Low-Cost Itineraries for Multi-Hop Agents Designed for Scalable Monitoring of Multiple Subnets

Damianos Gavalas[*], Christina (Tanya) Politi[†]

[*] Department of Cultural Technology and Communication,
University of the Aegean
Arionos & Sapfous St., Mytilini, Lesvos Island, Greece
E-mail: dgavalas@aegean.gr

[†] School of Electrical and Computer Engineering,
National Technical University of Athens,
9 Heroon Polytechniou Street, Zographou, 15773, Athens, Greece
E-mail: tpoliti@telecom.ntua.gr

Abstract - Mobile agent (MA) technology represents a recent trend for implementing distributed management architectures, as an answer to the flexibility and scalability problems of centralised models. Management scalability limitations though, are not adequately addressed when monitoring tasks requiring the employment of multi-hop agents are considered. This is because agent-based architectures lack mechanisms that guarantee near-optimal agents itineraries so as to minimise the total migration cost in terms of the round-trip latency and the incurred traffic. This is of particular importance when the management of networks spanning multiple subnets is involved. To address these issues, we have adapted an algorithm originally designed to solve network design problems to the specific problematic of MA itinerary planning. The algorithm suggests the optimal number of MAs that minimise the overall cost and also constructs optimal itineraries for each of them. A Java-based implementation of the algorithm has been tested on realistic applications over a laboratory testbed, demonstrating significant cost savings. Simulation tests verified the algorithm's validity and competence over large enterprise networks.

1. Introduction

Mobile Agents (MA) [21], defined as autonomous programs with the ability of moving from host to host and acting on behalf of users towards the completion of a given task, attract increasing attention within the distributed computing field. One of the most popular topics in MA research community has been distributed Network and Systems Management (NSM).

Traditionally, NSM systems rely on centralised, client/server approaches wherein the functionality of both clients (managers) and distributed servers (management agents) is defined at design time. This centralised model is exemplified in the IETF Simple Network Management Protocol (SNMP) [25]. The centralised paradigm is known to exhibit severe scalability problems as it involves massive transfers of management data, which cause considerable strain on network throughput and processing bottlenecks at the manager host. Moreover, the system is highly dependent on the central management station. If the latter goes offline or a key network link fails, the system is no longer functional.

These problems have motivated a trend towards distributed management intelligence that represents a rational approach to overcome the limitations of centralised NSM [6]. A new trend in NSM involves using MAs to manage distributed network systems [5][17][24][27]. An MA can be used to locally retrieve and filter management data to monitor systems health and networking conditions in distributed environments. In particular, management tasks are assigned to an agent, which delegates and executes management logic in a distributed and autonomous fashion. After completing these tasks, the results are either communicated through a messaging mechanism or carried back to the manager by the MA.

Delegation of management logic may be realized with agents bound to *single-hop* mobility: the agents move from the managing node to remote managed nodes, where they statically complete their tasks [3][4]. What is not commonly exploited in management is the MA *multiple-hop* capability, where agents may move several times as they adapt to the changing circumstances. While single-hop mobility can improve flexibility and scalability in the context of relatively static networked systems, it is the multiple-hop capability offered by MAs that needs to be exploited to meet the requirements of future networked systems, i.e. large scale and dynamics [17]. In addition, network monitoring based on multi-hop (itinerant) MAs is advantageous for short-term monitoring tasks and also in cases that a global (domain)-level rather than a local (device)-level view of managed resources is required. For a complete discussion of these issues, the reader may refer to [7] and [18].

However, while in single-hop mobility, agent itinerary control is straightforward (the itinerary is restricted to the single destination host), this is not the case in multi-hop mobility, where slight variations on the set of visited hosts or even on the order that a specific set of nodes is visited may result in dramatic changes of the overall trip latency and migration traffic. In this article, we focus on multi-hop mobility, aiming at devising methods to optimize MA itineraries.

On that direction, we introduce an algorithm that addresses the issue of MA optimal itinerary planning. Our proposed algorithm determines the optimal number of MAs and their corresponding itineraries, which may be useful in a variety of distributed applications. The main motivation for designing these optimal agent itineraries is to minimize the overall cost (network overhead) associated with MA transfers and also to maintain low latency for completing their task.

The remainder of the paper is organised as follows: Section 2 explains the importance of optimal agent itinerary planning in management applications and Section 3 reviews works relevant to the research presented herein. Section 4 discusses the background, design and functionality of an algorithm for optimal itinerary planning. A qualitative evaluation of our proposed algorithm against an alternative approach is given in Section 5. Experimental results on real and simulated environments are presented in Section 6 and Section 7, respectively, while Section 8 concludes the paper.

2. Importance of Itinerary Planning in Agent-Based Monitoring

The main objective in MA-based distributed computing is to minimize the volume of network traffic exchanged between distributed systems while maintaining relatively low task execution time, especially for time-critical tasks [6]. Despite the potential of agent mobility in distributed applications, inappropriate use of MAs may lead to a highly inefficient design. In network monitoring applications for instance, using a single MA object that sequentially visits all managed devices, regardless of the underlying topology (see Figure 1a) may actually lead to performance worse than the conventional SNMP-based approach. The performance further declines, when the monitoring MA collects monitoring data from multiple subnets, often interconnected by low-bandwidth links. In such cases, the traffic associated with management tasks typically traverses several network segments and, when summed up, results in increased bandwidth waste [24].

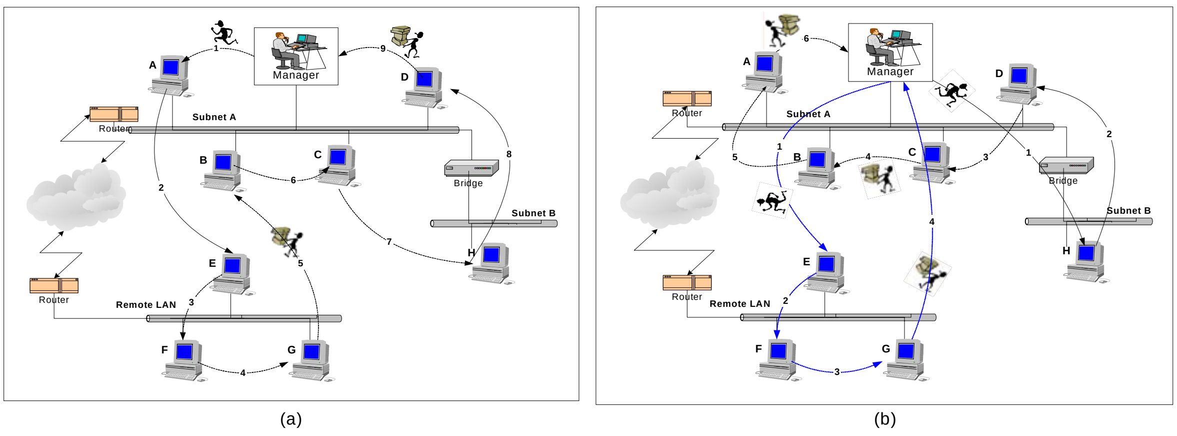


Figure 1. (a) 'Flat' MA-based monitoring; (b) Optimized partitioning of the network into two management domains

This inefficient approach, known as 'flat' MA-based monitoring [7], presents serious scalability problems: First, in large networks the *round-trip delay* of the MA greatly increases as the overall travel time depends on the number of hops realized by the MA. Second, the *network overhead* imposed by the MA transfers grows exponentially with the network size [7][24]; the slope of the overhead curve becomes steeper in the case of high selectivity values (selectivity $0 \leq \sigma \leq 1$ is a metric defined as the proportion of data *maintained* to that *retrieved* from each host).

A rational approach to overcome such scalability problems is to partition the managed network into several logical/physical domains. For instance, in Figure 1b, an MA object polls the devices of the remote LAN, whereas a second MA is assigned to the subnet local to the manager host as well as to another subnet, which is part of the same LAN. Through management traffic localization, unnecessary usage of expensive networking resources is restricted, thereby improving management scalability. The partitioning criteria could be the number of nodes assigned to each MA, the physical distribution of polled devices, or a combination of the previous criteria (see [7]).

