it sends a *trap* message to the predefined destination. This is the approach used in the OSI management model [ISO], and in the TMN model [Int] that extends OSI for telecommunication networks. Yet in other popular architectures, only a predefined set of traps corresponding to a limited set of events can be generated, as is the case with the SNMP framework [Sta99]. Finally, the local instrumentation may not support event-driven traps at all, and it may only be possible to extract their values in response to external polling requests.

In the next section, we discuss the standard and emerging network management architectures with respect to these three key characteristics, and point out their weaknesses.

1.2 State of the Art

1.2.1 SNMP

The de-facto standard network management framework for IP networks is *Simple Network Management Protocol (SNMP)*, the set of standards that is being developed by IETF. The literature covering all aspects of SNMP is extensive [MM99, RM95, Sta99], to mention just the few excellent sources. Therefore, in this section, we only mention the main facts about SNMP that are more relevant to our research.

SNMP Basics

SNMP originated as a network monitoring protocol in 1987. It was then called SGMP (Simple Gateway Monitoring Protocol). It provided a straightforward means for monitoring TCP/IP routers. The need for a more comprehensive approach has been identified soon. SGMP has been extended and adapted to meet this need. The result was SNMP, first issued in 1988. SNMP provided a minimal, yet powerful architectural framework for monitoring and control of network elements in IP networks by defining the following components:

- Structure of Management Information (SMI)(RFC1155 [RM90]) that describes how managed objects maintained in the Management Information Base (MIB) are defined;
- Management Information Base for TCP/IP-based networks (RFC1213 [MR91]) that describes the managed objects contained in the MIB;
- Simple Network Management Protocol (RFC1157 [CFSD90]) that is used to access and control the objects maintained in the MIB.

It is important to stress that SNMP itself does not provide network management solutions at all. It only provides the *architectural framework* that can be used to develop tools for the five functional areas of management (see our discussion in the beginning of the previous section). Therefore, it is important for us to understand what solutions to the generic monitoring problem are feasible within this architecture.

SNMP was initiated by the IETF community being frustrated by the complexity, implementation inefficiency, and slow rate of adoption and standardization of the more comprehensive OSI Network Management Model [OSI91, Yem93]. SNMP was conceived as a *temporary* solution that, in fact, borrowed heavily from the OSI approach, while reducing it to the absolute minimum. However, the rapid success of SNMP turned it into the de-facto permanent standard that is not to be abandoned in the foreseeable future. Since the first version of SNMP, two more versions, SNMPv2, and SNMPv3 have been issued. SNMPv1 constitutes the least common denominator, and is the most widely deployed SNMP version yet.

SNMPv1 defines a two-tier *thick client/thin server* architecture in which monitoring (and control) of a network element is performed through *in-band* interactions between the client, called *manager*, and a management server called *agent* that executes at the monitored resource. The agent maintains a virtual datastore, called MIB, containing instances of management objects. The objects (also called variables) are defined using the subset of ASN.1 notation as prescribed by the SMI specification. Variables are of two types: *scalars*, and *columnar* (also called *tables*). The term *scalar* indicates that there may be at most one instance of this object in the MIB. In contrast, there may be multiple instances of the columnar variable in the same MIB.

Each variable is assigned a globally unique object identifier. This identifier is an ordered set of sub-identifiers (called *components*) that are, if fact, non-negative integer numbers separated by dots. The components play the role similar to directory names in the global naming structure defined by SMI. Each instance of the variable, is assigned the globally unique object instance identifier (OID) that consists of the object identifier with appended instance identifier suffix. The MIB is a lexicographically ordered tree of the OIDs of the object instances that are available at the resource. The object instances are in the leaves of this tree. When an agent is asked to render the value of a specific object instance, it uses the local (usually proprietary) instrumentation to access the corresponding information within the resource. SNMP does not allow aggregate operations on the information in the MIB. For example, one cannot operate with the whole table, but only with the given instances of the table objects. Similarly, one cannot operate with an arbitrary group of different scalar object, but is limited to the per-instance access. SNMP MIBs do not have transactional semantics. This is one of the reasons why SNMP is used primarily for monitoring.

MIBs define the standard access points for the managers. For example, all SNMP-compatible network elements should support the managed objects definitions of MIB-II (RFC1213 [MR91]). This facilitates a clean separation between the managed resource interface (defined by the MIB), and their internal implementation exclusively controlled by the vendor of a network element. This separation allows independently developed agents and managers to inter-operate seamlessly using the SNMP protocol.

To collect the information from the agent's MIB, the manager can use two basic SNMP protocol requests: Get and GetNext. A Get(x) request retrieves the value of the object instance identified by the given OID x. If such OID is not present in the agent's MIB, Get(x) request returns with an error indicating this. A GetNext(x) request receives an *OID approximation* as its input. The term approximation is used in SNMP literature to indicate the fact that x does not have to identify an existing leaf in the MIB tree. The GetNext request will always return either the value of the immediate lexicographic successor of x in the tree, or the end of MIB indication.

Get is primarily used to retrieve values of the scalar varaibles, and GetNext is used to traverse the tables. SNMPv2 introduced a new request called GetBulk in order to optimize retrieval of large tables. The GetBulk request generalizes the GetNext operation. Conceptually, it is just n GetNext requests packed into the same protocol data unit (PDU). Notably, SNMP does not provide GetPrev request for retrieval of the lexicographic predecessors in the MIB tree. This may result in bandwidth inefficiency for some monitoring applications, such as *MIB browsers* when large tables (*e.g.*, accounting tables) are browsed on-line. However, as we show in [BRS02a], the functionality of GetPrev can be efficiently simulated using only Get and GetNext resulting in dramatic efficiency improvements for MIB browsing[6].

SNMP defines that the protocol requests initiated by managers are *synchronous*, *i.e.*, the manager cannot proceed to the next request before it either has received response on the previous request, or an application-specific timeout condition has occurred. The agent may generate predefined *asynchronous* messages called *traps* to inform the management station on specific local occurrences (*e.g.*, link outage, or element reboot). SNMP defines only a limited number of traps because the purpose of traps is to alert the manager to a specific problem area. The manager is then supposed to perform polling of the problematic resource to determine the exact conditions. This approach is sometimes called *trap-driven polling*. SNMP does not allow dynamic specification and enabling of the application-specific trap triggers.

Currently standardized transport mappings for SNMP are specified by RFC3417 [PCM+02]. This RFC defines that the preferred transport mapping is using SNMP over the User Datagram Protocol (UDP). Although using the Transmission Control Protocol (TCP), is allowed in principle (among other options), there is no corresponding RFC standardizing this mapping. In practice, TCP is not used for the following reasons[7].

- It is important for network management to work under stressful condition. This is particularly true for the fault and performance management that must diagnose the cause of the adverse conditions. Being datagram-oriented removes the need for setup and maintenance of transport *connections* that are likely to fail under such conditions.
- TCP connection set-up/tear-down requires much more overhead than sending a UDP datagram.
- TCP stack is considerably more complex and resource demanding than UDP. This may be inappropriate for small network devices with limited resources.

Let us now summarize the SNMP architectural framework using the three key factors (*i.e.*, in-band or out-of-band communication, reliability of the access protocol, and local instrumentation versatility), that were identified in the end of Section 1.1.

[6] In [BRS02a] we demonstrated that our GetPrev monitoring tool that is fully compatible with all versions of SNMP is two orders of magnitude faster and two to three orders of magnitude less bandwidth consuming than traditional tools that use Get, GetNext, and GetBulk in a straightforward manner.

[7] In [SMF99, Sch02], SNMP over TCP transport mappings were suggested. These proposals, however, are still experimental, and are not widely used.

- **SNMP defines an in-band monitoring architecture.** In fact, this is common to all existing management frameworks for data communication networks. Therefore, bandwidth efficiency is one of the more important metrics characterizing scalability of the specific monitoring facilities developed to address the five functional areas of network management in IP networks.

- **Asynchronous SNMP traps are intrinsically unreliable, because they are communicated over the unreliable connectionless transport protocol.** Protocol requests issued by managers to agents can be lost. However, since these requests are synchronous, reliability can be achieved at the application level through retransmissions of the lost request if the expected response is not received after some application-specific timeout period.

- **Access to local information at a network element as defined by SNMP, is limited to polling and a small number of the predefined unreliable traps that may help focusing the polling operations to a specific problem area.**

Some SNMP Drawbacks

Let us now come back to our "online versus statistical" classification of the monitoring mechanisms, and identify the problems with the SNMP approach regarding these two types of monitoring.

- **Bandwidth inefficiency due to thick client/thin server architecture**:
  - In case of statistical monitoring, the absence of preprocessing capabilities at the agent side requires transferring large volumes of the raw data.
  - Since it is impossible to configure application-specific trap triggers, and since even the predefined traps are unreliable, the manager in many cases has to use polling to satisfy the strict timing requirements of the online tools and capture the abnormal conditions as soon as possible. However, if most of the time resources behave normally, then periodically polling them brings no useful information while consumes valuable network bandwidth.

- **Manager station becomes a processing bottleneck for both types of monitoring as the size of the managed network increases.**

- The manager that uses polling can support much smaller number of network elements than the one that is based on reliable event-driven traps, and only polls the managed resources when indeed necessary [BACW90].
- Even if a reliable and versatile trap mechanism would be available at the agents, the manager can get overwhelmed by the number of simultaneous traps initiated by different network elements. This is known as the *feedback implosion* problem. This is an intrinsic problem of all event-driven monitoring mechanisms based on the flat control structure in which multiple agents communicate directly with the single manager.

- **Manager station is a single point of failure, which may jeopardize the availability of the monitoring services.** In general, monitoring cannot operate in SNMP framework in a disconnected mode[8]. This type of operation, however, is essential for scaling monitoring services, reducing communication overhead, and increasing survivability of management services as explained later in this work.

- **The unavoidable network distance between the management station and (some of) the network elements makes it very hard to control the elements, due to the inherent instability imposed by long control loops.** This may become an especially acute problem when online monitoring is used for control.

- **Since SNMP transport mapping is UDP, the size of messages exchanged between the agent and the manager is limited by the size of a single UDP datagram.** This makes SNMP bandwidth inefficient and sometimes unnatural, especially for statistical monitoring [SMF99].

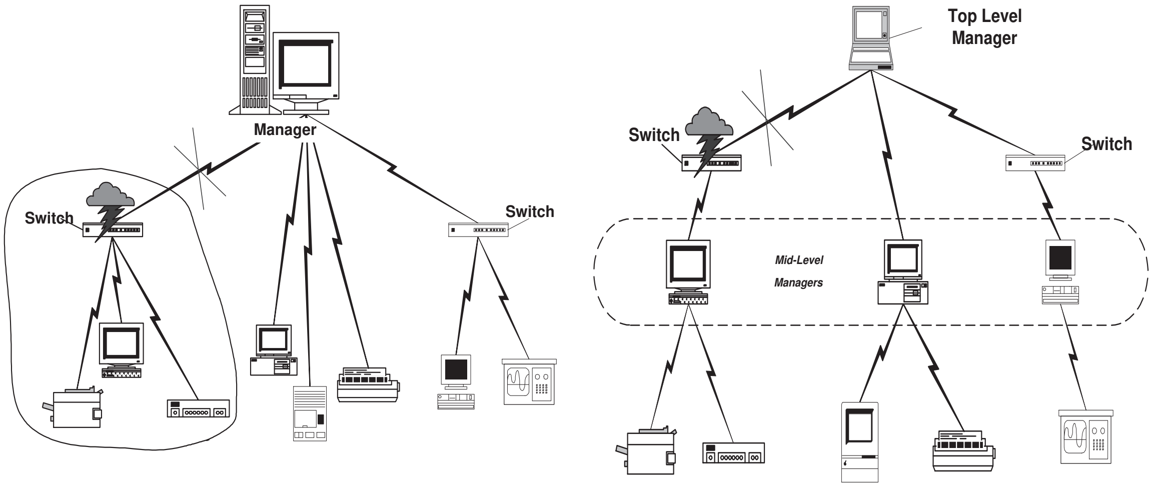
Some of these deficiencies affecting the scalability of the monitoring have been addressed within the SNMP framework itself by adopting the three-tier architecture. Ignoring specific details, a typical three-tier monitoring facility looks as depicted in Figure 1.1(b). The network element agents constitute the lowest tier, and serve as sources of the management data. The monitoring components are dynamically dispatched at the middle tier. They monitor the target agents (and, possibly, communicate with other monitoring components) accumulating and pre-processing the information collected from them. The end-consumers of this informa-

[8]Although for some types of management data disconnected operation of monitoring can be achieved using the hardware probes implementing RMON MIB [Sta99], the disconnected monitoring operation is not available in the general case.

tion, managers, constitute the uppermost tier. The entities in the middle tier may be further organized into hierarchies, leading to the *multi-tier* solutions.

To understand why the three-tier organization of the monitoring services solves the problems above, compare it with the two-tier organization depicted in Figure 1.1(a). Note that the components populating the middle tier can partially or fully implement some of the processing functionality that was previously residing exclusively on the manager side. This way, the management logic is brought closer to the managed devices. This reduces the traffic overhead, shortens the control loops, and extends the management functionality. In particular, the middle-tier components can implement arbitrarily complex application-specific threshold-driven traps, that allow deployment of efficient event-driven monitoring schemes.

Survivability and availability of the network monitoring services are also improved by the three-tier organization. The mid-tier components can operate autonomously of the first-tier managers. Consider an example given in Figure 1.1(b). When certain parts of the monitored network become unavailable, *e.g.*, due to a network split, the mid-tier monitoring components can continue monitoring in their respective partitions, and later merge the results. This is impossible in the traditional centralized two-tier architecture.

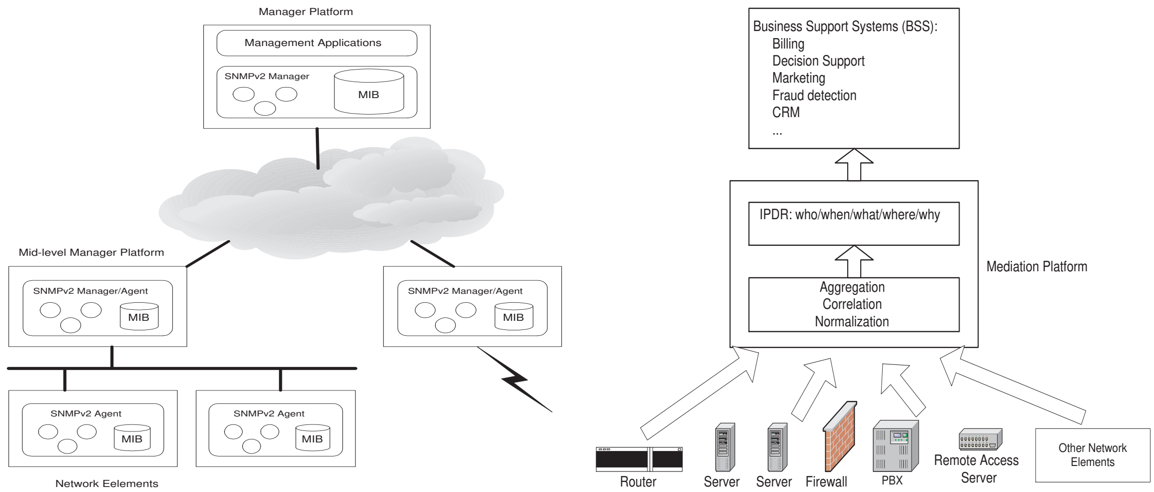


(a) Two-tier architecture (b) Three-tier architecture

Figure 1.1: Monitoring Architecture

On the down side, the fully distributed three-tier client/server applications are much more difficult to control than the centralized two-tier facilities. This calls for special *meta-management* tools, providing the mid-level monitoring components with a run-time environment that allows dynamically to upload, activate, deactivate, upgrade, and remove them in a reliable manner.

Besides these meta-management operations, the execution of the monitoring components themselves should be reliable, because their failure may be hard to detect, and other components of the monitoring services may depend on the failed ones [BSD00]. In Chapter 4, we propose an efficient solution to high-availability of the multi-tiered monitoring services. Our solution can be deployed incrementally, *i.e.*, without the need to replace popular standard management frameworks, such as SNMP.



(a) Hierarchical Management in SNMPv2 (b) IPDR Architecture

Figure 1.2: Examples of Three-Tier Monitoring Architecture

To enable the multi-tiered distributed monitoring and control, SNMPv2 [Sta99] defines a dual-role entity, called *manager-agent*, or *mid-level manager*. As shown in Figure 1.2(a), the mid-level managers form a tier being interposed between the front-end monitoring applications, and the network element agents. A mid-level manager may act either as SNMP agent, or SNMP manager depending on the circumstances. SNMPv2 defines the manager-to manager (M2M) MIB to support interactions between the managers. This MIB contains two groups of objects: the alarm group, and the event group. The alarm group is used by a superior manager to define a set of threshold-based alarms. Each threshold alarm specifies some object instance in the local MIB. When the value of this object instance crosses a threshold, an event is triggered. This event is typically an asynchronous InformRequest PDU (defined in SNMPv2,v3) that is sent to the superior manager. Each alarm in the alarm group points to an entry in the event group, which in turn defines the information that is to be sent with this InformRequest PDU.

The positive observations about the framework enhancements introduced by SNMPv2,v3 are as follows.

- Mid-level managers remove the flat control structure of the two-tier architecture defined in SNMPv1, and lay a foundation for the hierarchical multi-tiered monitoring facilities that are potentially much more bandwidth-efficient, scalable and available. Bandwidth efficiency comes from the fact that local events of no global importance are processed in the mid-tier and thus are prevented from flooding the network. Availability is due to the reduced probability of the failure event in which all mid-level managers crash, or become unreachable simultaneously.

- Event-driven monitoring become possible through deploying the mid-level managers closely to the managed network elements (or even locally to them). The mid-level manager then can poll some set of managed devices storing the values of their variables in its own M2M MIB, and generating asynchronous notification when threshold events occur. This scheme scales much better than the traditional polling-based approach.

- Pre-processing of managed data and autonomous control actions within the network are possible. To this end a mid-level manager has to specify itself as the superior manager that has to be notified when a threshold event occurs in its alarm group.

To further extend this distributed hierarchical management model of SNMP, a means for dynamic deployment of management scripts at the mid-level managers has been proposed recently by IETF [LS99b, LS99a, JS99]. These proposals address the facilities for delegating management scripts (*i.e.*, the executable code) to mid-level managers, storing, accessing, and scheduling them through the specialized *Scripts MIB*. This way the functionality of mid-level managers can be dynamically extended. All these operations are performed using the SNMP protocol between the superior and subordinate managers.

These enhancements to the original SNMP framework have a potential to dramatically improve scalability of the SNMP-based monitoring. A promising prototype implementation of a hierarchical network management system within the SNMPv2 framework was presented in [MG96]. However, distributed management capabilities offered by the extended SNMP management framework are not widely utilized by the management community yet.

There are a number of reasons for this. We believe that one of the more fundamental problems with the distributed hierarchical management in general, and that provided by SNMPv2,v3, in particular, is the increased complexity of the so called *meta-management*. Meta-management refers to administrating the management system itself. In other words, practical network administrators often consider that the benefits of developing, deploying and maintaining a truly distributed SNMP-based management application, are being offset by the associated costs.

The following meta-management problems arising with the highly distributed multi-tired monitoring facilities in the SNMP framework are in the focus of this work.

- If a mid-level manager, (see Figure 1.1(b)), and not the communication link connecting it fails, some portion of the network may become unavailable until either the superior manager, or a peer manager would take over the responsibilities of the failed one. However, no support for efficient and reliable discovery of such failure events and recovery from them is provided by the SNMP framework. Similarly, SNMP lacks the means for automatic reconciliation of the monitoring information independently obtained in different network partitions that may occur when communication links fail. Therefore, developing highly available distributed monitoring facilities in this framework is notoriously difficult.

- Unreliable and inefficient communication in the mid-level managers' tier: although the manager-to-manager communication is supported in the advanced versions of SNMP by means of the M2M MIB, and the new report messages, this communication remains inherently unreliable. Moreover, since the mid-level managers usually are deployed on the full-fledged hosts with considerable processing and memory capabilities, using only UDP for communication among them is both bandwidth-inefficient and unnatural [SMF99]. This is especially problematic for statistical monitoring applications.

- The means provided by the extended SNMP standards for delegation and control of the management scripts have no advantage over the code mobility solutions that made a great progress over the recent years (*e.g.*, Java).

- Alarms involving more than one object cannot be directly specified in the M2M MIB. Although correlation of the values of multiple objects is technically possible, the standard framework lacks efficient means for doing this.

- SNMP MIBs that are adequate for modelling the elementary variables in the network elements, are insufficient when it comes to modelling the complex aggregate objects in the middle tier.

Realizing advantages of the distributed multi-tiered monitoring paradigm is crucial for the scalability of network management. However, in our opinion, the best results in this direction will be achieved by means that are complimentary to the SNMP framework, and not by stretching the SNMP framework itself. In all likelihoods, SNMP will continue playing the central role as the low-level means of network element access, the goal for which it was originally designed.

1.2.2 Emerging Paradigms

In this section we briefly discuss the recent advances in network management (external to SNMP) in the areas of on-line and statistical monitoring, and enabling technologies that were nonexistent, or premature at the time of SNMP standardization.

Online Monitoring

One important ingredient notably missing from the standard management framework is a general-purpose monitoring language allowing to specify the elementary and aggregate events of interest, the producers and consumers of these events, the temporal constraints, priorities, dependencies, delivery deadlines, and other attributes of the events guiding their generation and dissemination. The issue of designing such languages has attracted considerable attention of the researchers over the last decade. A number of experimental proposals for efficient and powerful monitoring languages exist today [ASAWM97, AS98, MSS97]. However, to the best of our knowledge, there is no standardized solution to this problem yet.

The experimental proposals are focusing on the online monitoring where the need for such means is most obvious. The prevalent trend in online monitoring languages is to cater for some kind of *hierarchical event filtering*. In hierarchical filtering, the programs called *filters* (also referred to as *scripts/tasks/agents/sub-managers* in the literature) execute in different network locations, and are being organized into some form of a hierarchy, cooperating on processing of the various events coming from multiple different sources. In the hierarchical filtering approach, a filter within the hierarchy acts as a dual role entity. On one hand, it is a consumer of the monitoring events of interest that are coming from the nodes being connected below the present one in the hierarchy. On the other hand, as a result of processing the received events, it produces its own events that are disseminated to the relevant consumers up the hierarchy. The hierarchical filtering approach can use the SNMP capabilities to extract the raw data from the SNMP-enabled devices in the bottom of the hierarchy. Some form of the hierarchical filtering can be implemented within the SNMPv2,v3 frameworks using the mid-level managers' capabilities provided by these frameworks [MG96]. However, in order to fully exploit the advantages of this approach, dynamic hierarchies of filters must be formed on-demand with minimal human intervention. It is also important that the nodes in the hierarchies would be highly available. Thus, a failure of a filter upon which other filters' operation may depend, would not stop the whole monitoring process[9]. Unfortunately, this is beyond the means offered

[9]The importance of high-availability of the processing entities in the hierarchical distributed approach was

even by the most updated versions of SNMP. Thus, the experimental proposals on hierarchical filtering are usually complimentary to SNMP.

This dissertation focuses on the common principles underlying all hierarchical monitoring approaches. An interested reader is advised to consult [AS98], for instance, for a more detailed comparative study of the online monitoring languages and architectures.

Statistical Monitoring

Statistical monitoring challenges are addressed to a lesser extent by the research community. One example of a comprehensive proposal being actively promoted by the industry is IPDR [Org02]. IPDR standards define the architectural framework for usage-based accounting of IP-based services. IPDR is being developed by a consortium of companies and research institutions. Its design reflects the post-SNMP paradigm shift that is gradually taking place in the management community. While SNMP is designed to work at the level of the single network elements, IPDR's primary focus is a service as whole [IPD02]. Thus, while IPDR may utilize SNMP-based management agents to obtain the raw data from the network elements (physical and logical) that are involved with the given service, at the level of the service itself, the management objects are modelled, monitored and controlled by the means that are external to SNMP. Recently, IPDR consortium has developed reliable and efficient accounting data collection protocol called CRANE [ZE02] that is based on TCP/IP and is intended as a replacement for SNMP-based element access. IPDR adopts a three-tier architecture as shown in Figure 1.2(b):

- **Network and service element tier**: This tier consists of all the network elements (hardware and software) required to provide an IP-based service to consumers. This includes routers, switches, access devices, and transmission devices, network and transport protocols to provide the basic connectivity; firewalls and accounting, authorization, and authentication (AAA) servers to provide security and basic accounting services; email, file transfer, print, storage services to facilitate the applications, *etc.* In addition to the network elements, the tools that configure and manage these elements are considered part of this tier. Examples may include a network management platform (*e.g.*, SNMP-based), RADIUS authentication server [Liv97], a bandwidth broker system, *etc.*
- **Mediation tier**: The mediation systems are positioned between the business support systems and the network and service element tier. The mediation tier should deliver the

first pointed out by us in [BSD00]

requested information about the network usage to the business support systems within the temporal constraints. To this end, a mediation system has to identify the relevant source service and network elements, and collect the raw usage data from them using the access protocol supported by the elements (*e.g.*, SNMP, RADIUS [Liv97], DIAMETER [CLG+03], or CRANE [ZE02]). The mediation system then executes the algorithms that aggregate and pre-process the usage information into the standardized form that is expected by the business support systems.

- **Business support systems tier**: This tier includes all the applications that are necessary to support the IP-based business service operation. These facilities include billing, fraud detection, capacity planning, customer relationships management, *etc.*

IPDR defines the functional requirements on the mediation systems, generic usage record format, IP based services accounting requirements, and the transport protocol that supports encryption of the IPDR documents, and defines how the documents are transferred between the mediation and business support systems.

IPDR utilizes XML for usage reports definition. The IPDR Transfer Protocol supports two modes of delivery: *push* and *pull*. In the push mode, the mediation system is responsible to transfer the relevant documents within the temporal constraints to the pre-specified business support system. In the pull model, the business support system requests specific IPDR document from the mediation system. IPDR defines the protocol state machine and maps it to the Simple Object Access Protocol (SOAP) [WWWC] to obtain the concrete implementation. SOAP operates on top of HTTP, and is, therefore, a reliable connection-oriented transfer protocol. Thus, the third and second tier entities in the IPDR architecture are relieved from the original SNMP restrictions resulting in scalable architecture. IPDR is a good example supporting our claim that advantages of the three-tier architecture should be, and will be be pursued outside of the SNMP framework, in spite of the fact that SNMPv2, and SNMPv3 also provide support for hierarchical monitoring (see Section 1.2.1).

Policy-Based Network Management

Proposals for automation of the network management process through dynamically evaluated rules (also known as *policies*) can be found in early nineties, *e.g.*, Meta [MCWB91]. The policy-based approach has been studied intensively throughout the last decade both in the academia, and in the industry [Slo94, IETF03, Ver02, Bor01, SW99].

Essentially, *policy* is a small program expressed in some *policy definition language*, and having the form of **if**(*some predicate on the network state is true*) **then** *take the following [management] action(s)*. A given set of policies being defined by the administrators determine the network resources' behavior under dynamically changing conditions. In our example of Section 1.1, the decision of whether to reduce the number of open connections when the average response time of the server exceeds a predefined threshold, naturally fits into the policy-based management framework. Policy-based management framework is part of the IETF approach [IETF03]. However, it is not being part of the SNMP management framework. Using policies necessarily implies using monitoring mechanisms that gather the data needed for evaluating the policy predicates. Both types of monitoring (*i.e.*, statistical, and online) may be used depending on whether the policy is a short-, or long-term one. Thus, the discussion above fully applies to policy-based management. The policy-based management would greatly benefit from the hierarchical and event-driven monitoring approach, three-tier organization, and improved reliability, the generic issues that are in the focus of this work.

Enabling Technologies

One promising approach to scalable network management is using Active Networks [RS00]. One of the great advantages offered by active networks, is the possibility of *incremental deployment* of the new management and monitoring solutions. In this paradigm, the packets are no longer passive entities, and they act as a low-level mobile agents carrying an executable code along with them. This code is executed either within the switching elements, or dedicated *active engines* attached to them [RS00]. Active networks allow a variety of efficient management applications ranging from on-line control to efficient in-band statistical monitoring as was proposed by us in [BRS02b].

Among the first proposals to overcome the limitations imposed by the SNMP framework (see Section 1.2.1) was the Management by Delegation (MbD) approach [YGY91]. In this approach, an executable code of a management task is shipped (*delegated*) to the execution environment provided by the managed network element. This way, the management (and monitoring) algorithms are brought close to the resources that they monitor and manage. This dramatically shortens the control loop (especially in the Wide Area Network), reduces the bandwidth requirements, and facilitates trap-based monitoring schemes in which the delegated code acts as a local event generator mechanism for application-specific events. In the original MbD proposal, the delegated code is confined to a single location. This is, essentially, a framework for dynamically extending functionality of the management agent by code with constrained mobility.

Since the early days of the MbD approach, the advent of the mobile code technologies, such as Java, made some of the specific implementation details found in [YGY91, GY95] obsolete. However, the general framework remains very relevant.

The original MbD framework has been extended recently in a number of ways. First, higher levels of agent mobility have been proposed [LPK02, GB02, PWW00, ZHGW99, BPW98, Mou98, RD99, ZHGW99], resulting in the highly distributed management solutions that are very flexible on the one hand, while more difficult to develop, deploy, and control, on the other. This again brings the problem of meta-management into the focus and places it high on the research agenda. The mainstream approach is to develop a *middleware*, a software layer, that should take care of various meta-management aspects involved with the highly distributed management solutions, allowing the network administrators to concentrate on the design of the management applications rather than on the peculiarities of their deployment. One proposal being specifically aimed at the meta-management problems is MEMAD [GG00]. The purpose of MEMAD is to enable *Peered Distributed Managers* model in which a set of distributed managers cooperate on load-balancing executing management tasks that either are delegated to them, or being predetermined. However, high availability of the state-full monitoring tasks was not thoroughly addressed by this proposal[10].

Deploying middleware in the network management context is very much in-line with the current proliferation of the middleware for distributed client/server computing such as Java Enterprize Beans [Mic01, EST02], for instance. The main goal of the middleware is to isolate the application developer from the details of networking allowing him/her to concentrate on the application details rather than on the problems associated with its deployment in the networked environment.

The standard way to create a three-tier client/server application is to use an *application server* that provides the mid-tier run-time environment for application-specific objects. This approach is adopted by the Web-based network management proposals such as JDMK [SM03]. One important issue that is not addressed well by the standard application servers is high availability of the monitoring processes. The application servers are not aimed at continuously executing tasks. When a request arrives from the client, it is served by a thread from the pool of available threads. The worker thread returns into the pool after serving the request. The application specific state usually is kept in a separate database. Thus, availability level of the application depends on the external database. To allow distinguishing between different

[10] A position paper on monitoring middleware specifically addressing the high availability aspects of meta-management was presented by us at the same time as MEMAD in [BSD00]

method invocation instances performed on the same object, application servers may maintain *active primary-backup* replication of the transient conversational state between the client and the server-side object, so that when the primary copy of the server object fails, the backup can take over. The level of replication is, usually quite low, 2. The failure model that is tolerated by the standard solutions is, usually, **Failstop**, which is the simplest failure model. In this model, communication is reliable, and a server fails by *halting*. Halting means that when a participant sends a message to a set of destinations, it may halt before sending this message to all of them. Once it halts, the server remains in that state. The fact that a server has failed is detectable by other servers and by the clients (see Schneider [Sch84]).

