A Heuristic Algorithm for Designing Near-Optimal Mobile Agent Itineraries

Damianos Gavalas

Abstract: **Several distributed architectures, incorporating mobile agent technology, have been recently proposed to answer the scalability limitations of their centralized counterparts. However, these architectures fail to address scalability problems, when distributed tasks requiring the employment of itinerant agents is considered. This is because they lack mechanisms that guarantee optimization of agents' itineraries so as to minimize the total migration cost in terms of the round-trip latency and the incurred traffic. This is of particular importance when MAs itineraries span multiple subnets. The work presented herein aspires to address these issues. To that end, we have designed and implemented an algorithm that adapts methods usually applied for addressing network design problems in the specific area of mobile agent itinerary planning. The algorithm not only suggests the optimal number of mobile agents that minimize the overall cost but also constructs optimal itineraries for each of them. The algorithm implementation has been integrated into our mobile agent framework research prototype and tested in real network environments, demonstrating significant cost savings.**

Index Terms: **Heuristic, itinerary planning, mobile agents, network monitoring, optimization, performance evaluation.**

I. INTRODUCTION

The scene of IP network and systems management (NSM) has been dominated, during the past 15 years, by the IETF simple network management protocol (SNMP) [1]. Despite its wide deployment base, SNMP follows the client/server paradigm, typically associated with massive transfers of management data, which cause considerable strain on network throughput and processing bottlenecks at the manager host. These problems have motivated a trend towards distributed management as an answer to the limitations of centralized NSM. A new trend in NSM involves using mobile agents (MAs) [2] to manage distributed systems [3]–[6]. 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 remotely communicated or carried back to the manager by the MA. Despite its potential though, MA technology has not yet achieved the anticipated commercial success, mainly due to security reasons [7]; however, several software developers have lately incorporated agent technology into their software products, including management product solutions [8]–[10].

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 execute their tasks [11], [12]. What is not commonly exploited in management is the MA *multiple-hop* capability, where agents may move several times as they adapt to changing circumstances. While single-hop mobility can improve flexibility and scalability in 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 significant structural dynamics [4]. In addition, network monitoring based on multi-hop (itinerant) MAs is advantageous for short-term monitoring tasks and also in cases where a global (domain)-level rather than a local (device)-level view of managed resources is required. This issue is discussed in detail in [7] and [13].

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, with the goal to devise methods for optimizing MA itineraries.

The remainder of the paper is organized as follows. Section II explains the importance of optimal agent itinerary planning in management applications and Section III reviews works related to the research presented herein. Section IV discusses the background, design and functionality of our algorithm for optimal itinerary planning. Experimental results on realistic networking environments are presented in Section V, while Section VI concludes the paper.

II. THE ROLE OF ITINERARY DESIGN IN MOBILE AGENT-BASED MONITORING

Despite the potential of agent mobility in distributed applications, a naive use of MAs may lead to a highly inefficient design. In network monitoring applications for instance, using a single MA object launched from the manager platform that sequentially visits all managed devices, regardless of the underlying topology may actually lead to worse performance than the conventional SNMP-based approach (see Fig. 1(a)). The performance further declines, when the monitoring MA collects monitoring data from multiple subnets, often interconnected by low-bandwidth links (see Fig. 1(b)). In such cases, the traffic associated with management tasks typically traverses several network segments and, when summed up, results in increased bandwidth waste [5].

Manuscript received March 3, 2005; approved for publication by Dongman Lee, Division III Editor, September 30, 2005.

D. Gavalas is with the Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Lesvos Island, Greece, email: dgavalas@aegean.gr.

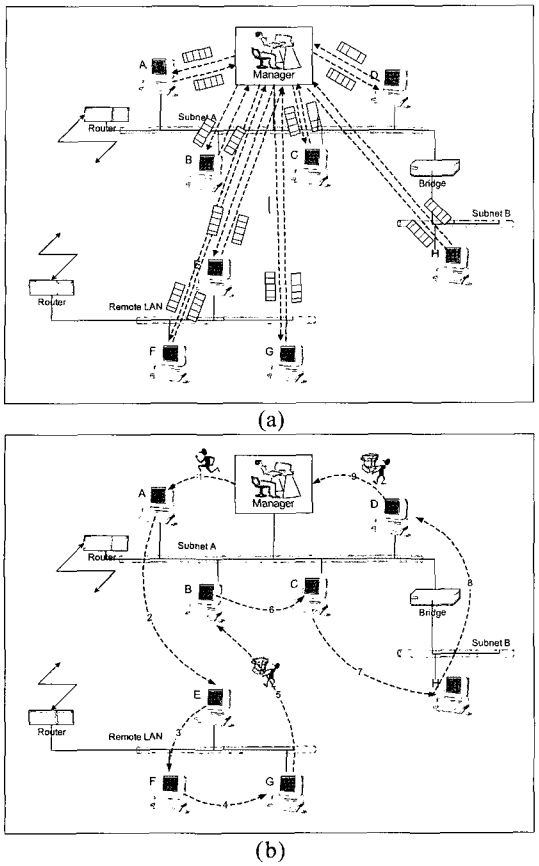


Fig. 1. (a) Centralized monitoring, (b) 'flat' MA-based monitoring.