However, the scenario illustrated in Figure 1 represents an ideal case in terms of the WAN link utilisation. That is because the link is traversed only twice per polling interval. A slightly different partitioning scheme or alteration on MAs itinerary would significantly increase total migration cost. Apparently, even when specific partitioning criteria are followed, the design of MA itineraries is almost random. Namely, itineraries scheduling process lacks a mechanism that would guarantee minimal use of links interconnecting individual management domains, hence an algorithm for itinerary planning is required. In Section 4, we describe a Heuristic algorithm for Itinerary Planning (HIP). The main objective of itinerary planning is to optimize the use of network resources, i.e. to minimize the cost associated with MA transfers; this optimization should not be achieved though at the expense of large MA round-trip delays, hence maintaining relatively low task execution times is a parallel objective.

3. Related Work

The problem of optimizing the itineraries of multi-hop MAs has not been sufficiently addressed in the literature. A first attempt to address this issue has been reported in [13]: Iqbal et al. developed a performance model that, given a specific communication pattern, allows agents to decide whether they should migrate to a site and communicate locally or the communication should be performed remotely. The decision is taken according to an 'optimal design graph'; in most cases, it has been indicated that the optimal performance of an agent is achieved by a critical sequence of mixed remote procedure calls and agent migrations. The same approach has been followed in [23], in the context of network and system management applications.

Rubinstein et al. [24] evaluated the scalability of MA-based management on large enterprise networks and compared the performance of this approach against that of centralized management paradigm. Recognizing the fact that "MA size increases with the number of visited nodes and, as a consequence, migration becomes difficult", they proposed a strategy in which the MA returns to the management station to deliver its collected data, thereby reducing its size before visiting the remaining hosts. Their simulation results indicated that for given network topologies there exists an optimum number of hosts that the MA should visit before returning to the management host to 'unload' its collected data, that minimizes the overall MA trip response time and cost (in terms of bandwidth usage). However, the possibility of using multiple MAs to perform management tasks is not investigated, nor is the issue of designing efficient agent itineraries.

A work conceptually relevant to the research presented herein has been presented in [22], where Qi and Wang propose the employment of MA paradigm in wireless sensor networks. To optimize agents itinerary, they derived a Local Closest First (LCF) algorithm according to which each MA searches for the next destination with the shortest distance to its current location. However, their cost function formulation does not take into account potential partitioning of visited hosts in multiple clusters, which would affect the calculation of the distance matrix. Also, LCF-like algorithms have been characterized as 'nearsighted', in the sense that their output highly depends on the MAs original location, while the nodes left to be visited last are associated with high migration cost [16]; the reason for this is that they search for the next destination among the nodes adjacent to the MA's current location, instead of looking at the 'global' network distance matrix.

Most importantly though, both the works presented in [13] and [22], deal with the problem of constructing near-optimal MA itineraries for given sets of network nodes, where a *single* MA visits the whole set. Also, they do not address the fundamental problem of partitioning network nodes in optimal clusters. On the other hand, the algorithm presented in the following section deals with the optimal clustering problem and subsequently uses the algorithm's output to construct near-optimal agent itineraries.

4. A Heuristic algorithm for Itinerary Planning (HIP)

Interestingly, the problem of designing optimised itineraries exhibits many similarities with the Multi-point Line Topologies or Constrained Minimum Spanning Trees (CMST) problems. A CMST is a Minimum Spanning Tree[1] with the additional constraint on the size of the subtrees rooted on the 'center' (there is an upper limit on the number of nodes included on each of the subtrees originated at the tree's root). CMST algorithms are used in graph theory, with the main application field being network design problems [16]. In such problems, the objective is the optimal selection of the links connecting terminals to concentrators (multiplexors) or directly to the network center, resulting in the minimum possible total cost. The output of CMST algorithms typically comprises topologies partitioned on several multi-point lines (or tree branches), where groups of terminals share a subtree to a specific node (center). For instance, Figure 2a depicts a set of nodes with a given network center and costs for connecting individual pairs of nodes, and Figure 2b presents the optimal multi-line topology that minimises the overall cost, where network nodes have been partitioned into two clusters or subtrees, each directed to the network center. In this particular scenario, the overall cost will comprise the sum of costs for connecting each link included into the problem solution:

$$c_{total} = c_{3,1} + c_{1,0} + c_{2,4} + c_{5,4} + c_{6,4} + c_{4,0} = 36 \qquad (4\text{-}1)$$

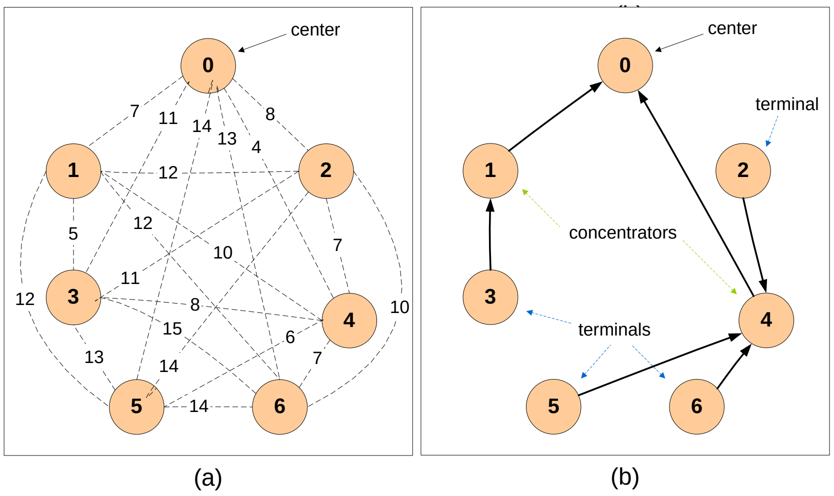


Figure 2. CMST problem: (a) The unconnected graph; (b) The optimal multi-point line topology (constrained minimum spanning tree)

Substituting the terms 'network center', 'link' and 'multi-point-line' with the terms 'manager station', 'migration' and 'itinerary' respectively, and following the observation that the output of CMST algorithms (group of multi-point lines rooted at the center) very much resembles a group of itineraries all originated at the manager station, the similarity of CMST and MA itinerary planning problems becomes evident. As a result, the idea of using algorithms originally devised for CMST problems in the application area of MA itinerary planning, naturally shapes up. It is noted that other tree structures, such as *Steiner* trees [12] (known to provide the shortest overall edge cost) could possibly be considered as alternatives of CMSTs for MA itinerary planning; however, we have chosen

[1] A Minimum Spanning Tree is defined as a tree (i.e. a connected graph without cycles) with the least total distance, cost, or some other metric of delay or reliability [16].

CMSTs (along with algorithms that deal with CMST problems), as they are more suitable for modeling MA itineraries [2].

CMST problems are NP-hard and as a result several heuristics have been proposed to efficiently deal with them; *Esau-Williams* (E-W) and *Sharma* are two popular algorithms that deal with such problems [16]. Our HIP (Heuristic for Itinerary Planning) algorithm adapts some basic principles of E-W algorithm in the specific requirements of itinerary planning problems. It is noted that some preliminary ideas and experimental results related to HIP algorithm have been presented in [10].

4.1. REQUIREMENTS OF ITINERARY PLANNING PROBLEM

The simple cost function of (4-1) fails to address the requirements of agent itinerary design problem since it considers the cost of link utilization as the only contributing factor to the total itinerary cost c_{total}. A key factor also affecting c_{total} is the agent size; more importantly, the agent size increment rate [6][24], which depends on the amount of data collected by the MA on every host. Let us assume that a set of itineraries $I = \{I_1, I_2, \ldots, I_n\}$ is constructed, each assigned to an individual MA object. Each itinerary I_i includes a sequence of hosts to be visited by its respective MA: $I_i = N_0, N_1, .. , N_n, N_0$. Note that all itineraries originate and terminate at the manager station host N_0.

