Scalable Message Routing for Mobile Software Assistants

Paweł T. Wojciechowski

Poznań University of Technology
60-965 Poznań, Poland
ptw@cs.put.poznan.pl

Abstract. In this paper we define an algorithm for location-independent communication of mobile software Personal Assistants (PAs). The algorithm extends the Query Server with Caching algorithm that we proposed earlier, with support of message routing in the wide-area networks. Our algorithm is suitable for two kinds of PAs collaboration: (1) within a local group of mobile individuals, who can communicate frequently using different computers connected to a local-area network (possibly via a wireless medium), and (2) some individuals may also communicate via the global network and move to other groups.

Key-words: mobile agents, distributed computing, protocols, middleware, P2P

1 Introduction

The ongoing growth of wide-area networks has brought up considerable interest in *mobile agents* [1–4]; mobile agents are units of executing code that can migrate between machines and perform tasks locally. It has been widely argued [4, 3, 5] that mobile computation provides a useful enabling technology for wide-area applications, such as web services, scientific computation, and collaborative work.

To ease application writing one would like to be able to use high-level *location independent* communication facilities, allowing the parts of an application to interact without explicitly tracking each other's movements. To provide these above standard network technologies (which directly support only location-dependent communication) requires some distributed infrastructure. Sewell, Wojciechowski, and Pierce [6] argued that the choice or design of an infrastructure must be somewhat application-specific – any given *infrastructure algorithm* will only have satisfactory performance for some range of migration and communication behaviour; the algorithms must be matched to the expected properties of applications and the communication network.

In [7], we described the *Personal Assistant (PA)* – an application that uses mobile agents to support collaborative work of mobile individuals. The PA application uses the *Query Server with Caching (QSC)* infrastructure algorithm for location-independent communication. We have prototyped our application using *Nomadic Pict* [6, 7] – a statically-typed, distributed programming language, which is based on the π-calculus [8] extended with distribution and agent mobility. The QSC algorithm however does not scale to wide-area networks, nor to many groups of PA agents, which would be required in the full-scale, practical implementation of the PA application.

In this paper, we therefore extend the QSC algorithm to support wide-area collaboration. We have done so with the following model of collaboration in mind: (1) mobile individuals within a local working group can communicate frequently using different computers connected to a local-area network (possibly via a wireless medium), and (2) some individuals may also communicate via the global network and move to other groups. This model of collaboration covers many real-world scenarios, e.g. think of people working closely on the same project or task within a local (indoor or outdoor) area, who may occasionally contact a distant expert or manager, or migrate to other working group.

We propose a *Federated Query Server with Caching (FQSC)* algorithm that fits well into the above model of collaboration. The algorithm uses a federation of servers. Each federated server is responsible for managing communication within a local group of PAs, and maintaining a dynamic forwarding pointers chain used for communication between groups. The FQSC algorithm behaves as well as the optimal-within-LAN QSC algorithm proposed in [7]. However, it avoids a single point of failure. Furthermore, cache information and compaction techniques are used so that also the communication between LANs requires only one network message in the common case.

Our paper is aimed at developers of mobile agent applications, and researchers interested in distributed (or peer-to-peer) algorithms. The algorithm has been specified formally as an executable encoding in Nomadic Pict; due to lack of space, we omitted this description in the paper but we made it available in the technical report [9]. The formal specification is concise but gives enough details to be directly translated by application programmers using their language of choice.

The paper is organized as follows. Section 2 presents the PA application. Section 3 describes the FQSC algorithm in several steps, beginning with two simple, centralized algorithms. Then we present our distributed algorithm. Section 4 proposes several extensions. Section 5 discusses related work, and Section 6 concludes.

2 The Personal Assistant Application

We consider the support of collaborations within (say) a large computer science department, spread over several buildings. Most individuals will be involved in a few collaborations, each of 2–10 people. Individuals move frequently between offices, labs and public spaces; impromptu working meetings may develop anywhere. Individuals would therefore like to be able to *summon* their working state (which may be complex, consisting of editors, file browsers, tests-in-progress etc.)

to any machine. These summonings should preserve any communications that they are engaged in, for example audio/video links with other members of the project. To achieve this, the user's working state can be encapsulated in a mobile agent, an electronic *personal assistant (PA)*, that can migrate on demand.