Orientation on serving on-demand client requests, limits applicability of the standard application servers as the middleware platforms for monitoring. The monitoring activities are, usually, continuously executing, state-full objects, and not short-lived client-initiated activities. It is also desirable in many cases that the monitoring process would be independent from the external database that may be unavailable. Fail-stop model may be unrealistically benign for the typical setting in which the monitoring facilities are required to operate. In fact, a far more challenging **General Omission** failure model is more relevant. In the original General Omission model (see Perry and Toueg [PT86]), a participant fails by receiving only a subset of messages that have been sent to it, by transmitting only a subset of the messages that it actually attempts sending[11], and/or by halting. In our work, we deal with this model suggesting a novel primary-backup approach to high availability of the state-full monitoring processes within the tree-tier architecture aimed at the network monitoring, efficiently catering for the *disconnected mode* of operation which is crucial for the ultimate success of highly distributed multi-tier management paradigm.

1.3 Methodology Summary

The guiding principles that we used throughout this research can be summarized as follows.

- **Optimize bandwidth consumption**: since the monitoring tools in IP networks are in-band, reduced bandwidth consumption is essential for scalability. We specifically concentrate on bandwidth efficiency throughout this work.

[11] These failures serve to model network splits and merges due to malfunctioning of the switching elements in a multi-domain network.

- **Generic solutions**: we pursue generic solutions and high level of abstraction. Therefore, our contributions are not technology-dependent, and also can be extended beyond the network management *per se*.
- **Realistic approach**: we achieve extended functionality through the solutions allowing *incremental deployment* rather than through the revolutionary ones, requiring universal acceptance.
- **Re-use rather than re-invent**: we treat network management in general, and network monitoring in particular, as an instance of distributed computing. Thus, we aim at re-using the methodologies and tools that have been accumulated in other distributed computing, and networking areas, over the last two decades. Specific examples, include using active networks [RS00] as a practical means of deployment of our intermediate parking optimization for statistical monitoring tools. We use group communication [AfCM96] to develop our highly-available monitoring middleware on top of it; we use the emerging peer-to-peer overlay networks approach to facilitate scalable our hierarchical and event-driven on-line monitoring schemes Internet-wide.
- **Thorough evaluation**: we evaluate our proposals through implementation wherever this is possible. If implementation is either too costly, or is not required to appreciate the ideas, or is not possible due to the limited capacity of our facilities, we employ rigorous analysis and extensive simulation studies.

1.4 Overview of Our Approach

In this section, we summarize the discussion above, and specifically point out the contributions of this work and how they fit into the general picture of the network management research.

Recall that the monitoring facilities for managing the IP-based networks usually should operate in the **in-band** mode. In particular this means, that the monitoring tools compete over the network bandwidth with the same user-level services that they help managing. The monitoring traffic is transparent to the regular users, and regarded by them as an overhead. To achieve scalability, this overhead should be minimized.

We identified two different categories of in-band monitoring: the **on-line** monitoring, and the **statistical** one. Both mechanisms are needed for different applications in all functional areas of network management (see Section 1.1). We claim that reducing bandwidth consumption of these two mechanisms should be pursued by different means as follows.

- **Bandwidth Efficiency of Statistical Monitoring**: although the total bandwidth consumption of the statistical monitoring facilities can be reduced substantially through the smart in-network aggregation mechanisms (*e.g.*, [BMB+99]), the total volume of the statistical data may still be very large. At the same time, statistical monitoring is characterized by the relaxed timing constraints. We propose a novel algorithmic framework to minimize the interference of this bulky traffic with the traffic generated by the regular user services at times of transient network congestions when the bandwidth is scarce. This way, the network *goodput* is improved, and the scalability of statistical monitoring is increased. The key idea is allowing the network nodes to delay monitoring data by locally storing it along the routing path. This can be done, for example, in the *Active Network* paradigm, and in the overlay peer-to-peer management networks.

  In Chapter 2 we show that intermediate parking of the management data with non-stringent timing constraints can improve the network's goodput dramatically even if a very simple scheduling algorithm for intermediate parking is used. The parking imposes additional load on the intermediate nodes. To obtain minimal cost schedules, we define an optimization problem that we call the *Travelling Miser Problem.*

  We concentrate on the *on-line* variant of this problem, which is of greater practical interest. For this variant, we develop a number of enhanced scheduling strategies, study their characteristics under different assumptions, and evaluate their performance through a rigorous simulation study.

  We prove that if only one link can be congested, then our scheduling algorithm is $O(\log_2 B)$ competitive, where B is congestion time, and is 3-competitive, if additional signalling is allowed. For the case when the congested node can signal link recovery along the reverse forwarding path, we prove that our scheduling strategy is 3-competitive.

- **Bandwidth Efficiency of On-Line Monitoring**: using the centralized event-driven monitoring schemes proposed in [DR02], we develop a highly distributed, hierarchical event-driven monitoring framework that overcomes the feedback implosion problem inherent to the two-tier centralized control structures, and scales Internet-wide.

  Bandwidth efficiency of our solution (as well as of all event-driven solutions) highly depends on the statistical nature of the data that is being monitored. Therefore, our framework is best appreciated in the context of a specific application. To this end, in Chapter 3, we study especially challenging problem of retaining multicast group size estimation for the purpose of pricing the multicast traffic transmissions.

  We show that for applications like pricing, estimating the group size using our bandwidthconserving scheme, rather than finding the exact number of receivers may be sufficient. In particular, it may suffice to know at any given moment that the number of receivers in the group is between some predefined bounds, *thresholds*. This estimation is disseminated to the multicast ISPs worldwide. The ISPs then locally compute the price of transmitting a unit of bandwidth to the group, and present this price to senders in their domain.

The multicast group size estimation is a representative example of an additive metric that is used to control the network and users behavior through the on-line pricing actuator. In general, our algorithms can be applied to calculating any additive network-wide metric. The exact bandwidth savings, however, will depend on the statistical nature of the monitored data.

For the group size metric, we provide a rigorous definition of the problem, theoretically analyze our algorithms, and investigate their relative performance using extensive simulation studies. We show that on real Internet traces, using our algorithms helps reducing the bandwidth overhead of the monitoring by a factor of 2-10, depending on the required accuracy and a specific behavior of the group size.

Based on the discussion held in the previous sections, we conclude that the scalable monitoring architecture will be a multi-tiered and a fully distributed one. However, its ultimate success is highly dependent on efficiently solving the meta-management problems.

To this end, in Chapter 4, we present the *Highly Available Monitoring Services Architecture (HAMSA)* that we developed, describe its functional requirements, implementation details, and evaluate its performance. Specifically, we demonstrate how our middleware can be easily deployed for several monitoring applications. HAMSA allows a high level of availability and abstraction, with relatively low overhead. It copes with the Extended General Omission failure model in which send and receive omissions may happen, and processors may fail by halting[12].

The efficiency of our solution comes from the following observation. It was shown in [BMST92] that the General Omission failure model implies the lower bound of $n > 2 \cdot f$ on the degree of replication in the primary-backup protocols, where n is the number of replicas, and f is the number of failures. However, this restriction pertains only to communication between the clients and the servers in a two-tier model. In our three-tier architecture, communication between the management clients (the uppermost tier entities), and the servers indeed obeys the restrictions imposed by the low bound on the primary-backup protocols [BMST92]. We allow

[12]To incorporate continuous operation in which previously halted computers may subsequently recover into the failure model, we partition the time into successive periods in which all participants behave as prescribed in the original General Omission failure model [Sch84].

the periods of unavailability of the mid-level entities performing continuous monitoring tasks to the management clients, when such communication may compromise a *single object* behavior of the monitoring task. However, for communication of the monitoring tasks with the target management agents (the bottom tier), we relax the demand of [BMST92], on having a single primary system-wide at any given time instance, since the *read* monitoring operations are idempotent.

Thus, we allow multiple primaries to operate simultaneously in non-communicating network partitions transparently to the high-level clients who still regard them as a single entity. Upon a re-merge of the previously split network, our middleware *automatically* obtains the most updated monitoring state about the disconnected network elements that have been available in some network partition, and were unreachable in the others. To the best of our knowledge, HAMSA is the only monitoring middleware with such high-availability guarantees, specifically designed to address the meta-management challenges of the three-tier architecture.

1.5 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, Chapter 3, and Chapter 4, we present our specific contributions regarding improving the goodput for statistical monitoring, reducing the bandwidth consumption in on-line monitoring through the hierarchical and event-driven process, and high availability of the monitoring respectively. The chapters are self-contained, and can be read in any order. Chapter 5 concludes our presentation, and provides some outlook. The results presented in these chapters have been previously published, and the reader is encouraged to consult [BRS02b, BDR03, BDRS03] for further details.

Chapter 2

In-Band Statistical Monitoring

2.1 Motivation

Statistical monitoring tools generate considerable amount of traffic. This traffic utilizes the same network resources as regular user traffic, and therefore, may affect the network *goodput*.

As argued in Chapter 1, statistical monitoring data does not have stringent timing requirements. Examples include various logging facilities that subsequently transfer the accumulated data over the network, accounting and billing, long term history monitoring of the large distributed systems, *etc.* In all these cases the applications have a considerable freedom regarding the exact transmission time.

Since in most cases the end-users are not directly interested in the services described above, the impact of the administrative traffic on the user-visible network services should be minimized. For instance, it would be beneficial to preempt some management traffic at the time of network congestion and schedule it for later transmission when the congestion condition is over.

Thus, it is useful to differentiate between the higher priority user-visible traffic, and the lower-priority management traffic. It is important to stress, however, that in this work, the terms *high priority* and *low priority* refer to the same best effort traffic being differentiated solely by the timing constraints.

The user traffic, *e.g.*, HTTP packets, has to arrive at its destination within a short period of time (2-3 seconds for the HTTP example) while the low priority management traffic can be delayed much longer. It is important to note that both the high priority and the low priority traffic compete for the same limited amount of network resources.

Given this model, it is our objective to improve the goodput of the network by preferring the high priority (user) traffic over the low priority (management) traffic at times of high load, if timing constraints of the management traffic are flexible enough to admit extra delays.

In other words, if there are plenty of resources in the network there is no difference between the low and high priority traffic. However, when the resources become scarce, the user traffic is given a priority over the management traffic wherever possible. In this work, the resource that interests us is bandwidth[1].

We characterize every management message[2] originating at a management agent, the *source*, by a single parameter: *deadline*. This parameter defines the latest time by which the message should arrive at the manager station, the *destination*. In this work, we are not concerned with a specific way the management data is obtained. In other words, we deal neither with the specific measurement techniques, nor with the data semantics. We simply view management applications as producers of the traffic that is a subject to the deadline constraint.

Based on this constraint, we attempt to create an individual *itinerary* for every low-priority message, along the existing routing path between the source and the destination in such a way that the message would meet its deadline, and at the same time would incur minimal additional load on the routing elements and links along the path. This, of course, can be done only when we allow the message to "park" at internal nodes for some time.

One framework in which this is possible is the *active networks* paradigm. This paradigm allows to equip each packet with an "autonomous intelligence" enabling the packets to react to the network conditions much like the rational car drivers do when they react to the road traffic conditions. In particular, an active packet is capable of detaining itself in the network in case of unfavorable conditions, scheduling its transition for a later time. Storing a packet in the network using the switching elements themselves is, obviously, infeasible. Fortunately, this need not to be the case since active packets may be diverted to a special purpose machine that is decoupled from the switching element, *e.g.*, the *active engine* in [RS00].

[1] One may notice that this approach is applicable also beyond the network management domain. For example, exactly for the same reasons as above, it may be beneficial to detain large e-mail messages in the network at times of congestion, to improve the overall goodput. Similarly, in peer-to-peer overlay networks it may be worthy to use intermediate peers in order to improve the overall efficiency of the data transfer. However, in order to focus the discussion, we will refer to the network management traffic as our primary motivation throughout the paper, and discuss other usages of the proposed framework in the concluding remarks.

[2] We use the term message to refer to an individual piece of management information that should be delivered in its completeness to a specific destination to be useful. Each variable length message may be fragmented into more than one smaller *packets* by the underlying packet-switched network.

Figure 2.1 shows a very simple algorithm that can be used by packets in active network to achieve the above objectives.

```
while next link is congested
      wait for time interval t and check again
proceed to next hop.
```

Figure 2.1: Simple Algorithm using the Intermediate Parking Ability

A link is considered congested when its utilization crosses some threshold value. However, it is important to notice that *different* threshold values may be configured for low-priority and high priority traffic. Thus, these administratively defined threshold values can serve as a network management mechanism that forces the low priority traffic to "give way" to the high priority traffic when the competition over the resources becomes acute. When this mechanism for prioritizing the traffic is used, an additional advantage of the active network paradigm would be a higher reliability for the low-priority traffic.

Consider the scenario described in Figure 2.2. The source S is trying to send some management information to the target T. There is cross-traffic between A and B, and E and F respectively, which alternates, *i.e.*, whenever the cross traffic from A to B stops, the traffic from E to F starts. If the cross-traffic is intensive enough to raise the utilization level of the respective links above the low priority threshold, all attempts of S to send the information to T using regular means will fail[3] as the packets will experience loss, and no retransmissions will help. In contrast, in our simple algorithm, the packet will just park at the intermediate nodes waiting for the cross traffic from E to F or A to B to abate, and then it will continue

[3] *E.g.*, due to an application/transport level time out, after too many retransmission attempts.

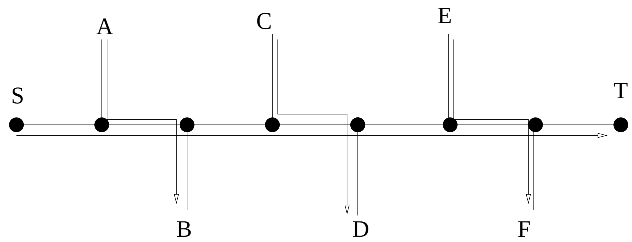


Figure 2.2: A simple cross traffic scenario

towards T. If delays introduced by waiting are not too large, the low-priority information will arrive in time to be useful for the management application, and the user traffic (that is responsible for transient congestion) is unaffected. This simple example shows that, indeed, the extra functionality introduced by intermediate parking, adds an additional value to the network.

It is reasonable to assume that the timing constraints of the management traffic will permit practical usage of the above scheme because while the round trip time is typically less than a second, the typical management traffic deadlines for non-real time management applications is, at least, in the range of minutes. This provides the required maneuvering space for the low priority messages.

However, since parking in the network consumes valuable resources of the hosting nodes, *e.g.*, memory, our scheme may introduce considerable overhead on the infrastructure. In order to prevent an excessive load on the hosting nodes, we associate some time-dependent *cost* with parking at a given node, and seek to optimize packet parking schedules in terms of these costs.

We formulate the problem of creating the low-cost parking schedules for the low-priority traffic in abstract terms as a graph theoretic problem and study its different variants. We call this problem the *travelling miser problem* because the behavior of packets resembles a strategy of a savvy traveler that is allowed to make stops in hotels offering different time-dependent prices along some route, trying to keep the total cost of the trip as low as possible while still being timely. We distinguish between the *on-line* variant, in which decisions should be made based on the partial information available locally at a given time, and the *off-line* variant in which all information about the network's behavior is known in advance.

Although solutions for both variants of the problem are relevant to the management applications, the on-line variant is of greater practical interest. For the off-line variant we explain its connection with other work done on the minimal weight path problem in time-dependent networks. For the on-line variant we present a scheduling algorithm called *Miser*. In addition to minimizing the interference of the management traffic with the user traffic (which is already achieved by the simple strategy above), this algorithm also minimizes the overhead on the nodes that contribute their local resources to enable intermediate parking. We prove that for the restricted case, in which only one link per route is congested at a time, the cost that our algorithm incurs on the active nodes is logarithmically competitive. We demonstrate our algorithm usefulness by studying its performance under other, more complex, network conditions, and comparing it to the simple strategy of Figure 2.1.

In this chapter, we study a simple model in which neither transmission costs, nor parking costs depend on the actual size of the low-priority message. This model is extended in

Appendix A to include the parking and transmission costs that are linearly dependent on the message size. It turns out that in this model, it is beneficial for the large management message to partition itself into the bulky main part, and a small "scout" part. This heuristics is called *Scout* algorithm. The Scout algorithm is at least as good as the Miser one, and under some conditions (*e.g.*, in case of an expensive network core) may result in schedules that are twice as cheaper than those produced by the Miser algorithm.

The rest of the chapter is organized as follows. In the next section we formally define the basic model. In Section 2.3 we formulate the problem, and then in Section 2.4 we describe and analyze the on-line and off-line algorithms. In Section 2.5 we present the results of our simulation study for the basic model. We survey the related work in Section 2.6, and provide a summary of the chapter in Section 2.7.

2.2 The Model

The motivation of the model presented in this section is to allow reasoning about the load of intermediate nodes and the cost associated with it. This model enables the definition and analysis of an optimization heuristics (see Section 2.4) that lowers the overhead imposed on the network nodes that accommodate the detained low-priority traffic[4].

Following [OR91] we consider a bi-directional *time-dependent* network $G = (V,\ E,\ W,\ P,\ L)$ with $V = \{1, 2, ..., n\}$ being a set of nodes, $E \subseteq V \times V$ being a set of links, and W, P, L being a set of *link weights*, *node parking weight densities*, and *link delays* respectively. $W = \{w_{i,j}(t) \mid (i,j) \in E\}$ are sets of time-dependent functions that represents the non-negative cost of traversing link (i,j) at departure time t. For simplicity, we assume that time is discrete.

$P = \{p_i(t) \mid i \in V\}$ are sets of time-depending functions representing the non-negative cost of spending one time unit at node $i \in V$ at time instant t. If a message arrives at node i at time t_1 and departs from the node at time t_2 where $0 \leq t_1 \leq t_2 < \infty$, then the *parking weight* is $P_i(t_1, t_2) \stackrel{\text{def}}{=} \sum_{t=t_1}^{t_2-1} p(t)$.

The time it takes to traverse link (i,j) is called *link delay* and denoted $d_{i,j}$. For simplicity we assume a constant delay of 1 time unit for every link.

Let R be a simple path from s to d, $R = \langle s \equiv v_0, v_1, ..., v_{n-1} \equiv d\rangle$. The *Travelling Miser Problem (TMP)* is defined (informally) as follows. A miser starts at node s at time 0. The

[4]In Appendix A, we study an enhanced model in which the parking costs linearly depend on the size of the low-priority message.

miser is required to reach the destination node d within the integer number $\mathcal{D}$ of time units, $0 \leq \mathcal{D} < \infty$, called *deadline*. At any given time instant t, the miser is at some node $v(t)$, and it can either park there for some time, or travel one of the outgoing links to one of the neighboring nodes.

Since we assume an underlying IP routing in the network, there exists only one relevant path from s to d on which packets can be routed. Thus, throughout this paper we concentrate on the single path case, *i.e.*, the message travels only along a single path R. However, although, we utilize a regular IP route, we enhance the basic IP routing mechanism by exploiting the active network capabilities along the route. Specifically, at any given time instant t, the miser is at some node $v_j \in R$, facing the following options for the next time unit:

- stay at the node v_j and pay $p_j(t)$ for parking;
- travel one link in the forward direction, *i.e.*, towards the destination, and pay $p_j(t) + w_{j,j+1}(t)$ for commuting;
- travel one link in the reverse direction, *i.e.*, towards the source, and pay $p_j(t) + w_{j,j-1}(t)$ for commuting;

The travelling miser problem is to devise a strategy that allows reaching the destination within $\mathcal{D}$ time units moving along a given path in either direction, and paying the minimal total cost for both commuting and parking. For convenience, we assume a fictitious self-link for every node in the path. The delay of this link is 1 (similarly to all other links), and its weight is 0. Traversing such a self-link corresponds to parking at the node for one time unit.

In order to state the problem more formally, we need the following definitions.

Definition 1. ***Itinerary:***
For a given $s, d \in R$ *an* **itinerary** $\mathcal{I}(s,d) \stackrel{\text{def}}{=} \langle s \equiv v(0), v(1), v(2), ..., v(T) \equiv d \rangle$ *such that*

- $\forall i \in [0,T] \quad v() \in R$;
- $\forall v(i-1), v(i) \in \mathcal{I}(s,d) \quad (v(i-1), v(i)) \in R$.

Thus, $\mathcal{I}(s,d)$ is a, possibly non-simple, path from source s to destination d consisting of all the original path links plus the fictitious self-links as described above.

The time T when the destination is reached is called the *termination time*, and the corresponding itinerary is referred to as ***T-termination itinerary***.

Definition 2. ***Parking Itinerary:***
For some node $j \in R$, an itinerary that consists only of $0 \leq t < \infty$ fictitious links (j, j) is referred to as **parking itinerary (or simply parking)** *at j for time t, and is denoted $\tilde{\mathcal{I}}(j, j)^t$.*

Definition 3. ***Itinerary Weight:***
For itinerary $\mathcal{I} = \langle s, v(1), ..., \ v(T-1), \ d\rangle$, itinerary weight *is $W(\mathcal{I}) \stackrel{\text{def}}{=} \sum_{t=0}^{T-1} w_{v(t),v(t+1)}(t)$.*

We are interested only in *finite weight* itineraries.

Definition 4. ***Feasible Itinerary:***
An itinerary $\mathcal{I}(s, d)$, $s, d \in R$ is feasible *for a given deadline $0 \leq \mathcal{D} < \infty$, being an integer number of time units, if and only if:*

- **The itinerary weight is finite***;*
- **The itinerary is timely***: $T \leq \mathcal{D}$.*

Definition 5. ***Reachable Node:***
Let M be an integer number of time units. Let R be some path in a time-dependent network G, and t be some time instance. A node $v_i \in R$ is **reachable** *at time t within M time units from another node $v_j \in R$, $j \neq i$ if and only if there exists at least one feasible itinerary $\mathcal{I}(R')$ for the deadline M and a sub-path $R' = \langle v_j, ..., v_i\rangle$, where $R' \subseteq R$.*

Definition 6. ***Time Distance:***
The total delay of a shortest finite itinerary $\hat{\mathcal{I}}(i, j)$, is called **time distance** *between the nodes i and j, and denoted $TD(i, j)$. Note that although in our case $d_i = 1$ for all i, $TD(i, j)$ may be different from $|j - i|$ due to infinite weights that some links may assume at certain time instances.*

Property 1. *Let $R = \langle 0, 1, 2, ..., n-1\rangle$ be a given path. $\forall \mathcal{I}(R)$*

$$W(\mathcal{I}) \geq \sum_{i=0}^{n-1} \min_t \{w_{i,i+1}(t)\}.$$

This property simply states that in order to obtain the minimal weight itinerary over the given path, each link in the path should be traversed at time instance for which the weight of the link is minimal. Note that this lower bound may be unattainable.

2.3 Problem Formulation

Now we can rigorously formulate the General Travelling Miser Problem (TMP) as follows.

For a given path $R(s,d) \subseteq G$, where G is a time-dependent network, and a finite deadline $\mathcal{D} \in \mathbf{I}^{+}$, find a feasible itinerary $\mathcal{I}(s,d) = \langle s \equiv v(0), v(1), ..., v(T) \equiv d \rangle$ for which $W(\mathcal{I})$ is minimal.

As explained in Section 2.6, the General TMP is an instance of a PSPACE-Complete problem, known as Canadian Traveler Problem. Fortunately, the general variant of TMP is of less practical importance since it corresponds to the cases when the network is highly unstable.

Therefore, in this work, we concentrate our efforts on a restricted variant of TMP, called K-block Recoverable TMP. This variation of TMP corresponds to scenarios in which the network is stable most of the time, but some links may become congested for certain time periods after which they recover. In K-block Recoverable TMP all parking weight densities are assumed to be constant in time $\forall t \leq \mathcal{D}$.

2.3.1 K-block Recoverable TMP

First, we define the notion of a *finite block*.

Definition 7. Finite Block:
*Let (i,j) be a link in a given time-dependent path $R(s,d)$. We say that a finite block of duration $t_2 - t_1$ occurs at link (i,j) at time t_1 (**block start time**), and terminates at time $t_2 \geq t_1$ (**block termination time**) if $\forall t$: $t_1 \leq t \leq t_2$ $w_{i,j}(t) \equiv \infty$.*

Note that fictitious self-links that correspond to parking at nodes are never blocked.

Let $\langle b_1, b_2, ..., b_k \rangle$ be an ordered sequence of triplets, where each triplet represents a single finite block by its link index, and start and termination times: $\forall l \in [1,k]$ corresponding to a link $l \equiv (i,j)$, $b_l \stackrel{\text{def}}{=} \langle (i,j), t_l^{start}, t_l^{end} \rangle$ where $0 \leq t_l^{start} \leq t_l^{end} < \infty$

The k-block recoverable travelling miser problem can now be formulated as follows.

Given a finite deadline $\mathcal{D}$ and a time-dependent path $R(s,d)$, a block sequence $\langle b_1, b_2, ..., b_k \rangle$, $\forall (i,j) \in R$ let link weight function be

$$w_{i,j}(t) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } \exists l \text{ s.t. } b_l = \langle (i,j), t_l^{start}, t_l^{end} \rangle \\ 0 < c_{i,j} < \infty & \text{otherwise.} \end{cases}$$

The k-block recoverable travelling miser problem is to find a feasible itinerary $\mathcal{I}^*(s,d)$ such that $\forall \mathcal{I} \neq \mathcal{I}^*$: $W(\mathcal{I}^*) \leq W(\mathcal{I})$.

Note that in order for a solution to exist, it is necessary that the the total sum of the blocking times be smaller than the given deadline. The relation of this problem to the actual network scenarios is straightforward. The infinite weight of a link corresponds to the link congestion. The level of link utilization that should be interpreted as a congestion is controllable through the administrative means. As one can easily verify, the k-block recoverable TMP is a special case of the general problem.

2.3.2 On-Line and Off-Line Problems

TMP can be formulated either as an *off-line* problem, or as an *on-line* one. In the off-line problem we assume that the input includes the full knowledge about the values of link weight functions along the path for any given instant of time.

The off-line variant is of interest for advanced planning of the optimal transmission schedules based on the long observed, relatively static load patterns. This is useful, *e.g.*, for the traffic that is generated in regular hours of the day, and when the usual network behavior is known in advance.

In the strictly on-line variant of the problem the values of time-dependent link weight functions are not known in advance, the miser has only a local knowledge about the path, and the current value of a link weight observed by the miser for a certain link does not tell anything about its future value.

Papadimitriou and Yannakakis showed in [PY89], that finding an algorithm with bounded competitive ratio for the strictly on-line minimal delay path problem in time-dependent networks is P-SPACE-complete. If we treat the parking and commuting costs as time-dependent delays, and assume a very large deadline parameter for the message, then the on-line TMP becomes an instance of the on-line minimal delay time-dependent path problem.

2.4 Algorithms

This section presents the algorithms for solving k-block recoverable TMP.

2.4.1 Off-Line Problem

The algorithm for finding minimal weight path in a general time-dependent network presented in [OR91] solves the general off-line travelling miser problem (as a special case) time for any constant deadline $0 \leq T < \infty$. Even though [OR91] solves the problem for the case of continuous (or piece-wise continuous) time-dependent weight and delay functions, the same algorithm may be used for our discrete case.

Since the off-line k-block recoverable TMP is a special case of the general off-line TMP, the algorithm presented in [OR91] also solves the off-line k-block recoverable TMP for any finite deadline.

It is useful to highlight some key ideas from the solution presented in [OR91]. The algorithm is based on the following *concatenation property* of the optimal (with respect to weight) time-dependent paths.

Lemma 1. Time-Dependent Path Concatenation Property

Let $R(s,d)$ be a time-dependent path. Let $T \in \mathbf{I}^+$: $0 \leq T < \infty$ be some time instant. Let $\mathcal{I}^*(s,d) = \langle s \equiv v(t_0), v(t_1), ..., v(t_{k-1}), ..., v(t_{N-1}) \equiv d\rangle$ be a minimal weight finite T-termination itinerary of length at most N among all T-termination itineraries of length at most N. Then $\forall k,\ 0 \leq k \leq L(\mathcal{I}^*)$, itinerary $\mathcal{I}_k \stackrel{\text{def}}{=} \langle s \equiv v(t_0), v(t_1), ..., v(t_{k-1})\rangle$ is a minimal weight finite T-termination itinerary of length at most k for destination $v(t_{k-1}) \in R$.

Proof. See [OR91]. □

In practice, we are interested in feasible optimal itinerary only with respect to some finite deadline T being an integer number of time units. Thus we can find optimal T-termination, $(T-1)$-termination, $(T-2)$-termination *etc.*, itineraries with length at most $\hat{l}$ using the concatenation property stated by Lemma 1, and choose a minimal-weight feasible simple itinerary among them.

Indeed, the optimal time-dependent path concatenation property allows recursive constructing of the optimal t-termination itineraries for every $t \in \mathbf{I}^+ : 0 \leq t \leq T$ and for every length l.

Suppose, for a given path $R(s,k)$, $k \in R(s,d)$, $\mathcal{I}(s,k)$ is an optimal t-termination itinerary of length at most $\hat{l}$ where $0 \leq t \leq T$. Let us denote its weight by $W_k(t)$. Then we have the following recurrent relations:

$$W_k(t) = \min_{i \in \{k-1,k,k+1 \in R\}} (W_i(t - d_{i,k}) + w_{i,k}(t - d_{i,k})) \tag{2.1}$$

$$W_s(0) = 0 \tag{2.2}$$

$$\forall t:\ 0 < t \leq T, \forall i \in R : W_i(t) = \infty. \tag{2.3}$$

From these recurrent relations we can compute the weights for optimal t-termination itineraries for every node in R and also the minimal feasible itinerary for reaching the destination before deadline T as shown in [OR91].

Note that if we are given an explicit representation of the parking weight functions in advance for time D (deadline), the algorithm is polynomial. However, if the weight functions are specified in a more concise form (*i.e.*, analytically), the algorithm is pseudo-polynomial, as its running time also depends on D.

2.4.2 On-Line Problem (Miser Algorithm)

In this section we present an algorithm that addresses the on-line k-recoverable TMP. We show that for the special case of $k = 1$ the presented algorithm is $5 + 4\log_2 B$ competitive where B is the block duration. For $k > 1$ the algorithm serves as a heuristic whose performance is demonstrated in Section 2.5 by means of simulation.

The idea of the on-line algorithm is as follows. The miser advances towards the destination using the non-fictitious links only, as long as he does not reach a block. At the blocked node, the miser has two options: either to stay at the node where the block has occurred (this stay is simulated by traversing the corresponding self-link), or move backwards on the reverse forwarding path that has been discovered so far. Assuming that the miser knows an upper bound U for the block duration, a competitive strategy for the miser is to pick up the cheapest node (in terms of the total cost of getting there and back plus spending a certain amount of time at the node as explained below) situated within roughly $U/2$ distance from the current location on the way back to the source, to spend the block time there. In other words, the miser has to pick up an itinerary with the minimal weight for the block duration. The intuition behind the algorithm is that if the miser has to spend some time en route on the way to the destination due to a block, he better do this using the cheapest itinerary. This way the overall cost of the trip is minimized.

However, the miser does not know the exact duration of any block, only an upper bound, that is not necessarily tight. Thus, it is beneficial to have a strategy for finding a tighter bound. This is done by doubling the existing estimation of the block duration at each stage of the algorithm. In order to explain the algorithm in detail we need the following definition.

