AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Autonomous Reflectors over Active Networks: Towards Seamless Group Communication

Lidia Yamamoto and Guy Leduc

Research Unit in Networking, University of Liège
Institut Montefiore, B28, B-4000 Liège, Belgium
yamamoto@run.montefiore.ulg.ac.be ; leduc@run.montefiore.ulg.ac.be

Abstract

We present a reflector service that seeks to maintain application-level connectivity in the presence of network-level multicast failures. The service is based on the dynamic deployment of autonomous reflectors modelled as mobile agents on top of an active network infrastructure. It is able to repair multicast tree failures by building a self-organising tree of reflectors, which will be connected to each other via unicast. The scheme is decentralised and takes into account node and link resources to find agent locations that lead to low cost tree configurations. We focus on the basic decision mechanisms related to code mobility during the tree construction and destruction phases, namely: cloning, migration, merging and termination. We show some preliminary simulation results that confirm the viability of the approach and settle directions for further research.

1 Introduction

The demand for multimedia group communication is growing, and multicast is widely recognised as an important service to enable efficient group communication. However, multicast protocols still face deployment obstacles, and the quality experienced by many users is still unsatisfactory.

One of the fallback solutions used is to establish multicast reflectors to serve users that have no multicast access. A reflector is a user-level gateway application that acts as a proxy between a multicast-enabled network and a set of unicast users. It forwards packets from the multicast group to all unicast clients, and from every unicast client to the multicast group and to all other unicast clients. This guarantees that connectivity is maintained within the session, in spite of the fact that some participants have no access to IP multicast, or in the presence of failures in the multicast tree.

Existing reflector software must typically be installed manually, which is an extra burden for the session organisers and users. Besides that, even the session organisers seldom have enough knowledge about the current network conditions in order to be able to choose an optimum location for a reflector. What happens then is that reflectors are typically placed close to the main session source, and all multicast disabled participants must connect to it as clients. This generates an amount of redundant traffic which is proportional to the number of reflector clients, and therefore obviously does not scale to large sessions where potentially large portions of the network might need the reflector service.

It would be interesting to be able to dynamically install reflectors when there are connectivity failures or administrative restrictions to multicast traffic. The location of reflectors should be automatically determined according to the network conditions observed during the session.

We have designed an autonomous reflector based on mobile code that runs on top of an active network execution environment. The candidate locations for such reflector agents are active network (AN) (Tennenhouse et al., 1997) or active server (AS) nodes (Amir, 1998; Fry and Ghosh, 1999). These nodes run an execution environment (EE) capable of downloading and executing the reflector's code, and of discarding it when no longer used.

Our reflectors are autonomous and decide when to migrate to other nodes, clone in order to cope with increasing demand, merge with other reflectors, or disappear when no longer needed. The decisions are based solely on local knowledge available at the terminals or active nodes where they run. This guarantees that reflector code is deployed only where needed and when needed, and that after the session finishes all the reflectors will be automatically eliminated. The idea is that reflectors progressively move from the affected session members towards a well-known main centre of interest, until failure points are successfully bypassed, such that data coming from the main centre of interest can reach such members.

Additionally, reflectors that do not receive sufficient demand die out, and those which are overloaded spawn others to less loaded nodes. Using such a scheme, a tree of reflectors emerges as a result of failure detection, and disappears by itself when the failure is repaired. It should be noticed that the reflectors are not able to diagnose nor repair failures by themselves. Their objective is only to maintain application-level connectivity in the presence of network-level multicast failures. Network management mechanisms to detect and repair such failures are orthogonal and outside the scope of this work.

The paper is structured as follows: Section 2 gives a brief overview of the relevant concepts in our context as well as related work in the area. Section 3 describes the autonomous reflector scheme to build reflector trees. Section 4 explains the decision model that each agent adopts while building the tree. Section 5 shows some simulation results. Section 6 describes the current state of our Java implementation. Section 7 concludes the paper.

This article is an extended version of an earlier work (Yamamoto and Leduc, 2001a). Sections 2 and 3 have been reorganised and enhanced to clarify some ambiguities detected in the initial article. The merge procedure has been revised and is briefly described in Section 4.5.

