RZ 2975 (#93021) 11/17/97
Computer Science/Mathematics 16 pages

IBM Research Report

Agent-based Multicast Routing

Stefan G. Hild and Jörg H. Bischof
IBM Research Division
Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland
{sgh, jrg}@zurich.ibm.com

IBM Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Abstract:

Agent systems are elegant, but it is difficult to find applications of practical importance that gain from being implemented with agent technology as opposed to conventional means: the prevailing opinion is that agents are a solution in search of a problem. In this short paper we state our definition of *agents*, briefly describe the *Aglet* agent system, and give details of an application that we believe is of practical importance and is more easily implemented with agent technology than other techniques: we introduce an agent that roams a network of arbitrary topology and configures a multicast network equivalent to the minimum spanning tree between the sender and the receivers.

1 Introduction

The emergence of Java's platform-independent byte code format has fueled research in agent technology. However, there appears to be some confusion within the research community with regard to the exact definition of 'agent'. In our view 'agents' are small pieces of executable code that can autonomously migrate from host to host across a network while preserving their (data) state and interact with their environment through abstract interfaces provided by the hosts. Thus, the properties we demand of an agent are *autonomy*, *mobility*, *data preservation* and *interactivity*. We prefer this definition over other conventions, such as the term *mobile agent* [BDJ96], which stresses the mobility aspects, or *intelligent agent* [WJ95], which stresses autonomy. We believe that both autonomy and mobility must be integral properties of agents, to whatever extent either is actually used for any particular application.

Although the agent architecture has much appeal, currently available implementations suffer in two respects: first, the environment provided to agents has to be agreed upon, standardized, and installed on all nodes within the network. Attempts to define such a standard have so far been less than successful [GMI95]. Second, for all its appeal it is difficult to find applications that utilize the full potential of the agent system (i.e. autonomy, mobility, data preservation and interactivity) and whose implementation as agents is intrinsically advantageous to an extent that justifies the additional effort over proven and readily available conventional client/server or other technologies [CHK97].

In this short paper we use the comparatively well established *Aglet* agent system [IBM] to implement such an application: we describe a model implementation of a network agent that establishes and dynamically maintains a minimum spanning tree network across a selected set of hosts on top of any network topology. We call our agent the *MST-Net Agent*.

We argue that our system not only utilizes the full set of facilities that define agent technology, but also uses these for an application of practical importance: minimum spanning tree topologies are desirable for a number of applications, such as multicast or groupware communications, but are difficult to establish with existing networking protocols [LGLA96, MBM96, Moy94, Dee91].

In Section 2 we briefly describe the general architecture of the Aglet system. The bulk of this paper is contained in Section 3, where we give details of our MST-Net Agent design. Section 4 describes our implementation and gives rudimentary performance data. Finally, Section 5 summarizes our work and briefly outlines possible future extensions to both the MST-Net Agent and the Aglet agent system itself.

2 The Aglet System

Aglets derive their name from *ag*ents and app*lets*. The Aglet system consists of the agent execution environment (which is called the *context*) and the aglets themselves. The context controls the execution of the aglets and provides the necessary abstractions for

```
public class AgletSkeleton extends Aglet {
    private int hopcount;

    public void onCreation(Object init) {
        hopcount = 0;
    }

    public handleMessage(Message msg) {
        if (msg.equals("GiveHopCount")) {
            AgletProxy p =
                getAgletContext().getAgletProxy(someURL, someAID);
            p.sendMessage(new Message(hopcount.toString()));
            return true;
        }
        return false;
    }

    public void run() {
        hopcount++;
        dispatch(someURL);
    }
}
```

Figure 1: The example skeleton aglet moves from location to location and returns the current hopcount on request. The source is incomplete and given for illustration purposes only.

the aglets to interact with their environment. Tools such as *Tahiti* (which provides a graphical user interface) or *Fiji* (which is web-based) are available to monitor the local context. Aglets are Java executables (which in turn are usually generated from a Java source) and are thus largely platform independent. The *Aglet Workbench*, which contains all the necessary tools, libraries and runtime systems to construct aglets, can be obtained as shareware from `http://www.trl.ibm.co.jp/aglets/`.