`variables:`
    $j \in [1, \lfloor \log_2 B \rfloor]$: `number of phase;` $dir \in \{FWD, BCKWD\}$: `state;`
    v_b:`currently blocked node in the forward direction;` v_i: `current node;`
    s, d: `source and destination;` v_j^*:`intermediate destination of phase` j;
    T_j: `parking time of phase` j; $\mathcal{W}$: `set of known weights;`
initially: $j \leftarrow 1$; $dir \leftarrow FWD$; $v_i \leftarrow s$; $v_b \leftarrow nil$; $v_j^* \leftarrow nil$; $T_j \leftarrow 0$; $\mathcal{W} \leftarrow \emptyset$;
while $v_i \neq d$
  if $dir = FWD \ \wedge \ w_{v_i,v_{i+1}} < \infty$
    $\mathcal{W} \leftarrow \mathcal{W} \bigcup \{w_{v_i,v_i}, w_{v_i,v_{i+1}}\}$;
    `proceed to node` v_{i+1};
  else if $dir = FWD \ \wedge \ w_{v,v+1} = \infty$
    if $v_i = v_b$
      $j \leftarrow j + 1$;
    else
      $v_b \leftarrow v_i$; $j \leftarrow 1$;
    COMPUTE_ITINERARY(j, v_b, $\mathcal{W}$);
  else if $dir = BCKWD \ \wedge \ w_{v_i,v_{i-1}} < \infty \ \wedge \ v_i \neq v_j^*$
    `proceed to node` v_{i-1};
  else if $dir = BCKWD \ \wedge \ w_{v_i,v_{i-1}} < \infty \ \wedge \ v_i = v_j^*$
    `park at node` v_j^* `for time` T_j;
  else if $dir = BCKWD \ \wedge \ w_{v_i,v_{i-1}} = \infty$
    $dir \leftarrow FWD$;
    $j \leftarrow 1$;
  upon `termination time of phase` j:
    $dir \leftarrow FWD$;
endwhile

compute_itinerary(j, v_b, $\mathcal{W}$)
  `using` $\mathcal{W}$, `find node` v_j^* `that is reachable from` v_b `within`
  $t \leq 2^{j-1}$ `time units, and such that` v_j^* `yields the the minimum weight`
  `round-trip itinerary with the parking time:` $T_j = max\{1, 2^{j-1} - 2 \cdot TD(v_j^*, v_b)\}$;

Figure 2.3: Miser Algorithm

A simple round-trip itinerary between the two nodes $j, k \in R$ is an itinerary that is a *concatenation* of a shortest, possibly empty, itinerary that goes from node j to node k, possibly empty finite itinerary that uses only the (k, k) fictitious links, and the shortest, possibly empty, itinerary that goes from node k to node j. More formally:

Definition 8. Simple Round Trip Itinerary:
Itinerary $\mathcal{I}_{rt}(j,k)^t \stackrel{\text{def}}{=} \hat{\mathcal{I}}(j,k)\&\tilde{\mathcal{I}}(k,k)^t\&\hat{\mathcal{I}}(k,j)$ *where* $\hat{\mathcal{I}}(j,k)$, *and* $\hat{\mathcal{I}}(k,j)$ *being the shortest itineraries between* j *and* k, *and vice versa, and* $\tilde{\mathcal{I}}(k,k)^t$ *being a parking itinerary at node* k *for time* t, *and* & *being concatenation operation, is termed* ***simple round trip itinerary*** *with parking time* t *between two nodes* $j, k \in R$.

Figure 2.3 shows the Miser algorithm. Suppose a block occurs at some node $v_b \in R$ on the link leading towards the destination. The algorithm works in stages. At stage $j \in [1, \lfloor \log_2 U \rfloor]$ the miser chooses node $v_j^* \in R$ reachable from v_b within $T \leq 2^{j-1}$ time units, such that

$$W(\mathcal{I}_{rt_j}(v_b, v_j^*))^{2^{j-1}-2\cdot TD(v_j^*, v_b)}$$

is minimal. Then, the miser follows this minimal weight simple round trip itinerary for stage j.

In order to gain a logarithmic factor, the miser spends exponentially increasing periods of time away from the blocked node. After each such period the miser goes back to the blocked link and checks whether the block is lifted. If the block is lifted then the miser goes through, otherwise he proceeds to phase $j+1$. If the miser encounters a block while moving in the backward direction, he forgets all the previous execution history, switches the direction, and starts moving towards the destination.

Analysis

Obviously, if only one block may occur on the path before the miser's deadline expires, he will not go back and forth more than $\lfloor \log_2 B \rfloor$ times and cannot spend more than $2 * B$ time units at v_j^* in any phase j, where B being the actual duration of the block. Thus, the algorithm terminates after at most $1 + log_2 B$ simple round trips, and the overall itinerary that he builds, is finite. Constructing the round trip itinerary for every phase clearly takes $O(n)$ time where n being the length of the path. This makes the running time of the strategy $O(n \cdot \log_2 B)$. As we show in Theorem 1, if $k = 1$ the Miser algorithm is logarithmically competitive.

In case of of multiple blocks, *i.e.*, when $k > 1$, the Miser strategy serves as a heuristics. When the Miser algorithm computes the round-trip itinerary for stage j, it assumes that all the links that have been discovered in all previous phases will not be blocked during stage j. Naturally, this may be not true for a certain phase. It is easy to see, that when $k > 1$, the strategy is not competitive. If two blocks occur simultaneously on the two real links leading from the node (one in the forward direction for time B_1, and one in the opposite direction for time B_2), the

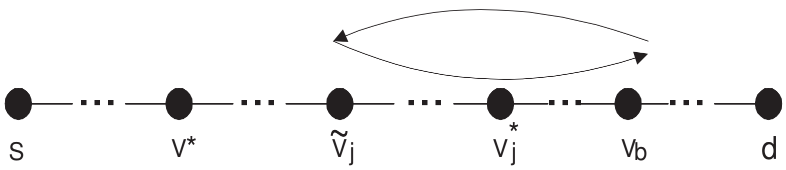


Figure 2.4: Simple Round Trip without Parking

weight of the resulting itinerary will increase by adding $B \cdot w_{v_b,v_b}$ where $B = \min\{B_1, B_2\}$. In contrast, the optimal off-line algorithm would build its itinerary in such a way that it will spend all the blocking time in the node with the minimal parking weight. Therefore, an adversary can always force the ratio between the on-line and the off-line algorithms to be the ratio between the minimal and the maximal link weights of the path. Fortunately, since blocks do not always follow the above worst case scenario in practice, the presented on-line strategy can be used as a heuristic. Its performance is studied using simulations in Section 2.5 under various assumptions on the blocks distribution.

Theorem 1. *Let $R(s, d)$ be a time-dependent path, as defined in Section 2.2. Let $\mathcal{D}$ be a finite deadline, and let U be the upper bound on the blocking time. If there exists only one block of duration B then the competitive ratio of the Miser algorithm is $5 + 4 \cdot \log_2 B$.*

Proof. There are two cases that should be inspected.
Case 1: The block happens at node $v_b \in R$ at time $t_{start} > TD(s, d)$.

It is easy to see that in this case, parking in the intermediate nodes renders no gain in minimizing the total cost of the itinerary, since all parking costs are positive. The optimal algorithm just advances directly to the destination. Since the Miser algorithm behaves identically, the competitive ratio is 1.

Case 2: The block happens at node $v_b \in R$ for time $B = t^{end} - t^{start}$, $0 \leq t^{start} \leq t^{end} \leq \mathcal{D}$, and t^{start} such that $TD(s, v_{b+1}) > t^{start}$. In other words, the miser cannot reach the destination without encountering the block en route.

According to the lower bound given by Property 1, the cost of the itinerary built by the optimal algorithm is *at least* the weight of the path (when all link weights are finite):

$$W(\hat{\mathcal{I}}(s, d)) \leq W(\mathcal{I}_{opt}). \tag{2.4}$$

Also note that every algorithm, including the optimal one, must wait in this case at least $B + t^{start} - TD(s, v_b)$. The optimal algorithm will do this (*i.e.*, wait) at some node called v^* (see Figure 2.4), and therefore its parking cost is at least $(B + t^{start} - TD(s, v_b) \cdot w_{v^*,v^*}$. Thus, we have:

$$W(\hat{\mathcal{I}}(s,d)) + (B + t^{start} - TD(s, v_b)) \cdot w_{v^*,v^*} \leq W(\mathcal{I}_{opt}). \tag{2.5}$$

Note, that since t^{start} is non-negative the following inequality holds:

$$W(\hat{\mathcal{I}}(s,d)) + (B - TD(s, v_b)) \cdot w_{v^*,v^*} \leq W(\mathcal{I}_{opt}). \tag{2.6}$$

For the on-line Miser algorithm we have:

$$W(\mathcal{I}_{miser}) \stackrel{\text{def}}{=} \sum_{j=1}^{\lfloor \log_2 B \rfloor \ +1} [(2^j - 2 \cdot TD(v_j^*, v_b)) \cdot w_{v_j^*,v_j^*} + 2 \cdot W(\hat{\mathcal{I}}(v_b, v_j^*))] + W(\hat{\mathcal{I}}(s,d)) \tag{2.7}$$

Let us find the upper bound for the cost of each phase j, $\forall j \in [1, \lfloor \log_2 B \rfloor]$ of the Miser algorithm. There are two cases.

Case 1: $1 \leq j \leq \lfloor \log_2 TD(s, v_b) \rfloor$.
Let $\tilde{v_j} \in [v^*, v_b]$ be the farthest node that is reachable from v_b in phase j by a round-trip itinerary without parking (see Figure 2.4). Since the Miser algorithm always chooses minimal weight round-trip itineraries for each phase, the cost of Miser itinerary in this phase is no greater than the cost of the round trip itinerary without parking to $\tilde{v_j}$. Therefore, for a *single* phase j:

$$W(\mathcal{I}_{miser}^j) \leq W(\mathcal{I}_{rt}(v_b, \tilde{v_j})) \leq 2 \cdot W(\hat{\mathcal{I}}(s,d))$$

.

From Inequality 2.4 we obtain that for a single phase j $W(\mathcal{I}_{miser}^j) \leq 2 \cdot W(\mathcal{I}_{opt})$.

Case 2: $\lfloor \log_2 TD(s, v_b) \rfloor < j \leq \lfloor \log_2 B \rfloor$.

Observe that Miser *always* parks at v_j^* that belongs to the sub-path $R(v^*, v_b)$ (including both ends). Let us consider a simple round trip itinerary from node v_b to node v^* with parking time $2^j - 2 \cdot TD(v_b, v^*)$ for each phase j. Since Miser chooses minimal weight itineraries for every phase, the cost of this round trip itinerary bounds the cost of phase in the Miser algorithm from above: $W(\mathcal{I}^j_{miser}) \leq 2 \cdot W(\hat{\mathcal{I}}(v_b, v^*)) + (2^j - 2 \cdot TD(v^*, v_b)) \cdot w_{v^*,v^*}$.

Since $W(\hat{\mathcal{I}}(v_b, v^*)) \leq W(\hat{\mathcal{I}}(s, d))$, we have:

$W(\mathcal{I}^j_{miser}) \leq 2 \cdot W(\hat{\mathcal{I}}(s, d)) + (2^j - 2 \cdot TD(v^*, v_b)) \cdot w_{v^*,v^*}$.

The maximal phase duration is $2 \cdot B$. Therefore, for any single phase j we have: $W(\mathcal{I}^j_{miser}) \leq 2 \cdot W(\hat{\mathcal{I}}(s, d)) + (2 \cdot B - 2 \cdot TD(s, v_b) \cdot w_{v^*,v^*} + 2 \cdot TD(s, v^*) \cdot w_{v^*,v^*}$.

Observing that $TD(s, v^*) \cdot w_{v^*,v^*} \leq W(\hat{\mathcal{I}}(s, d)) \leq W(\mathcal{I}_{opt})$, and using Inequality 2.6, we obtain that for a single phase j:

$W(\mathcal{I}^j_{miser}) \leq 4 \cdot W(\mathcal{I}_{opt})$

Then from Equality 2.7 we obtain (using 2.4 again)

$$W(\mathcal{I}_{miser}) \leq (5 + 4 \cdot \log_2 B) \cdot W(\mathcal{I}_{opt}). \tag{2.8}$$

□

2.4.3 Trap over RFP Model

Can we improve the Miser's worst case performance? The logarithmic competitive ratio of Miser comes from the fact that it requires the whole message to traverse the entire path between the blocked node, and the node yielding the lowest cost round-trip itinerary of phase j of the algorithm.

If our model can be slightly extended to accommodate small signal messages, that are propagated over the reverse forwarding path each time a transient congestion condition on the previously blocked link is over, the Miser strategy can be modified as shown in Figure 2.5, to obtain the competitive ratio of 3. We call this modified algorithm *Trap*, to distinguish it from the original version of the Miser strategy. Since trap messages do not park at the intermediate nodes, we assume their cost to be 0.

To analyze the competitive ratio of the Trap algorithm, we first prove the following lemma.

```
while next link is not congested
      proceed to next hop;
if next link is congested:
      for each phase j ∈ [1, ⌊log_2 B⌋] find node v_j^* that is reachable from v_b within
      t ≤ 2^{j-1} · T time units, where T is the time since the message departed
      from the source node, so that v_j^* yields the the minimum weight round-trip
      itinerary with the parking time 2^{j-1} · T − t;
      proceed to v_j^*;
if by the end of phase j no trap
      message signalling end of blockage at v_b is received, proceed to phase j + 1;
else //a trap message arrived
      proceed towards the destination;
```

Figure 2.5: Trap Algorithm using the signaling ability of intermediate active nodes.

Lemma 2. *Let v_b be the blocked node. If in phase j, message in the Trap algorithm parks at node v_j^*, then in phase $j+1$, the node $v_{j+1}^* \in [s, v_j^*]$*

Proof. By the algorithm definition, phase $j+1$ starts when no trap message is received from v_b at the end of phase j. Assume by contradiction that the statement of the lemma is wrong. *I.e.*, let v_{j+1}^*, the intermediate parking destination for phase $j+1$, be located between v_j^* and the blocked node (see Figure 2.4). This means that the round trip itinerary yielded by v_{j+1}^* has lower cost than the one that would be yielded by v_j^*, and any other node being located between the source s and v_j^* (including both these nodes). But since the parking costs are strictly positive, and all nodes between the source and the blocked node are reachable in every phase, then node v_{j+1}^* would have been chosen as the intermediate parking destination in phase j. Thus, we have a contradiction that proves the lemma. □

Theorem 2. *In the single block case, the competitive ratio of the Trap on-line strategy is* 3.

Proof. Note that according to Lemma 2, and by the fact that Trap builds minimal weight itinerary in every phase, the total cost of Trap for the entire block duration B is bounded from above by the cost of a simple round trip itinerary from node v_b to node v^* (that would have been chosen by the optimal algorithm) with parking time $B - 2 \cdot TD(v_b, v^*)$. Therefore the total cost of Trap is bounded from above as follows:

$W(\mathcal{I}_{trap}) \leq W(\hat{\mathcal{I}}(v_b, v^*)) + (B - TD(s, v_b)) \cdot w_{v^*,v^*} + W(\hat{\mathcal{I}}(s, v^*))$.

Noticing that $\hat{\mathcal{I}}(v_b, v^*)) \leq \hat{\mathcal{I}}(s, d))$, and $W(\hat{\mathcal{I}}(s, v^*)) \leq \hat{\mathcal{I}}(s, d))$, and using the lower bounds given by the inequalities 2.4, and 2.6, we obtain that:
$W(\mathcal{I}_{trap}) \leq 3 \cdot W(\mathcal{I}_{opt})$. □

2.4.4 Connection Oriented Transmission

The heuristics being discussed so far proposed a connectionless mode of operation. The active messages equipped with the autonomous intelligence took care of their own delivery to the specified destination. An arrival (or, alternatively, a loss) of an active message was not acknowledged by its destination. Since messages can get lost in the network, this approach applied as is may compromise reliability. Adding reliability does not imply having a connection oriented communication between the source and the destination. To add a fair level of reliability at the transport level, it is required to store copies of the active messages in the source node for possible retransmission. The destination is required to supply either a negative (or a positive) acknowledgement when a specific message is lost (or received) in order to enable retransmission and garbage collection at the source. A detailed discussion of the reliability mechanisms for the proposed algorithms is out of the focus of this work. It should be stressed, though, that reliability requires additional storage and bandwidth resources which may not always be available.

If, however, connection-oriented communication between the source and the destination is needed, and can be facilitated by the capabilities of the communicating parties, a simple active algorithm that would retransmit a message after a predefined time-out can be considered as an alternative to the proposed *Miser* and *Trap* heuristics. This algorithm may improve reliability (except for the pathological cases like the one explained in Section 2.1), and facilitate the shorter control loops by having the destination informing the source immediately after getting the message.

We look at one specific algorithm of this kind that we call *Reset*. In this algorithm, a message is dropped at the blocked node, and is retransmitted by the source when twice the round-trip time-out between the source and the destination elapses, and no acknowledgement for the message is received.

Performance of this algorithm serves as a helpful yardstick in our simulation studies allowing to identify the favorable and unfavorable conditions for the proposed on-line heuristics.

Note that while this algorithm can be modified to gain a logarithmic factor, by introducing the exponentially increasing retransmission timeouts similarly to *Miser*, and *Trap*, it is not competitive. The total cost of a schedule in the *Reset* algorithm depends on the parking cost of the source.

If, however, the source node is the cheapest node of the path, and there is only one block per path during the message lifetime, the exponential *Reset* algorithm saves an additive factor of $\log_2 B$ compared to *Miser*, because in *Reset* the message is simply dropped at the congestion point, and does not backtrack. At the same time, the exponential *Reset* is by the additive factor of $\log_2 B$ more expensive than *Trap* because in *Trap*, the message is "retransmitted" only once when the blocked link recovery is signalled.

2.5 Simulations

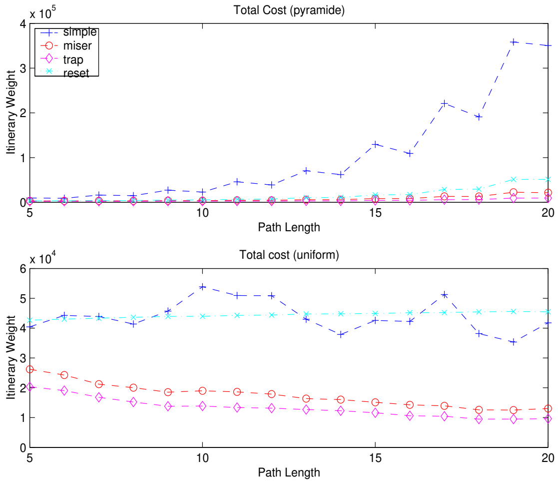
In this section, we present a simulation study of the on-line miser algorithm proposed in 2.4.2, and its variation proposed in 2.4.3. We refer to them simply as *Miser*, and *Trap* for brevity. Since competitive analysis provides indication only about the worst case, it is beneficial to study the actual performance of the active algorithms through simulation comparing them to other alternatives. We compare *Miser*, and *Trap* against each other, and also against the simple active algorithm explained in Section 2.1 (see Figure 2.1), referring to the latter as *Simple*, and the simple connection-like *Reset* algorithm of Section 2.4.4.

The overhead imposed on active nodes is quantified by the itinerary weight, *i.e.*, by the cost of the respective itineraries built by the algorithms (see Definition 3). In addition, we are interested in the propagation time of a packet through the network.

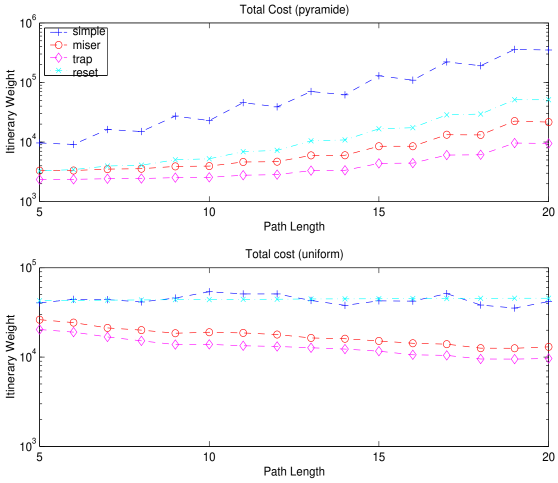
Our simulations consists of two sets. Section 2.5.1 investigates the behavior of the algorithms in a limited case when only one block may occur along the path during the specified deadline (*i.e.*, for 1-block recoverable TMP). Section 2.5.2 studies the algorithms for k-block recoverable TMP.

2.5.1 1-block Recoverable TMP

We start by comparing averaged itinerary weights for *Simple*, *Reset*, Miser, and Trap algorithms when only one block may occur along the path during a specified deadline (*i.e.*, for 1-block recoverable TMP). Although each one of these algorithms works under slightly differ-



(a) Average Itinerary Weight



(b) Semi-log Scale: Average Itinerary Weight

Figure 2.6: Average Itinerary Weight (1 block, sample size 100)

ent assumptions on the model (see discussion in Sections 2.4.3, 2.4.4), showing them all in the same figure also helps understanding the relative benefits of the models.

Figure 2.6(a) represents the simulation results as the averaged itinerary weight (cost) being a function of the path length for two different cost models *pyramid*, and *uniform* respectively. Figure 2.6(b) represents the same results with logarithmic scale on axis Y. The meaning of the cost models is as follows.

- ***Pyramid***: this cost model represents a deterministic distribution of parking densities to the nodes in the path simulating "cheap" edges and "expensive core". The parking weight density increases exponentially with the distance of a node from the edge, *e.g.*, $(1, 2, 4, 8, 16, 8, 4, 2, 1)$. The motivation for studying the pyramid cost model is that, usually, the load on the core switching elements increases sharply of the as we move deeper into the network cloud.

- ***Uniform:*** this cost model represents a random distribution of the parking weight densities to the nodes. Each cost is computed as 2^i, where i being an integer value uniformly distributed between 0 and 6.

For every cost model the graphs are obtained by averaging 100 runs of each algorithm for every path length ranging from 5 hops to 20 hops. In all the experiments the deadline was chosen to be $\mathcal{D} = 10000$ time units.

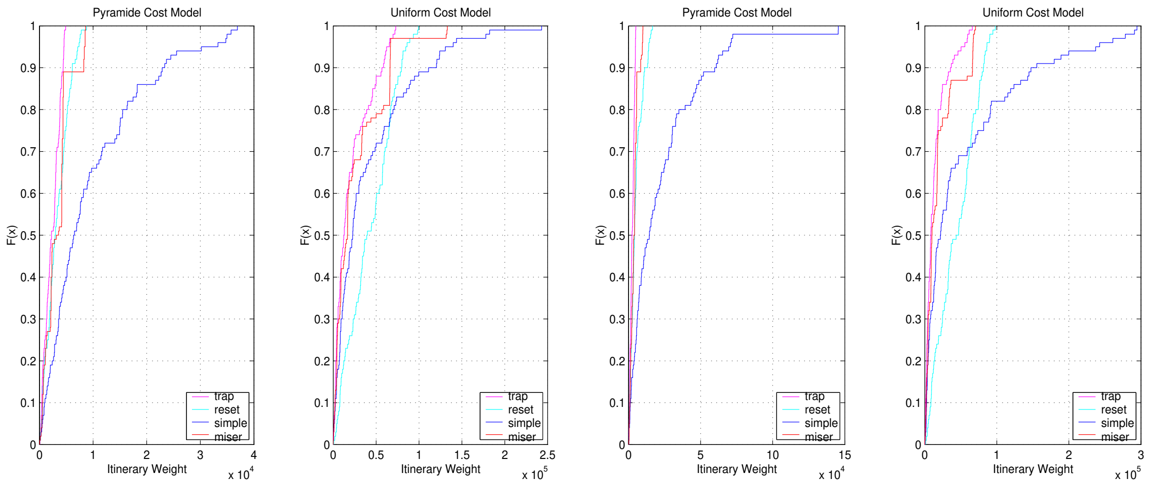
Every run for a given path length and a cost model is performed as follows. At the beginning of the run, we re-generate a path by allocating the parking weight densities to its nodes. Then, we randomly choose a link that will be congested. After this, we draw a random congestion duration b from $U(10, \mathcal{D}/2)$ distribution, and initiate a block at time 0 that will be removed after b time units. When this scenario is formed we execute all algorithms on the same scenario recording their performance: itinerary weight (cost), and total transit time.

Figure 2.6(a) shows that *Miser* is always superior to *Simple*, and *Reset* for both cost models while being always inferior to *Trap* which is consistent with our analysis. The difference is larger for the *pyramid* cost model. The explanation is that the cost difference between the core and and the edges is very sharp. Therefore when in *Miser* the message is caught by a long block in the middle of the path, after a few iterations it will migrate far towards the source, dramatically decreasing its costs while the simple algorithm remain parked in a more expensive location. In case of the *uniform* model these differences in the parking costs are alleviated.

To better appreciate the difference between *Trap*, *Reset*, *Simple* and *Miser* cost model, consider Figure 2.6(b) showing the same graphs using the the logarithmic scale on the Y axis. First, consider the Pyramid cost model. As one can observe, the *Reset* algorithm is only marginally better than *Miser* for shorter paths, and is about twice more expensive than *Miser* as paths become longer. The reason for this behavior is that since the blocked hops are drawn from the uniform distribution, approximately half of the blocked hops is beyond the most expensive node in the middle of the path (recall that the nodes' cost grow exponentially towards the middle, and then decrease exponentially towards the edge). Even if a blocked link is on the "other side" (*i.e.*, more than half-way towards the destination), *Reset* would still drop the message, and then retransmit it later paying all the way to the blocked node once again. *Miser*, in contrast, would not backtrack in this case, because all the nodes in the backward direction yield more expensive round-trip schedules. Thus, in approximately half of the blocks, *Miser* saves a factor of $\frac{B}{2 \cdot p}$ where B is the block duration, and p is the path length. In another half of the blocks *Miser* backtracks to the source and pays exactly the same parking price as *Reset*. The commuting price is different though. *Miser* pays $O(\log_2 B)$ for commuting while *Reset* pays $O(\frac{B}{2 \cdot p})$. Since *Miser* may over-estimate the block duration in the last phase, *Reset* may have marginal advantage when the blocks and the paths are short in approximately half of the cases.

As expected, the *Trap* heuristics results in the best performance in both models being approximately twice cheaper on the average than *Miser* for any fixed path length. The exponential *Reset* would yield a curve that would lie in between that of *Miser*, and that of *Trap* (it is not shown in the figure).

The simulation results for the uniform cost model show the same relative performance on the average for *Miser*, *Trap*, and *Simple* while *Reset* is often inferior on the average to *Simple*. This is due to the fact that since the nodes are priced randomly, the source is not necessarily cheap.



(a) Path Length: 5 (b) Path Length: 10

Figure 2.7: CDF of Itinerary Weight (1 block; sample size 100)

The variability of the itinerary weights obtained in different runs is high, because in each run the weight depends on the random block's location and duration. Therefore, although the average provides a useful indication, it is more informative to characterize the results by means of *empirical cumulative distribution function (ECDF)*. Figures 2.7(a), 2.7(b), 2.8, show the ECDFs for itinerary weights for both parking cost cost models for path length of 5, 10, and 15 respectively.

As one can clearly see, in the *pyramid* cost model, *Simple* exhibits high probability of very large itinerary weights. In this cost model, the total weight of the path increases exponentially with the path length. This means that *Simple* would impose much higher overhead on the active infrastructure if accidentally caught by the block at an expensive (*i.e.*, loaded) active node. *Miser*, *Trap*, and *Reset*, in contrast, would smooth these outlying cases.

In case of the *uniform* cost model, *Reset*'s performance deteriorates due to the fact that the source node may have a very high parking cost. As one can see, in this model *Simple* has high probability for large outliers (heavy tail), which makes its performance worse on the average. However, *Reset* builds itineraries of larger weights more often. For instance, as Figure 2.8 shows, in over than 80% of the runs *Simple* builds cheaper itineraries than *Reset*.

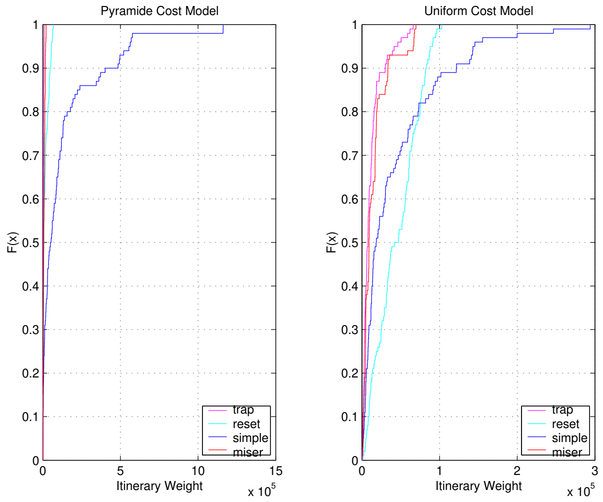
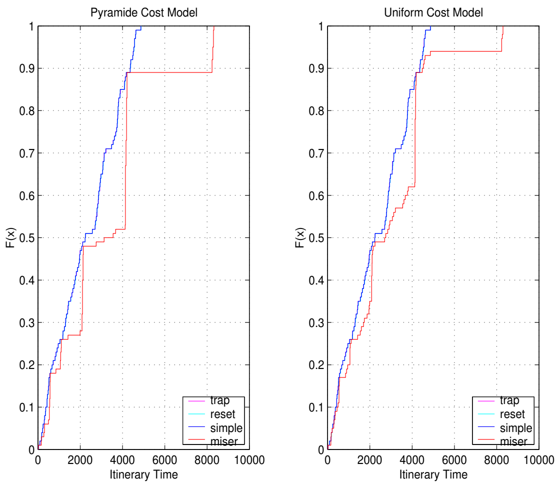
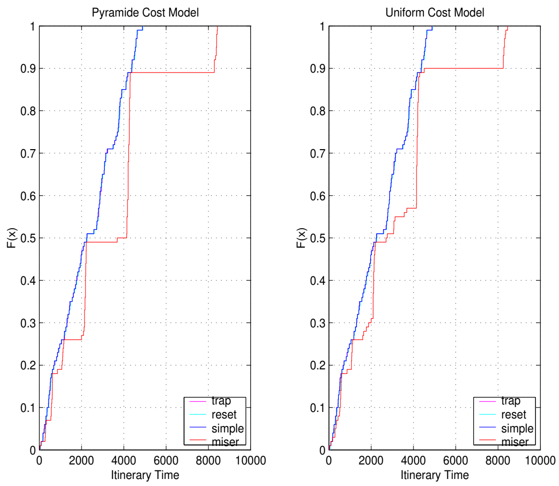


Figure 2.8: CDF of Itinerary Weight (1 block; path 15; sample size 100)

In addition to the total cost of the itinerary we might be interested in the total time required by it. Figures 2.9(a), 2.9(b), 2.10(a) represent ECDFs for the total transit time of the messages for both cost models when the path length are 5, 10, 15. As one can see, *Simple*, *Trap*, and *Reset* spend approximately the same time for message transmission. In fact the differences in total time between these three algorithms are so small that their curves appear as a single one in the ECDF graphs.



(a) Path Length: 5

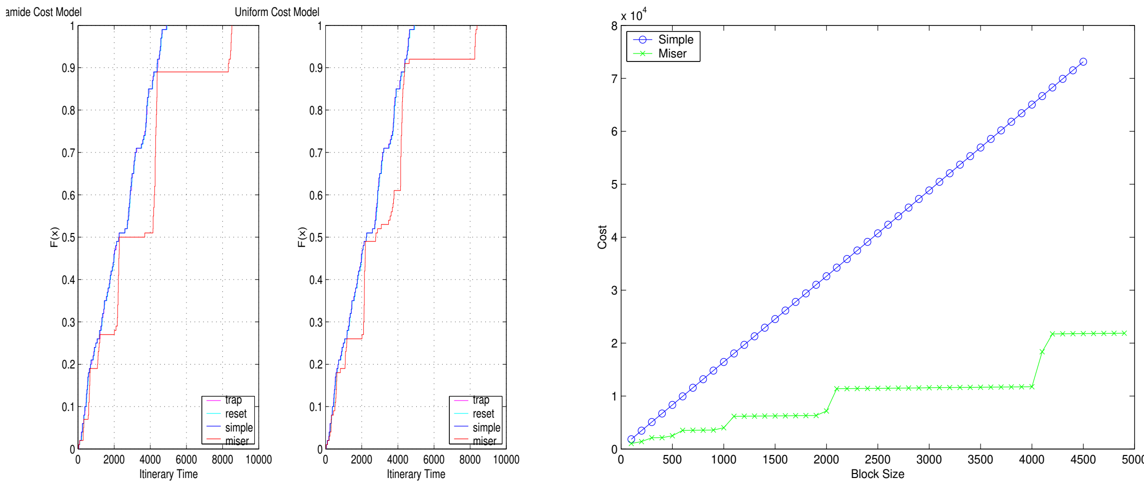


(b) Path Length: 10

Figure 2.9: CDF of Itinerary Time (1 block; sample size 100)

Miser, however, pays by time for the cost optimization it achieves. This happens because in many cases *Miser* overestimates block duration (consider the last phase of the algorithm). In the *Trap*, and *Reset* algorithms, no overestimation happens. This is the reason why they behave similarly to *Simple* with respect to the total transit time. One should remember, however, that this is achieved through the slight relaxations of the model as discussed in Sections 2.4.3, 2.4.4.