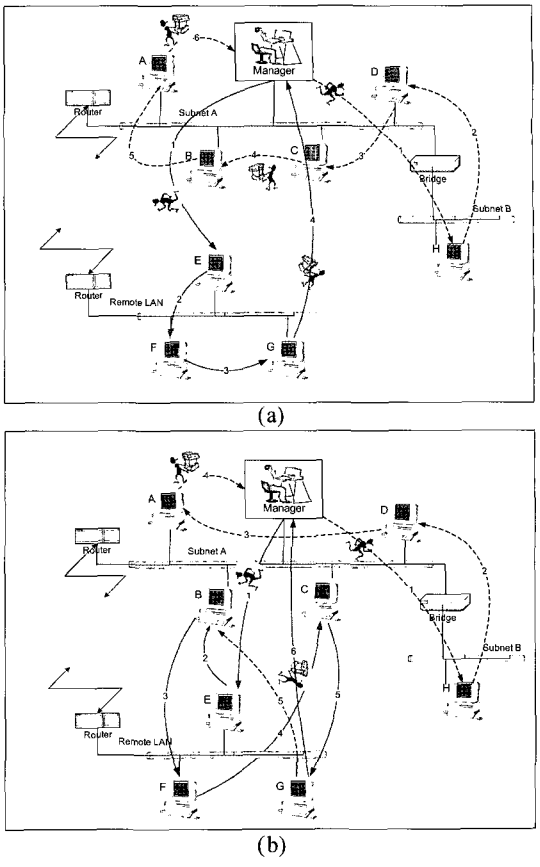


Fig. 2. Partitioning the network into two distinct domains: (a) Optimized, (b) non-optimized itineraries design.

This inefficient approach, known as 'flat' MA-based monitoring [13], presents serious scalability problems: In large networks the *round-trip delay* of the MA greatly increases as the overall travel time depends on the number of hops executed by the MA; additionally, the *network overhead* imposed by the MA transfers grows exponentially with the network size [5], [13]; the slope of the overhead curve becomes steeper in the case of high selectivity[1] values.

A rational approach to overcome such scalability problems is to partition the managed network into several logical/physical domains. For instance, in Fig. 2(a), an MA 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 reduced, 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 these criteria (see [13]).

However, the scenario illustrated in Fig. 2(a) represents an ideal case in terms of WAN link utilization, as the link is traversed only twice per polling interval. For instance, in the partitioning scenario pictured in Fig. 2(b), MAs traverse the WAN link six times (per polling interval, in both directions). This clearly inefficient usage of networking resources is due to the fact that traveling MA routes have not been optimally designed: Even when specific partitioning criteria are followed, the design of MA itineraries is almost random. Namely, the itinerary 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 IV, we describe an heuristic algorithm for itinerary planning (HIP).

[1]Selectivity σ ($0 \leq \sigma \leq 1$) is a metric defined as the proportion of data *maintained* to that *retrieved* from each host. For low selectivity values (the major part of the obtained data being filtered at the source), the MA's state size practically remains constant, otherwise the state rapidly grows [13].

III. RELATED WORK

Multi-hop MAs itinerary optimization has not been sufficiently addressed in the literature. A first attempt to address this issue has been reported in [14]: 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, they indicate 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 [15], in the context of network and system management applications.

Rubinstein *et al.* [5] evaluated the scalability of MA-based management of large enterprise networks and compared the

performance of this approach against that of the 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 in its itinerary. Their simulation results indicate 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, thus minimizing 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.

The work presented in [16], in which Qi and Wang propose the employment of MA paradigm in wireless sensor networks, is conceptually most similar to the work presented herein. To optimize an agent's itinerary, they derived a local closest first (LCF) algorithm according to which each MA searches for the next destination with the shortest distance from 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 depends significantly on the MAs original location, while the nodes left to be visited last are associated with high migration cost [17]; this occurs because 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.

It should be emphasized that the results presented in both [14] and [16], deal with the problem of constructing near-optimal MA itineraries for a given set 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. 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.

IV. BACKGROUND AND DESIGN ISSUES OF A HEURISTIC ALGORITHM FOR ITINERARY PLANNING (HIP)

The problem of designing optimized itineraries exhibits striking similarities with the multi-point line topologies or constrained minimum spanning trees (CMST) problems. A CMST is a minimum spanning tree[2] with an additional constraint on the size of the subtrees rooted at the 'center' (there is an upper limit on the number of nodes included on each of the subtrees originating from the tree's root). CMST algorithms are used in graph theory, with the main application field being network design problems [17]. In such problems, the objective is the optimal selection of the links connecting terminals to concentrators[3] or directly to the network center, resulting in the minimum

[2]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 [17].