2 Background

2.1 Multicast over the Internet

Multicast communication models for the Internet have received considerable attention since early 1990s. However, multicast protocols are still not widely available on the global Internet, and the experimental Multicast Backbone is slow to take off. Wide-area MBone sessions still fail due to multicast problems in some sections of the network. It is very difficult for the session participants to diagnose a failure and eliminate it during the lifetime of the session. The network conditions are unstable, it is difficult to monitor traffic and to detect points of failure, etc. The result is poor quality for the users.

There are several reasons for such a situation. One of them is the design of the protocols that usually requires modifications in the network routers and little support for incremental deployment. For a new protocol to be deployed over the Internet, it needs to be agreed upon, standardised, and manufacturers must implement compatible versions. Incremental deployment is difficult in this context, due to different paces of development and upgrade in different parts of the global network.

In the case of multicast, security concerns are also an obstacle to deployment, since multicast reinforces the risk that attackers easily flood the network with unwanted data. Therefore many providers are reluctant to allow IP Multicast in their networks, and firewalls can block multicast traffic. Some instabilities come from the fact that multicast is still considered as an experimental service in many places, and therefore it is given low priority over the operational tasks.

New network-layer solutions such as REUNITE (Stoica et al., 2000) and HBH (Costa et al., 2001) propose the use of the standard unicast addressing model to build multicast distribution trees, such that unicast-only regions can be supported in a transparent way, and therefore facilitate incremental deployment. However, such solutions still require compatible peers, and would need to be agreed upon and standardised as other protocols, before being deployed. While these new protocols are being discussed, application-layer solutions such as Narada (Chu et al., 2000) offer the users an alternative to waiting for a larger scale availability of network-layer solutions. However, due to the lack of network support, application-layer solutions are difficult to implement and often lead to inefficient overlay topologies.

2.2 Active Networks and Active Servers

Many of the deployment difficulties described for multicast are also shared by other Internet protocols such as IPv6, Mobile IP, etc. Research on Active Networking (AN) (Tennenhouse et al., 1997) came as a response to such deployment difficulties, among other motivations. Active networking enables the dynamic deployment of protocols and services over a set of programmable routers. The nodes of an active network are capable not only of forwarding packets as usual but also of loading and executing mobile code. The code can come in the form of active extensions or capsules. Active extensions are complete modules that implement a given service, while capsules contain small pieces of code (or a reference to the code) that are executed in every AN node they visit.

Since AN raises security issues which are still being studied and debated upon, some researchers have proposed the Active Server (AS) approach (Amir, 1998; Fry and Ghosh, 1999) as a shorter term alternative to AN. AS nodes are end systems that allow the secure downloading of mobile code such that new services can be deployed on-demand.

2.3 Existing Reflector Systems

Several commercial and non-commercial reflector systems are available, e.g. (Highfield, 1998; Live Networks, Inc., 2000; Kirstein and Bennett, 2000) (more references on p.27 of (Kon et al., 2000)). These systems are typically software packages that must be manually installed at the sites that will provide the reflector service. Therefore the location of reflector sites must be decided beforehand, and cannot be easily changed during the lifetime of the multimedia session. Changes in reflector location or configuration generally lead to temporary service disruption.

In (Baldi et al., 1998) mobile reflectors that can clone or migrate appear as part of a videoconference architecture for active networks. The authors focus on software design issues, and the actual algorithms and criteria for placing such reflectors are not covered.

A multicast reflector has been mentioned as an example application over the AS environment ALAN (Fry and Ghosh, 1999). It has been used to transmit MBone sessions via unicast to sites not connected to the MBone, and has the potential to move across ALAN nodes. However, dynamic reflector placement algorithms seem not to have been proposed in this context yet.

In (Kon et al., 2000) a distributed framework to manage a network of reflectors is proposed, based on dynamic code distribution and (re)configuration. Reflectors are used to support network and terminal heterogeneity. Within this framework, it is possible to manage networks containing a large number of reflectors. Each reflector has a limited degree of autonomy, such as reconfiguring the neighbour nodes to bypass a reflector that failed. For most other operations, however, the reflector elements need to be managed by a privileged user, the reflector administrator, who decides where to install new reflectors or to remove reflectors from the network. To take such decisions, the administrator needs to have a global view of the network topology and characteristics, which is not trivial to obtain from the wide-area Internet or the MBone. Another difficulty is to cope with participants dynamically joining and leaving the session, and the corresponding reflector tree reconfiguration in order to maintain a tree which always tracks the optimum. (Kon et al., 2000) report that in one experiment they were forced to deny approximately one million connection requests due to lack of bandwidth on the reflector sites. This could have been avoided if reflectors were able to automatically clone themselves to other sites in order to cope with the additional demand.