Figure 1 shows a skeleton aglet and illustrates its basic architecture. The context executes the methods defined in the aglet at certain stages during the lifetime of the aglet (the list is not exhaustive; for a complete introduction to aglets, see [LOKK97] and [LO]):

- the `onCreation()` method is called only once during the lifetime of each aglet immediately after the aglet is created by some parent aglet or the user;

- the `run()` method starts execution whenever the aglet is created (i.e. following the `onCreation()` method) or has arrived at a remote destination after a `dispatch()` call (see below);

- the `handleMessage()` method is executed whenever a message has arrived for the aglet.

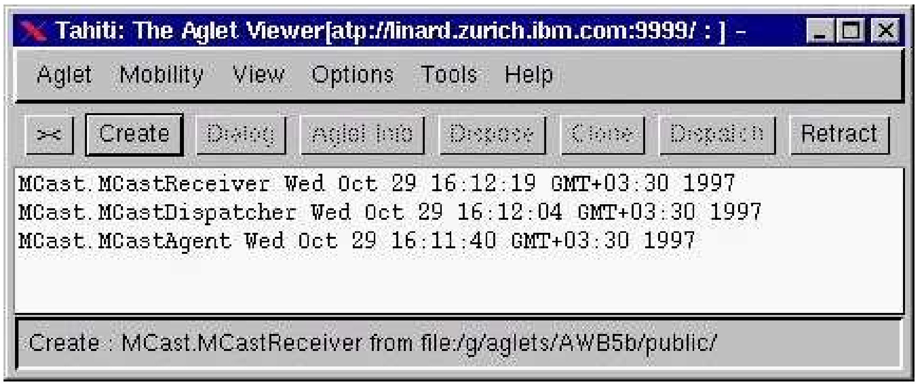


Figure 2: The Tahiti tool, showing three aglets running within the context on the local host.

The aglet's behavior is implemented by overriding the default methods defined in the `Aglet` class (see code skeleton in Figure 1).

Inter-aglet communication is possible through the `sendMessage()` call, which can be executed against the receiver aglet's local *proxy*. The proxy is retrieved by specifying the aglet's identifier and current location (see Line 10 in Figure 1).

Aglets can move to other locations by calling the `dispatch()` method with the destinations URL as the argument.

Figure 2 shows a screen dump of the Tahiti window interface, which signals the list of locally running aglets together with some additional information to the user.

3 The MST-Net Agent

Given the aglet's ability to distribute knowledge efficiently by roaming from node to node, it is tempting to use agents for network administration purposes. In our sample application, we use an aglet to distribute and configure a system consisting of a data source, receivers and dispatchers to generate an optimal multicast network infrastructure in terms of minimizing some network parameter for the entire network, for example cost, delay or transmission distance.

Each dispatcher receives data from either one other dispatcher or the data source, and forwards to a set of other dispatchers and/or receivers. There may only be one data source within the network at any given time (see Figure 3).

We assume that the parameter for which the network is to be optimized (we call this the *target parameter*) can be measured reliably by the agent while roaming from node to node. The problem of constructing minimum spanning trees is well studied, and a number of algorithms are available [GH85, GKK92, CGM80]. Our agent uses the well-known Prim's method [Pri57] to determine the dispatcher configuration.

Figure 4 shows a data source located at node *New York* and a set of receivers located

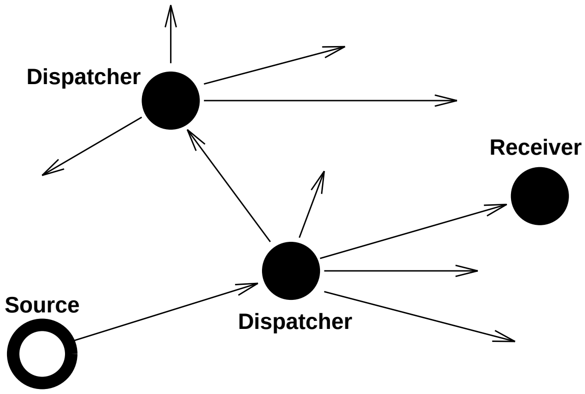


Figure 3: The multicast network layout consists of one sender and many dispatchers and receivers. Each dispatcher may send to many other dispatchers or receivers.