We also consider the support of remote collaborations. Individuals can either visit other institutions and summon their personal assistants there, or the PAs can be temporarily *delegated* to other groups. For example, a personal assistant agent encapsulating a buggy program (which may include source files, makefiles, and test data) can be delegated to language experts, who can analyse the program while interacting remotely with the program developer who launched the PA agent, then modify code, check the modified code using the original test data, and finally send the PA (with corrected program) back to the developer.

A usable infrastructure for location-independent communication of PA agents can only be designed in the context of detailed assumptions, both about the system properties and about the expected behaviour of the PA agents. We assume that the application is running over a collection of large LANs, which are connected to a wide-area network, or intranet. In each LAN reliable messaging can be provided by lower-level protocols and all machines are at roughly the same communication cost distance from each other. Machines are also basically reliable, although from time to time it is necessary to reboot or turn off.

We suppose that the number of PA agents is of the same order as the number of people in the labs. Each PA will migrate infrequently, with minutes or hours between migrations. The path of migrations is unpredictable – it may range over the whole LAN, some PAs may occasionally migrate between LANs. The migrations of different PAs are essentially uncorrelated in time. It is common for people to work for extended periods at machines out of their offices. PAs communicate between each other frequently, with significant bandwidth – e.g. audio/video messages or streams, and other data (that must be delivered reliably).

3 Design of Appropriate Infrastructure

We develop our infrastructure in several steps, beginning with two simple, centralized algorithms. Then we present our distributed algorithm.

3.1 Central Server and Query Server with Caching

The *Central Server* algorithm has a single server that records the current site of every agent. Agents synchronize with the server before and after migrations. The location-independent, application messages are sent via the server. The central server is however a bottleneck for all inter-PA communication. Furthermore, all application messages must make two hops (and these messages make up the main source of network load).

Adapting the Central Server so as to reduce the number of application-message hops required, we have the *Query Server with Caching* algorithm, described in [7]. As before, it has a server Q that records the current site of every

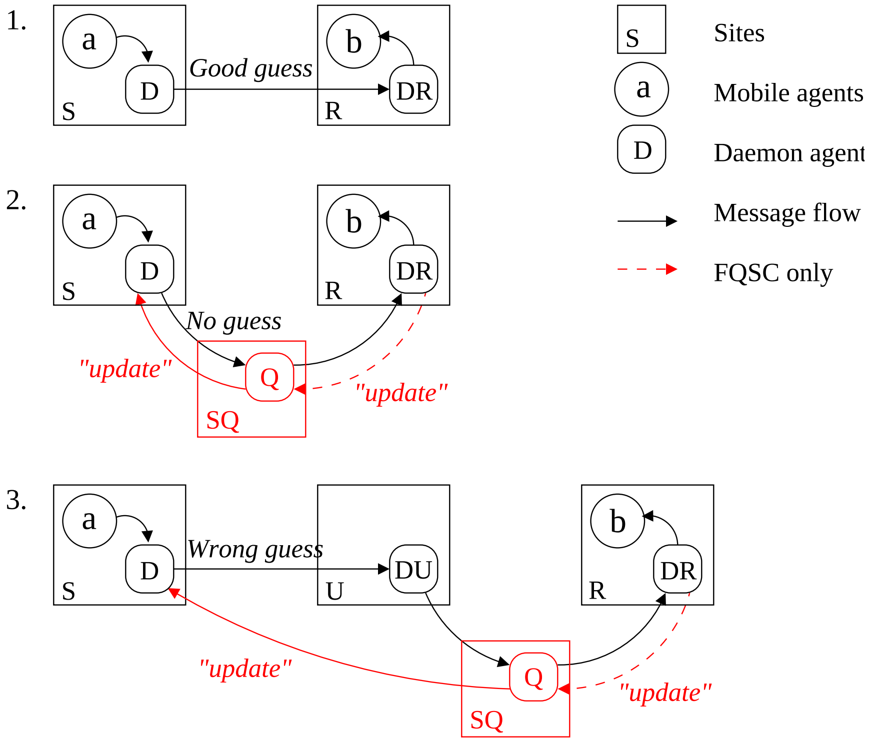

Fig. 1. The QSC and FQSC algorithms: the delivery of a message from agent a to b

agent, and agents synchronize with it on migrations. In addition, each site has a daemon that maintains a cache of location data.