[3]Concentrators (or multiplexors) are nodes that consolidate low speed lines into higher speed lines, directed to the network center [17].

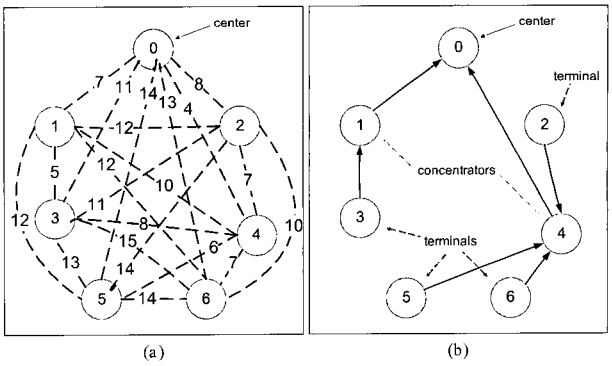


Fig. 3. CMST problem: (a) The unconnected graph, (b) the optimal multi-point line topology (constrained minimum spanning tree).

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, Fig. 3(a) depicts a set of nodes with a given network center and costs for connecting individual pair of nodes; Fig. 3(b) presents the optimal multi-line topology that minimizes 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 in the problem solution

$$c = c_{3,1} + c_{1,0} + c_{2,4} + c_{5,4} + c_{6,4} + c_{4,0} = 36. \quad (1)$$

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 originating from the manager station, the similarity of CMST and MA itinerary planning problems becomes evident. It is natural, therefore, to use algorithms originally devised for CMST problems in the application area of MA itinerary planning.

As CMST problems are NP-hard, 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 [17]. Our HIP algorithm adapts some basic principles of the E-W algorithm in the specific requirements of itinerary planning problems.

A. Particularities of the Itinerary Planning Problem

The simple cost function of (1) fails to address the particularities of the 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 [2], [5], which depends on the amount of data collected by the MA on each visited host.

A.1 Problem's Definition

In the itinerary planning problem, we model the network as an undirected graph G (N, E); N denotes the set of all hosts visited by MAs (*vertices*) and E denotes the set of links (i, j)

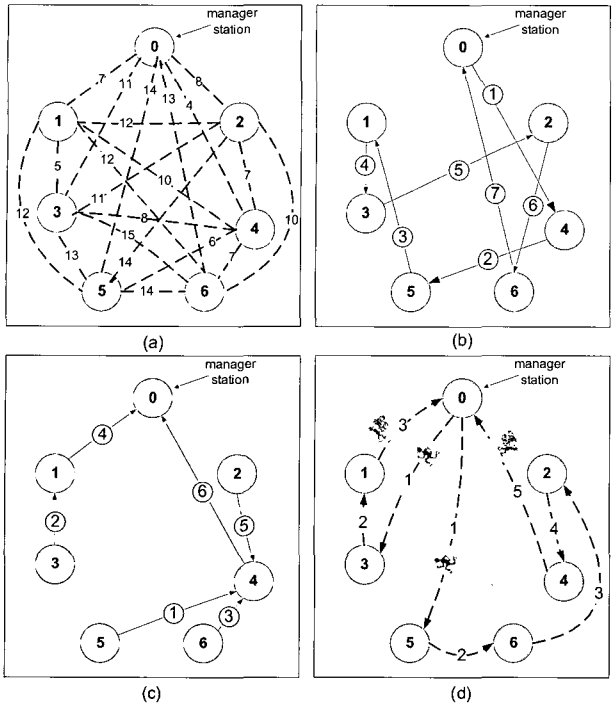


Fig. 4. The MA itinerary planning problem: (a) The original network graph, (b) the output of LCF algorithm, (c) the output of HIP algorithm, (d) two MA itineraries derived from the output of HIP algorithm.

traversed by MAs (*edges*) where $i, j \in N$. Let us assume that a set of itineraries $I = \{I_1, I_2, \cdots, I_n\}$ is constructed, each assigned to an individual MA; itineraries I_i formally represent a set of vertices: $I_i = \{N_i | N_i \in N\}$, $N = \cup I_i$.

In particular, each itinerary I_i includes a set of hosts to be sequentially visited by its respective MA: $I_i = \{N_0, N_1, \cdots, N_n, N_0\}$. Note that all itineraries originate and terminate at the manager station host N_0. The total cost per polling interval over all itineraries $|I|$ becomes

$$C_{\text{total}} = \sum_{i=1}^{|I|} \sum_{j=0}^{|I_i|-1} (d_{ij} + s)c_{ij}, \text{where } d_{i0} = 0 \qquad (2)$$

where 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 that minimize the cost function of (2).