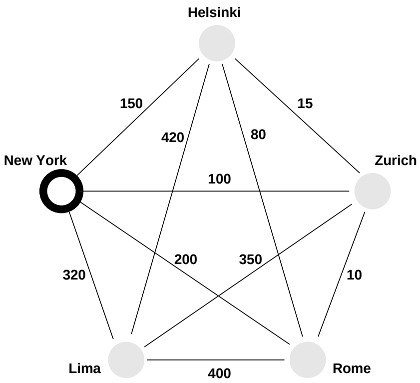


Figure 4: The initial configuration with one source New York, and destination locations Helsinki, Zurich, Rome and Lima.

at nodes *Helsinki*, *Zurich*, *Rome* and *Lima*. The underlying network links are also shown, together with the target parameter. The network is bootstrapped by the data source (New York), which generates the MST-Net Agent and passes to it a list of all receivers (we call this list the TODO list, as opposed to the DONE list, which holds the set of nodes already reached by the network and initially only contains the data source node). See Figure 5 for an illustration of the initial configuration.

Note that the aglets' data structures, which include the TODO and DONE lists and the link parameters measured by the agent, are retained as the agent moves from node to node: as we will see, the agent roams the network, takes additional measurements at each new node, and gradually builds up a complete table with all link parameters in the entire network.

This allows us to formulate the agent in a compact manner and removes the requirement for an extensive infrastructure (which would measure the target parameter of all local

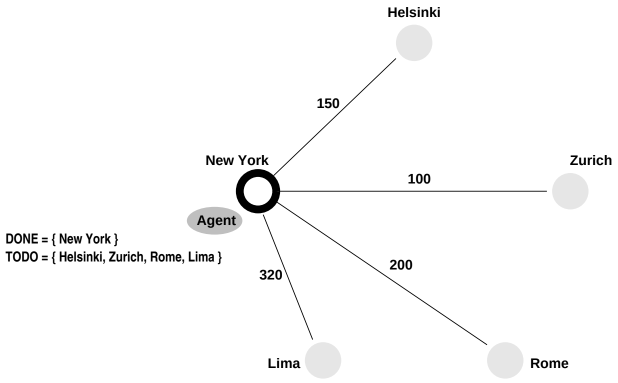


Figure 5: The source node New York measures the target parameter for all hosts in the TODO list.

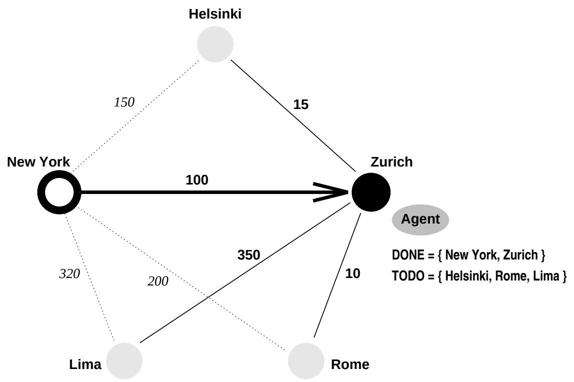


Figure 6: Adding a new link to the network.

links and report these measurements to a remote server) and a central server (which would receive and store this information).

The minimum spanning tree algorithm proceeds as follows:

- From its current location the MST-Net Agent determines the target parameter of all links from the current location to receivers not yet reached by the network, i.e. those nodes which are in the TODO list, and adds these measurements to its data structures.

- The agent selects the *best* link (in terms of the target parameter), which links a *new* node to the already established network, i.e. a node from the TODO list to a node from the DONE list, by examining all target parameters stored in its data structures. The two nodes being linked are the *source* node and the *target* node.

  In Figure 5 the link between New York and Zurich is chosen, resulting in the network configuration shown in Figure 6.

The agent does not necessarily have to reside on either the source or target node to establish the link.

- The agent then moves to the target node, i.e. the node just added to the network, taking with it its entire data structures and hence the state of the algorithm (data preservation).