2.4 Market-based resource control

A considerable amount of research results in the area of market-based control are available mainly in the agents field (Clearwater, 1996). It provides algorithms inspired by optimisation and economy theories for distributed control of resource usage, with many applications to computer and telecommunication networks. In (Gibney et al., 1999) a market-based mechanism to set up circuit switching paths with resource reservation is described. Closer to the AN perspective, in (Tschudin, 1997) an open resource allocation scheme based on market models is applied to the case of memory allocation for mobile code.

In (Najafi, 2001) a cost model for active networks is proposed, that takes into account the cost of processing a flow in the active nodes as well as the transmission costs. The author includes an algorithm that converges to the minimum cost for a flow that may be transformed in several active nodes before reaching its destination. He has also proposed a single agent positioning algorithm in which an agent can decide to reposition itself in the network in order to reduce the session cost.

2.5 Placing functionality and routing in active networks

The problem of choosing active nodes to place a given programmable functionality has been identified as a routing problem (Najafi, 2001; Choi et al., 2001). The potential of active routing is broadly discussed in (Maxemchuck and Low, 2001). The extensive simulation results in (Kiwior and Zabele, 2001) show that the performance of a reliable multicast protocol that makes use of active network nodes heavily depends on the location of the active nodes.

Several proposals in this area such as (Akamine et al., 2000; Duysburgh et al., 2000; Safaei et al., 2001; Choi et al., 2001; Partridge et al., 2001) require knowledge of the whole network graph in order to compute the active paths. Solutions of this kind are feasible to place generic services to be used by wide range of active applications within a domain. But they generally do not scale beyond a single domain, and are not feasible when the applications themselves need to find optimum locations for their active elements, depending on their own specific characteristics and constraints.

In (Wen et al., 2001) a framework is proposed for composing customised multicast protocols for active networks out of elementary building blocks. Our approach could benefit from such a framework to build the reflector service using similar blocks.

2.6 Self-deploying services

In the context of agents and active networking, a number of proposals for self-deploying services have been made. In (Shehory et al., 1998) a framework is proposed in which agents deal with overload by cloning, passing tasks to others, merging, or dying. Agents decide when to clone according to the loads of the different resources they use, such as memory, processing and communication resources. The possible decisions that the agent can take are described by a decision tree, and the optimum decision is calculated via dynamic programming. In (Tschudin, 1999a) an election service based on active packets is developed, that deploys itself to every reachable node. In a later work (Tschudin, 1999b) the same author addresses the security issues involved with a necessary self-destruction mechanism for such kind of services.

In (Roadknight and Marshall, 2000) the issue of quality of service differentiation is addressed by using a distributed genetic algorithm inspired by the behaviour of bacteria. The authors show that the amount of servers and their location in the network evolve according to the user demand for a given type of service and a requested trade-off between latency and packet loss. The potential of genetic techniques such as the ones proposed by (Roadknight and Marshall, 2000) is the increased variability to find new solutions and adapt to new situations not envisaged at the beginning. However, in an environment where nodes and links are heterogeneous, propagating successful rules ("genes") to neighbouring nodes might not necessarily be a good idea, since a rule that is successful in one node might fail completely in another node due to different resource constraints. In the context of self-organising systems, biologically-inspired and market-based techniques seem complementary, and an interesting research challenge would be to combine the best of both worlds to obtain new adaptation mechanisms.

3 Autonomous Reflectors

In this section we describe our autonomous reflector scheme. We start with some definitions and assumptions, and then describe the basic mechanisms for building and destroying trees of reflectors.

3.1 Definitions and terminology

- Reflector, or reflector agent: Software package that implements the reflector functionality and can be loaded on the active nodes on-demand. Each reflector is uniquely identified within the session. Each active node may hold only one instance of a reflector for a given session (although several reflector instances for different sessions running on the same node may share the same code).

- Client reflector: Reflector A is said to be a client of reflector B when A is a child of B on the tree.