Finally, it is beneficial to measure the dependency of itinerary weight on the block duration. Figure 2.10(b) depicts the cost as function of deterministically chosen block durations and *pyramid* cost model. As expected, the cost of *Miser* increases logarithmically, while the cost of *Simple* increases linearly. As one can clearly see, for the longer congestion periods, *Miser* is definitely superior. Another interesting behavior is the exponential height and width of the steps in the performance of *Miser*. This is a clear depiction of the 'nature' of the algorithm operation, that doubles its waiting time if the block is still present.



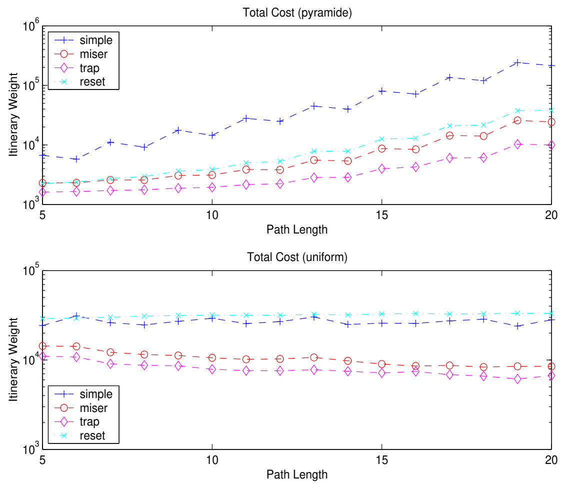
(a) CDF of Itinerary Time (b) Itinerary Weight dep. on Block Duration

Figure 2.10: Itinerary Time; Itinerary Weight (1 block; path length 15; sample size 100)

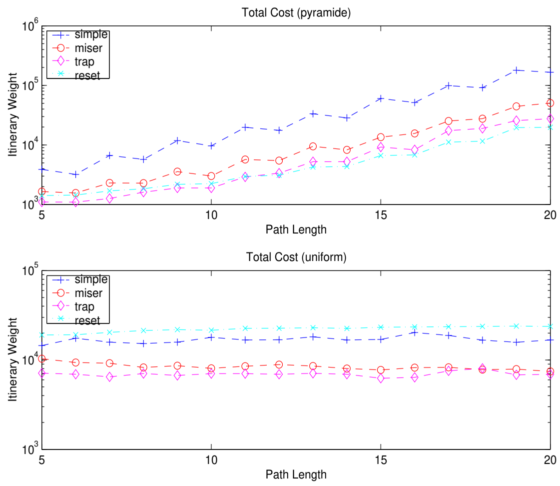
We want to attract the reader's attention to the fact that the encouraging results have been obtained even for relatively short paths which correspond to real topologies. The deadline assumed was also realistic given that the round trip time for such topologies is order of hundreds of milliseconds while the low-priority traffic deadlines is at minimum order of minutes. Finally, it is reasonable to assume that during a few minutes, indeed only one congestion would occur in the network, and the load on active nodes would not change dramatically, so that fixed parking weights are also reasonable. All this provides a strong motivation for the actual implementation of the proposed algorithm.

2.5.2 k-block Recoverable TMP

In this subsection we study the behavior of our algorithms for k-block Recoverable TMP. Figures 2.11(a), 2.11(b), 2.12 (a), and 2.12(b) show the averaged total costs of the itineraries built by the algorithms as a function of path length for k, the number of blocks, being 2, 4, 8, 14. respectively for each of the two cost models. The costs are shown on the semilogarithmic scale. The presented graphs have been obtained by averaging 100 runs for every value of k above, and path length varying from 5 to 20.



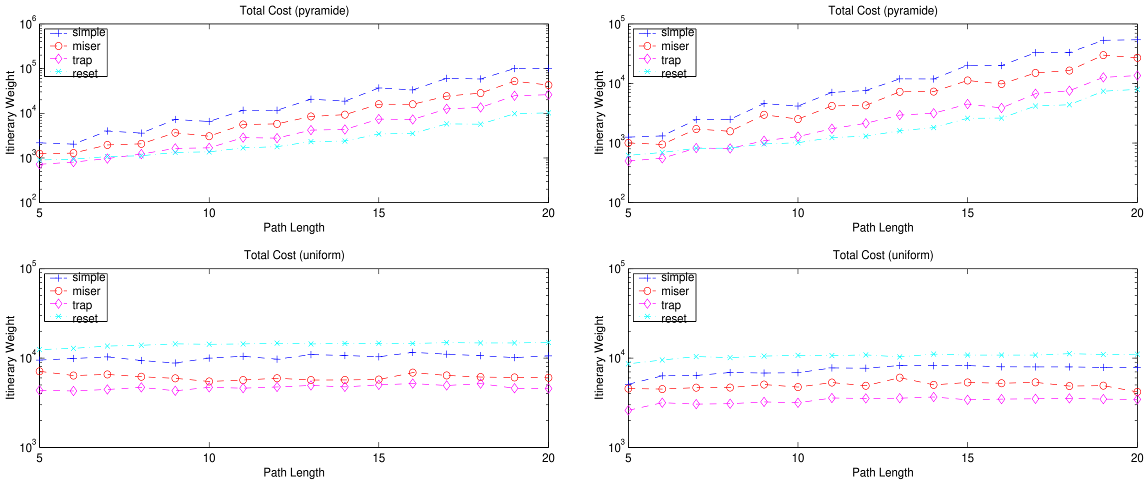
(a) 2 blocks



(b) 4 blocks

Figure 2.11: Average Itinerary Weight (sample size: 100)

Each run is performed as follows. In the beginning of the run, k links that will fail in this run are randomly chosen (repetitions allowed). Then a total block time is randomly chosen from $U(10, \mathcal{D}/2)$ as before. This total block time is split into k shorter blocks that are assigned to the chosen links. Then fail and recovery times are drawn from the exponential distribution for every link chosen in the first step. To make sure that there is at least one meaningful block, the earliest link failure time is treated as 0 time, and all failure and recovery times are shifted left by subtracting the earliest link block time from them. As one can observe from figures 2.6(b), 2.11(a), 2.11(b), 2.12(a), and 2.12(b), the relative behavior of *Miser*, *Simple*, and *Trap* algorithms remains similar for any fixed number of blocks. However, the absolute weights of itineraries built by the algorithms decrease with the number of blocks. This is natural since the more blocks can occur per fixed deadline D, the higher is the probability that the blocks will overlap. This effectively reduces the total blocking time by the additive factor proportional to the number of blocks.

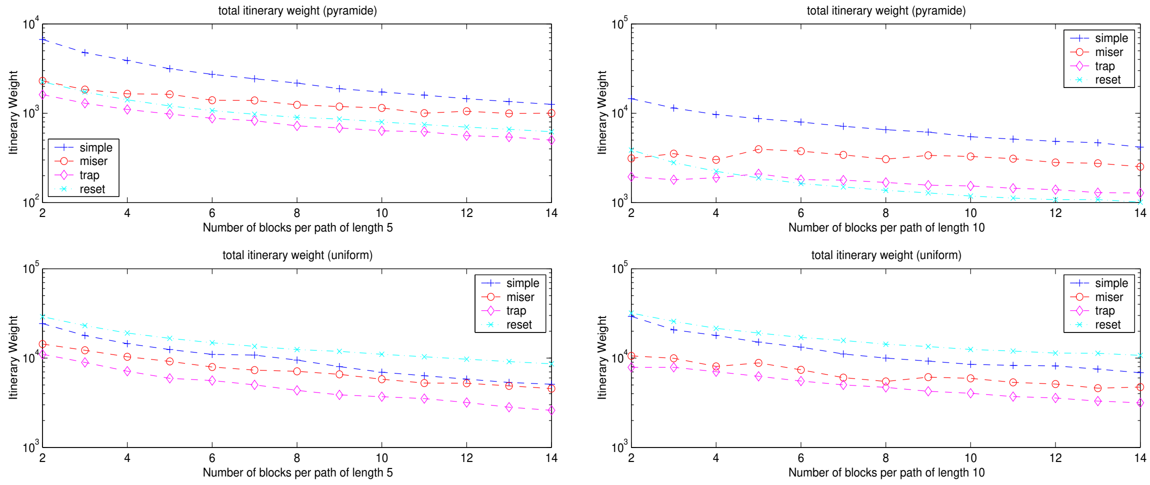


(a) 8 blocks (b) 14 blocks

Figure 2.12: Average Itinerary Weight (sample size 100)

Reset exhibits somewhat more interesting behavior that requires a more involved explanation. As one may see, in the *pyramid* cost model, *Reset* becomes clearly superior as the number of blocks grows. In the *uniform* cost model, however, *Reset* becomes clearly inferior even to the *Simple* algorithm as more blocks occur. To understand this behavior, we must recall that in our simulations, the total block time remains the same throughout all the runs. Thus, on the average, the block time per link decreases linearly as the number of blocks increases. Figures 2.13(a), and 2.13(b) show the total itinerary weights of the algorithms as a function of the number of blocks for path lengths 5, and 10 respectively. Figures 2.14(a), and 2.14(b) depict CDFs of the total itinerary weight for path length 5 for 2, and 14 blocks respectively. And Figures 2.15(a), and 2.15(b), present the CDFs for the total itinerary weights for path length 10 for 2, and 14 blocks respectively.

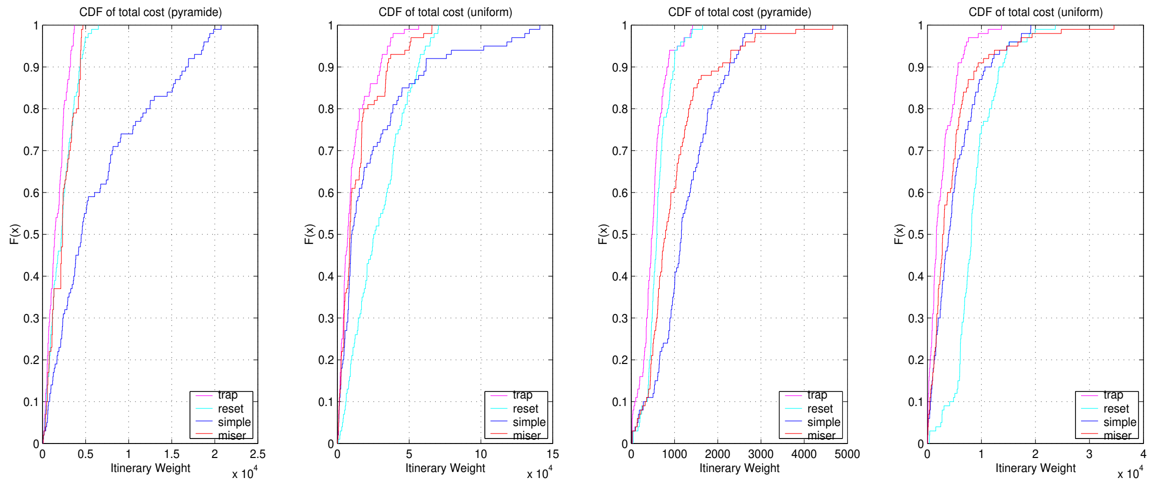
When more blocks occur per given path length, the blocks become shorter, and the probability of repeated failures occurring on the same link increases. This increases the probability for *Miser*, and *Trap* to build more expensive itineraries in both parking cost models since either repeated backtracking occurs in these algorithms due to multiple link failures, or *Miser*, and *Trap* have no effect since the packet gets locked between two blocks, and thus both behave more like *Simple*. In case of the *pyramid* cost model, source node is always the cheapest for roughly half of the path. Thus, in case of many short-term blocks it may be beneficial in this model simply to wait at the source until the failures abate. Note that *Reset* does exactly that. In case of the random cost model, the source's parking cost may be very high. Thus, spending the



(a) Path Length: 5 (b) Path Length: 10

Figure 2.13: Average Itinerary Weight (sample size: 100)

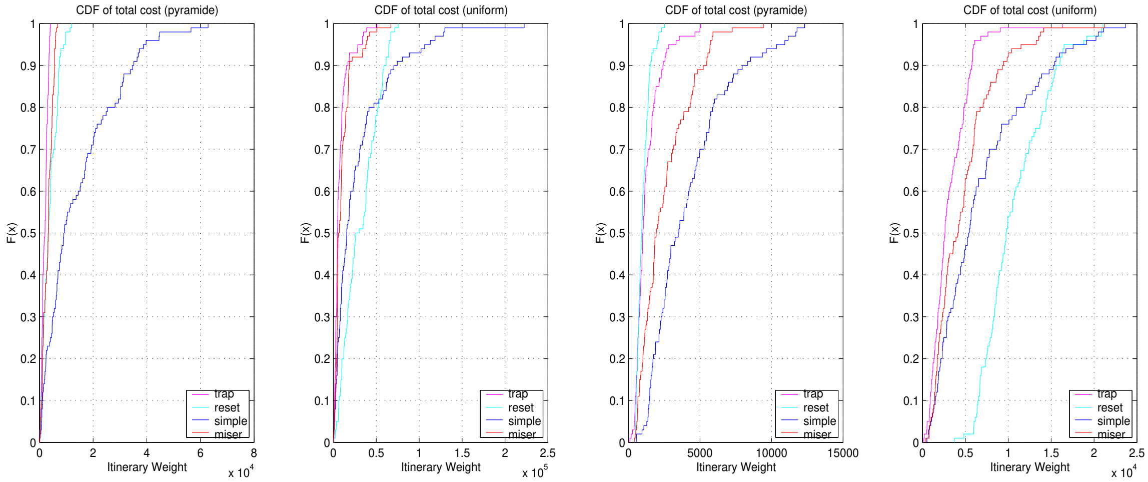
blocking time in the cheap place inside the network may be advantageous. This is the strategy pursued by *Miser*, and *Trap*.



(a) 2 blocks (b) 14 blocks

Figure 2.14: CDF of Itinerary Weight (path length: 5; sample size: 100)

Therefore, the trade-off between the strategies can be put as follows. When the network is very unstable, and the resources in the source node are sufficient (*i.e*, this node is cheap for parking), *Reset* is likely to yield the cheapest schedules. In cases when the network is unstable,



(a) 2 blocks (b) 14 blocks

Figure 2.15: CDF of Itinerary Weight (path length: 10; sample size: 100)

but the source does not have sufficient resources (*i.e.*, it is expensive), the on-line strategies like *Miser*, *Trap*, and even *Simple* are likely to be more economical.

2.6 Related Work

The problems studied in this chapter have much in common with the well known problem of finding a minimal weight path. Finding a minimum weight path in a static network has long been the subject for extensive research [Bel58]. The same problem for networks with time-dependent edge lengths has been studied by [Kla72, CH66, Hal77, OR96], and most extensively by [OR90]. Polynomial time algorithms for computing the minimal delay path in such networks when delay functions are known, have been demonstrated.

As was pointed out by [OR91], if real (even if strictly positive) time-dependent link weight functions are associated with traversing the links and parking in the nodes, the problem of finding a minimal weight path is, generally speaking, different from that of finding a path with minimal delay. The resulting paths are not necessarily simple and not necessarily minimal delay paths since in general delays are independent from the weights. The well-known optimality principle [Bel57], stating that a sub-path of any optimal path is itself an optimal path widely used for finding a minimal weight path in a static network, is not valid for time-dependent networks. This apparently prohibits an efficient solution. Different models have been presented to circumvent this problem: [SBSP00, IGSD98].

In [OR91], the most general model has been presented, and similar, but more restricted optimality principle has been established. We have discussed this principle in Section 2.4.1 along with the algorithm that is based on it. In [OR91] a polynomial time algorithm for finding a minimal weight path in time-dependent network when the weight functions are known in advance, has been presented. Thus, this work, essentially solves our *off-line* problem.

In practice, however, the weight functions may be not known in advance. This is the *on-line* variant of the problem in which the values of link weight and delay functions become known only upon an arrival to a vertices or a link, and their future values are unknown.

Very few studies of the on-line variant of the problem exist. One special case is known as the *Canadian Traveler Problem* [BNS91, PY89]. In the original version of this problem, studied in [PY89], a topology of the network (the map) is known in a advance. Each link is assigned a strictly positive finite delay. However, the map is unreliable. When a traveller following a minimal delay path, arrives to a certain hop, it may be impossible to proceed to the next hop since the next link is blocked (has infinite weight). Once a link is blocked it remains blocked forever. The problem is to travel from a source to a destination minimizing total trip time. In [PY89], it was shown that if the number of blocks is infinite then finding an on-line strategy with a bounded competitive ratio is P-SPACE Complete. The problem remains hard even if the blocks later recover as shown in [BNS91] for *Recoverable CTP*. In [BNS91] an exponential algorithm for a variant of *Recoverable CTP* that was called *k-block recoverable CTP* was presented. The difference of this problem from the original variant is that the number of blocks is fixed, k, and that links recover after a finite period of time.

Also, in [BNS91], a polynomial time strategy minimizing the total travel time from one vertex to another in such time-dependent network was presented. The crucial assumption made for achieving this solution was letting the link down time periods be *smaller* than the all internodal delays. This allowed an elegant recursive solution. Indeed, it is obvious how to solve the problem for $k = 1$. We just need to construct a minimal delay path, between the source and the destination, and for every vertex on this path, we need to construct a minimal delay path from this vertex to the destination. As long as there is no block, the traveller just advances along the minimal delay path. If a blocked link occurs, the traveller switches to an alternative path built from this vertex to the destination. Since only 1 block in the *whole network* may occur when $k = 1$, this algorithm is guaranteed to work. And it, obviously, has a polynomial running time. Now assume that we solved the problem for $k = m$, and wish to solve it for $k = m + 1$. If the traveler gets stopped by the block it may use an alternative (next minimal delay path), and because of the assumption on the inter-nodal delay being larger than any blocking time, when the traveller arrives to another node there are $k = m$ blocks in the networks. The case that we already solved.

Unfortunately, the assumption on the blocks being shorter than link delays renders the solution impractical for most networking environments where the situation is just the opposite: an internodal delay of a non-faulty link is smaller than the typical down time period.

In our study we, in fact, revisit the k-block Recoverable Canadian Traveler Problem for the special case of the topology being a single path, and we do not make any assumptions on the ratio between the internodal delay and the down time. We limit ourselves to finding only the "interesting" itineraries along this path. Namely, the itineraries that take less than the finite predefined time for the traveller to arrive from the source to the destination.

In [IS96] studies source-based routing approach to minimizing delay in time-dependent networks by trying several alternative paths in sequence. In this work the network topology is assumed to be known in advance along with the probabilities of the link failures. Under the assumption that during sufficiently long time no new link failures occur, [IS96] shows that a simple greedy strategy is optimal in a tree-like network. For general network they show that the problem of devising minimal source-based routes can be solved in polynomial space and is #P-Hard.

In [OR96] a distributed protocol for minimal delay routing in time-dependent networks oriented on the traditional routing model in which routers exchange control information about delays of the locally known links, is presented. Deployment of such protocol requires enhancements to the existing routing software, and, possibly, hardware.

Recently a new paradigm known as *active networks* has emerged [Pso99]. This paradigm allows seamless deployment of routing protocols at the level of individual *active packets*. Active networks turn out especially useful for wide range of network management applications [RS00, KS00, SJS+99]. In this work we maintain that this paradigm may be successfully deployed for transferring large volumes of management traffic while minimizing its impact on the user applications that consume the same network resources.

2.7 Chapter Summary

We presented a novel framework for handling low priority statistical monitoring traffic with non-stringent timing constraints. We demonstrated that intermediate parking may help in significantly reducing the competition for network resources between this low priority traffic and the user application traffic at times of congestion. Intermediate parking enables to increase the network's goodput, and, as a side effect, makes the management traffic more robust in presence of network congestion.

We proposed an efficient scheduling algorithm that minimizes the overhead imposed on the active network infrastructure itself by the intermediate parking. We showed that the desired effect may be achieved without the routing path modifications by constructing efficient parking schedules along a fixed routing path. In restricted, but practical, case the proposed algorithm is logarithmically competitive. In less restricted cases it serves as a useful heuristic, as demonstrated by simulations. Interestingly enough, the encouraging results have been obtained for relatively short paths which corresponds to the real networks' dimensions.

In Appendix A, we refine our basic model by introducing the parking and communication costs that depend linearly on the message size.

Chapter 3

In-Band On-Line Monitoring

3.1 Motivation

In [BACW90, DR02], it was shown that event-driven monitoring is much more scalable than the polling-based one. In [BACW90], Ben-Artzi *et al.* argue that a single manager can support by two to three orders of magnitude more threshold-driven agents than it can support the polling-based ones in realistic scenarios. However, in [BACW90], no specific mechanism for calculating the local thresholds was suggested. Dilman and Raz in [DR02], presented a comprehensive study of the threshold-based monitoring algorithms for solving Generic Monitoring Problem (see Section 1.1) for the target function $f(x_1, x_2, ..., x_N) \stackrel{\text{def}}{=} \sum_{i=1}^{N} x_i$ where x_i, $i \in [1, N]$, are the local variables managed by the network element agents. This is, indeed, a very general problem that is arising repeatedly in different network management applications. In our case, the value of variable x_i represents the number of active receivers in the domain of the mulicast ISP i.

In this chapter, we use findings of [DR02] as a starting point, and develop a generalized *reactive* monitoring mechanism that combines the threshold-based event reporting with a hierarchical control structure, allowing Internet-wide scalability of on-line monitoring.

Since specific cost benefits of reactive monitoring strongly depend on the statistical nature of the monitored variables, we need a specific management application context, within which to evaluate our proposed mechanisms. To this end, we study an important, and especially challenging problem of *multicast group size estimation* for the purposes of pricing, and billing multicast transmissions. In this context, a pricing function acts as an *on-line* actuator that influences behavior of the users by providing them with incentives to choose the most cost-efficient IP-based transport service (*i.e.*, either unicast, or multicast) for their transmissions.

Usually, IP multicast tree construction is guided by receivers. When the first receiver from a domain joins a specific group, the network provider should bear the control overhead of joining that group's tree and maintaining the new tree branch for as long as there exists at least one interested receiver in the domain.

In general, network providers may be interested in recovering the overhead costs of enabling multicast in their domains while at the same time encouraging both receivers and transmitters to use the most cost-efficient network service type for carrying their traffic. Not less important, the transmitters need to assess the benefits of transmitting into a specific multicast group, *i.e.*, the size of the audience that they can reach.

Recently, a number of proposals [CS01, HB00, HC99] suggested to address these issues through membership size-dependent pricing of multicast transmissions. In general, the price of a transmission grows, as the group size increases. However, for the reasons below, it is extremely difficult to find out an exact number of individual receivers in a multicast group at any given moment.

- The membership of a group is changing dynamically as receivers join and leave the group. Significant changes of the group size often occur over very short periods of time.
- The membership information is not confined to any particular location, but is distributed over a wide area network.
- Standards do not require the multicast routers to maintain an exact count of the local receivers.
- Centralized solutions for gathering the membership information are infeasible. In particular, if all membership notifications are being sent directly to a single management station, we get the problem known as *feedback implosion*.
- The communication overhead caused by distributed monitoring may be substantial.

We argue that for applications like pricing, it is sufficient to know at any given moment that the group size is between the two predefined threshold values. The main reason for this is that the senders are interested in stable pricing functions being insensitive to the minor fluctuations in the membership size. Accordingly, ISPs are not interested in tracking membership size changes when the cost of accounting exceeds the expected benefit from it.

We propose that the price of a multicast transmission into a specific group would be calculated locally by each ISP using the *globally consistent estimation* of this group size[1]. A specific pricing scheme requires knowing the group size up to some predefined constant factor $1 \leq \gamma < \infty$. Suppose that initially the group size is known to be S_0. As long as the group size S remains such that $\frac{S_0}{\gamma} < S < \gamma \cdot S_0$, there is no need in any additional accounting activity. We say that γ defines *low* and *high* thresholds values.

Multicast ISPs need to be notified about the new group size, *if and only if*, a global *threshold violation event (TVE)* occurs, which means that either low, or high threshold value is violated. This way, each ISP knows the group size up to factor γ at all times. Given this estimation, each ISP locally computes the price of transmitting a unit of bandwidth to the given group, and presents it to the local senders.

There exists a clear trade-off between the accuracy of the group size accounting, and communication cost paid for it. Thus, different pricing schemes may choose different QoS levels of accounting for controlling their overhead, while still maintaining a precision that suits them.

An exact procedure for thresholds calculation depends on the specific demands of the pricing scheme. For instance, the threshold values may reflect the maximal allowed change from the last known size of the group as above. Alternatively, the thresholds may be computed as certain percentiles of the maximal population size.

When we consider the typical lifespan of a multicast group, we identify three successive periods: membership build-up, saturation, and membership termination. Membership build-up and membership termination phases are characterized by rapid membership changes. However, these two periods are short relatively to the saturation phase that constitutes most of the transmission duration. Therefore, it is natural to consider an event-driven (*reactive*) monitoring scheme that generates more traffic in the first and last phases of the multicast transmission, and conserves bandwidth in the second stage. Thus, the total communication overhead of event-driven accounting for the multicast group size will be lower compared to a monitoring scheme based on periodic polling.

When we consider multicast, it is very natural to assume that we have a (bounded out-degree) tree structure where the leaves are the end routers that have a local count regarding the number of participants in the group. Since we want to know the total size of the group, our aim is to calculate the sum of the values of the leaves. This can be done in a hierarchical way in the internal leaves of the tree. In this way, every polling step consists of calculating the sum

[1]Additional factors such as spatial distribution of users, time of day, *etc.*, may be taken into account by the pricing scheme. However, in this work, we focus specifically on the group size estimation.

of the values up the tree and disseminating the value from the root to all the leaves if needed. This hierarchical polling process is a natural way to overcome the feedback implosion problem, and it also saves a factor of $\log N$ messages in each polling interval, where N being the number of ISPs. The next step is to combine it with the reactive monitoring process. This leads us to constructing a hierarchical and reactive monitoring algorithm that allows to maintain the multicast membership size estimation with very low communication overhead.

It is important to notice that making the membership monitoring function to use the tree constructed by multicast routing protocols requires changes to the latter. This approach has been taken by Holbrook and Cheriton in their EXPRESS proposal [HC99]. In this work, we are not proposing any changes to the existing routing protocols, but rather suggesting that the accounting function should be implemented as a separate on-line monitoring service, being orthogonal to the routing functionality. Other key differences between EXPRESS and our proposed algorithm are summarized in Section 3.12.

Section 3.3 explores the trade-off between the reactive and non-reactive monitoring using the model defined in Section 3.2. In Section 3.7 the basic form of our hierarchical reactive monitoring algorithm is described. In Section 3.8 we analyze its performance theoretically. Section 3.9 provides simulation methodology and results. Section 3.10 extends the basic hierarchical reactive monitoring algorithm and provides more simulation results. Section 3.11 validates our simulation and theoretical results with the real traces. Section 3.12 provides an overview of the related work. Section 3.13 provides a summary of this chapter.

3.2 Model and Problem Definition

Let $G(V, E)$ be synchronous network. Let $V' \subseteq V$ be a collection of nodes representing the ISPs participating as leaves in a multicast group tree. We assume that there exists a reliable unicast channel between each pair of nodes $v_i, v_j \in V'$ in addition to the spanning tree of the multicast group connecting all the nodes in V'.

Each $v \in V'$, maintains variable $x_i \in I^+ \bigcup \{0\}$, $x_i < \infty$. By $x_i(t)$ we denote the value of x_i at time $t \in I^+ \bigcup \{0\}$. The physical meaning of this variable is the number of active receivers in the domain represented by node v.

The system proceeds in rounds, where all messages sent at the beginning of a round are received by their destinations at the end of the round. Each round is completed during one time unit. In practice, the actual value of the time unit may be important. This is the maximal round trip time between any two neighbors in the network. In addition, each node in V' can send or receive no more than a fixed number of messages. All messages that exceed this threshold get lost. All messages are assumed to be of a fixed limited length.

Let $f(t) = f(x_1(t), x_2(t), ..., x_N(t))$ be a *target function* depending only on the values of the variables at time t. At any given time, every node in V' is interested in knowing whether the value of the target function exceeds a given value, or falls below another given value. These values are termed *upper*, and *lower thresholds* respectively. In this work, we are interested in a specific target function, $f(t) = \sum_{i=1}^{N=|V'|} x_i(t)$.

Let $\omega(n)$ be some function assuming non-negative integer values, and bounded from above.

Definition 9. *$f(t) = f(x_1(t), x_2(t), ..., x_N(t))$ is ω-constant function if and only if it is being constant on the semi-open intervals of length ω: $[0, \omega), [\omega, 2\omega)$, etc*

In fact, it is the same function after discretization on time intervals of length ω. Every node possesses only the value of its local variable. Thus, it takes some time for a node to assess the value of the target function, since it needs to communicate at least with some of the other nodes.

In this work, we assume that the membership size is $O(\log^2 N)$-constant. In other words, we do not account for the fluctuations in the target function value if these changes last for less than $O(\log^2 N)$ time units. Under the assumptions made above, we are interested in assessing the value of $f(t) = \sum_{i=1}^{N=|V'|} x_i(t)$, an ω-constant function.

Definition 10. *If at time t, $f(t) \leq H$, and at time t_a, t_a being minimal such time that $t_a > t$ and $f(t_a) > H$, we say that a* global upper threshold event, *or simply* cross-over *has occurred.*

We define a *global lower threshold event*, or *cross-down* similarly. Now we are ready to formulate our monitoring problem. Figure 3.1 provides the definition.

Given synchronous reliable network $G(V, E)$, a collection of independent random variables $x_1(t), x_2(t), ..., x_N(t) \in I^+ \bigcup \{0\}$, each being maintained by some $v \in V' \subseteq V$, a target function $f(t) = \sum_{i=1}^{n=|V'|} x_i(t)$, being ω-constant function, a set of threshold values $X_1, X_2, ..., X_m$, find an algorithm that guarantees that if and only if at time t_a a global threshold violation event occurs, all nodes in V' are notified about this event by the time $t_a + \omega$.

Figure 3.1: Problem Definition

Monitoring algorithms differ on the set of variables they measure, the actual timing of measurement, and the means of diffusing the information. We propose to compare different monitoring algorithms in terms of the total communication cost incurred by them over the time of their operation. Let $C(A, \delta)$ denote the total number of messages sent by algorithm A over time δ. *It is important to stress that we count traversing of each hop between any two neighbors in the network as a separate message.*

3.3 Reactive Monitoring versus Polling

In this section, we analyze the benefits provided by a reactive monitoring scheme versus a polling-based one in which the values of the monitored variables are being solicited at deterministic time intervals irrespective of the actual conditions in the network.

To separate this issue from all others, we consider a flat structure for information dissemination. More specifically, we consider the often used *many-to-one, one-to-many* structure. In this scheme the nodes use unicast channels to communicate their values to the relay, and the relay uses the multicast tree to disseminate the sum of the values to all ISPs at the edges [2].

In general, in this work, we are concerned neither with the actual topology of the network, nor with the exact topology of the multicast tree constructed by the underlying multicast protocol. To simplify matters, we assume that the height of a spanning tree constructed by the underlying multicast routing protocol is $O(\log N)$. To further simplify the analysis, we assume that the multicast delivery tree is a full binary tree. In our calculations, we defined the cost of a single multicast to be the number of hops in the tree, $c_m \stackrel{\text{def}}{=} 2(N-1)$ (because there is a total of $2(N-1)$ hops in the full binary tree with N leaves). Consequently, we define the cost of unicast, $c_u \stackrel{\text{def}}{=} \log N$.