The agent continues until all nodes are reached (as shown in Figure 7). At this point in time the network is guaranteed to be a minimum spanning tree between the data source (New York) and the set of receivers (Zurich, Rome, Helsinki and Lima) in terms of the target parameter.

The agent returns to the data source and awaits further instructions.

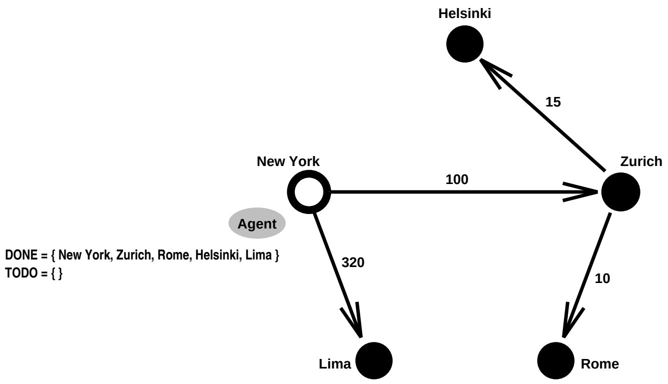


Figure 7: The minimum spanning tree over the given network in terms of a given target parameter.

Links are implemented by installing dispatchers and receivers as required, and updating the dispatcher configuration at the origin of the new link such that the origin is feeding data to the target location of the new link.

Should the target have previously been fed by another dispatcher, that dispatcher is notified to stop sending data to that location (this occurs if the topology is dynamically adapted to include a new receiver node, as described below). If a host serves as receiver and dispatcher, the dispatcher receives the feed and forwards the data to the local receiver in addition to any remote dispatchers or receivers. In Figure 7, node Zurich acts as receiver and dispatcher. The dispatcher at Zurich receives the feed from New York, and forwards data to both the local receiver and Helsinki and Rome.

Retaining the minimum spanning tree property while the network characteristics undergo usual fluctuations and/or nodes are added or removed requires a rerun of the algorithm. We describe such a rerun in detail in the following section for the specific case of adding a new node to the network, before briefly outlining the procedure for removing a node and adjusting to changing network characteristics.

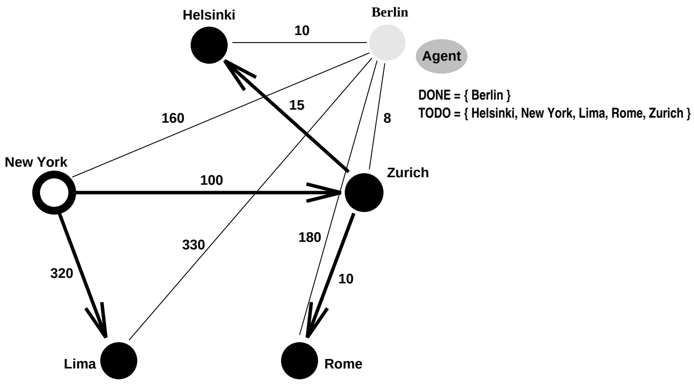


Figure 8: The network configuration at time of adding the new node Berlin.

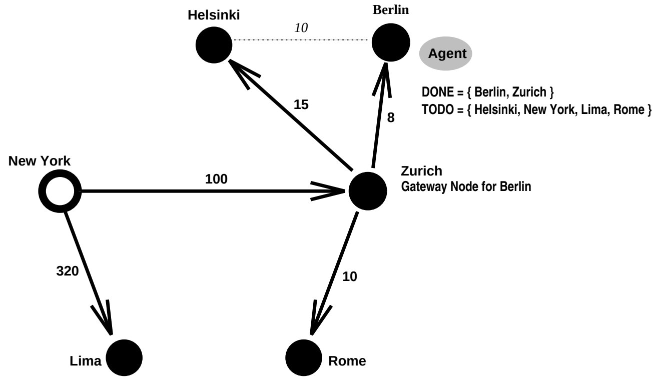
3.1 Adding a Node

The new receiver contacts the sender, and in turn is sent the MST-Net Agent. Figure 8 depicts the new node *Berlin* which is to be integrated into the network. Upon arrival at the new node, the agent determines the 'best' link into the old network (by measuring the target parameter of all links to all prospective entry nodes and determining the *gateway* node with the 'best' possible link), and establishes an appropriate delivery route (see Figure 8).