- Server reflector: Reflector B is said to be a server for reflector A when B is A's parent on the tree.
- Connection: A logical client-server connection between two reflectors A and B. Packets are exchanged over this logical connection using direct unicast addressing.
- Terminal reflector: A reflector that is a leaf of the tree.
- Intermediate reflector: A non-leaf reflector.
- Root reflector: The root of a reflector tree. It is a reflector agent that is either located at the RP or receiving data from the RP via native multicast.
- RP or rendez-vous point: Predefined target node towards which the reflector trees will be built.
- Downstream: The flow direction from the RP towards the root reflector, or from the root reflector towards the leaves of the reflector tree.
- Upstream: The direction from the tree leaves towards the root reflector or from the root reflector towards the RP.
- Clone: To send a copy of a reflector to another node; the clone will initially have its original reflector as a client.
- Migrate: To move to a given destination node, by sending a clone there, transferring the agent state to the clone (mainly its client list) and terminating.
- Merge: To combine two reflectors into a single one, by merging their client lists and terminating one of the reflectors. A merge operation is often used only as an abstraction, as two reflectors that intend to merge may negotiate a different configuration before the merging actually takes place (see Section 4.5).
- Terminate: To terminate the execution of a reflector at a given node (its code may remain cached for a certain amount of time until it is garbage collected).

3.2 Assumptions and Limitations

We assume that some basic default unicast routing service interconnecting all active nodes is available, such that at any moment it is possible for an active application to obtain the next hop to a given destination. This service can be either provided by the Execution Environment itself, or installed as active extension code with a well-defined interface exported to the active applications that need it. By default it can simply map directly to IP routing, but more sophisticated techniques such as application-layer routing (Ghosh et al., 2000) could also be available to provide optimised paths according to specific criteria. We assume that unicast routing is robust: rerouting around failed unicast links is out of the scope of this work.

The current failure detection mechanism is very simple: a reflector simply attempts to join the multicast group, listen for a while, and if no multicast packets arrive then it assumes that there is a failure. It also assumes a failure when there is an error while attempting to join the group at a given network interface, meaning that no multicast support is available for this interface.

Another assumption is that there is only one main centre of interest that generates content for the session (e.g. the lecturer's site). This centre of interest or main source will also be called the rendez-vous point (RP). All other session participants are also allowed to generate content to the session, as it is generally the case with RTP sessions (Schulzrinne et al., 1996), that nowadays are widespread on the MBone. So the system is not constrained to Single Source Multicast. However, these other sources of data will be considered as secondary from the point of view of the mobile reflectors, which means that when there is a multicast failure affecting only the reception of secondary sources, the system of reflectors will not attempt to repair it.

It is assumed that all session members learn the RP address in advance, together with other group information, that is generally advertised by standard session announcement mechanisms such as SIP or SAP (Handley et al., 1999; Handley et al., 2000), which are outside the scope of the paper.

One might argue that assuming a single centre of interest is not a realistic approach, but in practice it is often the case that a single centre of interest exists or can be defined close to where most of the "action" occurs in the session. Besides that, if multicast is down for a particular member, it is likely to be down for other members too, but detecting it for each member individually would be too costly for large sessions. Therefore the repair tree of reflectors is bidirectional, so that all participants affected by a failure share a single tree to distribute and receive content to/from the rest of the session.

The reflectors we propose are only able to repair multicast trees using unicast: multicast tree failures will cause a tree of reflectors to be formed, which will be connected to each other via unicast. Since multicast routing and active network unicast routing are independent, the unicast tree of reflectors might not coincide with the corresponding multicast subtree for a given set of session members. One can imagine that it would be interesting to use unicast only to bypass "broken" segments of a tree, using multicast everywhere else. This would result in a significantly smaller amount of reflectors being deployed. While this is an interesting possibility, it raises many new difficulties related to the self-organisation of disjoint subgroups within a session, with corresponding allocation of multicast group addresses, underlying topology discovery to make sure subgroups don't overlap, etc. Therefore we leave this possibility open for future study.

3.3 Tree Construction

The reflectors start at leaf nodes co-located or close to the session members, and then progressively move, clone and merge with other reflectors along their respective unicast paths towards a rendez-vous point (RP). As discussed in the previous paragraph, the RP role is typically assigned to the main source of content in the session. This leaf-initiated approach is similar to filter placement schemes based on RSVP, such as the AMNet prototype described by (Wittmann et al., 1998).