We define a straightforward monitoring algorithm OB_c as shown in Figure 3.2.

Each predefined time interval every node sends its value to the dedicated relay-node. The relay computes the sum, and compares it against the global thresholds. If it detects a TVE, it multicasts the new size of the group. Otherwise, it sends no messages. If a node does not receive any message from the relay in response before a predefined timeout, it assumes that the size estimation of the group remains unchanged.

Figure 3.2: Straightforward Monitoring Algorithm (OB_c).

$C(OB_c, \delta) \stackrel{\text{def}}{=} O(\delta \cdot N \cdot c_u + e_\delta \cdot c_m)$, where e_δ being the number of threshold events detected by the relay node during time period δ, and N being the number of the monitored variables.

[2] The alternative, *many-to-many* communication pattern is not being considered since it is impractical in the WAN setting.

In [DR02] four different reactive monitoring algorithms using similar flat control structure were presented, albeit in a different context. The key idea of these algorithms is that instead of inspecting the local values each predefined time interval, the monitored nodes are assigned local thresholds. When a local threshold is violated, this triggers a report toward the dedicated manager that decides whether to poll other nodes.

With minor modifications this idea can be applied to our problem. Let $f(0)$ be the sum of the local values, *i.e.*, the size of a group at time 0. Let L and H be the threshold values *s.t.*, $L \leq f(0) \leq H$. Then every node representing a leaf router is assigned local thresholds $l = \frac{L}{N}$, and $h = \frac{H}{N}$.

- **Simple Value (SV):** every predefined time interval, each node updates its local value. If it detects a local TVE, it sends a trap to the dedicated manager. If at least one trap is received, the manager multicasts an unconditional polling request, collects the reported values, and calculates the sum. If either H, or L, is violated, the manager recalculates the local thresholds, and multicasts them along with the new sum. Otherwise, no messages are sent.

- **Simple Rate (SR)**: instead of checking absolute values, the nodes inspects the local rate of change. The manager makes an assumption that the rate of the change is bounded by ρ. The value of ρ is multicast to the nodes. Under this assumption the manager calculates the time interval during which polling can be safely omitted before the global sum has a chance to reach either H, or L. If the manager does not hear from the nodes until this time, it multicasts a polling request to validate the current thresholds. If a node detects that the assumption is locally violated, it sends a trap to the manager. If the latter receives at least one trap, it polls all the nodes and acts upon the received values similarly to SV above.

- **Improved Rate (IR):** Similar to SR, but instead of checking for the rate change at each step, the local node calculates the average change since the last report it sent. If this average is below the rate threshold, no trap is sent to the manager. This improvement increases the chances to save the cost of communication from the nodes to the manager, which may be very significant.

- **Improved Value (IV):** Similar to SV, but instead of acting upon a single trap, the manager lowers the local thresholds for the nodes a little bit, *i.e.*, instead of l, h, the nodes receive local thresholds $l + \lambda$, and $h - \lambda$. A single trap does not trigger the global polling. Instead, the manager calculates the minimal and maximal possible values of the target function assuming the minimal and maximal values for the nodes that have not reported. If there is a chance for a global TVE, the manager polls the nodes as in SV. The basic idea in this improvement is that although the possibility of an individual trap slightly increases, the possibility of the global poll decreases which, in turn, decreases the total cost of the algorithm.

For all the algorithms above, the communication cost calculation is, essentially, the same. We refer to them as *flat* algorithms, since the control structure they use is flat. Let r_δ be the total number of local traps sent by all nodes to the manager over the time interval δ. Let e_δ be the number of global events detected by the manager, and k_δ be the number of global polls initiated by the manager. Then $C(A) = O(r_\delta \cdot c_u + c_m \cdot (k_\delta + e_\delta) + c_u \cdot k_\delta)$.

In the following section we explain the simulations methodology that is common to all simulation studies performed in this chapter. Section 3.6 presents the comparative simulation study of the flat reactive algorithms versus the polling-based algorithms for different methods of threshold.

3.4 Simulation Methodology

3.4.1 Synthetic Dataset

In this work, we focus on the short-lived multicast sessions. The reason for this is that shorter sessions are less stable, and, therefore, are more challenging. Following [AA97, CA01], we model the inter-arrival and stay times of the multicast receivers as exponentially distributed variables with means μ_{arr}, μ_{stay}. The inter-arrival time is the time between consecutive join operations of the same user. The stay time is the time between consecutive leave operations of the same user. We assume that the users are uniformly distributed over ISP domains. Each user is assigned a fixed domain number throughout a single trace. We assume a fixed ratio of users to domains, namely, 30. Thus, for each fixed number of domains, we know the total number of receivers. For instance, for 256 domains we have the total of $256 \cdot 30 = 7680$ potential receivers.

Each synthetic multicast trace that we create consists of 10 fixed length *sessions*. Each session models a multicast transmission of a specific popularity. As explained earlier, a session lifespan consists of three consecutive stages: membership build-up, membership saturation, and membership termination. The popularity of a session refers to the average number of receivers in the membership saturation stage. To model sessions of different popularity we compute 15%

regions of the receivers space. In our example, for 7680 users the threshold values computed this way will be $0, 1152, 2304, ..., 7680$. Each pair of neighboring thresholds defines a different *popularity index (rating)* of the group.

We draw the popularity rating for each session out of the uniform distribution. Then, we randomly choose a point within the interval defined by the rating and set the expected average number of receivers for this session to be that point. For instance, if we draw 0 as the group rating for a given session, the average number of users for this session will be set as a random number between 0 and 1152, in our example.

Let U_{total} denote the total number of potential receivers. We fix μ_{arr} throughout all sessions, to be $0.1 \cdot d$, where d is the session duration. Then, we find μ_{stay} for a given session in the multicast trace using the following formula:

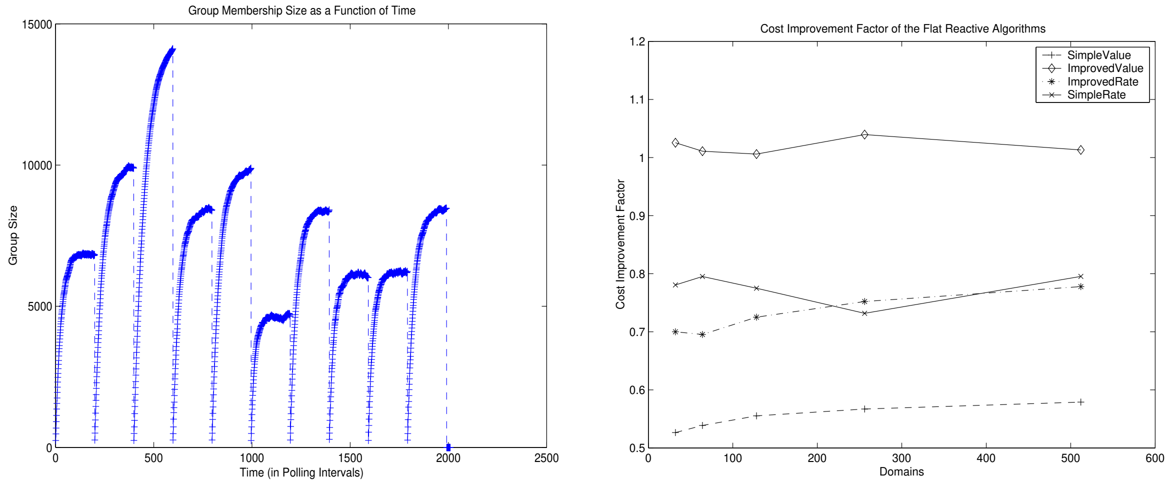
$$U = \frac{\mu_{stay}}{\mu_{arr} + \mu_{stay}} U_{total}. \tag{3.1}$$

This methodology removes a statistical dependency on the specific parameters of the model. Figure 3.3(a) provides a sample trace of the group membership for 512 multicast domains, 15360 potential receivers acting according to the exponential model above, and being uniformly distributed among the domains. There are 10 different multicast sessions per trace. All sessions are of equal length, 1800 time units. Each session starts to build its membership from 0, and at the end of transmission all users leave the group simultaneously. There are no inter-session idle periods, and the sessions themselves are relatively short. This is, of course, a very extreme scenario. In reality, the membership not necessarily drops to 0 at the end of each session, and exist larger periods of stability. Our model serves us as a basic framework, and the basic scenario can be changed by adding stability periods that freeze the membership activity to the trace. We express these stability periods as a certain percentage of the total trace duration. As suggested in [SA00] in today's MBone it is common to observe bursts of activity around some well publicized events interleaved with the long inactivity periods.

The cost of a single unicast in this trace, $c_u = 9$ messages, the cost of a single multicast transmission, $c_m = 1022$ messages.

3.5 Estimation Methodology

The requirements on estimation come from the application. Two following general methodologies are conceivable:



(a) Sample Membership Dynamics (b) Cost Improvement Factor (Maximal Load)

Figure 3.3: Performance Gain of Reactive Monitoring

- **Relative to Total (R2T)**: the estimation purpose is to show what percentage of the total maximum population is active. The total space is partitioned into percentiles. These percentiles also define precision of the estimation.
- **Relative to Last (R2L)**: the estimation purpose is to show the change of the group size relatively to the last known size. Precision of the estimation, γ, should be specified separately. At every given moment the estimation provides the group size up to a multiplicative factor γ.

In general, R2T is more applicable to the large groups while R2L is applicable to small and large groups alike. Since R2T is less precise than R2L, it incurs smaller communication overhead.

3.6 Performance of the Flat Reactive Algorithms

Figure 3.3(b) presents the average ratio (cost gain factor) between the total message cost of OB_c (the straightforward monitoring algorithm based on the dedicated manager) and the flat reactive monitoring algorithms. The cost gain factor, $C(OB_c)/C(A)$ is shown as a function of the number of simulated domains. The estimation methodology used for obtaining these results was R2T with 15% precision. One thing to notice is that due to the synthetic dataset creation methodology, performance of R2T and R2L for the same numerical values of precision

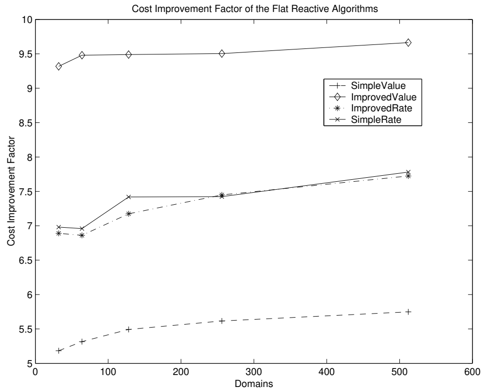


Figure 3.4: Cost Improvement Factor (90% Stability Time)

(in this case %15) is very close. R2L estimation methodology is explored in greater detail in Section 3.11 where the data comes from the real traces while the total population is unknown.

To eliminate the statistical dependency, 200 different traces have been generated per each number of domains. Each point in the graph represents the averaging of these runs. For all algorithms standard deviation of the resulting ratios ranges between 0.012, and 0.02. Therefore, the averaging indeed makes sense.

First thing to notice about the presented graph is that the results are rather disappointing. Simple Value and Simple Rate algorithms are about twice more expensive than OB_c. Improved Rate is slightly better, and Improved Value (the most efficient algorithm being presented in [DR02]) is only marginally better than OB_c.

For Simple Rate and Improved Rate algorithms, the initial choice of the upper bound on the rate of changes, ρ, considerably influences the performance. Similarly to [DR02], we performed a number of simulations in which this parameter has been varied in order to determine the value rendering the best performance. In the figure, only the best performance of the algorithms is shown. The performance maximum has been attained for Simple Rate and Improved rate algorithms for $\rho = 0.0015 \cdot total\_number\_of\_users$. The preliminary simulations used to determine this parameter for attaining the maximal performance are not shown here. Under the maximal load conditions these two algorithms almost do not differ. Similarly, for the Improved Value algorithm the best performance was attained with $\lambda = 0.02 \cdot total\_number\_of\_users$.

Why these reactive algorithms that have been demonstrated to be so efficient in [DR02] perform so poorly for multicast group size monitoring? The primary reason for this is a different statistical nature of the workload.

Throughout all the traces, the probability that at least one variable violates its local threshold has been measured. We found that it is 0.99 on the average. Moreover, on the average, $\approx 1/2$ of the variables send their traps to the manager in *each* round. For Simple Value and Simple Rate algorithms even a single trap message triggers the global poll in almost all rounds. However, when we measure the probability of detecting a global threshold event at the manager, we find that it is only 0.02 on the average.

This means that although there may be a lot of continuous activity in the leaf routers with many users joining and leaving the group, these changes do not affect the total size of the membership considerably. Therefore they are not worthy of accounting.

However, in the Simple Value, and Simple Rate algorithms the global polling operation is triggered by even a single trap received by the manager. The global polling means multicasting of the polling request and collecting the unicast replies from the nodes. Since about half of all nodes have already sent their traps in the previous round, this operation makes the algorithm at least twice as expensive as the straightforward convergecast and multicast performed by OB_c. Improved Value algorithm performs better because a single trap does not trigger the global polling operation, although the probability of the local threshold violation is slightly higher. The increase in this probability does not have any significance in this scenario though, because the probability of local trap was almost 1 anyway.

The lesson that we learn from this extreme scenario is that all flat reactive algorithms suffer from the same intrinsic problem. Namely, *the threshold events of no significance are communicated directly to the manager.*

Given this, does the reactive scheme provide any benefits at all when the flat control structure is used? When we consider the more realistic scenarios when the traces include periods of "no membership activity" that correspond to either a very popular event *e.g.*, the transmission of the final game of the World Cup, or to the temporary absence of events, we obtain a considerable cost improvement even in the flat model.

Figure 3.4 shows the cost ratio of the same algorithms measured on the traces with 90% stability time being attributed to the reasons above. As one can see (note the change in scale on axis y) the cost ratio is substantially better showing that even the simplest reactive monitoring algorithm, Simple Value, renders the improvement factor of over 5. The Improved Value algorithm performs almost an order of magnitude better than the straightforward algorithm.

Observing this behavior of the flat reactive monitoring algorithms, leads us to developing a different monitoring algorithm that combines the benefits of reactive monitoring with the hierarchical information dissemination scheme that suppresses the local threshold events from being being globally propagated. By the same token, the hierarchy solves the problem of feedback implosion that have been ignored so far for the sake of discussion. While this problem, indeed, can be ignored for the current MBone whose actual size roughly corresponds to our simulations, the feedback implosion problem will become acute when the multicast will be deployed on the large scale.

variables:
  $left$, $right$, $thresh[m]$:left and right children in the tree, and
  the global thresholds respectively;
initially: $L = thresh[i] < f(0) < thresh[i+1] = H$; $\alpha_1 \leftarrow H/f(0)$; $\alpha_2 \leftarrow f(0)/L$;
if (REPORT(s) m_l, m_r from $left$, or $right$, or both)
  act as an intermediate node;
if ($m_l.x + m_r.x \geq h$ or $m_l.x + m_r.x \leq l$) //global event
  determine new L and H using $thresh[m]$ array;
  calculate new α_1, and α_2;
  send UPDATE with new L, H, α_1, and α_2 to $left$ and $right$;

Figure 3.5: HRMA (the Root Node)

variables:
  $left$, $right$, $thresh[m]$:left and right children, and the parent in the tree.
if (UPDATE from $father$ containing values α_1, α_2)
  $h \leftarrow x_i \cdot \alpha_1$; $l \leftarrow x_i \cdot \alpha_2$;
every $2\log^2 N$ **rounds update the local value**, x_i;
if ($x_i \leq l$ or $x_i \geq h$)
  send REPORT to $father$;
else
  sleep for $2\log^2 N$ **rounds**;
if (received a REPORT-REQUEST from $father$)
  respond with REPORT containing x_i, l, and h;

Figure 3.6: HRMA (Leaf Node)

```
variables:
  left, right, father:  left and right children, and the parent in the tree;
  l, h, f:  low and high thresholds for the subtree rooted at this node;
  l_left, l_right, h_left, h_right:  l and h values of the left and right children;
  x:  the total weight of the subtree rooted at this node;
if (received REPORT msgs m_l, m_r from left and right)
  l_left ← m_l.l; h_left ← m_l.h;
  l_right ← m_r.l; h_right ← m_r.h;
  x ← m_l.x + m_r.x; //compute the total weight of this subtree
  l ← l_left + l_right; h ← h_left + h_right;
  if (x ≥ h or x ≤ l)
  send REPORT message with values x, l, and h to father;
if (received REPORT message m_l from left)
  l_left ← m_l.l; h_left ← m_r.h;
  if (m_l.x + right.h ≥ h or m_l.x + right.l ≤ l)
  send REPORT-REQUEST message to right;
if (received REPORT message m_r from right)
  l_right ← m_l.l; h_right ← m_r.h;
  if (m_r.x + left.h ≥ h or m._r x + left.l ≤ l)
  send REPORT-REQUEST message to left;
if (received REPORT-REQUEST message from father)
  h ← h_left + h_right; //request missing values from children
  l ← l_left + l_right; //if needed
  respond with REPORT containing left.x + right.x, h, l
if (received UPDATE message from father, with α_1, α_2)
  forward this message to both children;
```

Figure 3.7: HRMA (Intermediate Node)

3.7 Hierarchical Reactive Monitoring Algorithm

This section presents Hierarchical Reactive Monitoring Algorithm (HRMA) that overcomes the problems of feedback implosion and poor performance under the maximal load conditions that are common to all flat algorithms.

The basic idea is as follows. Instead of using the direct unicast communication channels for reporting local values of ISP variables, and the multicast communication channel for propagating the polling requests, we organize the nodes into a logical hierarchy in which the same node may act at different levels. The inter-nodal links are realized as reliable unicast FIFO channels.

The optimal hierarchy formation and maintenance needs a special study since one can look for several optimization functions. In this work, for the sake of simplicity, we assume that the logical hierarchy is a full binary tree[3].

The leaf nodes (ISPs) report their traps up the hierarchy, and the intermediate nodes act as local managers deciding whether to propagate the trap further toward the root depending on the total of the values reported by their children. This scheme leads to a logarithmic gain in communication toward the root (that acts as the global manager), and, also avoids the feedback implosion problem. In fact, the communication overhead is even lower because not all links in the hierarchy require actual network transmissions.

Figures 3.6, 3.7, and 3.5 present the pseudo-code of the local algorithms run by the leave nodes, intermediate nodes, and the root of the hierarchy respectively. Each node in the hierarchy has a notion of its father, and children in the tree. Each node has values l and h that hold the current low and high thresholds for the subtree rooted in this node.

The root also has an array of threshold values, $thresh$, as determined by the rating service.

Initially, a convergecast is performed to compute the total number of active receivers at time 0. Depending on this value, the local thresholds for each node are calculated, and multicast down the tree. See Figure 3.5. The intermediate nodes do not have l and h initially. They compute these values on the fly when they receive traps from their children.

When a node sends a trap to its father in the tree, we say that it starts verifying the threshold condition. This process ends either at an intermediate node, or at the root of the whole tree. A node stops the process if it discovers that neither high, nor low threshold of the tree being rooted in it is violated. In this case, the node does not send any further messages. The leave nodes in its subtree will discover that no global event has occurred after a timeout.

If the root discovers that a global threshold event has happened, it sends an *update* message down the hierarchy informing the ISPs about the new group size estimation.

Lemma 3. *Verification of any local threshold event is completed after $O(\log^2 N)$ time units, where N being the number of leaves (ISPs).*

Proof. The worst case happens when a global threshold event occurs, but only a single leaf node sends a trap to its father because at all other nodes no local thresholds were violated. In

[3] Currently, many protocols for creating and maintaining efficient internet-wide logical hierarchies exist, *e.g.*, [vRBDV03, ABDL97]. These protocols may be re-used in our context.

this case the father requests a report from its other child, and when this report is received, the father discovers that either lower or upper threshold is violated. This forces the father to send report to its own father *etc.*, until the report reaches the root. The process of checking the total value of a subtree is repeated at most $\log N$ times. At most two messages are send per edge in each subtree. Hence, the whole algorithm takes $2\log^2 N$ time units. This completes the proof. □

Lemma 4. *Message cost of HRMA per time window of duration $O(\log^2 N)$ is $O(N)$.*

Proof. It is sufficient to observe that REPORT-REQUEST message is sent at most once, REPORT is sent at most once, and UPDATE is sent at most once on each edge. Since there are $2(N-1)$ edges in the full binary tree with N leaves, the lemma follows. □

Theorem 3. *Let $\sum_{i=1}^{N} x_i$ be $O(log^2 N)$-constant function. Then HRMA is a correct monitoring algorithm (See Section 3.1).*

Proof. Sufficiency is straightforward since the all ISP nodes get notified about the TVEs only by the root, and the root only informs about the TVEs that indeed occur. To prove the necessary condition, observe that if there is a global TVE, then there is at least one local threshold violation event. Then necessity follows by Lemma 3. □

3.8 HRMA Analysis

In this section we analyze the expected cost gain factor of *HRMA* per round. To simplify the problem we study a restricted case where each monitored variable may assume a binary value. In other words, this is the same monitoring problem when the group size estimation is done at the granularity of ISPs. For the general case, performance of *HRMA* is studied through simulations. The simulation results are summarized in the next section.

3.8.1 Yardsticks

First, we need to establish a baseline algorithm against which the cost gain ratio of *HRMA* will be computed. It would be unfair to compare *HRMA*'s performance to that of OB_c since they use different information dissemination structures. Instead, we define a *Hierarchical Straightforward Monitoring Algorithm*, denoted OB_h that uses the hierarchical structure for communication as explained in Section 3.8.

The leave nodes initiate a convergecast of their local values toward the root of the logical hierarchy every $2\log^2 N$ time units. The root checks whether a global TVE occurs, and if so, it multicasts the new group size estimation using the same hierarchy.

Figure 3.8: Hierarchical Straightforward Monitoring Algorithm (OB_h).

Obviously, $C(OB_h, \delta) = 2(N-1) \cdot (\frac{\delta}{2\log^2 N} + e_\delta)$, where e_δ being the number of TVEs detected by the root. It is easy to observe that $C(HRMA, \delta)$ may be higher than $C(OB_h, \delta)$. In the worst case the performance of *HRMA* is about twice as bad as that of OB_h. However, the worst case is not necessarily the one that occurs most often. As our simulations show, *HRMA* is outperforming OB_h, and also the flat reactive algorithms even when the load is very high, *i.e.*, when the membership is very dynamic in the leaves, while being relatively stable in total.

To see that OB_h indeed provides a reasonable baseline for assessing communication cost gain of *HRMA*, we should consider the existing alternatives. One such alternative is offered by probabilistic polling techniques [BTW94, LN00, NB98, FT99, AAN02, NSB03] A generic probabilistic estimation procedure communicates directly with the receivers, and not with their ISPs. It proceeds in rounds. In each round a probability of response is multicast by the dedicated node, usually the source. The receivers respond with the advertised probability. Based on the number of responses, an estimation of the total number of receivers is performed. Although there are many more details involved, this *Naive Probabilistic (NaP)* algorithm captures the essence of the methodology. *NaP* avoids the problem of feedback implosion. However, it is easy to see that the communication cost of *NaP* approximately equals that of OB_h. Therefore, we do not use probabilistic methods for comparison. The comparison of *HRMA* versus OB_h applies also to *NaP*.

Another alternative is ECMP of EXPRESS. Nonetheless, it is again easy to observe that the communication cost incurred by ECMP is *at least* that of OB_h when the maximal silence period is set to be $O(2\log^2 N)$. Therefore comparing *HRMA* performance to that of OB_h is sufficient to draw the conclusions about ECMP versus *HRMA* performance.

3.8.2 Analysis

For the sake of simplicity we assume that each leaf node assumes value 1 with an independent probability p, and value 0 with probability $q = 1 - p$.

To further simplify the analysis we assume that all leaf nodes have equal local thresholds, and that the target function is non-decreasing (*i.e.*, the membership only builds up).

Let n_i denote number of leaves in a full binary subtree of height i. Let n_i^1 denote number of leaves in a binary subtree of height i for which the values of local variables equal 1. Let $0 < \gamma < 1$ be a given value. Then each node in the hierarchy is allocated a local threshold value $\gamma_i \stackrel{\text{def}}{=} \gamma \cdot n_i$. A threshold value γ_i means that the root of the full binary subtree of height i should either issue a REPORT message to its father, or declare a global TVE if it is the root of the whole tree, if and only if $n_i^1 \geq \gamma_i$.

Let $p_i(\gamma_i)$ be the probability that a node at level i in the hierarchy sends a REPORT message to its father at level $i+1$. Under our assumptions

$$p_i(\gamma_i) = 1 - \sum_{k=0}^{\gamma_i \cdot n_i} \binom{n_i}{k} p^k (1-p)^{n_i - k}. \tag{3.2}$$

For the sake of brevity we will omit γ_i from our notation, and use simply p_i to denote $p_i(\gamma_i)$, and $q_i = 1 - p_i(\gamma_i)$. From observing the *HRMA* operation it is easy to compute an expected cost for an edge at level i (*i.e.*, the expected number of messages per step of the algorithm being sent on this edge). Obviously, this cost, denoted $C(e_i)$, lies between 0 and 2 since at most two messages are sent by *HRMA* over any edge per step. There are two possibilities. Either a REPORT message is sent over e_i by the node being at level $i-1$ (in this case $C(e_i) = 1$), or a REPORT message has been solicited by a node at level $i+1$ (in this case $C(e_i) = 2$). Consequently, $E[C(e_i)]$ is computed as follows:

$$E[C(e_i)] = p_i + 2[q_i \cdot [p_i + q_i \cdot [p_{i+1} + q_i \cdot [p_{i+2} + q_i \cdot [p_{i+3} + \ldots \cdot [p_{\log N - 1}]\ldots].$$

Or in a more concise form:

$$E[C(e_i)] = p_i + 2 \sum_{j=i}^{\log N - 1} q_i^{j-i+1} p_j. \tag{3.3}$$

Since there are $N/2^i$ edges at level $i \in [0, \log N - 1]$, the total expected cost of *HRMA* per step is

$$E[C(HRMA)] = \sum_{i=0}^{\log N - 1} \frac{N}{2^i} E[C(e_i)] + 2(N-1) \cdot p_{\log N}. \tag{3.4}$$

The total expected cost of OB_h per step is

$$E[C(OB_h)] = 2(N-1) \cdot (1 + p_{\log N}). \tag{3.5}$$

The second term in Equation 3.4, and Equation 3.5 accounts for the multicast of a global TVE notification. Since this cost is the same for both algorithms, it can be omitted from the calculation. Consequently, the expected per step cost gain factor of *HRMA* over OB_h (and therefore also over ECMP, and Probabilistic Naive) is given by the following equation.

$$E[C(OB_h)/C(HRMA)] = \frac{\sum_{i=0}^{\log N-1} \frac{N}{2^i} E[C(e_i)]}{2(N-1)}. \tag{3.6}$$

If we let $p = 0.5$ and $\gamma = 0.5$, then $p_i = 0.5$ for all $i \in [1, \log N]$ regardless of N, and the computation becomes easy. According to Formula 3.3, $E[C(e_i)] = 0.5 + 2\sum_{j=1}^{\log N-i-1} 0.5^j \cdot 0.5 = 0.5 + \sum_{j=1}^{\log N-i-1} 0.5^j = 0.5 + \sum_{j=0}^{\infty} 0.5^j - 1 - \sum_{j=\log n-i}^{\infty} 0.5^j$. By neglecting the last term we obtain that $E[C(e_i)] \approx 1.5$. Thus, $E[C(OB_h)/C(HRMA)] \approx \frac{2(N-1)}{3(N-1)} = 0.6667$.

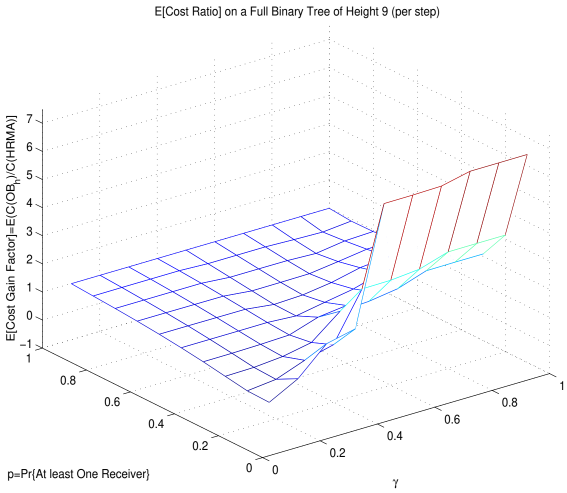
In the general case, it is difficult to compute 3.6 analytically. Therefore we present the numerical results in Figure 3.9(a). This figure shows the behavior of the cost gain factor as a function of γ and p for $N = 512$. When p approaches 1, the cost gain approaches 1 irrespective of the threshold allocation. When p approaches 0, the cost gain factor goes to ∞, as *HRMA* pays 0 in communication ($p = 0$ is not shown in order not to distort the visibility of the graph).

Figure 3.9(b) shows the isoline plot for $E[C((OB_h)/C(HRMA)]$ when $N = 512$. One can see that as threshold grows and the probability of having at least one receiver in an ISP domain decreases, the cost gain factor of *HRMA* increases. However, as probability of having at least one receiver increases, *HRMA* becomes more costly than OB_h. Given an apriory knowledge of the probabilities (e.g., through long term measurements that identify typical user behavior patterns, or through on-line probability estimation procedure), an administrator can decide which monitoring algorithm cheaper to deploy to meet the given precision requirements.

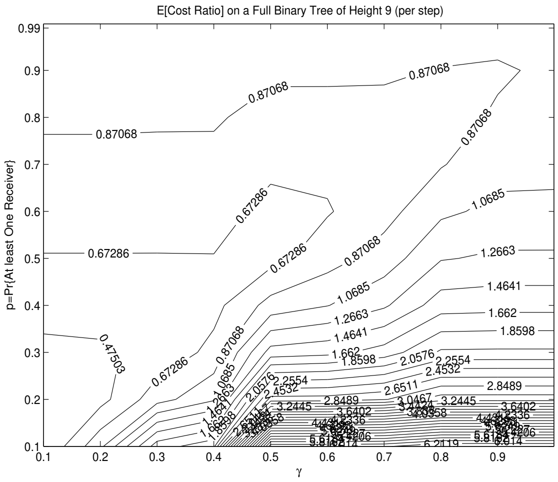
3.9 Performance of HRMA (Simulation Study)

In order to evaluate performance of *HRMA* in the general case we use the same synthetic dataset as in Section 3.4.