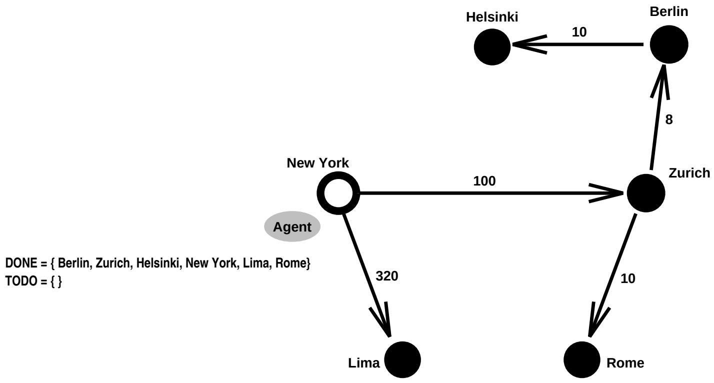
At this point in time, while the new node is linked to the old network through the best possible link, the network as a whole may no longer be a minimum spanning tree: other nodes within the new network may be reached more cheaply through the new node than through their existing feeder path. It is therefore necessary to continue the algorithm by setting the TODO list to contain the remaining network, the DONE list to contain the new node and the gateway node (see Figure 9). We then re-enter the minimum spanning tree algorithm outlined above.

By first establishing the link between the new node and the existing network we ensure that the new node is linked quickly and is receiving data. This allows us to postpone the execution of the bulk of the MST algorithm if desired (for example if the network load is to be kept at a minimum during critical operations), but have to accept a temporary loss of the minimum spanning tree property in the meantime.

Note that the algorithm is formulated such that links are only installed if they are new, i.e. were not already available, and that old feeder lines are removed as new links are configured. Also note that the direction of data flow must remain correct, i.e. from the data source outwards. Upon termination of the algorithm we can once again guarantee the minimum spanning tree property.

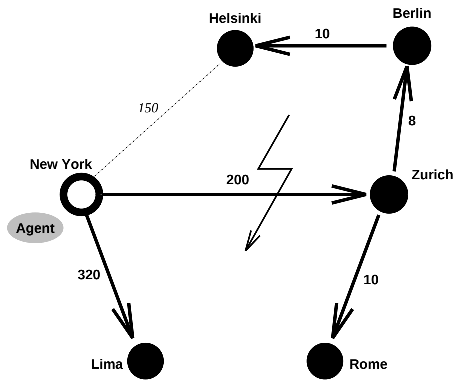


(a) Adding the new node Berlin to the existing network through the best possible link.

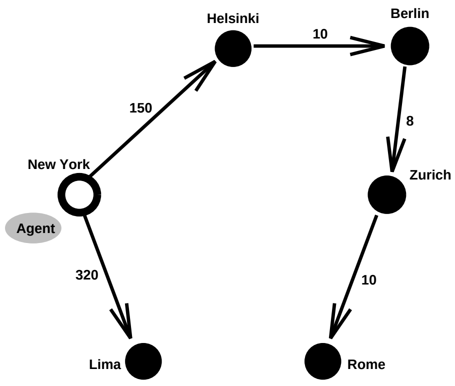


(b) The resulting new network configuration after reexecuting the minimum spanning tree algorithm.

Figure 9: Network configuration while adding the node Berlin.



(a) One target parameter of the network has changed.



(b) The resulting network configuration after adjusting to the new network characteristics.

Figure 10: The network configuration before and after a link parameter has changed.

3.2 Removing a Node

Nodes that serve only as receivers but not as dispatchers and are therefore leaves within the multicast network may be removed without impact on the remaining network (as each connected subset of a minimum spanning tree is still a minimum spanning tree). However, if the node serves as dispatcher and feeds other locations, the algorithm needs to be rerun to guarantee the minimum spanning tree property.

The process is similar to that described for adding nodes – in this case, however, we move the agent to the feeder node of the removed node before removing the dispatcher node from the TODO list and running the algorithm as described above in Section 3.