Consider the delivery of an application message from agent a to agent b. The message (see Fig. 1-1) is first sent to a daemon D on the current site S, which then forwards the message to the daemon DR on the target site R, which delivers the message to b. When a daemon D does not know the agent b (see Fig. 1-2), or when a daemon DU receives a mis-delivered message, for an agent b that has left its site U (see Fig. 1-3), the message is forwarded to server Q. The server both forwards the message on to the agent's current site R and sends a cache-update message to the originating daemon. In the common case application messages will here take only one hop (the "good guess" case in Fig. 1-1).

3.2 Federated Query Server with Caching

The QSC algorithm however does not scale to a large number of PAs and a global network. Consider PA migration to a remote working group. The obvious defect in this case is the need to send control messages between the daemon and the server over the Internet, even if migrations and communications of the

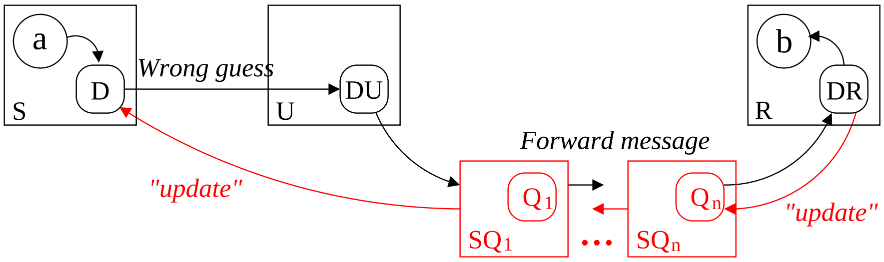

Fig. 2. The FQSC algorithm: the delivery of a message in the "wrong-guess" scenario

PA would be local within a LAN of the new group. Furthermore, the QSC algorithm has single point of failure. To overcome these drawbacks, we may have many servers, each one dealing with agents of a single user or collaborative group. Mobile computation introduces however an interesting problem: how to synchronize migrations and communications on these servers, so that the number of messages between servers and daemons is optimized ?

In this paper, we therefore propose the *Federated Query Server with Caching (FQSC)* algorithm, which removes the bottleneck. It employs a collection of query servers Q_1, ..., Q_n for the specific migration and communication pattern of PA agents. Each server maintains locations of agents that are present in a local domain, where a *domain* can range from a single computer to a LAN.

For each agent there is at least one server, which records the current site of the agent; a *local server* in agent's current domain is an example of such a server. Agents synchronize with the local server before and after migrations. When an agent migrates away to a new domain, it must register at the query server in this new domain, which now becomes the agent's new local server.

As before, each site has a daemon that maintains a cache of location data. Application messages are sent via the daemons, much like in the QSC algorithm (see Fig. 1, 1-3). When a message cannot be delivered using cache information, the message is forwarded by query servers, using forwarding pointer chains that are collapsed when possible upon receipt of an "update" message (see Fig. 2). By compaction of the pointer chains, in the common case application messages are delivered in only one hop, as in the case of QSC.

If a server has no pointer for the destination agent b, then it will forward the message to b's home server, which has the pointer. An agent's *home server* is the query server on which the agent was originally registered upon its creation. The address of this server is recorded as part of the agent's high-level ID.

This may seem well-suited to the PA application, but the textual description omits many critical points – it does not unambiguously identify a single algorithm. For example, it is difficult to explain in prose the following details:

- How are the migrations and communications synchronized (if at all) ?
- What data about agents are actually locked and for how long ?

– How to collapse the pointer chain on servers without loosing messages ?