Figure 3.10(a) shows performance of *HRMA* versus OB_h under the maximal load conditions (*HRMA* curve), and for 90% inactivity time trace (*HRMA* 90 curve). As one can see, the cost gain factor behaves similarly to the flat model. Under the maximal load conditions (high probability of violating a local threshold), *HRMA* produces a small message cost gain ranging from 14% to 20%. To understand why this gain is relatively small, please, recall from Section 3.6

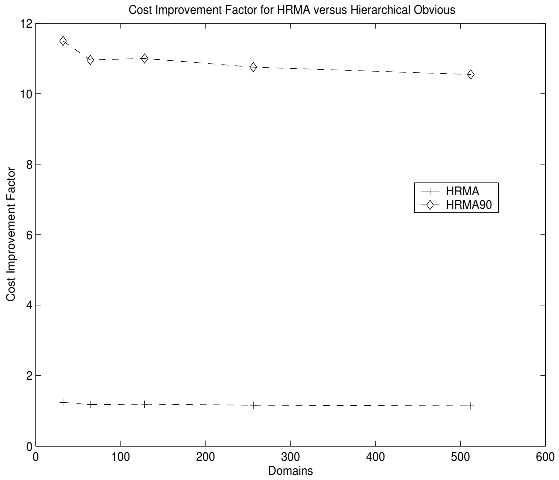


(a) Expected Cost Gain Factor

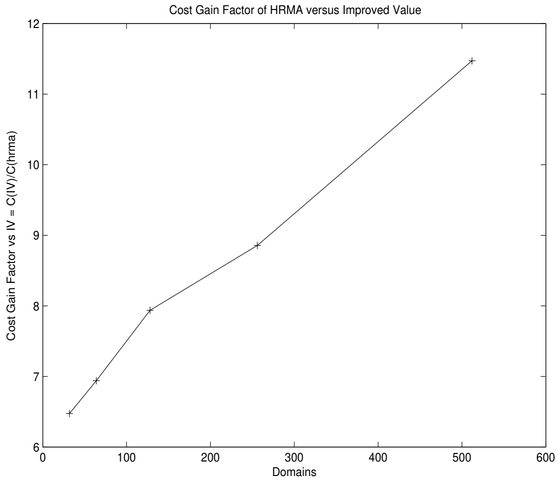


(b) Isoline Plot

Figure 3.9: Cost Gain Factor Analysis



(a) *HRMA*vs OB_h



(b) *HRMA*vs Improved Value

Figure 3.10: Relaive Performance of *HRMA*

that per step, almost $0.5 \cdot N$ leaf nodes trap to their fathers in the hierarchy. Since users are distributed uniformly, we can expect $0.25 \cdot N$ nodes in the second level to send the *report-request* messages to one of their children. This makes the cost of a single step $O(N)$. The relative cost gain of *HRMA* versus Improved Value (the cheapest flat monitoring algorithm) for maximal load trace is shown in Figure 3.10.

As one can see the relative cost gain of *HRMA* grows with the number of domains. *HRMA* does not save communication on multicasting of the notifications about the global TVEs. It saves communication by suppressing the non-significant local threshold violation events though. This is *HRMA*'s main feature. In our traces, *HRMA* pays $\approx 1.5 \cdot N$ messages per round. This is due to the fact that some edges are traversed twice: one message to solicit the value, and another one to report the value itself. In the same circumstances, IV algorithm pays $\approx 2 \cdot N \log$ messages per round. Thus, the expected improvement is around $1.33 \cdot \log N$ and is indeed expected to grow with the number of domains, N. This is very important observation since it suggests that *HRMA* is scalable.

3.10 Adaptive HRMA

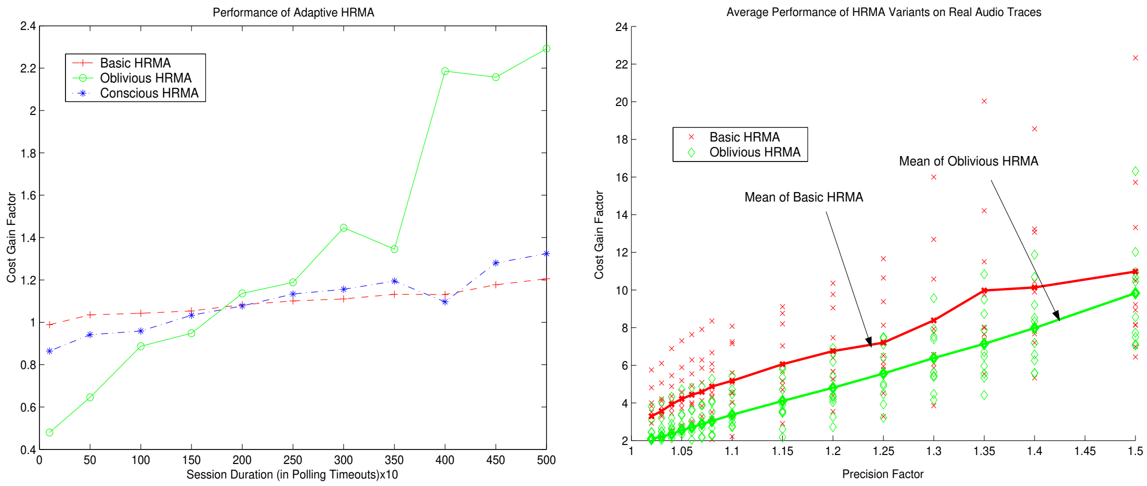
One may observe, that in its current form *HRMA* is not adaptive. Namely, if there is no global TVE, but there exist local threshold events, then at some level in the hierarchy, some intermediate node stops the alarm verification process. However, the algorithm learns no lesson from this. Therefore, if the situation remains similar also for the next step, the unnecessary verification process will repeat itself since the local thresholds of the leaves did not change. This is pure overhead. If we treat false alarms similarly to actual TVEs events, this overhead seemingly can be reduced.

In particular, there is an apparent trade-off between leaving the local thresholds in the subtree intact in case of a false alarm in this subtree, and changing them. The latter requires multicasting in the scope of this subtree, and therefore increases the communication cost of the algorithm at this round. However, if it succeeds in tuning the thresholds in such a way that no false alarms would follow, the total communication cost of the algorithm will be reduced over time. To this end we define two simple modifications to the basic *HRMA*.

- **Oblivious Adaptive HRMA:** When a false alarm is detected at time t by the root of some subtree (an intermediate node i), this node computes $\alpha_1 = f_i(t)/l$, and $\alpha_2 = h/f_i(t)$ where $f_i(t)$ being the sum of the leaf variables of the subtree rooted at i, and l, h being the lower and higher local thresholds of this node respectively. Then, i multicasts the UPDATE message containing α_1, α_2, and indicating that there was no global threshold event. The nodes in the subtree recalculate their local thresholds using these values as they would upon reception of this message from the global root. However they do not change their current estimation of the group size.
- **Conscious Adaptive HRMA:** Similar to Oblivious. The main difference is that the root of the subtree assess the chances of a successful thresholds tuning by comparing the

distance of the current value of the sum for its subtree with the median of the interval defined by its l, and h threshold values. The chances of suppressing false alarms are highest when the current value of the subtree is close to the median, and decrease as it gets close to either l, or h. Accordingly, Conscious Adaptive *HRMA* sends UPDATE message with the higher probability in the former case, and lowers the probability of sending an UPDATE message in the latter case.

Figure 3.11 demonstrates the adaptivity trade-off. For a fixed number of domains, we vary the trace duration. As one would expect, for shorter sessions the adaptivity heuristics are inferior to the basic form of *HRMA*. However, as sessions become longer, they become advantageous over the basic *HRMA*. Oblivious Adaptive *HRMA* is always inferior to the the Conscious and and Basic variants when sessions are short. This is very natural since Oblivious Adaptive *HRMA* needs more time to compensate for the extra cost it spent on updating the thresholds.



(a) Performance of Adaptive Heuristics (b) Performance on the Real Traces

Figure 3.11: Adaptive Heuristics and Real Traces Validation

3.11 Validation with Real Traces

In this section we evaluate *HRMA* vs Ob_h using the real membership traces collected using *mlisten* tool [AA97] in 1996. The purpose of this study is to inspect the behavior of *HRMA* when statistical assumptions on the user behavior used so far are not necessarily preserved.

MListen generates a separate record for every host receiver. Each record indicates the receiver's IP address, the class D address of the group, port, join time, and stay time in seconds. For each class D address, we treated the network portion of the IP address as an individual multicast ISP. Then for every such ISP we have calculated the number of host receivers at any given moment by counting the different host parts of the IP addresses of receivers. In total there were about $44,000$ "ISP" domains identified this way, and about 200 different multicast addresses. As indicated by [SA00] the measurements performed in 1999 suggest that MBone did not grow substantially larger since then. In fact, it even shrank down.

Most multicast groups that were observed by us were quite small ranging from a few users to a few tens of users at any given moment, and have not exhibited particularly high activity. See [SA00] for discussion on the reasons for this.

However, this makes this data even more interesting. So far our algorithms have been evaluated on the synthetic data that simulated very large groups (thousands of highly dynamic users) spanning hundreds of domains. This may be way too far from the current state of the Internet where the multicast groups are much smaller and less active. In a sense, this is the opposite of our maximal load scenarios being inspected so far. The question that we asked was whether the hierarchical reactive monitoring will be useful also for estimating the size of very small groups in such a way that it would be useful for pricing applications.

To this end we revisit the technique for global thresholds calculations. The percentages of the total user population are meaningless for small groups. Far more important is to detect changes in the group from its current size with a predefined precision γ. Instead of having a full scale of global thresholds we define only a single pair of thresholds $H \stackrel{\text{def}}{=} \gamma \cdot f(0)$, $L \stackrel{\text{def}}{=} f(0)/\gamma$, where $f(0)$ is the total size of the group as measured at time 0. The local thresholds l, h are computed as before. If at some time instance t, either of the global thresholds is violated, *HRMA* re-calculates L, H using $f(t)$. This way we always know the size of the group with precision γ as being set by the pricing application.

Out of ≈ 200 different traces that we created using the methodology above, we found only 11 traces with considerable activity. All these traces were comprised out of 256 ISPs. We analyzed only the first $300,000$ seconds ($\approx$ 80 hours) of these traces. Figure 3.11 illustrates performance of *HRMA* variants for various values of γ. One thing to notice is that on the average Conscious Adaptive *HRMA* obtained exactly the same results as the Basic *HRMA*. This means that the distribution of values in the tree was always such that the values of the subtrees were far from the median of the intervals defined by their local thresholds. We do not have an explanation for this. Also, it would be difficult to come to a decisive conclusion without inspecting more traces. However, the traces we used represent a certain type of real life traffic, and *HRMA* performance on these traces is very satisfactory. The cost gain factor of *HRMA* versus other known techniques represented by OB_h ranges from 5 to 10 on the average. As one would expect, the variance is high because the sample is too small. The variance is lower when threshold values are lower which generates more threshold events irrespective of the nature of the trace. It gets higher when thresholds grow. The variance diminishes as threshold grow even more since large thresholds prevent any local threshold violation events from occurring, and the cost gain factor eventually goes to infinity.

3.12 Related Work

Holbrook and Cheriton proposed a new IP multicast routing protocol called EXPRESS [HC99], that incorporates the membership size accounting function at the granularity of individual hosts as part of its operation. The native support for this accounting constitutes the primary motivation for EXPRESS. The price of transmission is assumed to be dependent of the size of the group as accounted by the routing and management protocol called ECMP.

The event-driven version of ECMP is close in spirit to our hierarchical reactive monitoring algorithm of Section 3.7. There are important differences, though. In ECMP a dedicated multicast tree is built for each active source. If the source wants to find out how many active receivers exist in its reach, it multicast counting request with parameters τ and α. Each multicast router in the tree computes the *error tolerance curve* $e_{max} = \frac{-log(dt/tau)}{\alpha}$, where dt is the time interval since the last report to the father router in the tree, τ is the maximal time that the router is allowed not to report, and α controls the rate of decay of e_{max}. In addition, at each node the current error is computed as $e = max(\frac{cur}{adv}, \frac{adv}{cur})$, where cur being the current number of receivers in this router's subtree known to this router, and adv being the last value reported to the father in the tree.

As long as $e < e_{max}$ holds, the router does not report anything. When the inequality no longer holds, the router should report its current notion of the number of receivers to its father. These values are being summed up along the tree, and eventually the sum reaches the source. Thus the source obtains the estimation of the group size with the precision being defined by e_{max} curve.

One important observation about this algorithm is that it generates accounting traffic even when there are no actual changes in the membership size. This happens because e_{max} decays with passage of time, and eventually reaches 0 after maximum τ time units. In [HC99] an initial simulation study of ECMP have been presented. However, no compelling comparative study, or analysis of the performance have been shown.

Since our primary target application is pricing, a short discussion on multicast pricing proposals is called for. Chuang and Sirbu [CS01] were the first to express the cost-effectiveness of multicast as a simple power-law function of the active membership size. Namely:

$$N^{k-1} = \frac{L_m}{L_u N}. \tag{3.7}$$

L_m is the total number of links in a multicast tree, L_u is the average length of the unicast route, and N is the number of leaf routers. The exponent k is called *Economy of Scale* (EoS) factor, $0 \leq k \leq 1$. In [CS01] it was demonstrated through simulations that $k \approx 0.8$ for realistic networks. This result was later re-validated mathematically by Phillips *et al.* [PST99].

Equation 3.7 suggests a straightforward multicast pricing scheme. It is assumed in [CS01] that the multicast routers can be discovered using distributed monitoring. However no specific accounting mechanism has been suggested.

In [DMS03], Dolev *et. al.* presented a fast and cost-effective algorithm for estimating the number of leaf multicast routers. This algorithm leverages on the empirical properties of the realistic multicast topologies that exhibit a linear ratio between the number of high degree nodes, and the number of leaves. The algorithm as is does not help finding the number of individual hosts. However, it can be used to provide a better initial estimation for the probabilistic methods, so that they converge faster.

Additional important discussion on accounting and pricing of the multicast traffic can be found in [HB00, HSE95], and [Bri99].

3.13 Chapter Summary

We presented simple and generic algorithmic framework for multicast group size monitoring. This framework, being realized as generic on-line management service independent from the routing, is useful for usage-based pricing of multicast traffic. We demonstrated through analysis and simulations that hierarchical reactive monitoring is a powerful and scalable technique for on-line calculation of the internet-wide additive metrics (in our case, multicast group size). The communication costs it incurs are low, and it reduces management bandwidth requirements by factor of 2 to 10 on the average compared to other existing techniques without degrading the precision of estimation. These results have been validated with the real datasets that were available to us.

Chapter 4

High Availability of Monitoring

4.1 Motivation

As argued in Section 1.2.1 (Chapter 1), the traditional two-tier structure of network management applications based on the rigid client/server paradigm with thick managers and thin target agents suffers from severe scalability and availability limitations. One alternative is adopting the popular three-tier architecture. A typical three-tier monitoring application as depicted in Figure 1.1 (b) (see Chapter 1).

While considerably more scalable, the three-tier client/server applications are much more difficult to control. Providing high availability of the mid-tier components in presence of General Omission failure model [PT86] in which hosts may fail by crash (subsequently recovering), and the network may split and merge, is especially challenging. A standard way of creating a three-tier client/server application is using *application server* that provides the mid-tier run-time environment. Usually, however, application servers are not oriented on the continuous operation of the mid-tier components, but rather on a short-live activities performed by them upon client requests. To the best of our knowledge, no existing application server provides a highly available run-time environment that copes with the kind of failures found in the general omission model, and can transparently (to the external client) restart failed *state-full* monitoring components from a consistent point while providing disconnected continuous monitoring of the target network elements.

Given the complexity of handling distributed three-tier applications in an unpredictable environment prone to various network failures, it is both important and challenging to provide a maximally transparent infrastructure that allows a manager to deploy the needed monitoring components of the second tier in a highly available manner. This improves the overall failure behavior of the management applications, allows more efficient applications (such as event-driven monitoring applications), and therefore contributes to better provisioning of network services in general.

In this work, we propose a novel middleware architecture, HAMSA, that facilitates reliable and efficient deployment of three-tier monitoring applications. Our architecture pertains to both types of monitoring, *i.e.*, statistical, and on-line, as was discussed in Chapter 1. We describe the main building blocks of this architecture, and demonstrate it power for efficient and reliable monitoring by describing and analyzing the performance of monitoring applications implemented using HAMSA.

4.2 Architecture Overview

HAMSA is an architecture that defines the interfaces, and functionality of a highly available run-time environment for HAMSA-Compatible Components. We use the term *components* (as done in many other middleware software systems) to describe objects that implement a set of the predefined interfaces allowing dynamically to mix and match this object with other objects that conform to the same set of interfaces. The following guarantees on the execution of these components in the second tier generate the primary added value of HAMSA.

- Failures of the components are masked from the outside entities. As long as HAMSA has sufficient resources for executing the components, components function *continuously* despite component host failures (*i.e.*, the machine running the component), arbitrary asynchronous network splits, merges, and host recoveries.
- In case of network splits, a single instance of each component is executed per network partition.
- The component whose host machine has failed, is guaranteed to resume operation from the last consistent state, *i.e.*, the last state of this component known to the outside world.
- Component's interactions with the environment may potentially influence the state of other components, and/or external entities. In this case, interactions (messages, method invocations) are termed *non-idempotent*. Failure, and a subsequent recovery, of a component being in the middle of a non-idempotent interaction may violate the original interaction semantics. An advantage of HAMSA over other middleware architectures is

that it preserves the original *at-most-once*, or *at-least-once* semantic of the component interactions in spite of asynchronous network failures.

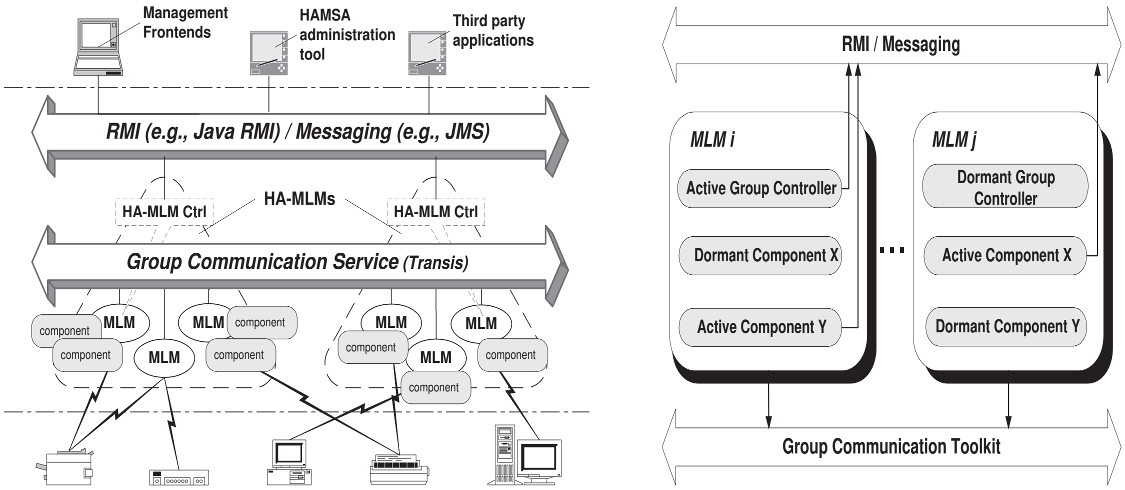
These advantages come at a certain price in bandwidth, processing, and degree of replication. In Section 4.4 we discuss the trade-offs between the extended functionality of HAMSA and this overhead. Also HAMSA restricts the inter-process communication model to asynchronous communication and messaging. This communication model, though, fits well into the Network Management domain.

It was shown in [BMST92] that the general omission failure model implies the lower bound of $n > 2 \cdot f$ on the degree of replication in the primary-backup protocols, where n is the number of replicas, and f is the number of failures. However, this restriction pertains only to communication between the clients, and the servers in a two-tier model. In our three-tier architecture, communication between the management clients (the uppermost tier entities), and the servers indeed obeys the restrictions imposed by the low bound on the primary-backup protocols [BMST92]. We allow the periods of unavailability of the mid-level entities performing continuous monitoring tasks to the management clients, when such communication may compromise a *single object* behavior of the monitoring task. However, for communication of the monitoring tasks with the target management agents (the bottom tier), we relax the demand of [BMST92], on having a single primary system-wide at any given time instance, since the *read* monitoring operations are idempotent.

Thus, we allow multiple primaries to operate simultaneously in non-communicating network partitions transparently to the high-level clients who still regard them as a single entity. Upon a re-merge of the previously split network, our middleware *automatically* obtains the most updated monitoring state about the disconnected network elements that have been available in some network partition, and were unreachable in the others. To the best of our knowledge, HAMSA is the only monitoring middleware with such high-availability guarantees, specifically designed to address the meta-management challenges of the three-tier architecture.

4.2.1 Highly Available Mid-Level Managers (HA-MLMs)

The run-time environment with the above properties is provided by a set of virtual servers termed *Highly Available Mid-Level Managers (HA-MLMs)*. HA-MLMs are logical entities that are comprised of one or more physical servers called MLMs (see Figure 4.1(a)). To its users, every HA-MLM appears as a single object following the (k, Δ)-bofo server semantics defined



(a) Detailed Logical Structure of HAMSA (b) Detailed Logical Structure of HA-MLM

Figure 4.1: HAMSA Architecture

in [BMST92][1]. The bofo semantics (bounded outages, finitely often), essentially, states that all server outages[2] can be grouped into at most k intervals of duration at most Δ each. This implies that even though some requests made to a (k, Δ)-bofo can be lost, the number of such losses is bounded.

HA-MLM's interface is exported by the special stateless component called *HA-MLM Controller*. Each MLM in a given HA-MLM is capable of running the HA-MLM Controller, but at any given moment only a single MLM is executing it being primary for the controller. At other MLMs (backups) this component is being *dormant* (see Figure 4.1(b)).

HAMSA-compatible component is dynamically delegated to a specific HA-MLM through its controller interface using the HAMSA administration tool. The identity of the HA-MLM to which this component is being delegated, is part of the run-time identity of this component.

The executable code of the component is being reliably propagated to all MLMs in the HA-MLM through the group communication service (*e.g.*, Transis [ADKM92]), see Section 4.2.4. However, similarly to the HA-MLM controller component, this component will be executed only at one MLM at any given moment. Other MLMs within the same HA-MLM keep dormant

[1] See Appendix C for discussion of the (k, Δ)-bofo semantics and primary-backup replication protocols implementing in general omission model.

[2] *Service outage* is a situation when a non-faulty client sends a request to the server at time t, but does not receive the expected response by $t + \delta$, where δ is an upper bound on the time interval required for the round trip time, and processing time at the server.

replicas serving as warm backups for the components whose host MLM may fail, see Figure 4.1. This is different from the more common practice of keeping components on a component server, and downloading them on demand. HAMSA performs component propagation as above to increase their availability.

The administrator has a limited direct control over the physical location of the components within HA-MLM. This is motivated by the fact that components may need to be relocated automatically in case of failures. However, it is possible to influence the HA-MLM placement decisions by supplying some suggestive *policies* that are followed as long as no failures occur.

HA-MLMs may be organized into logical hierarchies. This feature is important in order to address the challenges of providing multi-tiered monitoring solutions efficiently. In particular, this feature may be needed for the hierarchical filtering approach to on-line monitoring, and multi-tier mediation platforms for accounting management that were discussed in Chapter 1. Naturally, HA-MLM hierarchies may be used to facilitate reliable hierarchy of ISPs that we assumed in Chapter 3 of this thesis. Additional details on the HA-MLM administration can be found in Appendix B.

4.2.2 HAMSA-compatible Components

To render warm backups of the executing components, MLMs transparently replicate the state of the components delegated to their HA-MLM. To achieve high availability, the state is co-located with the component. This is another difference between HAMSA, and more traditional approaches in which a dedicated database is used to store the state of the components.

The state of a component consists of *Interaction State*, and *Component-Specific State*. Interaction State consists of all inbound and outbound unprocessed interactions between this component and external entities. Component-Specific State consists of arbitrary application-specific objects (*e.g.*, files) that implement a predefined interface. The state objects are managed by the components themselves. The predefined interface allows components to demand state replication without knowing any details of the replication mechanism. Similarly, MLMs handle a component's state with no knowledge of its semantics. Components may interact with other components executing within the same HA-MLM, in different HA-MLM, and outside HAMSA, *e.g.*, with the front-ends residing in the first tier, and the network elements.

To provide its guarantees, HAMSA requires that all non-idempotent interactions between the HAMSA-compatible components, and any external entities are made through HAMSA Message Service. This allows to replicate the interaction part of a component state transparently.

However, using HAMSA Message Service is feasible only for interactions between the entities of the second and first tiers. It is not feasible to restrict in this way the interactions with the managed devices. Therefore, HAMSA is primarily targeted to monitoring, and not to other kinds of management activities that may change the state of the target devices.

4.2.3 HAMSA Messaging Service

Allowing direct interactions of the components with their environment is not always safe if we wish to comply to the (k, Δ)-bofo semantics. HAMSA defines that each component is assigned an *interaction approver* that policies its interactions. In particular, it may defer interactions with the outside entities depending on the specific state of the network, *e.g.*, when the network splits in such a way that there is no majority of replicas in a given partition.

However, the interactions between the HAMSA-compatible components and the network elements are not restricted in any way. This is the key feature distinguishing our solution from the existing approaches [BMST92], and making our proposal especially well suited for monitoring. Interactions with the target devices are supposed to be implicitly reflected in the component-specific state objects, which get replicated on demand from a component that owns it, or transparently to the component, upon an outgoing interaction demanded by the component. When messages are sent to a component by external entities, this does not imply immediate update of the replicated state among all the MLMs of the HA-MLM where this component executes. However, upon an external interaction attempted by the component itself (*e.g.*, by sending a message, or invoking RMI on the external entity), the replicated state of the component is synchronized among all the MLMs, and only after that, the outgoing message is allowed to continue. This leads to the following key property of HAMSA-components: *a component that potentially made its internal state publicly available through an interaction with external entity, would never return to a more outdated state in spite of the errors found in the general omission failure model.*

Each component is given a unique symbolic name consisting of its host HA-MLM name and a unique prefix within this HA-MLM. Each time a component is (re-)activated at an MLM, it updates the external Naming and Directory Service to renew the binding between the component's name and its communication handles. The HA-MLM assigns two types of communication handles for each component: *Mailbox*, and *Proxy*.

Mailbox is needed for directly sending a message to a component. Therehamlm:admin is one mailbox per HA-MLM that is shared by all components within this HA-MLM, and their clients. In order to support remote method invocations while using the HAMSA consistency and ordering mechanisms transparently, we use the standard proxy approach. Any remote invocation between a HAMSA component and any other party is intercepted, and processed by the per-component proxy. The proxy creates a message from the method call performed on it, and relays it to the HAMSA Messaging Service.

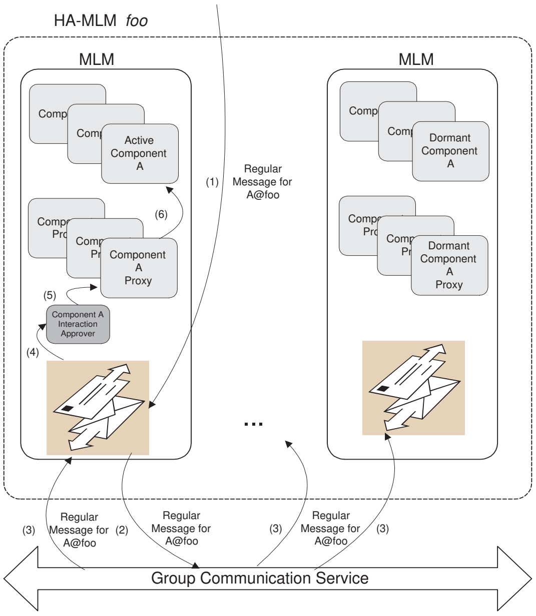


Figure 4.2: Interacting through HAMSA Messaging Service

Figure 4.2 depicts how a message is communicated to a component. The message is placed into the mailbox of this component by the MLM hosting it, (1). This message is not delivered to the target component immediately. Instead, (2), it is being propagated to all MLMs in the HA-MLM using the group communication service (see the next subsection). When the message is received at the group communication service level at all operational MLMs including the sender, (3), this message is assessed for delivery to the target by consulting the component's interaction approver, (4). If the interaction approver permits the interaction, the message is delivered to the active component proxy, (5). Finally, (6), the proxy delivers the message to the target component.

One restriction of this approach is that HAMSA-compatible component cannot support synchronous method invocations with non-void return values. HAMSA communication model requires that if a caller wants to receive information from a component it has to supply either a callback interface or to be registered as a recipient at the mailbox serving this component.

We use the Java programming language to implement HAMSA. We support Java RMI as an instance of the RMI technology. In our prototype, we implemented HAMSA Messaging Service as part of Java Messaging Service (JMS).

4.2.4 Group Communication Service (GCS)

The replication of the components state within a HA-MLM is facilitated by a group communication toolkit that is not visible outside HA-MLM, as shown in Figure 4.1(a)). Such toolkit systems usually allow processes to form *groups* that can be addressed by a single logical name, so that messages can be sent to the group using this name as an address, and all operational members of the group receive them. HA-MLMs are realized in HAMSA as process groups.

- Reliable multicast FIFO delivery of messages.
- Per-group notification of membership changes either due to network failures, or members (*i.e.*, MLMs in the context of HAMSA) voluntarily joining/leaving the group.
- *Virtual Synchrony* model of message delivery, which, simply stated, means that members of the group that go together through the same set of membership changes receive the same set of messages.
- *Partitionable Membership Model* which means that although members of the same group can find themselves in different network partitions (due to asynchronous network splits), each connected component can continue its operation, and when a network merge occurs, the members can resume operation from a consistent point in the message stream so that the Virtual Synchrony model is preserved.

There exist several group communication toolkits that supply this functionality [AfCM96, CKV01].

4.3 Monitoring Applications

In this section we present two typical NM applications, demonstrate how one can deploy them using HAMSA, and explain the benefits NM applications gain from taking the HAMSA approach.

The first NM application is a highly available *post mortem* analysis system. In this application, several MIB scalar variables from each network element are being kept in a centralized repository, and when a network failure occurs, the management system searches this repository for the relevant variables whose values may suggest the source for the failure (see for example [JdS01]).

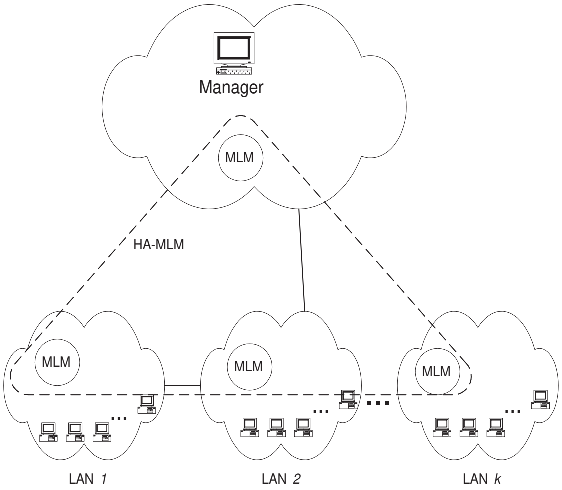
In a typical two-tier scenario such a system is deployed at a single station, and the MIB variables of all network elements are accessed from it. The collected data is kept in the local file system. When a failure of a monitored element (or of several elements from the same network region) is detected the collected data is searched and the behavior of the relevant MIB variables is examined in order to identify the cause of the problem.

In HAMSA, the centralized polling application and its repository are being handled transparently by the middleware. The administrator chooses a set of MLMs by either selecting an existing HA-MLM, or defining a new one, and delegates the polling component to this HA-MLM. Based on the component placement policy, the controller activates this polling component at one of the MLMs, while the replicas are kept for warm backup at other MLMs.

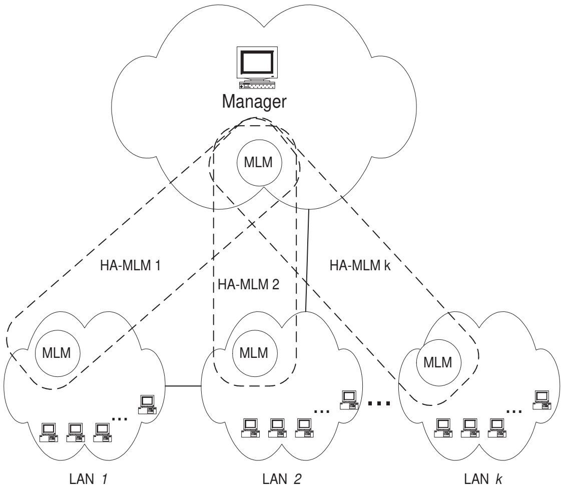
If the network splits, the monitoring continues automatically in each network partition where at least one MLM of the split HA-MLM is present. The state (*e.g.*, the collected MIB variables), is kept locally per replica of the monitoring component in each network partition. When the network re-merges these autonomously collected states become available to the administrator.