3.3 Adjusting to Changing Network Characteristics

Owing to fluctuations in network load, individual link performance in terms of the target parameter may change during the lifetime of the multicast network, as shown in Figure 10. Our system pays tribute to such fluctuations by triggering a rerun of the minimum spanning tree algorithm from the data source. The algorithm is rerun as described in Section 3. Existing links may be confirmed (i.e. no changes are necessary), or alternative links may be chosen, in which case the receiver, the old dispatcher and the new dispatcher are informed of the new configuration, as shown in Figure 10(b). The sequence of messages is such that no data is lost during the transition.

In our current implementation we trigger the rerun periodically. Alternative methods may be used. For example, each dispatcher can monitor the target parameter on each data link and trigger a rerun if performance drops below a certain threshold. The threshold itself may be defined by the agent at the time of dispatcher configuration, for example by setting the threshold for each installed link to the equivalent of the second-best choice. Once the performance of the configured link has dropped below the performance of the second-best link the dispatcher may reasonably assume that a better network configuration could be found and thus trigger the rerun of the minimum spanning tree algorithm.

Alternatively, the dispatcher can dynamically compare the performance of the chosen link with alternative links (the second-best choice, the n-best choices, or another randomly selected link) and trigger reruns as required.

Unfortunately, this requires that the dispatcher can autonomously monitor link performance and trigger reconfigurations. This may not be possible if the dispatcher is implemented as a distribution list within the node's native protocol routing table. It may thus be reasonable to define a *monitor* agent, which remains at each dispatcher location to perform this task, or visits selected dispatcher locations occasionally to perform the monitoring task before moving on to other dispatchers to avoid any degradation in dispatcher performance due to the additional workload imposed on each dispatcher location.

In practical implementations we expect that a mixture of techniques may be used. Our model implementation can easily be extended to use any of the described methods.

4 Implementation and Results

We have implemented the MST-Net Agent on top of IBM's Aglet agent system. For simplicity, we have also implemented the data source, receivers and dispatchers as aglets, although none of them is an 'agent' according to our own definition given in the introductory sections of this paper. They may therefore well be implemented as static programs. We hope that the dispatcher functionality will eventually be implemented as part of each host's native networking implementation.

In our first implementation we have opted to use the transmission delay as our target parameter. We determine the delay by measuring the roundtrip time of a UDP packet by using TCP/IP's ECHO facility on the target host.

Our source code is available for downloading at `http://www.zurich.ibm.com/~jrg`. For illustration purposes, we include the main loop of the minimum spanning tree algorithm contained in the MST-Net Agent in Figure 11. The system is bootstrapped by the data source, which reads the list of receivers from a file held in a specified location. We require that a context (through either Tahiti or Fiji) be available at all specified receiver locations, although our implementation is resilient against a number of network failure modes and will simply ignore receiver locations to which no connection can be established due either to link failure or the lack of an accessible aglet context.

We have conducted a number of tests with a source and multiple receivers located in IBM labs in the UK (*Mobilers* is located at IBM Hursley), Switzerland (*Linard*, *Klingenstock* and *Piot*, all located in Zurich) and Japan (*Artemis*, in Tokyo), and depict the

```
public void run() {
    // Agent is now at location ‘curLoc’
    if (TODO.size()>0) {
        // More work to do!
        while (TODO.size()>0) {
            // Measure target parameter to required destinations
            doMeasure(targetParam, curLoc, TODO);
            // Find the ‘best’ next link, anywhere in the network
            MCastLinkInfo nextlink =
                    findBestLink(targetParam, DONE, TODO);
            if (nextlink==null) {
                // trouble.  Host not reachable?
                System.out.println("*** No more hosts reachable.");
                TODO.removeAllElements();
                break;
            }
            // Establish link:  ‘nextlink.from()’ to ‘nextlink.to()’
            updateDispatcher(nextlink.from(), nextlink.to(),
                nextlink.from().feeder());
            // Update ‘DONE’ and ‘TODO’ lists
            DONE.addElement(nextlink.to());
            TODO.removeElement(nextlink.to());
            // Move on to next host.
            dispatch(nextlink.to());
            // Error:  We shouldn’t get here
            dispose();
        }
        // Now move back to Source
        dispatch(dataSource);
    }
    // All is done, inform data source.
}
```

Figure 11: Main loop of the MST-Net Agent.