We distinguish two types of reflectors: terminal and intermediate. Terminal reflectors are located as close as possible to the end systems and do not move, while intermediate reflectors are dynamically placed on other active nodes in the network, and might move from one node to another according to the network conditions.

A terminal reflector ideally serves one local client, which is the user application that generates and/or treats session content (e.g. MBone tools such as `vic` or `rat`). A terminal reflector must be installed at each end system that wishes to make use of autonomous reflectors, or as close as possible to the end system (or set of end systems) to be served. The terminal reflector works as a proxy between the actual multicast group and the user application, so that the direct use of multicast or the use of reflectors is hidden from the application. This allows the use of reflectors based on mobile code without requiring any change to existing applications. Terminal reflectors are intermediate reflectors that have their migration rules disabled. They must be fixed because the existing tools are not able to detect moving peers.

The tree of reflectors organises itself in a client-server hierarchy. Each intermediate reflector serves a number of clients that are located downstream from it and directly connected to it via unicast. All intermediate reflectors are both servers for a number of clients and clients of an upstream reflector, except the root of the tree which only acts as a server.

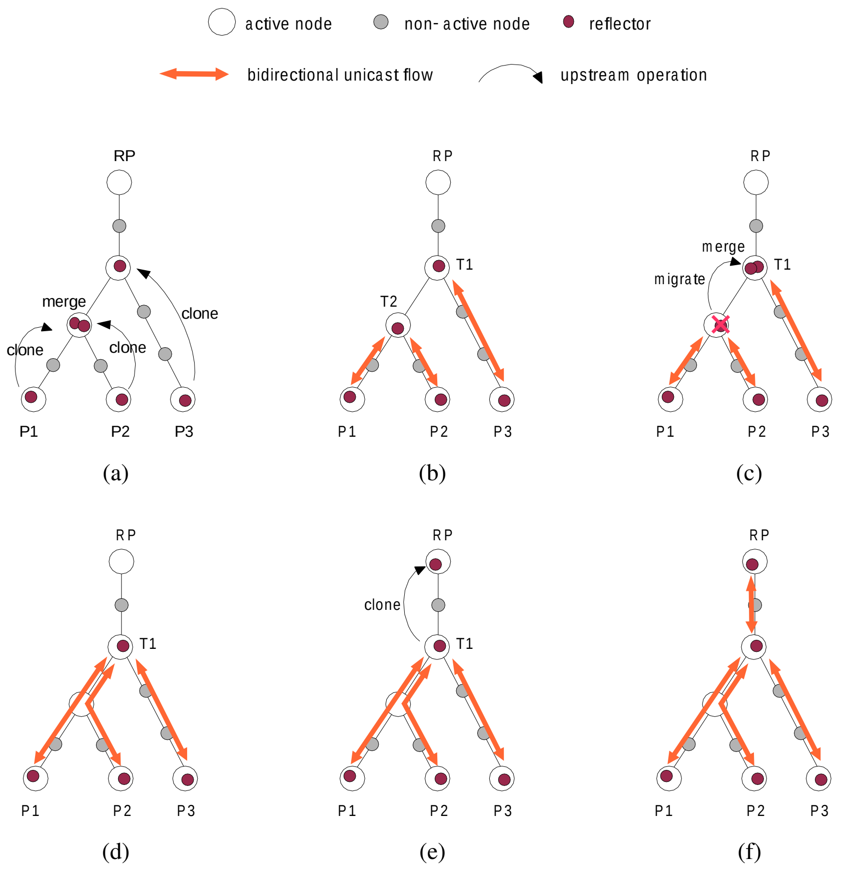

Figure 1: Tree construction example.

Figure 1 shows a tree construction example for a session consisting of an RP and three session members P1, P2, and P3. When a participant detects the absence of native multicast connectivity to the RP (by trying to listen to the multicast group as described in Section 3.2), its terminal reflector sends another reflector to the next AN hop towards the RP. This operation is called *upstream cloning* (Fig. 1(a)). The clone reflector spawned in this way is not an exact copy of the original reflector, since it is an intermediate reflector, but it serves the same session and has the same goals. In our case, the goal is to maintain application connectivity when native multicast fails or is absent, such that at least the data coming from the main session source reaches all session members.