A.2 Demonstration of Algorithm's Execution

In order to be able to compare the performance of our HIP algorithm with the previous state of the art, we have also implemented the LCF algorithm, as described in [16]. For instance, for the network of Fig. 4(a) with cost matrix detailed in Table 1, the itinerary constructed by LCF is shown in Fig. 4(b). The sequence numbers enclosed within circles indicate the order in which individual links (or migrations) become accepted in the corresponding algorithm steps.

Table 1. Cost matrix of the network shown in Fig. 4(a).

	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							

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,[4] i.e., 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.

B. The HIP Algorithm Implementation and Execution

The 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' while visiting managed devices without returning back to the manager site to 'unload' their collected data [5]. 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.

Specifically, the aim of the HIP algorithm is, given a set of hosts $N = \{N_0, N_1, \cdots, N_{n-1}\}$, the manager station N_0 and the cost matrix C, to return a set of near-optimal itineraries $I = \{I_0, \cdots, I_k\}$, all originated and terminated at the manager station. Initially, we assume $|N|(= n)$ itineraries, as many as the network nodes: $I_0, \cdots, I_{n-1}$, each containing a single host (N_0, N_1, $\cdots$, 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 III, LCF-like algorithms usually perform poorly as they tend to leave hosts located far from the center stranded since they prioritize the inclusion of hosts closest to the 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. The HIP algorithm accomplishes this by borrowing and extending the concept of 'tradeoff function'[5]

[4]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).

[5]The concept of the tradeoff function is introduced in E-W algorithm, defined as follows: $t_{ij} = c_{ij} - c_{iN_0}$. Equation (3) extends and adapts this function in the specific requirements of agent itinerary planning problem. In particu-

```
HIP (n, c, d, N0) // n: Total number of hosts, c: Cost matrix,
// d: Data collected per host, N0: Origin station
    initialize I // I: The list of itineraries to be constructed
    current = N0
    N_connected = 0 /* N_connected: The number of hosts
     already included into an itinerary */
    while (N_connected < n)
      /* I(i) is the group of hosts (itinerary) where host I
         has already been included and |I(i)| is the
         number of hosts already included within I(i)*/
      compute t_{i,j} = c_{i,j} + (d(|I(i)| + |I(j)|)) - C_{i,N0},
         where I(i) ∩ I(j) = ∅ and C_{i,N0} = min_{k∈I(i)} C_{k,N0}
      merge (I(i), I(j)), for (i, j) minimizing the tradeoff
         function min_{i,j} t_{i,j}
      N_connected ++
return I
```

Fig. 5. Pseudocode implementation of HIP algorithm.

$t_{i,j}$ associated with each link (i, j), defined by

$$t_{i,j} = c_{i,j} + (d(|I(i)| + |I(j)|)) - C_{i,N_0} \quad (3)$$

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

$$C_{i,N_0} = \min c_{k,N_0}, k \in I(i). \quad (4)$$

Equation (3) 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. Fig. 5 lists a pseudo-code implementation of the HIP algorithm.

The HIP algorithm execution steps for the test network graph of Fig. 4(a) are demonstrated in Table 2, where the links (agent migrations) selected are underlined; 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 of itineraries including *all* hosts is constructed. Table 2 presents the values of $t_{i,j}$ for pairs (i, j) minimizing the tradeoff function for each host i (min $t_{i,j}$, $\forall j$); the pair (i, j) that minimizes $t_{i,j}$ over all hosts (min $t_{i,j}$, $\forall i, j$) 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: $I(5) \cup (4) = \{5, 4\}$. On the next step, $t_{i,j}$ values are re-calculated, for instance, $t_{2,4} = c_{2,4} + (1 \times (|I(2)| + |I(4)|) - c_{2,N_0} = 7 + (1+2) - 8 = 2$

lar, 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. In network design problems, the term 'tradeoff' is used to reflect the tradeoff between connecting each component directly to the center and interconnecting two components [17]; likewise, in the itinerary planning problem, it reflects the tradeoff between allowing the MA to return to the manager station (to deliver collected data) and instructing it to continue its itinerary to another managed device.

Table 2. HIP algorithm execution steps for the network of Fig. 4(a).

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

(note that the set elements of the itinerary including host 4 have increased: $I(4) = \{5, 4\} \Rightarrow |I(4)| = 2$). At the end of step 6, two sets are constructed, forming two subtrees rooted at the manager host: {6, 5, 4, 2} and {3, 1} (see Fig. 4(c)). It is then straightforward to form the itinerary plan of the two MAs: $I_1 = \{0, 5, 6, 2, 4, 0\}$ and $I_2 = \{0, 3, 1, 0\}$ (see Fig. 4(d)). These itineraries correspond to a post-order traversal[6] of the two subtrees.