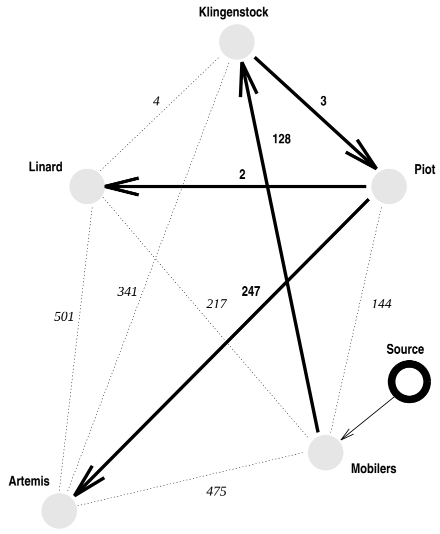


Figure 12: The network configuration with a source located in Hursley (UK).

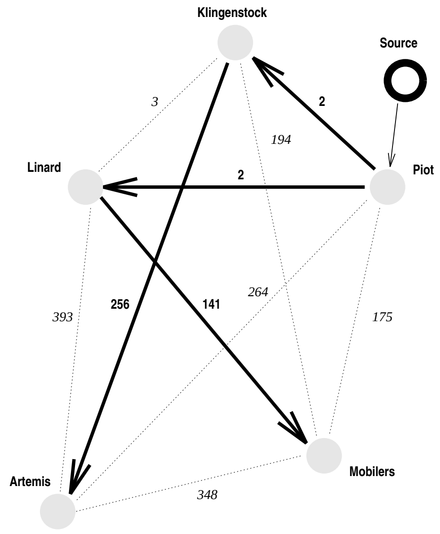


Figure 13: The network configuration with a source located in Switzerland (Zurich).

identified network configurations in Figures 12, 13 and 14. Please note that the three hosts in Switzerland are interconnected across a fast LAN; the measured ping times are therefore very small and the resulting topology between these hosts is randomized by very small fluctuations in these measurements.

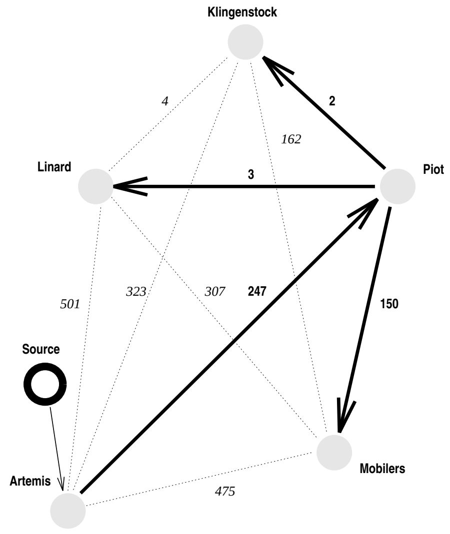


Figure 14: The network configuration with a source located in Japan (Tokyo).

5 Conclusions

We have presented an agent-based system that automatically configures a minimum spanning tree network between an arbitrary set of nodes on an arbitrary network. The system is truly agent-based in that it uses the agent's ability to roam the network (mobility), retains the algorithm's state while doing so (data-preservation), and installs and configures dispatchers autonomously based on parameters measured or retrieved from the local host (autonomy, interactivity). We believe the application is of practical importance; in our model implementation we target multicast applications, for example the Internet-based *M-Bone* [KV95, SRL96].

We also believe that non-agent-based implementations are less efficient: first, they require dedicated infrastructure to gather the global network state necessary for the minimum spanning tree algorithm as opposed to a generic and multi-purpose agent execution platform. Second, they either execute centrally at a central server which is a single point of failure, or impose high network loading while distributing global network state across the network to allow the minimum spanning tree to execute in a distributed fashion. In contrast, the agent-based solution combines the simplicity and compactness of a non-distributed algorithm implementation in a single MST-Net agent with the elegance and robustness of gathering global network state gradually and incrementally in a distributed fashion while the agent is roaming the network. This minimizes network traffic and exposes no single points of failure.