When two reflectors belonging to the same session meet at the same active node, they merge into a single reflector (Fig. 1(a)(c)). Two reflector trees, T1 and T2 (Fig. 1(b)), result from the operations shown in Fig. 1(a). If native multicast traffic from the RP is detected by an intermediate reflector, it becomes a root reflector and stays in the node.

Otherwise, it either clones or migrates to the next active hop towards the RP. Figure 1(c) shows an *upstream migration*. A hierarchy of reflectors results from this process (Fig. 1(d)(e)(f)).

Since each agent reflects every packet received, the result is a bidirectional shared tree such as CBT (Ballardie, 1997). For example, on Figure 1(d) if T1 is receiving multicast from RP, then the flow coming from RP will be sent in unicast to P1, P2, and P3; and the flow coming from P1 will be copied to P2, P3, and to the multicast group.

While network-layer protocols typically deal with outgoing interfaces, the reflectors deal with unicast client connections directly: a copy of each packet is sent to each client, even when several clients share the same outgoing interface. A packet is never looped back to a client, even in the presence of route and interface asymmetry such that the incoming and outgoing interfaces to a given client are different. This strict hierarchy must be observed at all times in order to ensure that loops in the bidirectional tree do not occur.

Each reflector includes a local selector that selects data from either multicast or unicast channels, to ensure that no duplicate packets are forwarded downstream, and that packets coming from downstream are forwarded only to the selected channel (either multicast or parent reflector).

Applying this bottom-up tree construction algorithm, a reflector that succeeds bypassing the failure point becomes the root of its reflector tree. This method that not only one tree but several ones might arise in response to an absence of native multicast, in case multiple reflectors cross a failure point at different nodes. This can happen, for example, if the failure "point" is not a single link but a whole network with multicast capabilities disabled for some reason.

The repair trees will be located as close as possible to the concerned participants, and will not interfere with the rest of the session running in native multicast. The drawback is that the tree is built following the reverse path to the RP, which might lead to suboptimal downstream paths when routes are asymmetric.

3.4 Termination

A reflector that runs out of clients automatically terminates itself. If native multicast connectivity is somehow restored, reflectors start receiving data from the main source via the multicast channel, and disconnect from their upstream reflectors. The latter will eventually die out due to lack of clients. Each reflector contains a local selector module that is responsible for discarding duplicate packets, and for ensuring that packets going upstream are forwarded to the selected channel (either native multicast or parent reflector).

If a reflector terminates abnormally, its children will detect the absence of traffic coming from their parent and will restart the tree construction process to rebuild the affected portion of the tree.

3.5 Communication among reflectors

AN Capsules are used to implement a signalling mechanism among neighbouring reflectors so that a given intermediate reflector can inform its downstream clients of its current location, and to keep the reflector tree alive. Capsules are also used to prospect the state of an upstream node before making a decision to clone or migrate, and to enable two agents to take merge decisions jointly, as will be explained in Sections 4.4 and 4.5.

4 Reflector decisions

Reflectors can use resource control based on market mechanisms (Clearwater, 1996) to make decisions to either clone, merge, migrate or terminate themselves. Such mechanisms can also be used to dynamically decide on the maximum number of clients to accept at a given machine, in order not to cause link or CPU overload. Another usage is to make downstream cloning decisions: for example, in the case of an overloaded reflector, a number of clones can be sent downstream to handle part of the clients.

In this paper we concentrate on the basic decision mechanisms related to code mobility during tree construction and destruction, that is, cloning, migrating, merging and terminating. We are currently working on the additional mechanisms related to tree reshaping and load control.

Each reflector has costs associated with its consumption of node and network resources. The tree of reflectors is organised in a client-server hierarchy, such that server reflectors sell session data to their clients, and buy data from their server reflector. A reflector uses the revenues that come from its clients to pay for resource usage in the active nodes and for the services of the upstream reflector.

4.1 Resource usage costs

Each reflector has associated fixed costs and variable costs for the use of node resources. The fixed costs do not vary with the number of clients that a reflector has, and correspond to the costs of using the mobile code platform. They represent the minimum processing plus storage cost that the mobile agent incurs, even when no clients are connected. Note however that the fixed costs are not constant in general, as they may vary as a function of the load level of the resource in question (CPU, memory).

The variable costs increase with the number of clients, and correspond to the link transmission costs to all clients, plus the processing costs for all packets. These costs may also vary according to the total load of the corresponding resource (link bandwidth or processing).