To explain the details of the FQSC algorithm and to develop reasonable confidence in its correctness, a more precise description is required, ideally in an executable form. In the extended version of this paper [9], we have described the FQSC algorithm formally as a Nomadic Pict encoding, thereby making all the details of concurrency and synchronization precise

The encoding involves three main classes of agents: the query servers `Q` (distributed on sites, so that there is at least one server in each LAN), the daemons `D` (one on each site), and the translations of high-level application agents (which may migrate). For each mobile agent name there is at least one server that has the site and daemon where the agent *is* currently located; servers store these data in a map `m`. Each daemon maintains its own map `m` from agent names to the site and daemon where they *guess* the agent is located. This is updated only when a message delivery fails. The encoding of each high-level agent records its current site and daemon, and the name and site of the local server. This is kept accurate when agents are created or migrate.

The messages sent between agents fall into three groups, implementing the location-independent messages, the high-level agent creation, and agent migration. Below we describe the first group of messages in more details.

Message delivery To send a location-independent message the translation of a high-level agent simply asks the local daemon to send it. Consider an output of a message in agent `a` on site `S` to agent `b`, where the local daemon is `D`. The message will be reliably delivered to agent `b`, irrespective of the current site of `b` and of any migrations. Suppose `b` is on site `R`, where the daemon is `DR`. There are three cases to describe. Either `D` has the correct site/daemon of `b` cached, or `D` has no cache data for `b`, or it has incorrect cache data.

In the first case (see Figure 1-1) `D` sends a message to `DR` which delivers the message to `b` atomically. For the PA application this should be the common case, including the cross-domain communication; it requires only one network message.

In the cache-miss case (see Figure 1-2) daemon `D` sends a message to the local query server `Q`, which forwards the message to a daemon `DR` at site `R`, which then delivers successfully and sends an `update` message back to `D` via `Q` (both `D` and `Q` update their cache). The query server's lock is kept until the message is delivered, thus preventing `b` from migrating until then. Two other variants are possible. If the forwarding pointer for the agent `b` is not found, `Q` forwards the message to `b`'s home server (the server's name/site are encoded as part of the name `b`). Similarly, if `b` has moved between domains and there has been no communication to `b` since then (and so no cache updates), `Q` will contain a pointer to the query server in the domain visited by `b`. In this case, the `message` message is forwarded between query servers until it eventually reaches `DR` (see Figure 2). The forwarding pointer chain is collapsed by sending the `update` messages which update caches with `b`'s current location.

Finally, the incorrect-cache-hit case (see Figure 1-3). Suppose `D` has a mistaken pointer to `DU` at `U`. It will send a message to `DU` which will be unable to deliver the message. `DU` will then send the message to the query server, much as before (except that the cache update message still goes to `D`, not to `DU`).

4 Further Extensions

The FQSC algorithm avoids sending too many cache updates over the Internet. As long as agent migrations are local, a cache-update message to other query servers is sent only in the case of incorrect-cache-hits from these servers. Consequently, the cost of forwarding a message to agents in other domains is paid only for the first message. Then, the forwarding pointer chain is collapsed and any subsequent messages (from the same location) are sent directly.

The above design choice reflects the expected behaviour of the PA agents: communications are more frequent than migrations, and the inter-domain migrations, which correspond to delegation or a physical movement of individuals, are less frequent than migrations within a local domain. If PA behaviour would be different, it may be worth to collapse the forwarding pointer chains more often. For example, upon each cross-domain migration, the cache of *several* daemons and servers could be updated, not just those last visited.

One can also analyse the application further. In fact, migrations of the PA agents may usually be within a small group of machines, e.g. those of a project group. More sophisticated infrastructures might use some heuristics to take advantage of this. For a critical application a quantitative analysis may be required. An exhaustive discussion is beyond the scope of this paper.

This paper does not explicitly address questions of security, fault-tolerance, or administrative domains. These should be addressed in the full-size implementation of the PA infrastructure. In order to tolerate machine crashes, the (logical) query servers can be replicated on several machines (e.g. using the *group communication* middleware [10]).

5 Related Work

Many authors present strategies for *locating* mobile objects and devices (see, e.g., surveys [11, 12]). Similar to locating objects are mechanisms for resource discovery, e.g. Dimakopoulos and Pitoura [13] describe cached-based distributed flooding approaches to locate a peer that provides a particular resource, with cache updates propagated either upon resource lookup or change.