The distinction made between data sources and receivers in our implementation is largely arbitrary and may be removed with only insignificant modifications. The system we have described may therefore also be used for other applications, for example groupware communication [Eff95].

Algorithms other than minimum spanning tree algorithms may also be explored; for example, within redundant server setups it may be desirable that one server be accessible through the fastest possible route, whereas the connection time to the redundant systems is less important and may be neglected.

With regard to the Aglet technology applied for this work, we would like to see the system extended to provide better support for network applications. In particular, the Java-based model does not yet allow direct access to TCP/IP sockets. This, in turn, prohibits the application from accessing and retrieving topological information regarding the surrounding network infrastructure, and does not allow the aglet to roam to under-specified hosts (as for example, " ...move to my next neighbor").

We are now working on extending the aglet's reliability in terms of dependability and security. It is hoped that this in turn will lead to a more widespread adoption of the aglet system in particular and agent technology in general within the business domain. This will open up a whole range of exciting applications. We believe that applications will ultimately lead the development of agent technology by defining the list of required, redundant, desired and undesired features.

References

[BDJ96] Y. Berbers, B. De Decker, and W. Joosen. Infrastructure for mobile agents. In *Seventh ACM SIGOPS European Workshop: System Support for Worldwide Applications*, pages 173–180, 1996.

[CGM80] P. M. Camerini, G. Galbiati, and F. Maffioli. On the complexity of finding multiconstrained spanning trees. *Disc. Applied Math.*, 5:346–352, 1980.

[CHK97] D. Chess, C.G. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? In J. Vitek and C. Tschudin, editors, *Mobile Object Systems - Towards the Programmable Internet*, number 1222 in Lecture Notes in Computer Science, pages 25–47. Springer-Verlag, Berlin Germany, 1997.

[Dee91] S. Deering. Multicast routing in a datagram internetwork. Technical Report STAN-CS-92-1415, Stanford, December 1991.

[Eff95] Wolfgang Effelsberg. Multicast and group communication – paradigms and protocols. In *Proceedings of the International Conference on Multimedia Networking 1995 (MMNET '95)*, October 1995.

[GH85] R. L. Graham and P. Hell. On the history of minimum spanning tree problem. *Annals of History of Computing*, 1(7):43–57, 1985.

[GKK92] F. Glover, D. Klingman, and R. Krishnan. An in-depth empirical investigation of non-greedy approaches for the minimum spanning tree problem. *European Journal of Operation Research*, 3(56):343–353, 1992.

[GMI95] GMI. Telescript technology: Scenes from the electronic marketplace. General Magic Inc., 1995.

[IBM] IBM Japan. IBM Aglets Workbench. http://www.trl.ibm.co.jp/aglets/.

[KV95] Kumar and Vinay. *MBone: Interactive Multimedia on the Internet.* Macmillan Publishing, 1995.

[LGLA96] B. N. Levine and J. Garcia-Luna-Aceves. A comparison of known classes of reliable multicast protocols. In *Proceedings of the International Conference on Network Procotols, (ICNP-96)*, October 1996.

[LO] D.B. Lange and M. Oshima. Programming Mobile Agents in Java – With the Java Aglet API.

[LOKK97] D.B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming mobile agents in java. In T. Masuda, Y. Masunaga, and M.Tsukamoto, editors, *Worldwide Computing and Its Applications (WWCA97)*, number 1274 in Lecture Notes in Computer Science, pages 253–266. Springer-Verlag, Berlin Germany, 1997.

[MBM96] S. Maffeis, W. Bischofberger, and K. Mätzel. GTS: A Generic Multicast Transport Service. Technical Report UBILAB96.6.1, University of Zurich, June 1996.

[Moy94] J. Moy. Multicast extensions to OSPF. RFC 1584, Proteon Inc., March 1994.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, (36), 1957.

[SRL96] K. Savbetz, N. Randall, and Y. Lepage. *MBone: Multicasting Tomorrow's Internet.* IDG Books, 1996.

[WJ95] M. Wooldridge and N.R. Jennings, editors. *Intelligent Agents — Theories, Architectures, and Languages*, volume 890 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 1995.