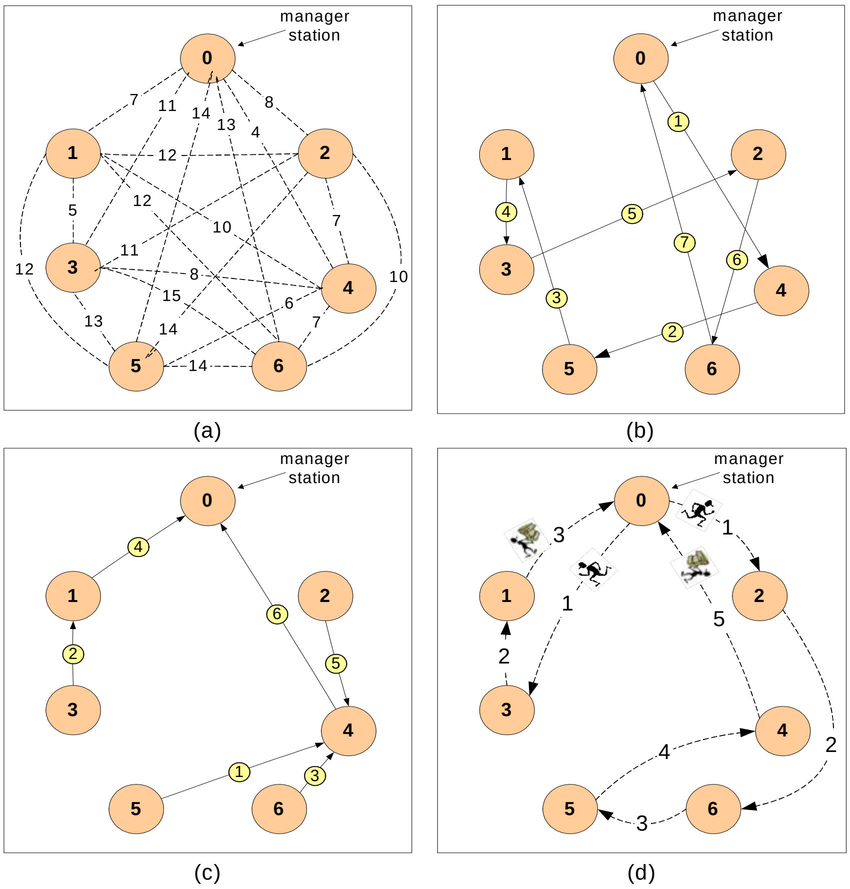


Figure 3. The MA itinerary planning problem: (a) The original network graph; (b) The output of LCF algorithm; (c) The output of HIP algorithm; (d) Two MA itineraries derived from the output of HIP algorithm

The total cost per polling interval over all itineraries $|I|$ becomes:

[2] The Steiner tree problem is remarkably similar to the minimum spanning tree problem: the objective is to interconnect a set *V* of points (vertices) by a network (graph) of shortest length. Unlike the minimum spanning tree problem though, new vertices (*Steiner vertices*) can be added to the network to reduce its length [12]. However, since in MA itinery planning, graph vertices correspond to nodes able of receiving MAs (similarly, graph edges correspond to physical links), those additional vertices cannot be physically projected. Hence, Steiner trees are not suitable for modeling the itinerary planning problem.

$$C_{total} = \sum_{i=1}^{|I|} \sum_{j=0}^{H(I_i)-1} (d_{ij} + s) * c_{ij} \qquad (4\text{-}2)$$

where $H(I_i)$ denotes the number of hosts included in itinerary sequence I_i, d_{ij} is the amount of data collected by the MA performing itinerary i on the first j visited hosts, s the MA initial size and c_{ij} the cost of utilizing the link traversed by the MA on its j^{th} hop, i.e. the link connecting hosts N_j and N_{j+1} (c_{ij} is given by the network cost matrix). In principle, HIP algorithm aims at constructing a set of itineraries I minimizing the cost function of equation (4-2).

In order to provide our HIP algorithm a fair performance metric we have also implemented the LCF algorithm, as described in [22] (LCF implementation details may be found in the Appendix). For instance, for the network of Figure 3a with cost matrix detailed in Table 1, the itinerary constructed by LCF is shown in Figure 3b. The sequence numbers enclosed within circles indicate the order in which individual links (or migrations) become accepted in the corresponding algorithm steps.

Although the entries of the cost matrix presented in Table 1 are random, when testing the efficiency of HIP and LCF algorithms in real or simulated environments cost matrices are constructed so as to reflect the cost of using the underlying networking infrastructure[3], e.g. we accept the fact that it is 'cheaper' for an MA to migrate within a high-speed LAN than over a low-bandwidth WAN link or a wireless connection.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0		7	8	11	4	14	13
1			12	5	10	12	12
2				11	7	14	10
3					8	13	15
4						6	7
5							14
6							

Table 1. Cost matrix of the network shown in Figure 3a

4.2. HIP ALGORITHM IMPLEMENTATION AND EXECUTION

HIP algorithm takes into account the amount of data accumulated by MAs at each visited host (without loss of generality, we assume this is a constant d), a parameter ignored by LCF algorithm. Namely, it recognizes that traveling MAs become 'heavier' (accumulate data multiple of d) while visiting managed devices without returning back to the manager site to 'unload' their collected data [24]. Therefore, HIP restricts the number of migrations performed by individual MAs, thereby promoting the parallel employment of multiple cooperating MAs, each visiting a subset of managed devices. The assumption of constant data accumulation d from each host affects the general formula (4-2), which becomes:

$$C_{total} = \sum_{i=1}^{|I'|} \sum_{j=0}^{H(I_i)-1} (d_j + s) * c_{ij} \qquad (4\text{-}3)$$

Specifically, the aim of HIP algorithm is, given a set of hosts $N = \{N_0, N_1, \ldots, N_{n-1}\}$, the manager station N_0 and the cost matrix C, to return a set of near-optimal itineraries $I = \{I_0, .., I_k\}$, all originated and terminated at the manager station. Initially, we assume $|N|$ (= n) itineraries, as many as the

[3] For hosts i and j located in subnets S_i and S_j respectively, the cost $c_{i,j}$ of an MA migration from i to j depends on the actual 'distance' between S_i and S_j, i.e. the number of intermediate subnets that need to be traversed, the bandwidth of their interconnecting links, etc. In general, for $S_i \equiv S_j$ we set $c_{i,j} = 0$, while for $S_i \neq S_j$ $c_{i,j} > 0$ (proportional to the inverse of the bandwidth of links interconnecting S_i and S_j).

network nodes: I_0, .., I_{n-1} , each containing a single host (N_0, N_1, …, N_{n-1}, respectively). On each algorithm step, two hosts i and j are 'connected' and, as a result, the itineraries including these hosts ($I(i)$ and $I(j)$ respectively) are merged into a single itinerary.

As mentioned in Section 3, LCF-like algorithms usually fail as they tend to leave hosts located far from the center stranded since they prioritize the inclusion of hosts closed to last selected host. As a result, relatively expensive links are left last to be included in the solution, significantly increasing the overall cost. A way of dealing with this problem is to pay more attention to nodes far from the center, giving preference to links incident upon them. HIP algorithm accomplishes this by borrowing and extending the concept of 'tradeoff function'[4] $t_{i,j}$ associated with each link (i, j), defined by:

$$t_{i,j} = c_{i,j} + (d * H(I(i)) + H(I(j))) - C_{i,N_0} \tag{4-4}$$

where C_{i,N_0} is the cost of connecting $I(i)$ to the manager station N_0. Initially, this is simply the cost of connecting node i directly to the manager station. As i becomes part of an itinerary containing other nodes, however, this changes to:

$$C_{i,N_0} = \min_{k \in I(i)} c_{k,C} \tag{4-5}$$

Equation (4-4) implies that the more hosts an itinerary already includes, the more difficult for a new host to become part of that itinerary, especially when the amount of data collected from each host (d) is large. Figure 4 lists a pseudo-code implementation of HIP algorithm.

```
1  HIP (n, c, d, N_0)  // n: Total number of hosts, c: cost matrix, d: data collected per host, N_0: origin station
2      initialize I   // I: the list of itineraries to be constructed
3      current = N_0
4      N_ connected = 0  // N_connected: the number of hosts already included into an itinerary
5      while (N_ connected < n)
           /*  I(i) is the sequence of hosts (itinerary) where host i has already been included
           and H(I(i)) is the number of hosts already included within I(i) */
6          compute $t_{i,j} = c_{i,j} + (d * H(I(i)) + H(I(j))) - C_{i,N_0}$, where $I^s(i) \cap I^s(j) = \varnothing$ and
               $C_{i,N_0} = \min_{k \in I(i)} c_{k,N_0}$ // $I^s(i)$ denotes the set corresponding to itinerary sequence I(i)
7          merge (I(i), I(j)), for (i, j) minimizing the tradeoff function ( $\min_{i,j} t_{i,j}$ )
8          N_ connected ++
9      return I
```