One question raised by this example concerns different configuration trade-offs available for the monitoring application that uses HAMSA. Consider the typical network configuration illustrated in Figure 4.3(a). In this scenario, the information arrives at the monitoring station from k LANs. If the monitoring is done by centralized polling from the management station, and the connectivity to one of the LANs is lost, the monitoring of its elements cannot continue. In particular, if the failure is caused by a wrongly configured access interface in the LAN's access router, the information about the root cause of the problem will not be available. This is because the connectivity may be lost before the values of the router's MIB variables suggesting the cause of the problem are retrieved.

If however the administrator configures HA-MLM in such a way that there is at least one MLM per LAN, the MLM in the disconnected LAN will re-start a separate copy of the monitoring as soon as it detects (through the underlying group communication service) that there is a network partition, and all variables in the router's MIBs will be polled. Once connectivity is re-established (say, through rolling back the configuration) the management station will be able to access this information, and the manager will be able to identify the source of the problem,



(a) All MLMs are put into a single HA-MLM



(b) Pair-wise Organization

Figure 4.3: HA-MLM Configuration Options

(*i.e.*, a wrong configuration) and to fix it.

This example also demonstrates the importance of proper HA-MLM configuration. The administrator may be tempted to have at least one MLM in each LAN, as in our example. However, since the state of each monitoring component is distributed by HAMSA to all members of the HA-MLM, the communication costs induced by the replication may become too high.

In fact, one may choose to create k separate applications, each having a different HA-MLM containing only a pair of MLMs, as described in Figure 4.3(b). In this case, the monitoring application for each LAN is running separately on the local MLM (according to the distance-based component placement policy), and thus being unaffected by a possible network partition. If, however, the local MLM itself fails, a copy of the monitoring process for that LAN will be initiated automatically by HAMSA on the MLM that is co-located with the management station. This configuration also reduces the overall monitoring traffic when there are no failures, since in this case the monitoring is done locally and the state of the monitoring components is synchronized among the two MLMs only upon the external interactions.

There exists a trade-off between the monitoring overhead traffic, and the overhead traffic induced by HAMSA due to replication it performs behind the scene. The actual amount of overhead depends on the total number of MLMs in a HA-MLM, the size of the application state in HAMSA, the frequency of external interactions, and the amount of data involved in these interactions.

For example, in the described post-mortem failure analysis application, one can choose to have a small state (*i.e.*, the serial number of the last poll), or a very large state (*i.e.*, the actual data of the last 10 minutes polling). Clearly, the latter choice allows a faster recovery after a failure of a monitoring component, but it generates much more overhead traffic. We study these trade-offs in Section 4.4, and show that the overhead required by HAMSA to provide the extra functionality is much smaller than the monitoring costs we saved.

A more complex monitoring application demonstrating the inter-process communication capabilities of HAMSA is an event-driven reactive monitoring NM application. In such an application we are required to detect when a function (typically the sum) of a number of MIB variables, each belonging to a different network element, exceeds a predefined threshold (see [DR02]).

A centralized realization of this application involves a polling station that monitors all variables at all network elements, computes the function and sets up an alarm if the value has exceeded the threshold. This solution induces both significant traffic overhead and computation load at the monitoring station that grow linearly with the number of polled elements.

To address these issues, several algorithms that combine local computation, traps, and a centralized monitoring station were proposed in [DR02]. However, in order to deploy these algorithms the agent should be able to carry on simple computation tasks and issue traps, which are in many cases beyond the ability of the standard SNMP trap framework. This is a very good example where the extended functionality of HAMSA can be utilized. The global reactive monitoring application is executed in HAMSA in a distributed way. Namely, a number of copies of the same monitoring component are launched at several HA-MLMs. Each HA-MLM is responsible for its own local set of devices. According to the algorithm of [DR02], if a local threshold event has been detected (in this case the "local" means "with respect to the local set of variables"), then the other copies of the monitoring component are being notified using HAMSA messaging service. Then according to the algorithm, a global poll may be initiated, and, if needed, an alarm is declared. If one of the local monitoring processes fails, HA-MLM controller restarts it on another MLM, and the system continue functioning. This, of course, comes with a cost of increasing the monitoring traffic, but paying such a cost is definitely better than losing the ability to carry on with the critical monitoring task.

4.4 Performance Evaluation

In order to understand the trade-off between the communication overhead induced by HAMSA, and the possible reduction in monitoring overhead, consider again the scenario described in figures 4.3(a), and 4.3(b). We want to compare the amount of traffic overhead generated by the monitoring application without HAMSA with the overhead induced by HAMSA and the underlying group communication service.

The group communication service is responsible for failure detection that is based on periodic broadcasting of short *I-am-alive* messages. In general, if only unicast communication is available among the MLMs comprising a HA-MLM, this overhead grows as k^2, where k being the size of the HA-MLM. If some of the MLMs in the HA-MLM share the same broadcast domain, then this message overhead can be reduced by factor l, the number of broadcast domains [ACDS00]. It is important to stress that the overhead of failure detection is inevitable, and cannot be strictly attributed either to HAMSA, or to group communication, because any application wishing to achieve the high availability guarantees of HAMSA on its own, would pay these costs anyway. The experiments performed in [ACDS00] with the current implementation of Xpand and Transis show that group communication scales to 200 hosts dispersed over WAN without visible impact on the regular traffic.

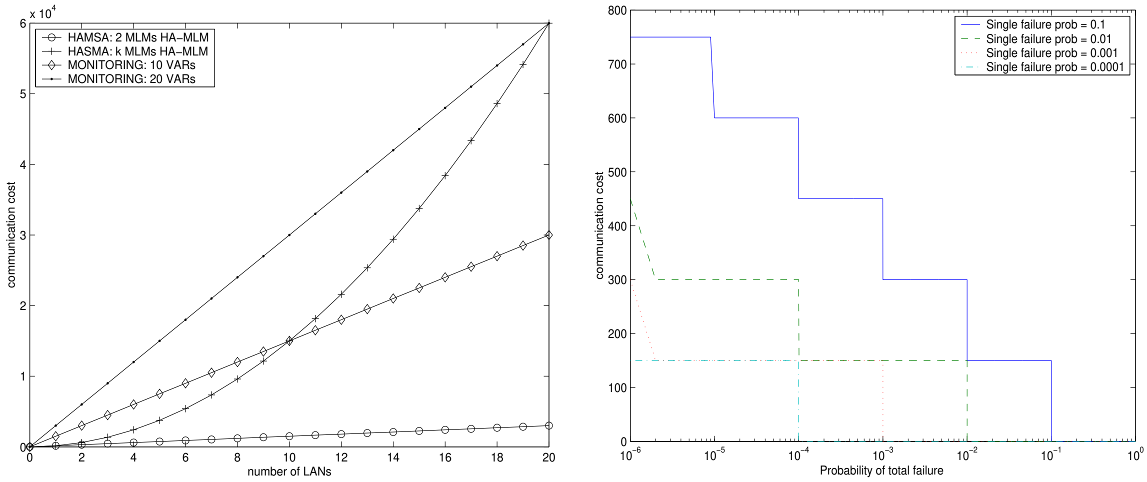
As described in Section 4.3 the overhead of HAMSA itself strongly depends on the way we configure HA-MLMs and on the size of the application *state*. In order to investigate the trade-off, assume that the state size sent by HAMSA is 150 bytes. This is a reasonable size, when one chooses to use a small state (like a measurement sequence number).

Figure 4.4(a) depicts the tradeoff for the two choices of HA-MLM configuration and for 10, and 20 scalar MIBs variables in each LAN. We assumed here that due to the SNMP encoding (SMIv1/SMIv2), polling of one variable takes about 150 bytes, and thus polling 10 or 20 variables per LAN will consume 1500, and 3000 bytes respectively for each LAN. HAMSA's overhead depends on the HA-MLM configuration. If all k MLMs are members of the same HA-MLM, we need to update all k states each polling interval. This takes $150 \cdot k^2$ bytes[3]. On the other hand, if we use k different HA-MLMs, each of size 2, HAMSA's overhead is reduced to $150 \cdot k$.

One can easily see that even for a very small number of monitored variables the overhead of HAMSA is significantly smaller than the monitoring overhead of a traditional application. This

[3]In this section, we assume that communication among the MLMs is unicast.

is a big advantage even without considering the HAMSA's main goals: extended functionality and reliability.



(a) Cost dep. on the number of LANs (b) Cost dep. on Availability Level

Figure 4.4: HAMSA Performance Evaluation

The main mission of HAMSA is increasing the system reliability. However, the high reliability comes with the cost of introducing more MLMs. In particular, this implies a higher communication overhead. Thus, there is a trade-off between the level of availability and the traffic overhead. In order to evaluate this trade-off, we consider the same scenario as above.

The current host MLM of an active component replica propagates its state to the rest of its HA-MLM through multicast. Thus, communication cost of HA-MLM replicating the state is linear in the number of group members. However, when we increase the number of MLMs in the group, we reduce the probability of a total system failure, since HA-MLM restarts the failed process on a different MLM as long as they are available.

Thus, if we have n MLMs in a HA-MLM, and the independent probability of a single MLM failure is p, the probability of the application failure is $\tilde{p} = 1 - p^n$. Since we have only one active component per network partition then the communication is $s \cdot (n-1)$ per each state in the component state, where s being the component state size. To obtain specific numbers, let $s = 150$ bytes, as in our previous example. Then, in order to get an application failure probability of $\tilde{p}$ we pay $150(\lceil \frac{\log(1-\tilde{p})}{\log p} \rceil - 1)$ bytes per change. This cost is plotted in Figure 4.4(b), for single MLM failure probability of 0.1, 0.01, 0.001, and 0.0001.

As one can see, in order to get the often desired "5 nines" reliability, starting from a very high error rate of 0.1 on a single machine, 6 MLMs are sufficient. The communication cost

$(150 \cdot (6-1) = 750$ bytes per state change) becomes much smaller when the reliability of a single machine increases.

4.5 Background and Related Work

The quest for more efficient and versatile management paradigms has been pursued by many researches over the last few years. One general line of approach suggests using mobile agents, active networks, or programmable networks for decentralizing and shortening the control loop [BPW98, RS00]. Usually, these proposals focus on the mechanics of the mobility and extended functionality rather than on the high availability and meta-management issues being in the focus of this work.

Several approaches for integrating the management by delegation approach [YGY91] into SNMP environment have been proposed recently [LS99b]. With the advent of Java, the delegation is easily implemented by exploiting its mobility and security features making Java a preferred language for developing delegated programs.

Java Management Extension (JMX) [Mic02] is an emerging Java standard for representing managed objects as Java Beans. JMX Bean is an object that serves as a Java wrapper facade for the actual managed object. JMX Beans may be co-located with the objects they represent at the agent side, or be deployed in a distributed fashion. In the latter case, JMX Beans need distributed object services of the second tier that are currently left unspecified by JMX. HAMSA components can be implemented as JMX Beans.

One of the more mature Java technologies for deploying three-tier Java applications is provided by Enterprize JavaBeans (EJB) [EST02]. EJB defines interfaces for Application Server, and Enterprize Java components (Beans) that execute in the environment of the application server that manages transactions, persistency, security, and naming services for the components.

The problems that HAMSA copes with are very similar to those of the state-full EJB clustering. Some of the existing EJB implementations provide fail-over models that allow replication of the beans' states, and support takeover of the failed beans by other servers in the cluster [EST02]. Most EJB servers perform state-full fail-over by using either in-memory replication, or persistent storage to a shared database. These solutions are inappropriate for the NM domain, since they rely on the fact that the network remains connected. To the best of our knowledge, there is no current implementation of EJB, or other application server technology that provide the high availability of the second-tier components execution to the level that allows their comparison with HAMSA.

4.6 Chapter Summary

Efficient monitoring of large and dynamic distributed systems becomes challenging. Current standard technologies scale poorly due to their inherently centralized approach. We presented a lightweight monitoring middleware called HAMSA that allows to enhance monitoring functionality dynamically, and decentralize it in a reliable and efficient manner. This work presents the architectural overview of the middleware, and the possible functional and performance trade-offs involved in its deployment. Our architecture uses a group communication middleware to increase availability, modularity, and scalability.

Appendix B presents the details of HA-MLM administration which includes creation, maintenance, and deletion of HA-MLMs. In Appendix C, we provide an overview of the (k, Δ)-bofo server semantics. Finally, in Appendix D, additional details on the main HAMSA algorithms can be found.

Chapter 5

Conclusions and Future Work

Data communication networks grow exponentially. One of the more challenging issues is management of very large networks to ensure that they operate within the required specifications, at a reasonable cost, and with optimal capacity. One of the more important factors affecting scalability of the network management system is the scalability of the monitoring mechanisms that it uses. Current standard solutions scale poorly for a variety of reasons. Among them are bandwidth inefficiency of the usually deployed polling-based in-band monitoring, and the difficulty of the meta-management involved with controlling highly distributed monitoring schemes.

In this work, we considered generic scalability problems associated with monitoring very large data communication networks. We proposed very general, technology-independent solutions, to these problems. Our solutions admit incremental deployment. In other words, our proposals are aimed at complementing the existing approaches, radically improving them, rather than on the universal replacement of the existing solutions, which would be impractical.

We suggest to treat differently the two different types of monitoring, *i.e.*, statistical and on-line, by different means. To address the bandwidth inefficiency of statistical monitoring, we proposed a novel algorithmic framework that optimizes intermediate parking of management data inside the network in such a way that the additional load on the intermediate nodes due to parking is minimized. In this work, we concentrated on optimizing the parking itineraries for a single management message. However, in practice, multiple management messages with similar or identical timing constraints may be generated simultaneously on different sources. If every message is acting independently of all other messages using only the locally available information about the current nodes' load when building its itinerary, then this may cause undesirable *convoy migration* effects when backtracking. This is similar to the problems occurring in distributed load balancing algorithms in clusters of workstations when tasks migrate to the least loaded node simultaneously and independently of each other, and rising the node's load sharply. Naturally, in order to optimize the network-wide behavior of the monitoring messages, additional signalling is required. In this research, we did not investigate this option leaving it as a future direction in which this work may continue.

Another interesting direction would be allowing the monitoring messages to explore alternative routing paths when the backtracking on the given path does not help for sufficiently long period of time. An alternative path to the destination either has to be pre-built in advance for every node in the given routing path, or be constructed dynamically in the *ad hoc* manner by diverting the message to a node with the maximal probability of having a non-congested routing path to the destination. In both cases, additional *a priori* information about the network topology would be needed, and processing overhead would be increased.

In case of on-line monitoring in the framework of multicast group size estimation for the purposes of pricing and billing we proposed using generic hierarchical event-driven algorithms that were shown to be superior in terms of bandwidth to other alternatives. Since our primary goal was establishing of a generic algorithmic framework that can be used for distributed calculation of *any* additive metric with minimal bandwidth overhead, and not just for group size estimation, we did not invest much effort into application-specific optimizations. However, additional bandwidth savings are possible if our scheme is optimized to exploit the statistical properties of the users' join/leave behavior. In particular, our recent studies of the actual user behavior[1] suggest that in the common case, users are not uniformly distributed over domains[2], but rather exhibit a heavy tailed time-dependent spatial distribution with a very small number of domains having a majority of the users at any given time instance. The "heavy" domains change with the passage of time reflecting the time differences in the world's regions. These patterns are quite stable. If they are taken into account, it is possible to devise a much more efficient scheme for distribution of thresholds over the virtual monitoring tree described in Chapter 3, and additionally reduce bandwidth consumption without hurting the accuracy necessary for billing.

Another direction that may be interesting to explore in the context of group size estimation for pricing and billing is extending our accounting mechanism to handle other factors besides group size. In particular, one may be interested in accounting the receivers' population only

[1]This is a work in progress, and its results are not shown in this work yet.

[2]This assumption is usually made by researchers for two reasons: (1) for the sake of mathematical tractability, (b) in some specific cases users are indeed distributed uniformly over domains.

in specific geographic areas, because having receivers there results in the highest utilization for the transmitter.

We plan to continue our work on HAMSA in the following direction. First, we would like to encourage researchers and practitioners all over the world to experiment with our prototype by allowing to download it freely, so that more applications would be developed for HAMSA, and it would be tried out in different settings. Second, we would like to generalize our experience with the existing applications, and optimize our implementation accordingly. Currently, HAMSA is a proof-of-concept prototype. We did not invest considerable efforts into optimizing and fine tuning so far, and we plan doing this in the near future.

Bibliography

[AA97] K. Almeroth and M. Ammar. Multicast Group Behavior in the Internet's Multicast Backbone (MBone). *IEEE Communications*, 35(6):224–229, June 1997.

[AAN02] Sara Alouf, Eitan Altman, and Philippe Nain. Optimal On-Line Estimation of the Size of a Dynamic Multicast Group. In *INFOCOM'02*, pages 23–33, New York City, New York, USA, June 2002.

[ABDL97] T. Anker, D. Breitgand, D. Dolev, and Z. Levy. CONGRESS: CONnection-oriented Group-address RESolution service. In *Broadband Networking Technology, International Society for Optical Engineering SPIE, Seyhan Civanlar, Indra Widjaja; Eds.*, volume 3233, pages 89–100, Dallas, TX, USA, October 1997.

[ACDS00] T. Anker, G. Chockler, D. Dolev, and I. Shnaiderman. The Design of Xpand: A Group Communication System for Wide Area. Technical Report HUJI-CSE-LTR-2000-31, The Hebrew University, July 2000.

[ADKM92] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A Communication Sub-System for High Availability. In *22nd Annual International Symposium on Fault-Tolerant Computing*, pages 76–84, Boston, MA, USA, July 1992.

[AfCM96] Association for Computing Machinery. *Communications of the ACM, Special Issue on Group Communication Systems*, 39(4), April 1996.

[AS98] Ehab Al-Shaer. *A Hierarchical Filtering Based Monitoring Architecture for Large Scale Distributed Systems*. PhD thesis, Old Dominion University, December 1998.

[ASAWM97] E. Al-Shaer, H. Abdel-Wahab, and K. Maly. High-Performance Monitoring Architecture for Large-Scale Distributed Systems Using Event Filtering. In *Third International Conference on Computer Science and Informatica*, volume 3, pages 42–46, March 1997.

[BACW90] A. Ben-Artzi, A. Chandna, and U. Warrier. Network Management of TCP/IP Networks: Present and Future. *IEEE Network Magazine*, 4(4):35–43, July 1990.

[BDR03] D. Breitgand, D. Dolev, and D. Raz. Accounting Mechanism for Membership Size-Dependent Pricing of Multicast Traffic. In *Group Communications and Charges; Technology and Business Models, 5th COST264 International Workshop on Networked Group Communications, NGC 2003, and 3rd International Workshop on Internet Charging and QoS Technologies, ICQT 2003, Munich, Germany, September 16-19, 2003, Proceedings*, volume 2816 of *Lecture Notes in Computer Science*, pages 276–286. Springer, September 2003.

[BDRS03] D. Breitgand, D. Dolev, D. Raz, and G. Shaviner. Facilitating Efficient and Reliable Monitoring through HAMSA. In *IFIP/IEEE Integrated Management (IM)*, pages 263–277, Colorado Springs, USA, March 2003.

[Bel57] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[Bel58] R. Bellman. On a Routing Problem. *Quart. Appl. Math.*, 16(4):87 – 90, 1958.

[BMB+99] Stephen M. Blott, Clifford E. Martin, Yuri J. Breitbart, Jose C. Brustoloni, Thomas R. Gramaglia, Henry F. Korth, David M. Kristol, Robert H. Liao, Eben L. Scanlon, and Avi Silberschatz. User-Level Billing and Accounting in IP Networks. *Bell Labs Technical Journal*, 4(4):237–252, oct–dec 1999.

[BMST92] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. Primary-backup Protocols: Lower bounds and optimal implementations. In *Third IFIP Working Conference on Dependable Computing*, pages 187–198, Mondello, Italy, 1992.

[BNS91] Amotz Bar-Noy and Baruch Schieber. The Canadian Traveller Problem. In *Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 261 – 270, San-Francisco, California, 1991.

[Bor01] S. Boros. Policy-Based Network Management With SNMP. `http://citeseer.nj.nec.com/379518.html`, 2001.

[BPW98] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1):2–9, Forth Quarter 1998.

[Bri99] B. Briscoe. The Direction of value flow in connection-less networks. In *NGC'99*, pages 244–269, Pisa, Italy, November 1999.

[BRS02a] D. Breitgand, D. Raz, and Yuval Shavitt. SNMP GetPrev: An Efficient Way to Browse Large MIB Table. *IEEE Journal of Selected Areas in Communications (JSAC), Special Issue on Advances in the Fundamentals of Network Management and Operations*, 20(4):656–668, May 2002.

[BRS02b] David Breitgand, Danny Raz, and Yuval Shavitt. The Travelling Miser Problem. In *INFOCOM'02*, New-York, NY, USA, September 2002.

[BSD00] D. Breitgand, G. Shaviner, and D. Dolev. Towards Highly Available Three-Tier Monitoring Applications (extended abstract), short papers session. `http://cs.huji.ac.il/~davb/abstracts/hamsa.ps`, April 2000. 11th IFIP/IEE Internationaal Workshop on Distributed Systems: Operations and Management, Austin TX, USA.

[BTW94] J-C. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *ACM SIGCOMM'94*, pages 58–67, London, UK, September 1994.

[CA01] Robert C. Chalmers and Kevin C. Almeroth. Modelling the Branching Characteristcs and Efficiency Gains in Global Multicast Trees. In *INFOCOM 2001*, pages 77–86, Anchorage, Alaska, USA, April 2001.

[CFSD90] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP), May 1990. RFC 1157.

[CH66] Kenneth L. Cooke and Eric Halsey. The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. *Journal of Mathematical Analysis and Applications*, 12(7):493 – 498, 1966.

[CKV01] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group Communication Specifications: a comprehensive study. *ACM Computnig Surveys*, 33(4):427–469, March 2001.

[CLG+03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. DIAMETER Base Protocol, September 2003. IETF RFC3588, Proposed Standard.

[CS01] John C.-I. Chuang and Marvin A. Sirbu. Pricing Multicast Communication: A Cost-Based Approach. *Telecommunication Systems*, 17(3):281–297, 2001.

[DMS03] D. Dolev, O. Mokryn, and Y. Shavitt. On Multicast Trees: Structure and Size Estimation. In *INFOCOM'03*, pages 344–355, San-Francisco, USA, April 2003.

[DR02] Mark Dilman and Danny Raz. Efficient Reactive Monitoring. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Recent Advances in Network Management*, 20(4):668–677, May 2002.

[EST02] Roman E., Ambler S., and Jewell T. *Mastering Enterprise JavaBeans(tm).* Wiley, 2nd edition, 2002.

[Fora] Distributed Management Task Force. Common Information Model (CIM). `http://www.dmtf.org/standards/cim`.

[Forb] Distributed Management Task Force. DMTF Published Standards. `http://www.dmtf.org/standards`.

[Forc] Internet Engineering Task Force. IETF Home Page. `http://www.ietf.org/`.

[FT99] T. Friedman and D. Towsley. Multicast Session Membership Size Estimation. In *IEEE INFOCOM'99*, volume 2, pages 965–972, New York City, NY, USA, March 1999.

[GB02] Manuel Günter and Torsten Braun. Internet Service Monitoring with Mobile Agents. *IEEE Network: The Magazine of Global Internetworking*, 16(3):22–30, May–June 2002.

[GG00] Ran Giladi and Merav Gat. Meta-Management of Dynamic Distributed Network Managers (MEMAD). In *11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM'00*, pages 119–131, Austin, TX, USA, December 2000.

[GY95] Germán Goldszmidt and Yechiam Yemini. Distributed Management by Delegation. In *International Conference on Disributed Computing Systems*, pages 333–340, Vancouver, British Columbia, Canada, June 1995.

[Hal77] J. Halpern. Shortest Route with Time Dependent Length of Edges and Limited Delay Possibilities in Nodes. *Zeitschrift fuer operations research*, 21(2):117 – 124, 1977.

[HB00] T. Henderson and S.N. Bhatti. Protocol Independent Multicast Pricing. In *10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'00)*, The University of North Carolina, USA, June 2000.

[HC99] Hugh W. Holbrook and David R. Cheriton. Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications. In *ACM SIGCOMM'99*, pages 65–78, Harvard, MA, USA, September 1999.

[HSE95] S. Herzog, S. Shenker, and D. Estrin. Sharing the cost of multicast trees: An axiomatic analysis. In *SIGCOMM'95*, pages 315–327, Cambridge, MA, USA, August 1995.

[IETF03] Internet Engineering Task Force. The IETF Policy Framework Working Group. `http://www.ietf.org/html.charters/policy-charter.html`, 2003.

[IGSD98] Irina Ioachim, Sylvie Gelinas, Francois Soumis, and Jacques Desrosiers. A Dynamic Prograamming Algorithm for the Shortest Path Problem with Time Windows and Linear Node Costs. *Networks*, 31(10):193 – 204, 1998. John Wiley & Sons, Inc.

[Int] International Telecommunication Union. ITU-T. TMN. `http://www.itu.int/ITU-T/studygroups/`.

[IPD02] IPDR: Internet Protocol Detail Record Organization. Service specifications. `http://www.ipdr.org/download-docs/register/df_servicespecs.html`, 2002.

[IS96] A. Itai and H. Shachnai. Adaptive Source Routing in High-Speed Networks. *Journal of Algorithms*, 20(5):218–243, 1996.

[ISO] ISO/IEC. DIS 10164-5, information processing systems - open systems interconnection - systems management - part5: Event report management function.

[JdS01] E. P. Duarte Jr. and Aldri L. dos Santos. Semi-Active Replication of SNMP Objects in Agent Groups Applied for Fault Management. In *7th IEEE/IFIP International Symposium on Integrated Network Management (IM)*, Seattle, WA, May 2001.

[JS99] J. Quittek J. Schöenwaelder. IETF RFC 2593: Script MIB Extensibility Protocol Version 1.0, May 1999.

[Kla72] Emil Klafszky. Determination of Shortest Path in a Network with Time-Dependent Edge-Lengths. *Math. Operationsforsch. u. Statist.*, 3(4):255 – 257, 1972.

[KS00] Ryutaro Kawamura and Rolf Stadler. Active Distributed Management for IP Networks. *IEEE Communications Magazine*, 38(4):114–120, April 2000.

[Liv97] C. Rigney Livinston. RADIUS Accounting, April 1997. IETF RFC2139, Informational.

[LN00] C. Liu and J. Nonnenmacher. Broadcast Audience Estimation. In *IEEE INFOCOM'00*, volume 2, pages 952–960, Tel-Aviv, Israel, March 2000.

[LPK02] Antonio Liotta, George Pavlou, and Graham Knight. Exploiting Agent Mobility for Large-Scale Network Monitoring. *IEEE Network: The Magazine of Global Internetworking*, 16(3):7–16, May–June 2002.

[LS99a] D. Levi and J. Schöenwaelder. Definitions of Managed Objects for Scheduling Management Operations, May 1999. RFC 2591.

[LS99b] D. Levi and J. Schöenwaelder. IETF RFC2592: Definitions of Managed Objects for the Delegation of Management Script, May 1999.

[MCWB91] K. Marzullo, R. Cooper, M. D. Wood, and K. P. Birman. Tools for Distributed Application Management. *IEEE Computer*, 24:42–51, August 1991.

[MG96] Siegl M. and Trausmuth G. Hierarchical Network Management; a Concept and its Prototype in SNMPv2. *Computer Networks and ISDN Systems*, 4(28):441–452, 1996.

[Mic01] Sun Microsystems. Enterprise JavaBeans(TM) Specification 2.0. `http://java.sun.com/products/ejb/doc.html`, August 2001.

[Mic02] Sun Microsystems. Java Management Extensions (JMX) Instrumentation and Agent Specification, v1.1. `http://java.sun.com/products/JavaManagement/doc.html`, March 2002.

[MM99] P. E. Miller and Mark A. Miller. *Managing Internetworks with SNMP*. M&T Books, 2 edition, November 1999.

[Mou98] Maria-Athina Mountzia. A Distributed Management Approach based on Flexible Agents. *Interoperable Communication Networks*, 1(1):99–120, January 1998.

[MR91] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets: MIB-II, March 1991. Internet RFC1213.

[MSS97] Masoud Mansouri-Samani and Moris Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[NB98] Jorg Nonnenmacher and Ernst W. Biersack. Optimal Multicast Feedback. In *IEEE INFOCOM'98*, pages 964–971, San-Francisco, California, USA, March 1998.

[NSB03] Maziar Nekovee, Andrea Soppera, and Trevor Burbridge. An Adaptive Method for Dynamic Audience Size Estimation in Multicast. In *NGC/ICQT'03*, pages 23–33, Munich, Germany, September 2003.

[OR90] Ariel Orda and Rafael Rom. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3):607 – 625, July 1990.

[OR91] Ariel Orda and Rafael Rom. Minimum Weight Paths in Time-Dependent Networks. *Networks*, 21:295 – 319, May 1991.

[OR96] Ariel Orda and Rafael Rom. Distributed Shortest Path and Minimum-Delay Protocols in Networks with Time-Dependent Edge-Length. *Distributed Computing*, 10(1):49 – 62, 1996.

[Org02] IPDR: Internet Protocol Detail Record Organization. Network Data Management Usage Specification - Version 3.1.1. `http://www.ipdr.org/download-docs/register/df_ndmu-311.html`, 2002.

[OSI91] Open Systems Interconnection. OSI, I.S.O. 10040 Systems Management Overview, 1991.

[PCM+02] R. Presuhn, J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport Mappings for the Simple Network Management Protocol (SNMP), December 2002. RFC3417.

[Pso99] Konstantinos Psounis. Active Networks: Applications, Security, Safety, and Architectures. *IEEE Communications Surveys*, 2(1), First Quarter 1999. `http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html`.

[PST99] G. Phillips, S. Shenker, and H. Tangmunarunkit. Comments on the Chuang-Sirbu Scaling Law. In *ACM SIGCOMM'99*, pages 41–51, Cambridge, Massachsetts, USA, August 1999.

[PT86] K. J. Perry and S. Toueg. Distributed Agreement in Presence of Processor and Communication Faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, 1986.

[PWW00] B. Pagurek, Y. Wang, and T. White. Integration of Mobile Agents with SNMP: Why and How. In *IEEE/IFIP Network Operations and Management Symposium*, pages 609 – 622, Honolulu, Hawaii, USA, April 2000.

[PY89] C.H. Papadimitriou and M. Yannakakis. Shortest Paths without a Map. In *16th ICALP*, pages 610–620, July 1989.

[RD99] Marcelo Gonalves Rubinstein and Otto Carlos Muniz Bandeira Duarte. Evaluating Tradeoffs of Mobile Agents in Network Management. *Networking and Information Systems Journal*, 2(2):237–252, 1999. HERMES Science Publications.

[RM90] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets, May 1990. RFC1155.

[RM95] Marshall T. Rose and Keith McCloghrie. *How to Manage Your Network using SNMP*. Prentice Hall, January 1995.

[RS00] Danny Raz and Yuval Shavitt. Active Networks for Efficient Distributed Network Management. *IEEE Communications Magazine*, 38(3), March 2000.

[SA00] K. Sarac and Kevin C. Almeroth. A Long-Term Analysis of Growth and Usage Patterns in the Multicast Backbone (MBone). In *INFOCOM'00*, pages 824–833, Tel-Aviv, Israel, March 2000.

[SBSP00] Kiseok Sung, Michael G.H. Bell, Myeongki Seong, and Soondal Park. Shortest Paths in a Network with Time-Dependent Flow Speeds. *European Journal of Operational Research*, 121(12):32 – 39, 2000.

[Sch84] F. B. Schneider. Byzantine Generals in Action: implementing fail-stop processors. *ACM Transactions on Computer Systems*, 2(2):145–154, 1984.

[Sch90] Fred B. Schneider. Implementing Fault-Tolerant Services using the State-Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[Sch02] J. Schöenwaelder. Simple Network Management Protocol (SNMP) over Transmission Control Protocol (TCP) Transport Mapping, December 2002. RFC3430, Experimental.