From the cost point of view, having many clients is good for a reflector because the fixed costs are shared among all the clients, but if the number of clients becomes too large, the demand for one or more resources might exceed the supply (congestion situation), leading to an increase in processing and link prices, with possible packet losses and consequent degradation in quality for the end user.

Every time a new reflector is added to the tree, there is an increase in costs corresponding to the resources that the new reflector needs. However, this increase might be compensated by a decrease in costs for other reflectors, e.g. because their load is alleviated.

The cost of processing at an active node is also related to the delay penalty imposed to the end user due to the use of a tree of reflectors. The delay penalty is the ratio between the actual delay experienced by an end user and the delay that would be experienced if the user could connect directly to the multicast session without the help of reflectors. If the processing power were infinite, the extra delay imposed by a reflector would be null. On the other hand, a very low processing power would incur a high additional delay. The same is valid for a machine with high processing power but which is overloaded, such that the processing time available to a reflector is very low. Therefore if a reflector tries to choose nodes that have low processing costs, it is likely to be moving towards a lower delay penalty for its users.

4.2 Definitions

We begin by providing some definitions of terms that will be used later in this section:

n_i: a reflector that runs at a given node i
nc: number of clients of reflector n_i
$r_{i,j}$: reflector j, the j-th client of reflector n_i, for $j = 1..nc$
$R_{i,j}$: data sending rate of reflector $r_{i,j}$ to n_i (upstream direction)
S: data sending rate of the main source
nk: number of terminal reflectors in session

SR: total rate of the session (session bandwidth)

$$SR = S + \sum_{1 \leq k \leq nk} R_k \tag{1}$$

for all terminal reflectors r_k with sending rate R_k.

cf_i: fixed costs at node n_i (do not vary with nc).
$cv_i(nc)$: variable costs at node n_i (vary with nc).
$cvp_i(nc)$: processing costs at n_i; depend on the amount of data treated per second.
$cvl_i(nc)$: total link costs at n_i: represent the costs associated with the total amount of bandwidth emitted by reflector n_i to each link that leads to clients of n_i, and to the parent reflector if any.

$ct_i(nc)$: total cost at node n_i when nc clients are present: the sum of fixed and variable costs, as follows:

$$\begin{aligned} ct_i(nc) &= cf_i + cv_i(nc) \\ &= cf_i + cvp_i(nc) + cvl_i(nc) \end{aligned} \tag{2}$$

4.3 Estimating costs

In order to make a decision to either clone or to migrate, a reflector first needs to estimate the costs that would result from choosing either option. A simple decision strategy would then be just to choose the configuration with the lowest cost. However, there are a number of difficulties in obtaining such estimation. Actually this is a typical problem of making decisions in the presence of risks, and decision analysis could be applied here as in (Shehory et al., 1998). In this section we present a first simplified approach to the problem. Further research is necessary in order to extend it to a more general case.

One of the main difficulties is that, at the beginning, when the reflector still hasn't reached the main source (directly via multicast or via another server reflector), it is not able to measure the actual resource consumption that will result when it reaches it. When that happens, it goes into full operation mode, but at this moment, it is too late to revise its previous decisions concerning cloning or migrating. Especially, if the reflectors underestimate the aggregate sending rates of all the session members beyond the multicast failure point while building the tree, several points of congestion might appear as soon as the tree becomes fully operational.

A solution to this problem would be to rely on an estimation of the total rate of the session (session bandwidth), that must be available somehow before the session starts. In practice it is possible to obtain such information by looking at the media types in the

SDR session announcements. Additionally, if RTCP is used (Schulzrinne et al., 1996), and assuming that only a limited number of session members send significant amounts of data to the group, the session bandwidth grows very little with the total session size.

Using such an upper bound, resources could be reserved at the active nodes along the path in order to guarantee that enough resources are available when the reflectors reach the main source. However, resource reservation might not be available at all nodes, and most of the nodes might not even be active. Besides that, if the session bandwidth is overestimated, too many costs might incur with little extra benefit for the end user. We adopt a simple solution that relies on an upper bound on the session bandwidth to simplify the cost calculations, but does not reserve resources on the nodes.

Now we quantify each cost component in our context. We begin with the processing costs.