C. Cost Comparison of HIP and LCF Algorithms

The total costs associated with LCF and HIP proposed solutions (shown in Fig. 4(b) and Fig. 4(d), respectively) are calculated using the generic cost function of (2)

$$\begin{aligned} C_{LCF} = {} & s \times c_{0,4} + (s + d)c_{4,5} + (s + 2d)c_{5,1} \\ & + (s + 3d)c_{1,3} + (s + 4d)c_{3,2} + (s + 5d)c_{2,6} \\ & + (s + 6d)c_{6,0} \end{aligned}$$

$$\begin{aligned} C_{\text{HIP}} = {} & [s \times c_{0,3} + (s + d)c_{3,1} + (s + 2d)c_{1,0}] \\ & + [s \times c_{0,5} + (s + d)c_{5,6} + (s + 2d)c_{6,2} \\ & + (s + 3d)c_{2,4} + (s + 4d)c_{4,0}] \end{aligned}$$

where the first term of C_{HIP} refers to the cost of the first MA itinerary while the second term to the cost of the second MA itinerary.

[6]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 is assigned devices spread among subnets A, B, and C, it will first visit all devices of B, then all devices of subnet C, 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.

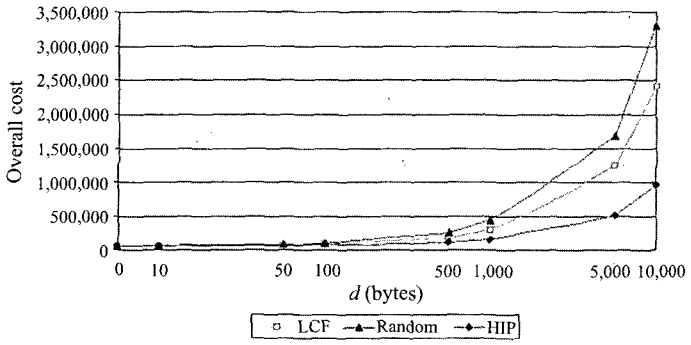


Fig. 6. Scaling of LCF and HIP overall costs as a function of collected data per host, for $s = 1000$ bytes.

Assuming an MA of initial size $s = 1000$ bytes that collects an amount of $d = 100$ bytes from each visited host and after substituting various costs with the corresponding values found in the cost matrix of Table 1, we get $C_{\text{LCF}} = 90,500$ and $C_{\text{HIP}} = 81,000$ cost units, which corresponds to a cost saving of 10.5% when employing the HIP 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 becomes 42.0%.

Fig. 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[7]). 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 the MA size growth rate is such that multihop itineraries become too costly. Also, a smaller value for s is expected to shift the exponential growth part of the curve left, while a larger value of s is expected to shift it right.

The cost saving percentages of HIP-proposed itineraries over LCF-proposed itineraries, for various-sized graphs with randomly generated cost matrices, are shown in Table 3. Each table entry corresponds to the average cost measured for the two algorithms on 5 different 'topologies,' i.e., cost matrices. Interestingly, the number of itineraries proposed by the HIP algorithm (i.e., the number of MAs that collect data in parallel) highly depends on the cost matrix values. For instance, for MA size of 1000 bytes collecting 100 bytes from a network of 50 nodes, the average number of proposed itineraries increases from 4 to 6 when increasing cost matrix values from the range $1 \leq c \leq 20$ to $21 \leq c \leq 40$ (this increment also implies itinerary cost growth, hence HIP motivates the construction of shorter itineraries, i.e., the employment of additional MAs).

[7]We initially measured the itinerary cost for a larger number N of random topologies ($N > 5$), however it has been noticed that the average cost stabilizes for $N > 5$, i.e., 5 random topologies provide an indicative average cost value.

Table 3. Cost improvement of HIP over LCF algorithm for agents of initial size $s = 1000$ bytes as function of the network size N and the amount of data collected from each host d (in bytes). The figures presented correspond to link utilization costs.

	$N = 6$	$N = 10$	$N = 20$	$N = 50$
$d = 10$	−8.09%	−2.34%	1.12%	4.48%
$d = 50$	−0.06%	1.52%	3.22%	8.38%
$d = 100$	10.50%	12.77%	19.10%	30.66%
$d = 500$	33.88%	39.12%	46.90%	59.10%
$d = 1000$	43.71%	50.45%	61.06%	83.63%

V. HIP ALGORITHM EXPERIMENTAL EVALUATION

It should be emphasized that the 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 with any available MAP (e.g., Aglets [2] or JADE [21]), constructing and supplying near-optimal MA itineraries. To prove the validity and effectiveness of the HIP, 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 [19].

ISM has been implemented in the Java programming language (JDK 1.4.1 [22]); 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 an event, ISM is notified through a callback method). The LCF algorithm has also been implemented for comparison purposes.