[SJS+99] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart Packets for Active Networks. In *IEEE OPENARCH'99*, pages 90–97, New York, NY, USA, March 1999.

[Slo94] M. Sloman. Policy Driven Management for Distributed Systems. *International Journal of Network and Systems Management*, 2(4):333–360, December 1994.

[SM03] Sun Microsystems. Java Dynamic Management Kit. `http://java.sun.com/products/jdmk`, 1998–2003.

[SMF99] Ron Sprenkels and Jean-Philippe Martin-Flatin. Bulk Transfers of MIB Data. *The Simple Times, The Quarterly Newsletter of SNMP Technology, Comment, and Events*, 7(1):1–8, March 1999.

[Sta99] William Stallings. *SNMP, SNMPV2, SNMPV3, and RMON 1 and 2*. Addison-Wesley, January 1999.

[SW99] Mark L. Stevens and Walter J. Weiss. Policy-based Management for IP Networks. *Bell Labs Technical Journal, Network Management*, 4(4), oct–dec 1999.

[Ver02] Danesh C. Verma. Simplifying Network Administration Using Policy-Based Management. *IEEE Network*, 16(2):40–50, March 2002.

[vRBDV03] R. van Renesse, K. P. Birman, D. Dumitriu, and W. Vogels. A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems (TOCS)*, 21(2):164–206, May 2003.

[WWWC] World Wide Web Consortium. SOAP and Web Services. `http://www.w3.org/2002/ws`.

[Yem93] Y. Yemini. The OSI Network Management Model. *IEEE Communications Magazine*, 31(5):20–29, May 1993.

[YGY91] Yechiam Yemini, Germán Goldszmidt, and Shaula Yemini. Network Management by Delegation. In *The Second International Symposium on Integrated Network Management*, pages 95–107, Washington, DC, USA, April 1991.

[ZE02] E. Zhang and E. Elkin. XACCT's Common Reliable Accounting for Network Element (CRANE) Protocol Specification Version 1.0, November 2002. IETF RFC3423, Informational.

[ZHGW99] Michael Zapf, Klaus Herrmann, Kurt Geihs, and Johann Wolfgang. Decentralised SNMP Management with Mobile Agents . In *Sixth IFIP/IEEE International Symposium on Integrated Network Management - IM'99*, Boston, MA, USA, May 1999.

Appendix A

Size-Dependent Cost Model

The basic model of Sections 2.2, 2.3 does not take into account the actual size of the messages. Thus, commuting and parking costs do not differentiate between large and small messages. In this appendix we refine our basic model by introducing costs that depend linearly on the message size. More specifically, in the definition of k-block recoverable TMP of Section 2.3, $w_{i,j}$ changes as follows.

Given a finite deadline $\mathcal{D}$ and a time-dependent path $R(s,d)$, block sequence $\{b_1, b_2, ..., b_k\}$, $\forall (i,j) \in R$ let link weight function

$$w_{i,j}(t) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } \exists l \text{ s.t. } b_l = \langle (i,j), t_l^{start}, t_l^{end} \rangle \\ c_{i,j} \cdot z + x_{i,j} & \text{otherwise} \end{cases}$$

where $c_{i,j}$, and $x_{i,j}$ being finite positive constants, and z being a size of the message.

As one can easily verify, the analysis of Section 2.4.2 still holds for the Miser strategy in our new model. In particular, the competitive ratio of the strategy in case of the single block per path throughout the deadline duration remains $O(\log B)$.

However, if one partition the message into smaller pieces, the Miser strategy can be improved in the new model. Indeed, in the Miser strategy, the message goes back and force between the blocked node, and the node resulting in the cheapest round-trip itinerary for phase j of the algorithm. The cost of these recurring round-trip itineraries can be minimized if we slightly change our basic on-line strategy. In the following section we describe a modified Miser algorithm that takes advantage of the extended cost model. We call this version *Scout*.

As the size of the minimal message that can be sent over the network decreases, so does the cost of probing the blocked link. Thus, the Scout heuristics asymptotically converges to the Trap strategy discussed in Section 2.4.3.

A.1 Scout Strategy

Let s_{min} be the *minimal MTU* allowed by the underlying communication network. Then we may take an additional advantage of the active network paradigm, by allowing the miser message to spawn a minimal size *scout* message on demand as explained below.

As long as no link *en-route* to the destination is blocked, the message moves forward, as it was the case with the Miser strategy. If a blocked link is encountered at node v_b, the algorithm proceeds in stages exactly as in the Miser algorithm with one important difference. When the parking time of phase j terminates, the message that parks at node v_j^* (of phase j) spawns a scout message of size s_{min} that follows the shortest round-trip itinerary $\mathcal{I}_{rt}(v_j^*, v_b)$ in order to check the status of the blocked link, and inform the parked message about it.

If the previously blocked link recovered, the message moves towards the destination. Otherwise, it behaves like the miser. It computes the cheapest round-trip itinerary for the next phase for all the nodes that are reachable from the current location of the message, and follows this itinerary. Then the process repeats itself at the end of the next phase.

To analyze this strategy, let us first notice that the running time of the algorithm remains the same as for the Miser algorithm up to a small constant factor. This is because each of $1 + \log_2 B$ phases of the algorithm may take slightly longer because of the delay introduced by the scout message.

It is important to observe that if the link is reported by the scout message as blocked at the end of phase j, in phase $j+1$, the message can either move backwards (*i.e.,* towards its original source), or stay in place. However, it would **never** move in the forward direction unless the scout reports that the previously blocked link became passable.

This means that the large message would not travel the path more than three times, while the scout message would not go back and forth more than $\log_2 B$ times. Let $k = \lceil \frac{z}{s_{min}} \rceil$ where z is the original message size. Following the logic similar to the one that was used in proving Theorem 1, we obtain:

$$W(\mathcal{I}_{scout}) \leq 3 \cdot W(\mathcal{I}_{opt}) + \cdot (4 \cdot \log_2 B + 5) \cdot \frac{W(\mathcal{I}_{opt})}{k}$$

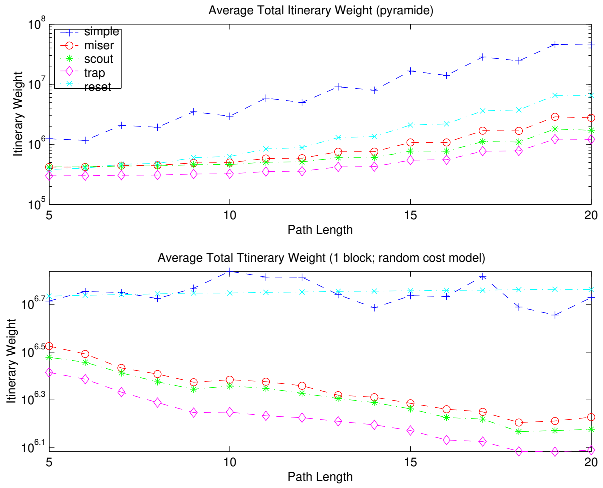


Figure A.1: Average Itinerary Weight as a Function of Path Length (1 block; sample 100)

.

Thus, the competitive ratio of the Scout algorithm is $\frac{(4 \cdot \log_2 B+5)}{k} + 3$. In other words, we obtain a linear gain in the total cost which is roughly the ratio between the minimal size active message (the scout), and the average size of the management message. Section A.2 shows a comparative simulation study of the *Scout* strategy in the size-dependent parking cost model.

A.2 Size-Dependent Model Simulation Study

The simulation study of this section is similar to the one of Section 2.5. We use the same two cost models, *pyramid*, and *uniform* as before. However, the parking weights generated at the beginning of each run are treated as coefficients $c_{i,i}$ (parking self-links weights) in the model above. Throughout all the simulations we use message deadline of 10,000 time units, the total blocking time is drawn from $U(10, \mathcal{D}/2)$, the message size is fixed at 1024, and the scout size is fixed at 128, *i.e.*, the ratio between the bulk of the message and the scout is 8.

Figures A.1, A.2, A.3, and A.4 show the relative performance of all algorithms in the size dependent linear cost functions model. Figures A.5, A.6, and A.7 depict the sample CDF function of itinerary cost for path length 5, and 1, 4, and 14 blocks respectively. This corresponds to *short* path being relatively stable (1 block), moderately unstable (4 blocks), and highly unstable (14 blocks). Figures A.8, A.9,and A.10 show the sample CDF function of itinerary cost for path length 10, and 1, 4, and 14 blocks. This corresponds to studying the algorithms' behavior over

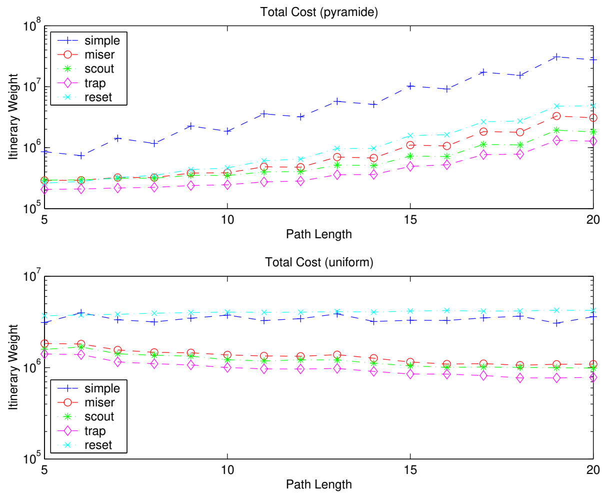


Figure A.2: Average Itinerary Weight as a Function of Path Length (2 blocks; sample size 100)

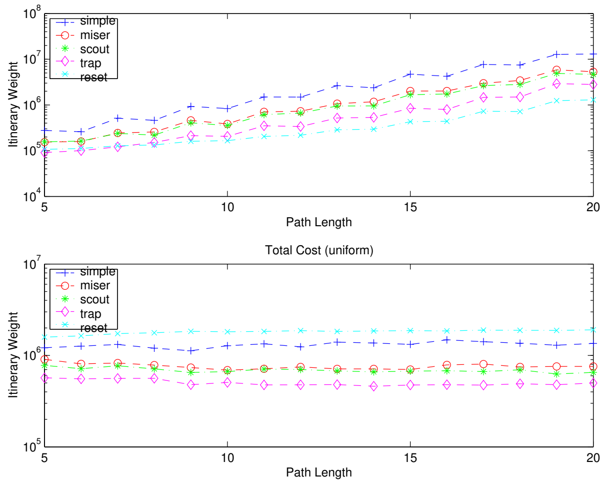


Figure A.3: Average Itinerary Weight as a Function of Path Length (8 blocks; sample size 100)

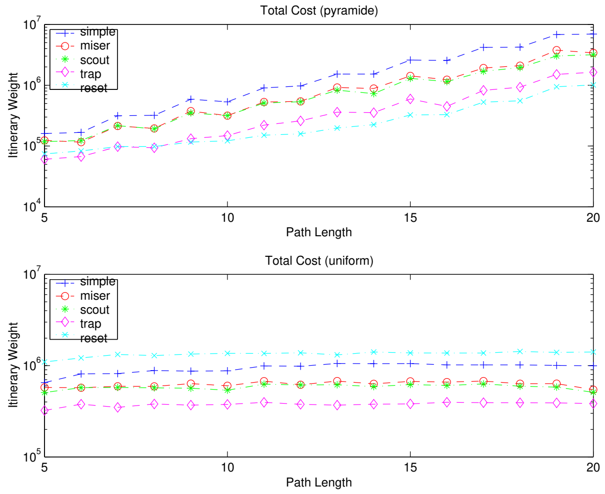


Figure A.4: Average Itinerary Weight as a Function of Path Length (14 blocks; sample size 100)

the *long* paths.

As can be observed from the figures, the *Scout* heuristics becomes more advantageous when path length grows. This is natural since larger paths exibit sharper differences in parking costs. *Scout* is superior also over short paths when there exist large differences in the parking costs.

Obviously, as the ratio between the whole message and the scout increases, the *Scout* algorithm saves more. We assumed ratio of 8 in our simulations, but the ratio may easily be at least two orders of magnitude in practice, which would make *Scout* a preferable on-line strategy.

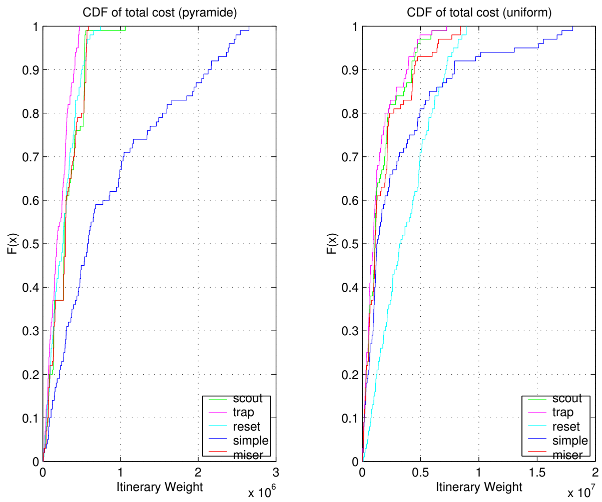


Figure A.5: CDF of Itinerary Weight (1 block; path length 5; sample size 100)

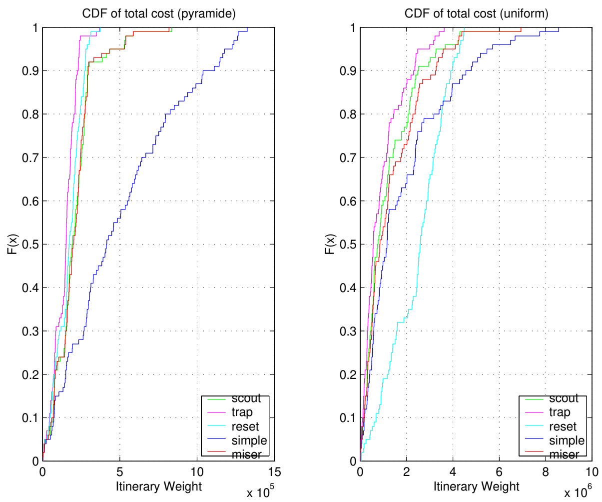


Figure A.6: CDF of Itinerary Weight (4 blocks; path length 5; sample size 100)

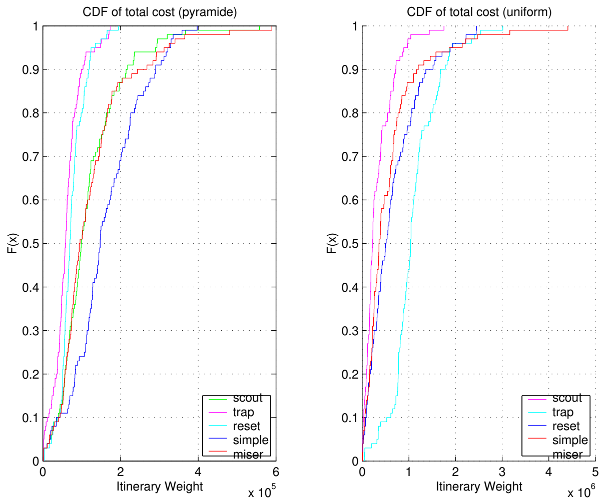


Figure A.7: CDF of Itinerary Weight (14 blocks; path length 5; sample size 100)

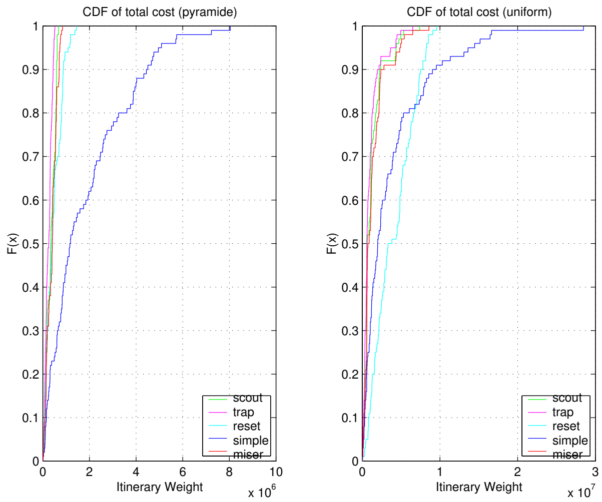


Figure A.8: CDF of Itinerary Weight (1 block; path length 10; sample size 100)

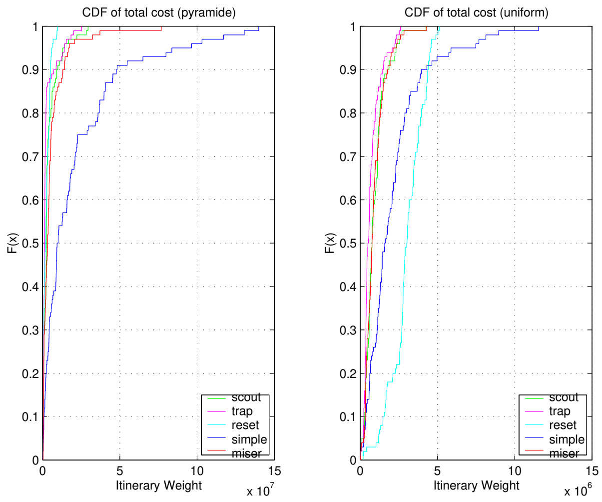


Figure A.9: CDF of Itinerary Weight (4 blocks; path length 10; sample size 100)

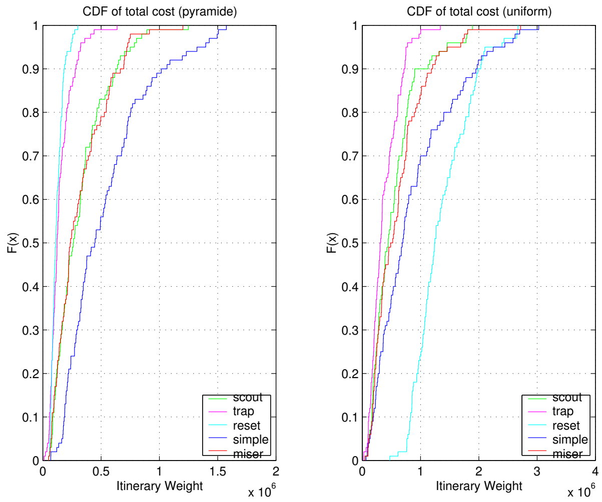


Figure A.10: CDF of Itinerary Weight (14 blocks; path length 10; sample size 100)

Appendix B

HA-MLM Administration

HA-MLMs are created through the HAMSA Administration Tool. A screenshot of HA-MLM creation operation is presented in Figure B.1.

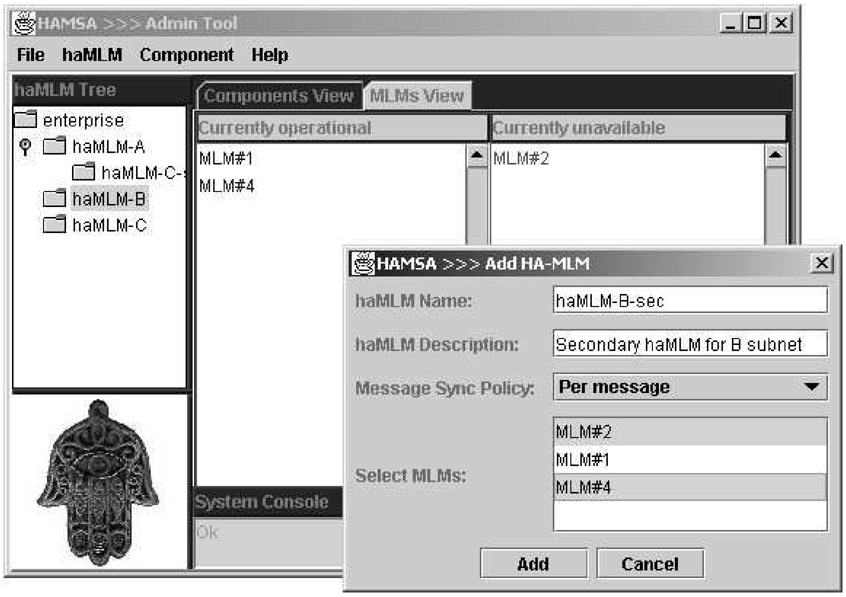


Figure B.1: Creating new HA-MLM

The administrator picks the MLMs she wishes and groups them into HA-MLMs with a unique name. The same MLM may be a member of different HA-MLMs. A logical hierarchy of HA-MLMs may be formed. The MLMs chosen by the administrator form the *nominal view* of the HA-MLM, as opposed to the *current view*, which is always less or equal to the nominal view due to failures. MLMs that are bundled into a HA-MLM join the process group with the same name. The joining is triggered by the HA-MLM controller that multicasts a JOIN message into the ENTERPRIZE process group specifying the nominal view.

One restriction that HAMSA imposes on the logical hierarchy is that all the nominal view of the new HA-MLM should be a proper subset of the nominal view of its parent HA-MLM in the hierarchy. Another restriction is that HA-MLM creation is not allowed unless at least one MLM from the nominal view is available.

Each time a new member joins HA-MLM, it receives the *state* of the group from someone that is already in the group. The state versions are identified using the pair: $<$ *current view* $>$, $<$ *epoch number* $>$, where the epoch number is advanced each time the membership changes.

The communications cost involved in creating a new HA-MLM are dominated by the following. A single multicast through Transis is needed to propagate a *join* message. There are at most k membership change notifications delivered by Transis, where k is the size of the nominal view of the new HA-MLM. Finally, there are at most k state exchange messages needed to accommodate each newly joining MLM.

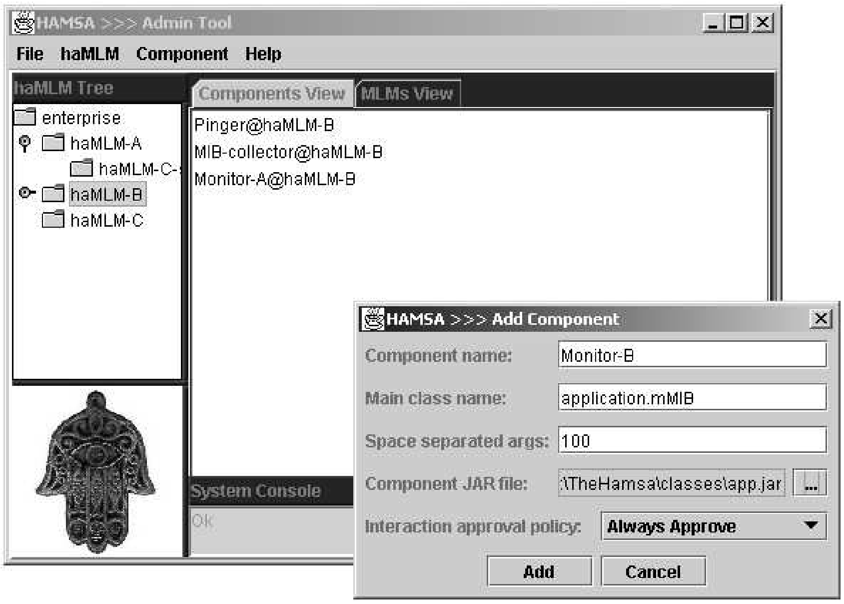


Figure B.2: Adding new HAMSA component

When a HA-MLM already exists, HAMSA compatible components can be delegated to it using the administration tool. A screenshot of the delegation operation is presented in Figure B.2.

The administrator specifies the target HA-MLM, component name, and the component specific parameters, such as the interaction approving policy, and placement policy. The HAMSA components are realized as JAR packages. Two interaction approval policies are supported currently. One policy is always to approve all interactions[1]. The second policy is to approve external interactions only if the majority of MLMs in HA-MLM are present in the network partition. This is also the default policy.

Placement policy defines either load-dependent, or distance-dependent placement of a component. These policies are best-effort ones.

The communication cost involved in the component delegation protocol is dominated by multicasting a component through Transis to all MLMs of the HA-MLM, and obviously, depends on the size of the component.

[1]Note that this policy is not safe, because it may lead to violating the (k, Δ)-bofo server semantics of Section 4.2.1. However, situations may exist where strong semantics may be traded for performance gain, provided that the management application is sophisticated enough to cope with the violations of the (k, Δ)-bofo semantics.

Appendix C

(k, Δ)-bofo Semantics

In a two-tier client/server architecture, a highly available service that is implemented using the primary-backup approach consists of a set of servers, in which only one server acts as a *primary* at any given time instance. A client initiates requests by sending a message to the server, that it believes is the current primary. For example, if the current primary is registered with the Naming Service using some well-known name, then a name resolution request to the Naming Service will return the current primary to the client. When the primary server receives the client's request it processes it, and sends a response. Let us assume, for simplicity, that all clients' requests are *synchronous*. This means that the client does not initiate a new request until it gets the response to the previous one. Let δ be an upper bound on the time that it takes the server to process the client's request, and for a message to travel in a round-trip between the client and the server. Then, if a request was issued at time t, and no response is received by the time $t + \delta$, we say that the *service outage* has occurred at time t. If at time $t + \delta + T$ client receives a response to its request, we say that service outage interval is T.

In the primary-backup replication protocol, other servers detect that the current primary became unavailable, and elect the new primary using some algorithm. The new primary continues providing the service from the last service-specific *state*, that is known to it. To allow this, primary should propagate the state changes that it makes as a result of processing the client's requests, to other servers. Various primary-backup protocols will differ on such issues as when the state changes are propagated, to which other members of the replication group, and on the exact sequence of the operations. The common part among all these protocols is that at some point, the primary notifies other members of the replication group about the state change to allow them transparently to take over in case of this primary's failure.

In [BMST92] Budhiraja *et. al.*, define the following four properties for the primary-backup client/server protocol[1].

- **PB1**: There exists a local predicate *I_Am_the_Primarys* defined on the set of identities of the servers comprising the replication group. At any time, there is at most one server s for which this which this predicate evaluates to TRUE.

- **PB2**: Each client i maintains a server identity $Primary$ that identifies the current primary server for this client. When the client makes a request, the corresponding message is sent to $Primary$.
  This property, actually, distinguishes between the primary-backup replication approach from the state-machine approach [Sch90], in which the client *broadcasts* its requests to all the servers, that should process them in the same order to arrive to the same consistent state.

- **PB3**: If a client's request arrives at a server i, for which *I_Am_the_Primarys* = FALSE, this request is not processed.

- **PB4**: There exist fixed values k and Δ such that all service outages can be grouped into at most k intervals of length at most Δ each.
  Essentially, this property rules out implementations in which the primary ignores all requests from the clients. This property also implies that only a bounded number of failures can be tolerated by the service during its lifetime. To implement a service that tolerates an unbounded number of failures, we should partition the operation time of the service into *epochs*. Only a bounded number of errors occur in each *epoch*, and the recovered servers are re-integrated at the epoch boundaries.

For the general omission failure, [BMST92] proves a lower bound of $2 \cdot f$ on the degree of replication, where f is the maximal number of failures that may occur during a given epoch.

Now, let us consider HAMSA's algorithms in light of this model. First, we observe that concerning the interactions between the external clients and the HAMSA components (and among the different components in the second tier), HA-MLMs that host these component behave like a (k, Δ)-bofo server with respect to every component. Indeed, for each component there is a single primary MLM in the HA-MLM group at any given time, providing communication with the component executing in its environment. If a majority of MLMs fails in HA-MLM, and

[1] We slightly change the notation from the original one in [BMST92].

the interaction approval policy is *majority approve* (see Appendix B), no communication with the component is possible. However, in HAMSA this does not imply the service outage. In fact, components continue running, and perform monitoring on the target network elements. It is just the results of their activity that are temporarily unavailable. We relax the property PB1 when it comes to communication between the components, and their target network elements. In fact, we allow multiple 'primaries operating in different partitions simultaneously, and gathering the monitoring information concerning the network elements in their respective partitions, with one primary per partition. However, to preserve the bofo server semantics to the higher-level clients of these components, we do not allow at most one primary (the one that executes in the majority partition) to communicate with entities other than the target network elements.

To appreciate HAMSA model, consider a non-replicated monitoring application that communicates with some manager at one hand, and with the network elements on the other hand. If this application is state-full, then it should be capable of storing its state on the non-volatile storage each time it considers that the loss of the information gathered thus far is undesirable. If such an application fails, it is subsequently re-started (automatically or manually). Upon a restart this application re-reads its state from the non-volatile storage, and continues its operation from the las consistent point that it exposed to the outside world. In HAMSA, we preserve this single object image of the application, but at the same time we shorten the service outage time (*i.e.*, impossibility to continue the monitoring process) by consistently replicating the application, so that the monitoring continues even if less than $2 \cdot f$ replicas exist in a given network partition. The most widely used standard framework for managing IP networks is Simple Network Management Protocol (SNMP) [Sta99]. It defines both the information model and the management protocol allowing monitoring and modifying of the managed objects. SNMP is based on a two-tier thick client/thin server architecture. With some exceptions, when SNMP is used for monitoring, a centralized platform called *manager* performs periodic *polling* of the remote objects. SNMP is an *in-band* management framework. This means that to function, the network management system uses the same network resources that it manages, *e.g.*, bandwidth.

Appendix D

HAMSA Recovery Algorithms

In this appendix we present a brief description of HAMSA procedures involved in handling host and network failures.

D.1 Recovery of a Failed Component

When a component host becomes unavailable either due to a network split, or to a machine failure, the MLMs that remain connected and the HA-MLM controller receive a membership change notification message from Transis. They advance epoch number and assign a new identifier to their state objects that reflects this.

The controller locally inquires its host MLM about the components that have been active at the failed MLM. Then, according to its load balancing algorithm the controller assigns the failed components to the remaining MLMs, and informs them through a single Transis multicast.

Thanks to the HAMSA component delegation protocol, and active replication of the components' state, the newly assigned MLMs can immediately activate their local dormant replicas of the failed components from the last known consistent state.

The message cost is harder to assess for this type of HAMSA activity. Upon each failure of an MLM, all the HA-MLMs that have this MLM as a member should go through the protocol above. The cost of state replication for a component is highly dependent on that component's nature.

D.2 Merging Previously Split HA-MLM

When a previously split HA-MLM re-merges, a single replica of each component remains operational and other replicas return to the dormant state. To ensure consistency, it is necessary to merge the states of the replicas.

In a simple case, the re-merging replicas worked with disjoint sets of the target devices being available in their respective network partitions. In this case per-device state is simply merged into a single collection.

In a pathological case component replicas could work with the same set of the target devices producing conflicting per-target device states. There is no conceivable way to merge these conflicting states automatically, since HAMSA does not know their semantics. Thus, in case of conflicts, HAMSA prefers the states that correspond to the replica that has completed the higher number of external interactions.

The communications cost involved in merging the split MLMs depends on the size of the state, the number of delegated components, and the number of simultaneously merging partitions.

D.3 Garbage Collection of Messages

As explained above, each MLM in a HA-MLM has all components delegated to it. The messages that are being sent and received by these components are being replicated at all MLMs in the HA-MLM, but since there is only one active replica of the component per network partition at any given time, only this replica will actually process the messages.

This raises the question of garbage collection of the messages at the MLMs, at which the component replicas are dormant. Current implementation of HAMSA defines two message service synchronization policies.

One policy never ages messages, and sets a *watermark* as a percentage of a total local mailbox capacity. Reaching the watermark triggers a garbage collection protocol within a HA-MLM. For example, if there are two MLMs in a single HA-MLM, one having an active component replica, and another one having a dormant one, the messages being replicated in the mailbox of the second MLM for the dormant component will accumulate (because they are not consumed by the dormant component). When the watermark of the mailbox is reached at the second MLM, it will ask the first MLM about the status of the not consumed messages with the specific numbers. When it discovers that those messages are already consumed by the active replica, the MLM evicts them from its local mailbox.

Another, simpler but less safe, policy currently supported by HAMSA is assigning a maximal time-to-live timeout to every message produced by a component. This will cause eventual garbage collection of the unconsumed messages.