Figure 4. Pseudocode implementation of HIP algorithm

HIP algorithm execution steps for the test network graph of Figure 3a are demonstrated in Figure 5, where the links (agent migrations) selected are highlighted; we assume that the amount of data collected per host is $d = 1$. On every algorithm step, a pair (i, j) minimizing $t_{i,j}$ is selected and, following that, the itineraries containing hosts i and j are merged into a single itinerary. This process is repeated until a set if itineraries including *all* hosts is constructed. Figure 5 presents the values of $t_{i,j}$ for pairs (i, j) minimizing the tradeoff function for each host i ($\min_j t_{i,j}$); the pair (i, j) that minimizes

[4] The concept of the tradeoff function is introduced in E-W algorithm, defined as follows: $t_{i,j} = c_{i,j} - c_{i,N_0}$. Equation (4-4) extends and adapts this function in the specific requirements of agent itinerary planning problem. In particular, the inclusion of a parameter representing the amount of data collected from each host (d) and also the number of hosts already included in the itineraries considered for merging, i.e. $I(i)$ and $I(j)$, obstructs the construction of large itineraries, thereby promoting the formation of multiple itineraries, assigned to separate MAs.

$t_{i,j}$ over all hosts ($\min\limits_{i,j} t_{i,j}$) is then selected. For instance, on step one, the pair minimizing $t_{i,j}$ is (i, j) = (5, 4), hence itineraries including hosts 5 and 4 are merged forming: $I^s(5) \cup I^s(4) = \{5,4\}$. On next step, $t_{i,j}$ values are re-calculated, for instance, $t_{2,4} = c_{2,4} + (1 * H(I(2)) + H(I(4))) - C_{2,N_0}$ = 7+(1+2)-8 = 2. Note that the elements of the itinerary set including host 4 have increased: $I^s(4) = \{5, 4\} \Rightarrow H(I(4))=2$. At the end of step 6, two itinerary sequences are constructed, forming two subtrees rooted at the manager host: 6, 5, 4, 2 and 3, 1 (see Figure 3c). It is then a trivial task to form the itinerary plan of the two MAs: I_1 = 0, 2, 6, 5, 4, 0 and I_2 = 0, 3, 1, 0 (see Figure 3d). These itineraries correspond to a post-order traversal[5] of the two subtrees. Note that itinerary 0, 2, 6, 5, 4, 0 is chosen amongst alternative post-order traversals (e.g. 0, 5, 6, 2, 4, 0 or 0, 6, 5, 2, 4, 0) as it provides the most cost-efficient solution; for instance, C_{056240} = (14 + 14 + 10 + 7 + 4) × d = 49 × d, whereas C_{026540} = (8 + 10 + 14 + 6 + 4) × d = 42 × d.

Step 1	**Step 2**	**Step 3**
t_{13} = 5+(1+1)-7 = 0	t_{13} = 0	t_{10} = 7+(2+1)-7 = 3
t_{24} = 7+(1+1)-8 = 1	t_{24} = 7+(1+2)-8 = 2	t_{24} = 2
t_{31} = 5+(1+1)-11 = -4	t_{31} = -4	t_{34} = 8+(2+2)-7 = 5
t_{40} = 4+(1+1)-4 = 2	t_{40} = 4+(2+1)-4 = 3	t_{40} = 3
t_{54} = 6+(1+1)-14 = -6	t_{51} = 12+(2+1)-4 = 11	t_{51} = 12+(2+2)-4 = 12
t_{64} = 7+(1+1)-13= -4	t_{64} = 7+(1+2)-13= -3	t_{64} = -3
Step 4	**Step 5**	**Step 6**
t_{10} = 3	t_{14} = 10+(3+3)-0 = 16	t_{14} = 10+(3+4)-0 = 17
t_{24} = 7+(1+3)-8 = 3	t_{24} = 3	t_{20} = 8+(4+3)-4 = 11
t_{34} = 8+(2+3)-7 = 6	t_{34} = 8+(3+3)-0 = 14	t_{34} = 8+(3+4)-0 = 15
t_{40} = 4+(3+1)-4 = 4	t_{40} = 4+(3+3)-4 = 6	t_{40} = 4+(4+3)-4 = 7
t_{51} = 12+(3+2)-4 = 13	t_{51} = 12+(3+3)-4 = 14	t_{51} = 12+(4+3)-4 = 15
t_{62} = 10+(3+1)-4= 10	t_{62} = 10	t_{61} = 12+(4+3)-4 = 15

Figure 5. HIP algorithm execution steps for the network of Figure 3a

5. Qualitative Evaluation of HIP and LCF Algorithms

The total costs associated with LCF and HIP proposed solutions (shown in Figure 3(b) and Figure 3(d), respectively) are calculated using the generic cost function of equation (4-3):

$$C_{LCF} = s * c_{0,4} + (s+d) * c_{4,5} + (s+2d) * c_{5,1} + (s+3d) * c_{1,3} + (s+4d) * c_{3,2} + \\ + (s+5d) * c_{2,6} + (s+6d) * c_{6,0}$$

[5] Post-order traversal (visit the left subtree, then the right subtree, then the root) is more efficient than pre-order (visit the root, then the left subtree, then the right subtree) or in-order (visit left subtree, then the root, then the right subtree) traversal, as it ensures minimal usage of inter-connecting links. If, for instance, the manager host is located on subnet A and an MA object is assigned devices spread among subnets A, B and C, it will first visit all devices of B, then all devices of subnet C (or vice-versa), leaving the devices of subnet A to be visited at the end of the MA's itinerary before returning to the manager host. That is, when crossing the link A→B, the MA has not yet collected any data, hence, it will have the minimum possible impact on network resources. The decision regarding which subnet (B or C) will be visited first is based on the overall cost of itineraries B→C→A and C→B→A. In particular, if C_{BC}, C_{CA}, C_{CB} and C_{BA} denote the cost of links B→C, C→A, C→B and B→A respectively and H(A), H(B) and H(C) denote the number of elements hosted in subnets A, B and C respectively, then the itinerary with smaller overall cost is chosen: $\min\{C_{BCA}, C_{CBA}\} = \min\{C_{BC}*H(B)*d + C_{CA}*H(C)*d, C_{CB}*H(C)*d + C_{BA}*H(B)*d\}$.

$$C_{\mathrm{HIP}} = \underbrace{[s*c_{0,3} + (s+d)*c_{3,1} + (s+2d)*c_{1,0}]}_{\text{cost of first MA itinerary}} +$$

$$\underbrace{[s*c_{0,5} + (s+d)*c_{5,6} + (s+2d)*c_{6,2} + (s+3d)*c_{2,4} + (s+4d)*c_{4,0}]}_{\text{cost of second MA itinerary}}$$

Assuming an MA of initial size s=1000 bytes that collects an amount of d=100 bytes from each node visited and after substituting various costs with the corresponding values found in the cost matrix of Table 1, we get: C_{LCF} = 90,500 and C_{HIP} = 81,000 cost units, which corresponds to cost saving of 10.5% when employing the HIP instead of LCF algorithm. However, it is clear that as the s/d ratio decreases (the MA accumulates a larger amount of data), HIP algorithm performance gain improves further. For instance, for s=700 bytes and d=500 bytes, the resulting cost saving of HIP over LCF in this scenario becomes 41.97%.

Figure 6 demonstrates how the overall cost of LCF and HIP algorithms scale as a function of the amount of collected data (for s = 1000 bytes); the cost of using single agents with randomly selected itineraries is also presented (the 'Random' curve represents the average cost among 5 different random itineraries). It is noted that when d = 5000 bytes, HIP suggests the parallel employment of 3 MAs, rather than two (four MAs for d = 10,000), since MA size growth rate is such that multi-hop itineraries become too costly.