Our work builds on the above, but is focused on the location-independent *message delivery*, which provides stronger properties than a pair of unsynchronized agent lookup and message sending actions. For instance, the FQSC algorithm guarantees that messages are not lost irrespective of agent migrations, and the upper bound on the number of hops required to deliver a message in case of local (within domain) migrations is known.

A number of agent systems provide a form of location independence; we briefly review some of them below. Comparisons are difficult, in part because of the lack of clear levels of abstraction and descriptions of algorithms – without these, it is hard to understand the performance and robustness properties of the infrastructures. Some mobile agent infrastructure algorithms are for locating agents only, which – as we explained above – provides weaker guarantees.

For instance, Mobile Objects and Agents (MOA) [14] supports four schemes for locating agents; these are used as required to deliver location-independent messages. Stream communication between agents is also described, with communicating channel managers informing each other on migration.

The MASIF proposal [15] also involves four locating schemes, but appears to build communication facilities on top. This excludes a number of reasonable infrastructures; it contrasts with our approach here, in which location-independent message delivery is taken as primary (some infrastructures do not support a location service).

The infrastructure work of Aridor and Oshima [16] provides three main forms of message delivery: location-independent using either forwarding pointers or location servers, and location dependent (they also provide other mechanisms for *locating* an agent).

Roth and Peters [17] propose a scalable global service for locating mobile agents, with encryption and decryption capabilities to prevent security attacks through agent impersonating.

The Join Language [18] provides location-independent messages using a built-in infrastructure, based on forwarding pointer chains that are collapsed when possible.

The Mobile Object Workbench [19] provides location independent interaction, using a hierarchical directory service for locating clusters of objects that have moved. There is a single infrastructure, although it is stated that the architecture is flexible enough to allow others.

Moreau [20] describes formally an algorithm for routing messages to migrating agents, which is based on distributed location directory service, with forwarding pointer chains that are collapsed when possible. In [21], he describes the directory extended with pointer redundancy to tolerate node crashes; the algorithm has been verified using the proof assistant Coq.

Our model assumes direct message routing, while other approaches are also possible, e.g. Murphy and Picco [22] present a distributed-snapshot-based algorithm. It attempts to deliver a message to every agent in the system using broadcast, and only the agents whose IDs match the message target actually accept the message. Cao *et al.* [23, 24] propose to separate agents and movable *mailboxes*, i.e. receivers of location-independent messages, with push and pull techniques that can be used by agents to obtain messages from their mailbox. They also discuss schemes to make the communication tolerant to mailbox crashes [23], and path compression for better performance [24].

The use of home servers in our FQSC algorithm resembles the *Internet Mobile Host Protocol (IMHP)* proposed by Perkins *et al.* [25] for transparent routing of IP packets to mobile hosts. By enabling sites to also cache bindings for mobile hosts (or mobile agents in FQSC) both protocols provide mechanisms for better routing which bypasses the default reliance on routes through the home server, and so they eliminate the likelihood that the home server would be a bottleneck. However, cache updates are performed differently, with FQSC optimizing the specific migration and communication pattern of PA agents. The FQSC protocol normally delivers messages to mobile agents in one-hop, while IMHP must route messages to mobile hosts via *care-of address* (which corresponds to the current local server of the target mobile agent in FQSC).

6 Conclusion

In this paper we have proposed a distributed algorithm for scalable location-independent message delivery to mobile agents, that is suitable for the Personal Assistants application. The algorithm reflects the expected behaviour of the Personal Assistant agents: communications are more frequent than migrations, and the inter-domain migrations, which correspond to delegation or a physical movement of individuals, are less frequent than migrations within a local domain.

Acknowledgments. We would like to thank Peter Sewell and Asis Unyapoth for many useful discussions. This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under KBN grant 3 T11C 073 28.