Although HIP has only been tested on realistic network monitoring application scenarios, it is suited to any application field which benefits from the distribution of intelligence and processing overhead offered by the MA paradigm. The experimental testbed includes several Windows NT and Linux Red Hat v. 6.1 stations. The managed network comprises two 10 Mbps LANs connected through a 1 Mbps leased line (see Fig. 7); the first LAN (where the manager application executes) hosts 5 managed elements while the second hosts 10 elements.[8] The network traffic generated by MA migrations has been measured using the WinDump[9] network analyzer [25]. 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 the UCD-SNMP package [24] have been installed on Linux stations.

The cost matrix used by the HIP algorithm reflects the cost of using network resources (the bandwidth of our LANs is ten

[8]Although providing a useful insight on the algorithm's operation, the 15 elements available on the experimental testbed are certainly not sufficient to evaluate the performance of the HIP algorithm on large networking environments. This small test network has been mainly used to measure the network overhead and timing parameters associated with MAs migration and interaction with legacy systems; the latter will be incorporated into a simulation model which is under development, used to extract realistic evaluation results regarding the performance of the HIP algorithm on large enterprise networks.

[9]WinDump, developed at the Politecnico di Torino, represents a graphical version of the well-known *tcpdump* analyzer. Windump is capable of capturing packets transferred over a LAN and decoding their contents, providing a useful insight of ongoing network interactions.

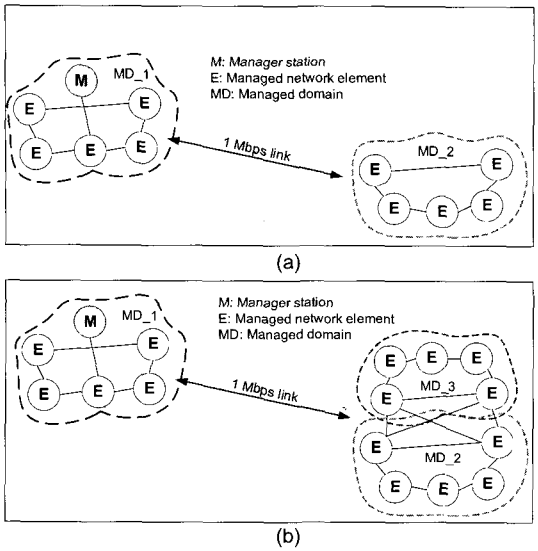


Fig. 7. HIP execution on the experimental testbed: (a) The remote subnet hosts 5 managed devices all included in a single managed domain, (b) the remote subnet hosts 10 managed devices separated in two managed domains.

times larger than the bandwidth of the leased line)

- $c_{i,j} = 1$, when i and j are located within the same subnet,
- while $c_{i,j} = 10$, when i and j are located on different subnets.

The performance of the HIP and LCF algorithms has been compared against an implementation of SNMP in Java [18]. The application scenario involves polling managed elements (SNMP agents) for the contents of the tcpConnTable MIB-II[10] 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) [23], while managed elements stored information about 46 TCP connections on average, i.e., included 46 table rows (46×5 values in total).

The interactions performed between MAs and legacy systems at each visited host are depicted in Fig. 8. MAs sequentially migrate to their assigned monitored elements; an SNMP agent is installed and executed at each element, providing access to MIB-II variables. The communication between MAs and legacy systems (SNMP agents) is achieved through an agent translator gateway interface, which converts MA requests (method invocations) to SNMP requests. In particular, the interaction involves the following steps: (1) The MA communicates through remote procedure calls (RPC) with the agent translator; (2) the latter issues a request (GetRequest PDU) to the SNMP agent for MIB-II variables; (3) the SNMP agent returns the requested variable values (GetResponse PDU); the translator agent returns variable values to the MA which in turn compresses (using the Java *gzip* facility) and encapsulates them into its state.

[10]A management information base (MIB) is a formal description of the set of network objects that can be managed using SNMP. MIB-II [23] is the standard MIB in the IP world, supported by virtually all SNMP-compliant network devices.

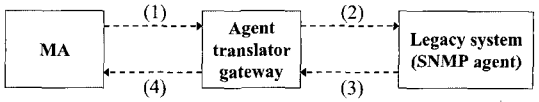


Fig. 8. Interaction of visiting MAs with legacy systems.

Table 4. Network overhead and time parameters.

Parameter	Value
MA initial size	1.08 kB
Average SNMP packet size	90 bytes
Average SNMP packet increment per value	17 bytes
MA state size increment per table sample	98 bytes
Average SNMP-based table retrieval time	102.1 msec
Average MA-based table retrieval time	71 msec

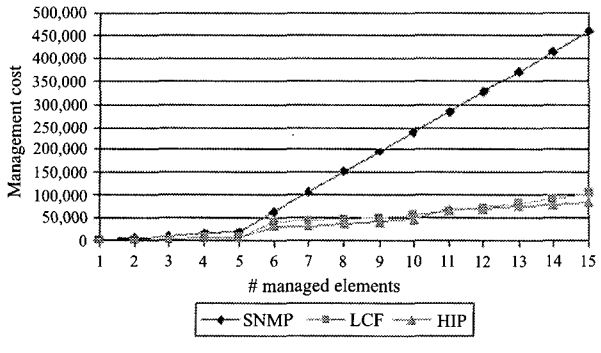
In contrast to the LCF algorithm, HIP involves the parallel execution of multiple MAs the number of which depends on network size. For instance, when 5 managed elements are hosted in the remote subnet, two managed domains are created, each assigned to an individual MA (Fig. 7(a)). However, when the number of managed elements increases to 10, HIP separates them into two distinct managed domain and engages an additional MA into the polling operation (Fig. 7(b)).