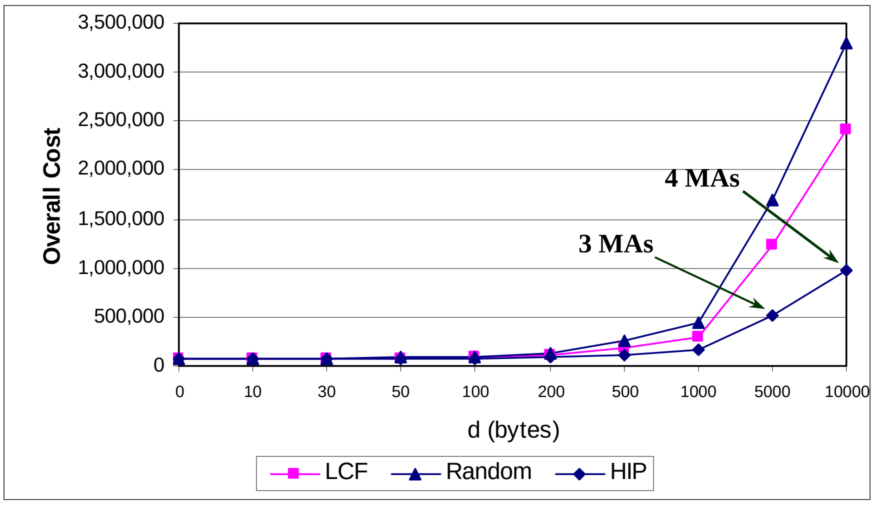


Figure 6. Scaling of LCF and HIP overall costs as a function of collected data per host, for s = 1000 bytes

6. Experimental Evaluation

It should be emphasized that HIP algorithm is platform-independent, i.e. it has not been designed having a particular mobile agent platform (MAP) in mind. In fact it can easily be integrated in any available MAP (e.g. Aglets [2] or JADE [14]), constructing and supplying near-optimal MA itineraries. To prove HIP algorithm validity and effectiveness, we have implemented and incorporated the algorithm as an add-in module, termed the *Itinerary Scheduler Module* (ISM), into our MAP research prototype presented in [8].

ISM has been implemented in Java programming language (JDK 1.4.1 [15]); ISM executes the HIP algorithm and informs the manager application on the number of MAs that need to be instantiated and their respective itineraries. Agent itineraries are reconstructed whenever a new managed device is 'discovered' (upon such event, ISM is notified through a callback method). LCF algorithm has also been implemented for comparison purposes.

HIP has only been tested on realistic network monitoring application scenarios; yet, it suits any application field which benefits from distribution of intelligence and processing overhead offered by MA paradigm. The experimental testbed includes several Windows NT (Pentium III 450MHz, 256

MB RAM) and Linux Red Hat v. 6.1 (Pentium MMX 233 MHz, 128 MB RAM) stations. The managed network comprises two 10Mbps LANs connected through a 1Mbps leased line (see Figure 7); the first LAN (where the manager application executes) hosts 5 managed elements while the second hosts a varying number (5 to 10) elements. The network traffic generated by MA migrations has been measured using the WinDump network analyser [29]. Network monitoring data are collected through interacting with SNMP agents hosted by managed elements; in particular, the standard SNMP service has been used in NT stations, while *snmpd* agents of UCD-SNMP package [28] have been installed on Linux stations.

The cost matrix used by HIP algorithm reflects the cost of using network resources (the bandwidth of LANs medium is ten times larger than the bandwidth of the leased line):

$$c_{i,j} = \begin{cases} 1, \text{when i and j are located on the same LAN} \\ 10, \text{when i and j are located on different LANs} \end{cases} \qquad (6\text{-}1)$$

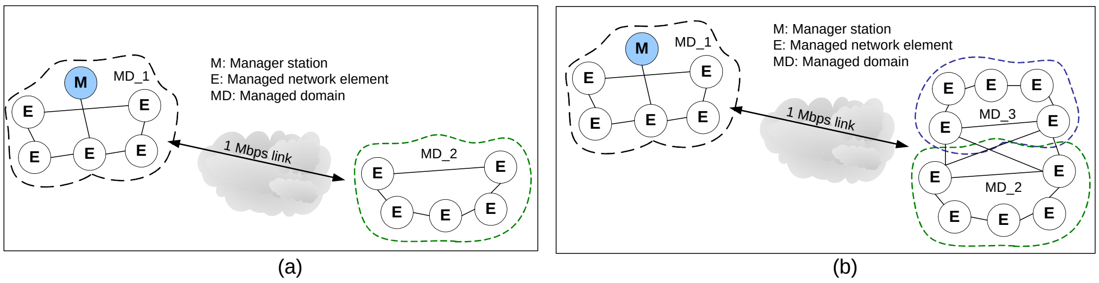


Figure 7. HIP execution on the experimental testbed. (a) The remote subnet hosts 5 managed devices all included in a single managed domain; (b) The remote subnet hosts 10 managed devices separated in two managed domains

The performance of HIP and LCF algorithms has been compared against an implementation of SNMP in Java [1]. The application scenario involves polling managed elements for the contents of `tcpConnTable` MIB-II[6] table, which lists information about all TCP connections of a host. In the SNMP-based implementation, individual MIB tables are remotely retrieved through exchanging request/response messages, in particular by issuing successive `get-next` requests (each retrieving a table row). Every tcpConnTable row contains *5* values (columns) [19], while managed elements stored information about *46* TCP connections on average, i.e. included *46* table rows (46 × 5 values in total).

Parameter	Value
MA initial size	1,08 KB
Average SNMP request/response packet size (including UDP header)	90 bytes
Average SNMP packet increment for each additional requested value	17 bytes
MA state size increment per table sample (compressed data)	98 bytes
Average SNMP-based table retrieval response time	102.1 msec
Average MA-based table retrieval response time	71 msec

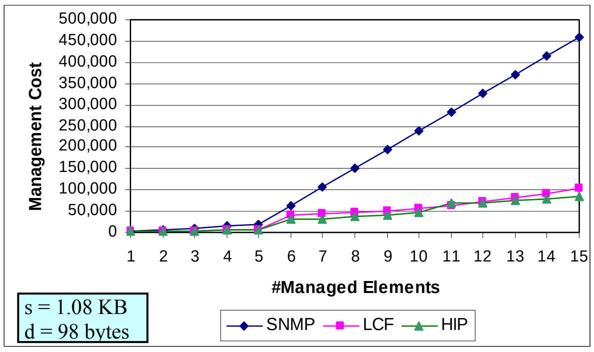
Table 2. Network overhead and time parameters

In MA-based approaches, travelling MAs obtain table values through local interaction with legacy systems; table contents are then compressed (using Java *gzip* facility) and encapsulated within MAs state. In contrast with LCF algorithm, HIP involves the parallel execution of multiple MA

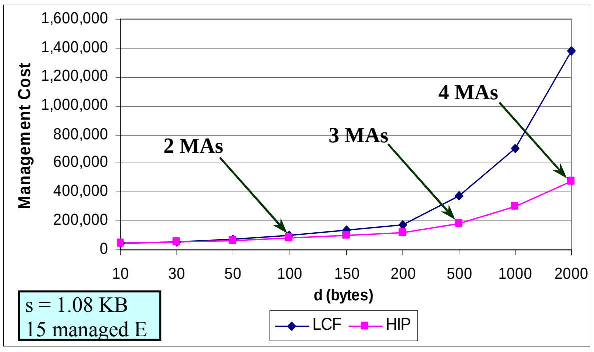
6 A management information base (MIB) is a formal description of the set of network objects that can be managed using SNMP. MIB-II [19] is the standard MIB in the IP world, supported by virtually all SNMP-compliant network devices.

objects which depends on network size. For instance, when *5* managed elements are hosted in the remote subnet, two managed domains are created, each assigned to an individual MA (Figure 7a). However, when managed elements increase to *10*, HIP separates them into two distinct managed domain and engages an additional MA into polling operation (Figure 7b).