References

1. D. Chess, C. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? In *Mobile Object Systems – Towards the Programmable Internet*, LNCS 1222, pages 25–48. Springer, 1997.
2. Dejan Milojičić, Frederick Douglis, and Richard Wheeler, editors. *Mobility: Processes, Computers, and Agents*. Addison-Wesley, 1999.
3. David Kotz, Robert Gray, and Daniela Rus. Future directions for mobile agent research. *IEEE Distributed Systems Online*, 3(8), August 2002.
4. Luca Cardelli. Abstractions for mobile computation. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, LNCS 1603, pages 51–94. Springer, 1999.
5. Anand Tripathi, Tanvir Ahmed, and Neeran M. Karnik. Experiences and future challenges in mobile agent programming. *Microprocessors and Microsystems*, 25(2):121–129, April 2001.
6. Peter Sewell, Paweł T. Wojciechowski, and Benjamin C. Pierce. Location-independent communication for mobile agents: A two-level architecture. In *Internet Programming Languages*, LNCS 1686, pages 1–31. Springer, 1999.
7. Paweł T. Wojciechowski and Peter Sewell. Nomadic Pict: Language and infrastructure design for mobile agents. *IEEE Concurrency*, 8(2):42–52, April-June 2000.
8. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100(1):1–77, 1992.
9. Paweł T. Wojciechowski. Scalable message routing for mobile software assistants. Technical Report RA-010/06, Institute of Computing Science, Poznań University of Technology, May 2006. Available electronically at `http://www.cs.put.poznan.pl/pawelw`.

10. Sergio Mena, André Schiper, and Paweł T. Wojciechowski. A step towards a new generation of group communication systems. In *Proc. Middleware '03*, LNCS 2672, June 2003.
11. Vincent Wong and Victor Leung. Location management for next generation personal communication networks. *IEEE Network*, 14(5):8–14, Sept./Oct. 2000.
12. Evaggelia Pitoura and George Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):571 – 592, July/August 2001.
13. Vassilios V. Dimakopoulos and Evaggelia Pitoura. A peer-to-peer approach to resource discovery in multi-agent systems. In *Proc. CIA '03: the 7th Workshop on Cooperative Information Agents*, LNCS 2782, August 2003.
14. Dejan S. Milojičić, William LaForge, and Deepika Chauhan. Mobile Objects and Agents (MOA). In *Proc. COOTS '98: the 4th USENIX Conference on Object-Oriented Technologies and Systems*, April 1998.
15. D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF: The OMG Mobile Agent System Interoperability Facility. In *Proc. 2nd Int. Workshop on Mobile Agents*, LNCS 1477, September 1998.
16. Yariv Aridor and Mitsuru Oshima. Infrastructure for mobile agents: Requirements and design. In *Proc. 2nd Int. Workshop on Mobile Agents*, LNCS 1477, September 1998.
17. Volker Roth and Jan Peters. A scalable and secure global tracking service for mobile agents. In *Proc. of the 5th Int. Workshop on Mobile Agents*, LNCS 2240, pages 169–181, December 2001.
18. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proc. CONCUR '96: the 7th Int. Conference on Concurrency Theory*, LNCS 1119, August 1996.
19. Michael Bursell, Richard Hayton, Douglas Donaldson, and Andrew Herbert. A Mobile Object Workbench. In *Proc. the 2nd Int. Workshop on Mobile Agents*, LNCS 1477, September 1998.
20. Luc Moreau. Distributed directory service and message router for mobile agents. *Science of Computer Programming*, 39(2–3):249–272, 2001.
21. Luc Moreau. A fault-tolerant directory service for mobile agents based on forwarding pointers. In *Proc. SAC '02: the 17th Symp. on Applied Computing: Track on Agents, Interactions, Mobility and Systems*, March 2002.
22. Amy L. Murphy and Gian Pietro Picco. Reliable communication for highly mobile agents. In *Proc. ASA/MA '99: 1st Int. Symposium on Agent Systems and Applications / 3rd Int. Symposium on Mobile Agents*, October 1999.
23. Jiannong Cao, Liang Zhang, Jin Yang, and Sajal K. Das. A reliable mobile agent communication protocol. In *Proc. ICDCS '04: the 24th Int. Conference on Distributed Computing Systems*, March 2004.
24. Jiannong Cao, Liang Zhang, Xinyu Feng, and Sajal K. Das. Path compression in forwarding-based reliable mobile agent communications. In *Proc. ICPP '03: the 32nd Int. Conference on Parallel Processing*, October 2003.
25. Charles Perkins, Andrew Myles, and David B. Johnson. IMHP: a mobile host protocol for the Internet. *Computer Networks and ISDN Systems*, 27(3):479–491, December 1994.