The network overhead and timing experiment parameters associated with our experiments are presented in Table 4. Network overhead parameters have been measured by the WinDump tool, while the System.currentTimeMillis() Java method has been used to measure response times.

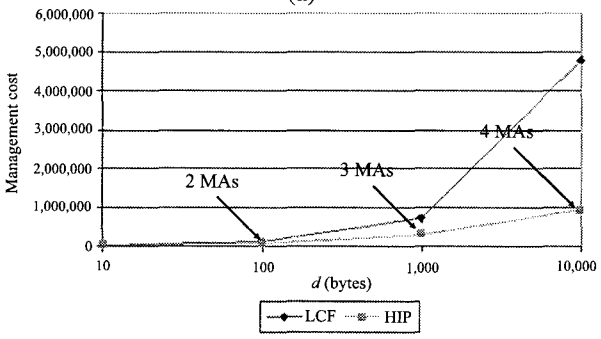
Experimental results are presented in Fig. 9. The overall management cost has been calculated according to the cost function of (2), where cost matrix coefficients $c_{i,j}$ either equal 1 or 10, as described above. Fig. 9(a) compares the management cost of SNMP against MA-based implementations, as a 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, especially 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.

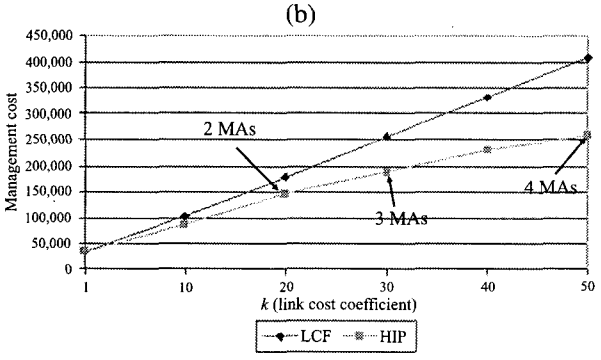
In Fig. 9(b), experiment parameters are adjusted to examine the effect of collected data amounts in the overall cost. The experiments have been performed on the test network of Fig. 7(b) (15 managed elements). HIP performance gain over LCF has



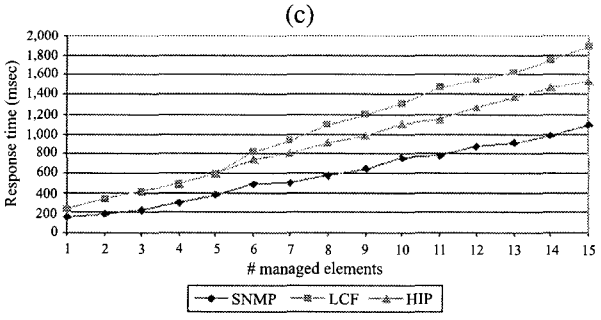
(a)



(b)



(c)



(d)

Fig. 9. (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 Fig. 7, (d) response time of SNMP monitoring against HIP and LCF implementations for varying network sizes.

shown to increase as the amount of data collected from each managed element increases. The HIP algorithm separates remote subnet hosts into 2, 3, and 4 'virtual' managed domains as data sample size increases to 100, 1000, and 10000 bytes, respectively. Fig. 9(c) shows how the management cost relates to the relative cost of the interconnecting link of Fig. 7 (for instance, when the bandwidth of that link is 10 times smaller than the bandwidth of the LANs, the cost coefficient is $k = 10$).

Regarding timing experiments, HIP performs better than LCF due to the parallel employment of MAs when the managed network size increases (see Fig. 9(d)). 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 desirable 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 [2], [13].

It should be noted that the graphs of Fig. 9 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 to filter 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 [20].

VI. CONCLUSIONS AND ONGOING RESEARCH

Methodologies for designing efficient MA itineraries have received little attention so far, despite the popularity of MA-based management approaches. This paper introduces an algorithm that borrows ideas and concepts from the area of network design to address the issue of optimal MA itinerary planning. Although only tested on network monitoring applications, it would comfortably address other application areas, such as network discovery, service management, etc. The HIP algorithm not only suggests the appropriate number of MAs that should be employed in parallel to minimize the associated cost, but also constructs near-optimal itineraries for each of them.