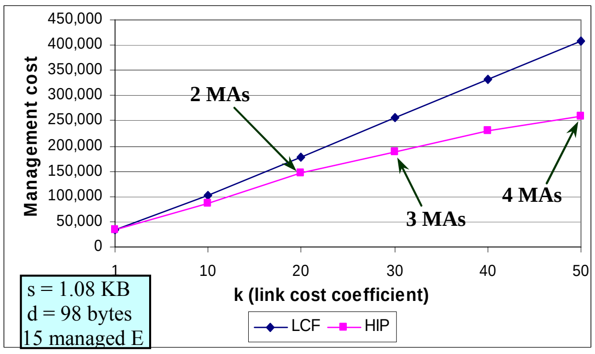
The network overhead and timing experiment parameters associated with our experiments are presented in Table 2. Network overhead parameters have been measured by WinDump tool. Experimental results are presented in Figure 8. The overall management cost has been calculated according to the cost function of equation (4-3), where cost matrix coefficients are given in equation (5-1). Figure 8a compares the management cost of SNMP against MA-based implementations, as a function of the number of polled devices, for the network monitoring application scenario described above. SNMP-based polling does not scale well as it involves heavy usage of the relatively expensive interconnection link, when increasing the number of polled devices located in the remote subnet (managed elements *6*-*15*). In contrast LCF and HIP algorithms require usage of the interconnecting link only when an MA migrates/returns to/from the remote subnet. HIP presents superior performance since the MA(s) assigned to the remote subnet managed domain(s) migrate through the low-bandwidth link with empty state; in the LCF-based solution, the unique MA first visits all the elements local to the manager host subnet and then migrates to the remote subnet (at migration time though, the MA has already collected an amount of data, therefore increasing network traffic). It should be emphasized though that for application scenarios involving collection of larger chunks of data or managed network topologies comprising large numbers of host, the performance gap of HIP against LCF is expected to grow (this is verified by the simulation results presented in Section 7).



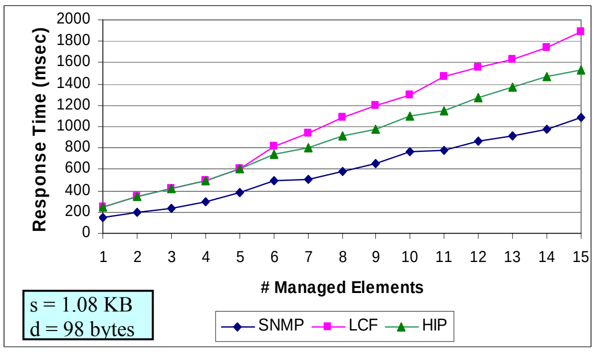
(a)



(b)



(c)



(d)

Figure 8. (a) Management cost of SNMP monitoring (per polling interval) against HIP and LCF implementations for varying managed network sizes; (b) Management cost of HIP vs. LCF as a function of the amount of collected data per managed element; (c) Management cost of HIP vs. LCF as a function of the relative cost of the interconnecting link of Figure 7; (d) Response time of SNMP monitoring against HIP and LCF implementations for varying network sizes

In Figure 8b, experiment parameters are adjusted to examine the effect of collected data amounts in the overall cost. The experiments have been performed on the test network of Figure 7b (15 managed elements). Apparently, HIP performance gain over LCF increases as the amount of data collected from each managed element increases. HIP algorithm separates remote subnet hosts into 2, 3 and 4 ‘virtual’ managed domains as data sample size increases to 100, 500 and 2000 bytes respectively. Figure 8c shows how management cost relates to the relative cost of the interconnecting link of Figure 7 (for instance, when the bandwidth of that link is 10 times smaller to the bandwidth of LANs medium, the cost coefficient is $k = 10$).

Regarding timing experiments, HIP performs better than LCF due to the parallel employment of MA objects when managed network size increases (see Figure 8d). Interestingly, the SNMP-based solution provided lower response time, which is attributed to the parallel request and collection of SNMP table snapshots. On the other hand, an invited side effect of MA-based table polling is the improved consistency of the acquired values due to the negligible time intervals between the retrievals of individual table rows [6][7].

It should be noted that the graphs of Figure 8 represent the worst-case scenario for MA-based implementations in terms of network overhead, as the agents store the entire SNMP table within their state without exploiting their ability of filtering management data. If, for instance, the network administrator is interested in acquiring only the table rows corresponding to established TCP connections (the column `tcpConnState` of `tcpConnTable` equals the value "established"), the amount of accumulated data may be significantly reduced. In addition, several MAP-related parameters may be adjusted to minimize agent migrations overhead and latency, as suggested in [9].

7. Performance Analysis by Simulation

HIP has also been evaluated in simulated network environments, using the *Network Simulator* (NS-2) tool [20], a discrete event simulator targeted at networking research. We have implemented SNMP and mobile agent modules, while the parameters presented in Table 2 have been incorporated into the implemented modules to enhance simulation results validity and accuracy.

Three different *transit-stub* topologies created by the topology generator GT-ITM [30] are used (GT-ITM output has been first converted to NS-2 format). The topologies comprise 272 nodes with interconnecting links of 2 Mbps bandwidth and latency of few milliseconds; 100 Mbps links are assumed within stub boundaries (see Figure 9). The management station controls 15 stub domains each hosting 16 network elements (240 managed elements in total). Two main simulations tests (network monitoring scenarios) have been performed and several measurements have been taken and evaluated.

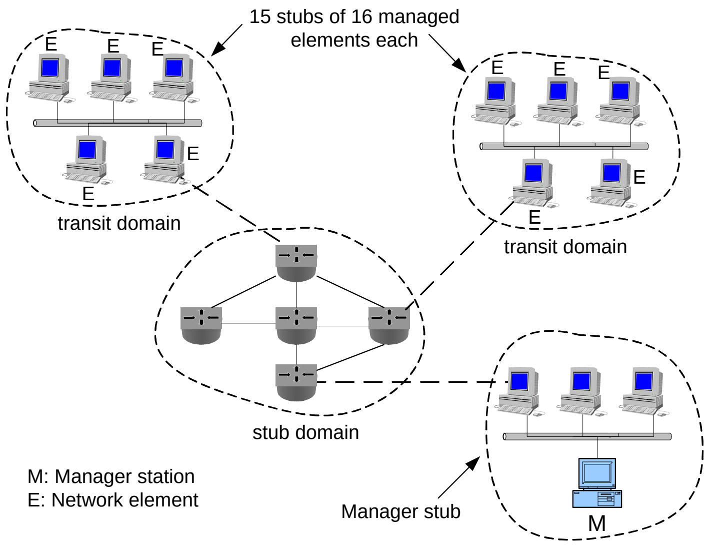


Figure 9. Network monitoring application on a transit-stub topology

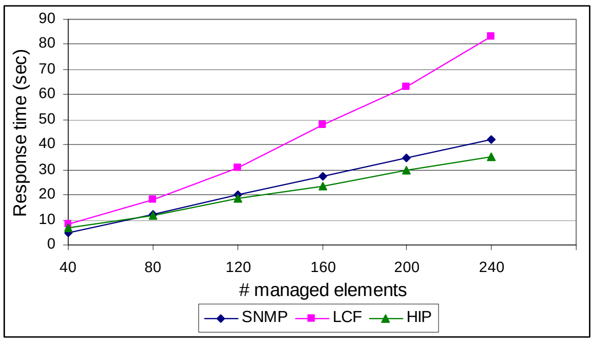
7.1. SIMULATION #1

The first test involves simulation of the same network monitoring application described in the preceding section. In the SNMP implementation, monitoring is performed in the following way: first, all elements of a stub are accessed; then, the next stub is managed until all the 16 stubs are accessed. In the LCF-based approach, a unique MA object visits sequentially all monitored devices, interacts with local SNMP agents and stores collected management data within its state. HIP-based monitoring involves the parallel execution of multiple MAs, each supplied with a 'near-optimal' itinerary; the number of monitoring MAs depends on the specific topological characteristics.

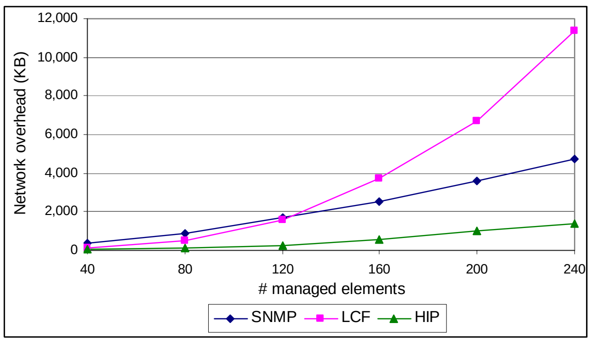
Simulation results are reported in Figure 10, where various curves represent the mean response time or bandwidth consumption for the three simulated topologies. Figure 10a presents mean response times for various managed network sizes. For a few sets of managed elements, SNMP performs better than HIP-based approach due to its lightweight nature and the fact that all the elements of a specific stub are polled in parallel. As the number of managed elements increases, SNMP response time grows proportionally since the time to manage a stub is approximately the same for all stubs. In the LCF approach, the response time increases faster when the number of managed elements grows due to the incremental size of the mobile agent which implies increased transmission delays. HIP presents better scalability as monitored elements are separated in 'virtual' managed domains, each managed by separate MA objects collecting management data in parallel.

Figure 10b shows that network overhead generated by HIP and SNMP implementations is approximately proportional to managed network size, with HIP achieving significantly better results. On the contrary the scalability of LCF algorithm is very poor (as the number of elements visited by the unique object increases, the amount of collected data becomes much larger than the size of MA code itself).

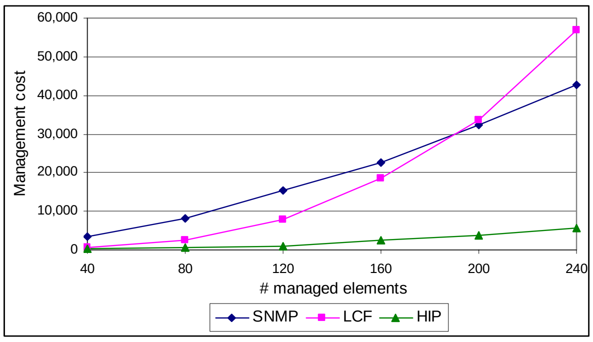
A comparison of the real management cost associated with monitoring operations (the cost of using low-bandwidth resources is taken into account) is given in Figure 10c where the performance gain of HIP-based approach is amplified. The drawbacks of the SNMP centralised approach becomes evident as it indiscriminately makes heavy use of relatively expensive network resources; as a result, the host limit where SNMP surpasses LCF approach increases, i.e. it provides better results for topologies comprising more than 200 hosts.



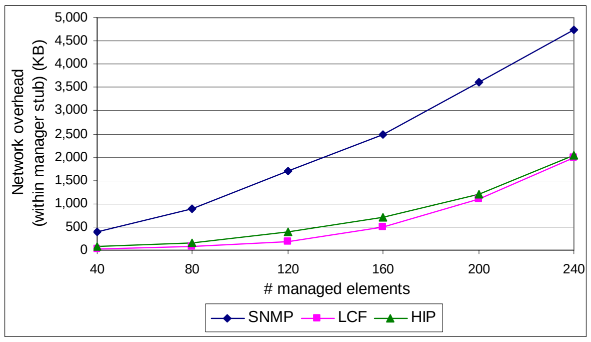
(a)



(b)



(c)



(d)

Figure 10. Monitoring of `tcpConnTable` *MIB-II variable: comparison of SNMP against HIP and LCF algorithms in terms of (a) Overall latency, (b) Network overhead, (c) Management cost, (d) Network overhead within manager station stub*

Finally, Figure 10d illustrates the usage of network resources within manager station stub. Interestingly, this metric represents the main asset of LCF algorithm, since it suggests that the unique monitoring MA will use manager stub link only twice: on its first migration and at the end of its itinerary (when it returns to deliver collected data to the manager station). On the other hand, SNMP routes all monitoring traffic through the manager stub links, implying a congestion point.

7.2. SIMULATION #2

Admittedly, a comparison of HIP against SNMP only based on retrieving a SNMP table variable (`tcpConnTable`)may seem biased, since table retrieval is a well-known weakness of SNMP [26]. Hence, our second simulation test involves the retrieval of 'atomic' (non tabular) MIB-II variables.

Cases often occur in monitoring operations, where one or two MIB variables are not a representative indicator of system state and hence an aggregation of multiple variables is required, known as a *health function* (HF) [11]. For instance, *five* MIB-II objects are combined to define the percentage *E(t)* of IP output packets discarded over the total number of packets sent within a specific time interval:

$$E(t) = \frac{(ipOutDiscards + ipOutNoRoutes + ipFragFails) * 100}{ipOutRequests + ipForwDatagrams} \qquad (7\text{-}1)$$

In the SNMP model, the least 'expensive' option would be to group the five variables into a single *get* request packet. The response packet would then include the variable IDs along with the requested values, with the IDs typically occupying more space than the actual values [26]. On the other hand, MAs are able to compute HFs locally, providing a way to semantically compress large amount of data in a single value returned to the manager; thus, the manager application is relieved from processing management data, while MAs state size remains as small as possible. MAs can also be instructed to accumulate computed values only when certain thresholds are exceeded.

In addition to the parameters shown in Table 2, we have also measured and incorporated into the simulation modules the following parameters: (a) MA state size increment per HF value sample (compressed data): 2 bytes; (b) Average SNMP-based retrieval response time of 5 MIB-II variables: 8 msec; (c) Average MA-based retrieval response time of 5 MIB-II variables: 4 msec.

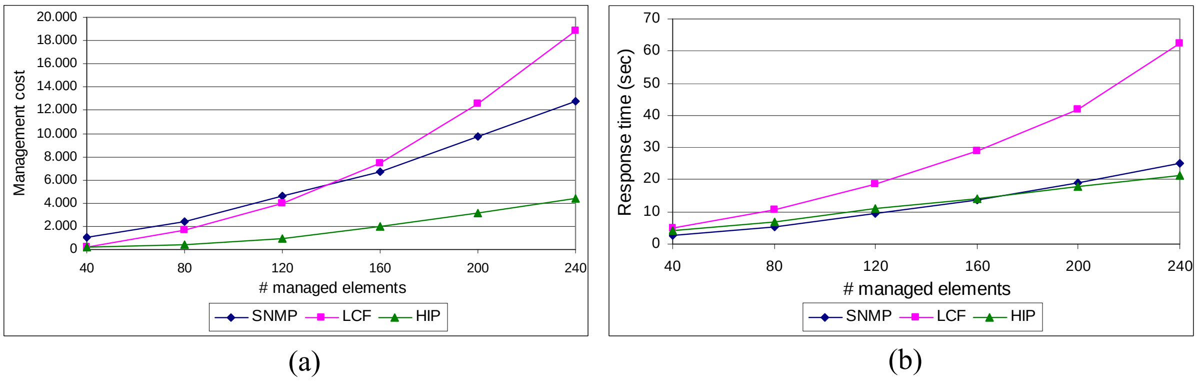


Figure 11. Monitoring of health function value: comparison of SNMP against HIP and LCF algorithms in terms of (a) Management cost, (b) Overall latency

The results of the second simulation test are shown in Figure 11. Evidently, HIP algorithm provides again a more scalable solution in terms of management cost, compared to SNMP and LCF-based monitoring (Figure 11a). This is mainly due to the heavy usage that SNMP implies over 'expensive' transit-stub interconnecting links. However, LCF demonstrates scalability improvement in comparison with the first simulation test, since the MA state increment rate is reduced (a smaller volume of management data is now accumulated). Finally, Figure 11b shows that HIP maintains comparable response time to SNMP, while clearly overwhelming the LCF-based approach.

8. Conclusions & Future Work

Despite the popularity of MA-based management applications, methodologies for designing efficient MA itineraries have received little attention. In particular, the number of hops realised by a multi-hop agent is not the only metric to evaluate the communication overhead of MA-based operations. The order in which MAs visit their assigned hosts (i.e. MA itineraries) is also a crucial factor, as slight modifications may result in dramatically variant costs.