Ongoing research involves extensive simulations of HIP algorithm to assess its performance in large-scale enterprise networks. Testing of HIP in several application areas, other than network monitoring, will also be considered. Appropriate modifications of HIP algorithm are also under examination to investigate its suitability for ad-hoc networking environments; in particular the suitability of MAs for message multicasting in electronic classrooms and ad-hoc collaborative computing scenarios is currently explored.

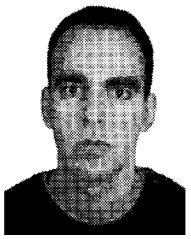
ACKNOWLEDGMENTS

Tryfonoula Korbi is acknowledged for correcting many grammatical and syntax errors and improving the readability of this paper. The author also wishes to thank the anonymous reviewers for their constructive comments which added value to the technical content as well as the presentation of the paper.

REFERENCES

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

[2] A. Fuggeta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Software Engineering*, vol. 24, no. 5, pp. 346–361, 1998.
[3] T. Du, E. Li, and A. P. Chang, "Mobile agents in distributed network management," *Commun. the ACM*, vol. 46, no. 7, July 2003.
[4] A. Liotta, G. Pavlou, and G. Knight, "Exploiting agent mobility for large scale network monitoring," *IEEE Network*, vol. 16, no. 3, pp. 7–15, May/June 2002.
[5] M. G. Rubinstein, O. C. Duarte, and G. Pujolle, "Scalability of a mobile agents based network management application," *J. Commun. Networks*, vol. 5, no. 3, Sept 2003.
[6] R. Stephan, P. Ray, and N. Paramesh, "Network management platform based on mobile agents," *Int. J. Network Management*, vol. 14, pp. 59–73, 2004.
[7] P. Marques, P. Simões, L. Silva, F. Boavida, and J. Gabriel, "Providing applications with mobile agent technology," in *Proc. IEEE OpenArch 2001*, Apr. 2001.
[8] Present Technologies, "JAMES project (Java mobile agent platform for the management of telecommunication and data networks)," available at http://www.present-technologies.com/james.jsp.
[9] SysteMATech—System Management Based on Mobile Agent Technology, available at http://www.eutist-ami.org/more_systematech.asp.
[10] Whitestein Technologies, available at http://www.whitestein.com/pages/index.html.
[11] C. Bohoris, A. Liotta, and G. Pavlou, "Mobile agent based performance management for the virtual home environment," *J. Network and System Management*, vol. 11, no. 2, pp. 133–149, June 2003, Kluwer Academic.
[12] T. Chen and S. Liu, "A model and evaluation of distributed network management applications," *IEEE J. Selected Areas Commun.*, vol. 20, no. 4, May 2002.
[13] D. Gavalas, "Mobile software agents for network monitoring and performance management," Ph.D. Thesis, University of Essex, UK, July 2001.
[14] A. Iqbal, J. Baumann, and M. Straßer, "Efficient algorithms to find optimal agent migration strategies," Universität Stuttgart, Fakultät Informatik, *Bericht Nr. 1998/05*, Apr. 1998.
[15] E. Reuter, F. Baude, "System and network management itineraries for mobile agents," in *Proc. MATA 2002 (Lecture Note in Computer Science)*, vol. 2521, Oct 2002, pp. 227–238.
[16] H. Qi and F. Wang, "Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks," in *Proc. IEEE Int. Conf. Wireless Commun. 2001*, July 2001, pp.147–153.
[17] A. Kershenbaum, *Telecommunications Network Design Algorithms*, McGraw-Hill, 1993.
[18] Adventnet, available at http://www.adventnet.com.
[19] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony, "Hierarchical network management: A scalable and dynamic mobile agent-based approach," *Computer Networks*, vol. 38, no. 6, pp. 693–711, Apr. 2002.
[20] D. Gavalas, "Mobile agent platform design optimisations for minimising network overhead and latency in agent migrations," in *Proc. Globecom 2004*, Dec. 2004.
[21] JADE (Java agent development framework), available at http://jade.tilab.com/.
[22] Java 2 Platform, Standard Edition (J2SE), available at http://java.sun.com/j2se/.
[23] K. McCloghrie and M. Rose, "Management information base for network management of TCP/IP-based Internets: MIB-II," *RFC 1213*, Mar. 1991.
[24] UCD-SNMP project, available at http://www.ece.ucdavis.edu/ucd-snmp/.
[25] WinDump: tcpdump for Windows, available at http://windump.polito.it/.
[26] Aglets Mobile Agent Platform, available at http://www.trl.ibm.com/aglets/.
[27] The Network Simulator—NS-2, available at http://www.isi.edu/nsnam/ns/.
[28] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 770–783, Dec. 1997.



Damianos Gavalas received his B.Sc. degree in Informatics (Computer Science) from University of Athens, Greece, in 1995 and his M.Sc. and Ph.D. degrees 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.