This article introduces an algorithm that borrows ideas and concepts from the area of network design to address the issue of MA optimal itinerary planning. Although only tested on network monitoring applications, it would comfortably fit in other application areas, such as network discovery, service management, etc. HIP algorithm not only suggests the appropriate number of MAs that should be employed in parallel to minimise the associated cost but also constructs near-optimal itineraries for each of them.

Future research will involve testing of HIP on several application fields, other than network monitoring, such as configuration management, performance management, service discovery, etc.

Acknowledgements

The authors are grateful to Dr. Tryfonoula Korbi for proof-reading this paper and correcting many grammatical and vocabulary errors. The authors also wish to thank the anonymous reviewers for their constructive comments which significantly improved the technical content and the presentation of the paper.

List of Acronyms

CMST:	Constrained Minimum Spanning Trees	MD:	Managed Domain
E-W:	Esau-Williams	MIB:	Management Information Base
HIP:	Heuristic for Itinerary Planning	MST:	Minimum Spanning Trees
IETF:	Internet Engineering Task Force	NE:	Network Element
IP:	Internet Protocol	NS-2:	Network Simulator, version 2
ISM:	Itinerary Scheduler Module	NSM:	Network and Systems Management
JDK:	Java Development Kit	RAM:	Random Access Memory
LAN:	Local Area Network	SNMP:	Simple Network Management Protocol
LCF:	Local Closest First	TCP:	Transport Control Protocol
MA:	Mobile Agent	UDP:	User Datagram Protocol
MAP:	Mobile Agent Platform	WAN:	Wide Area Network

References

[1] Adventnet, http://www.adventnet.com.

[2] Aglets Mobile Agent Platform, http://www.trl.ibm.com/aglets/.

[3] C. Bohoris, A. Liotta, G. Pavlou, “Mobile Agent Based Performance Management for the Virtual Home Environment”, Journal of Network and System Management, 11(2), pp. 133-149, June 2003.

[4] T. Chen, S. Liu, “A Model and Evaluation of Distributed Network Management Applications”, IEEE Journal of Selected Areas in Communications, 20(4), May 2002.

[5] T. Du, E. Li, A.P. Chang, “Mobile Agents in Distributed Network Management”, Communications of the ACM, 46(7), July 2003.

[6] A. Fuggeta, G.P. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering 24(5), pp. 346–361, 1998.

[7] D. Gavalas, "Mobile Software Agents for Network Monitoring and Performance Management", PhD Thesis, University of Essex, UK, July 2001.

[8] D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, "Hierarchical Network Management: A Scalable and Dynamic Mobile Agent-Based Approach", Computer Networks, 38(6), pp.693-711, April 2002.

[9] D. Gavalas, "Mobile Agent Platform Design Optimisations for Minimising Network Overhead and Latency in Agent Migrations", Proc. of the 2004 IEEE Global Communications Conference (Globecom'2004), December 2004.

[10] D. Gavalas, "Optimal Itinerary Planning for Mobile Agents-Based Management Applications", Proc. of the 2005 IEEE Global Communications Conference (Globecom'2005), in press.

[11] G. Goldszmidt, "On Distributed Systems Management", Proc. of the 3rd IBM/CAS Conference, 1993.

[12] F.K. Hwang, D.S. Richards, P. Winter, "The Steiner Tree Problem", Annals of Discrete Mathematics series, Vol. 53, Elsevier, 1992.

[13] A. Iqbal, J. Baumann, M. Straßer, "Efficient Algorithms to Find Optimal Agent Migration Strategies", Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1998/05, April 1998.

[14] JADE (Java Agent DEvelopment Framework), http://jade.tilab.com/

[15] Java 2 Platform, Standard Edition (J2SE), http://java.sun.com/j2se/

[16] A.Kershenbaum, "Telecommunications Network Design Algorithms", McGraw-Hill, 1993.

[17] A. Liotta, G. Pavlou, G. Knight, "Exploiting Agent Mobility For Large Scale Network Monitoring", IEEE Network, 16(3), pp. 7-15, May/June 2002.

[18] P. Marques, P. Simões, L. Silva, F. Boavida, J. Gabriel, "Providing Applications with Mobile Agent Technology", Proc. of the 4th IEEE International Conference on Open Architectures and Network Programming (OpenArch'01), April 2001.

[19] K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II", RFC 1213, March 1991.

[20] The Network Simulator - NS-2, http://www.isi.edu/nsnam/ns/

[21] V. Pham, A. Karmouch, "Mobile Software Agents: An Overview", IEEE Communications Magazine, 36(7), pp. 26-37, 1998.

[22] H. Qi, F. Wang, "Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks", Proc. of the 15th IEEE International Conference on Wireless Communications, pp.147-153, July 2001.

[23] E. Reuter, F. Baude, "System and Network Management Itineraries for Mobile Agents", Proc. of the 4th International Workshop on Mobile Agents for Telecommunication Applications (MATA'02), LNCS vol. 2521, pp 227-238, October 2002.

[24] M.G. Rubinstein, O. C. Duarte, G. Pujolle, "Scalability of a Mobile Agents Based Network Management Application", Journal of Communications and Networks, 5(3), September 2003.

[25] W. Stallings, "SNMP, SNMPv2, SNMPv3 and RMON 1 and 2", 3rd ed., Addison Wesley, 1999.

[26] Sprenkels R. and Martin-Flatin J.P., "Bulk Transfers of MIB Data", The Simple Times, 7(1), pp. 1-7, 1999, http://www.simple-times.org/.

[27] R. Stephan, P. Ray, N. Paramesh, "Network Management Platform Based on Mobile Agents", International Journal of Network Management, 14, pp. 59-73, 2004.

[28] UCD-SNMP project, http://www.ece.ucdavis.edu/ucd-snmp/.

[29] WinDump: tcpdump for Windows, http://windump.polito.it/

[30] E. W. Zegura, K. L. Calvert, M. J. Donahoo, "A quantitative comparison of graph-based models for internet topology," IEEE/ACM Transactions on Networking, 5(6), pp. 770–783, December 1997.

Appendix: LCF Algorithm Implementation

Local Closest First (LCF) algorithm has been implemented, based on the description given in [22], to provide a fair performance and comparison measure to HIP algorithm.

Let *n* be the overall number of nodes to be managed, *C* the cost matrix ($c_{i,j}$ is the cost of including in the itinerary an agent migration form host *i* to host *j*) and N_0 the node hosting the manager application. The unique MA itinerary proposed by LCF algorithm is formulated starting with host N_0; on every step, the so-far constructed itinerary is 'merged' with the host 'nearest' to the host previously added in the itinerary. This process is described by the pseudo-code implementation of Figure 12.

```
LCF (n, C, $N_0$)
        initialize ($I$)  // $I$ is the unique itinerary constructed by the algorithm
        N_connected = 0   // number of hosts already included in $I$
        merge ($I$, $N_0$)
        current = $N_0$   // 'current' is the host last included in $I$

        while (N_ connected < n)
            closest = find_closest (current), where $c_{\text{current, closest}} = \min_i (c_{\text{current}, i}), \forall i \notin I$
            merge ($I$, closest)
            current = nearest
            N_ connected ++
        return $I$
```

Figure 12. Pseudo-code implementation of LCF algorithm

Dr. Damianos Gavalas received his BSc degree in Informatics (Computer Science) from University of Athens, Greece, in 1995 and his MSc and PhD degree in electronic engineering from University of Essex, U.K., in 1997 and 2001, respectively. In July 2004 he was appointed as a Lecturer in the Department of Cultural Technology and Communication, University of the Aegean, Greece. His research interests include distributed computing, mobile code, network and systems management, network design, e-commerce, m-commerce, wireless sensor networks and mobile wireless ad-hoc networks clustering and routing.

Dr Christina (Tanya) Politi received a B.Sc. in Physics from the University of Athens in 1998 and a M.Sc. degree in the "Physics of Laser Communications" from the University of Essex in 2000. Subsequently, she joined the Photonic Network Research Group in the Department of Electronic Systems Engineering at the University of Essex where she obtained her PhD. She was involved in various projects including the IST-OPTIMIST and IST-BREAD projects. She has recently joint the Optical Networks Group in ICCS/NTUA. Her research interests include network design algorithms, optical packet and circuit switched networks, high speed optical networks and optical wavelength converters.

.