4.3.1 Processing costs

Network packets constitute the bulk of the data treated by a reflector. Therefore the processing costs during a given interval increase with the number and size of the packets treated. Every packet received is reflected to everyone else. Thus every packet from the upstream channel (reflector or multicast) is copied to every client, and every packet from a client is copied to the upstream channel plus all the other clients except itself. For a reflector that has already reached the main source (directly via multicast or via another server reflector), the total number of bits per second treated at n_i is:

$$st_i(nc) = sp_i(nc) + sc_i(nc) \tag{3}$$

where:
$sp_i(nc)$ is the data rate sent from the parent to all child reflectors of n_i
$sc_i(nc)$ is the data rate sent from all child reflectors of n_i to all others and to the parent.

$$sp_i(nc) = nc \cdot (SR - \sum_{1 \le j \le nc} R_j) \tag{4}$$

$$\begin{aligned} sc_i(nc) &= \sum_{1 \le j \le nc} ((nc-1) \cdot R_j + 1 \cdot R_j) \\ &= nc \cdot \sum_{1 \le j \le nc} R_j \end{aligned} \tag{5}$$

Substituting equations 4 and 5 in 3, we have:

$$st_i(nc) = nc \cdot SR \tag{6}$$

Assuming constant prices, and processing costs that increase linearly with the data rate treated, we have:

$$\begin{aligned} cvp_i(nc) &= pp_i \cdot st_i(nc) \\ &= pp_i \cdot nc \cdot SR \end{aligned} \tag{7}$$

where:
pp_i is the (constant) processing price per bit per second at node n_i.

4.3.2 Link costs

The link usage costs include the costs for bandwidth and queueing. Here we consider only the bandwidth costs for simplification. There are only costs associated with the transmission of packets, not with the reception of packets. Thus the link costs are the sum of the costs to reflect a packet from the parent reflector to all child reflectors, and from each child to every other child plus the parent.

Assuming constant link prices, the total link cost for n_i can be written as:

$$cvl_i(nc) = pl_{i,p} \cdot \sum_{1 \leq j \leq nc} R_j + \sum_{1 \leq j \leq nc} (pl_{i,j} \cdot (SR - R_j)) \tag{8}$$

where:
$pl_{i,j}$ is the price per unit of bandwidth on the link in n_i that leads to the client $r_{i,j}$.
$pl_{i,p}$ is the price per unit of bandwidth on the link in n_i that leads to the parent reflector of n_i (or the candidate parent in case a decision to clone or to migrate is about to be made).

If the link price is the same for all clients and equal to $pl_{i,l}$, or when all clients of n_i share the same link l, we can rewrite the link cost as:

$$cvl_i(nc) = pl_{i,p} \cdot \sum_j R_j + pl_{i,l} \cdot (nc \cdot SR - \sum_j R_j) \tag{9}$$

4.3.3 Cost of the cloning configuration

If we are going to send a clone from the origin node n_i to an upstream destination node n_d, the cost of the resulting clone configuration can be calculated as:

$$\begin{aligned} ctc_{i,d} &= ct_i(nc) + ct_d(1) \\ &= cf_i + cvp_i(nc) + cvl_i(nc) + cf_d + cvp_d(1) + cvl_d(1) \end{aligned} \tag{10}$$

Here $ct_i(nc)$ represents the total cost of running the agent at the current node when the agent is fully operational, while $ct_d(1)$ is the cost of a new agent running at the upstream node n_d with a single node (n_i) as a client.

4.3.4 Cost of the migration configuration

When migrating to an upstream destination n_d, a reflector n_i carries its client list along with it. Assuming symmetric unicast routing paths, the traffic will continue to go through node i, therefore consuming the same amount of bandwidth resources at the links leading to each client reflector. Since the reflector itself will disappear from node n_i, there are no fixed nor processing costs associated with it anymore at this node. Therefore the cost of resulting configuration after migration can be calculated as:

$$\begin{aligned} ctm_{i,d} &= cvl_i(nc) + ct_d(nc) \\ &= cvl_i(nc) + cf_d + cvp_d(nc) + cvl_d(nc) \end{aligned} \tag{11}$$

4.4 Making a decision

We would like to make a decision to either clone or migrate based on the total costs of resources for each configuration.

A simple decision strategy is to choose the configuration with the lowest cost:

$$\text{if } ctm_{i,d} > ctc_{i,d} \text{ then clone else migrate.} \tag{12}$$

In order to simplify the calculations, we rewrite the above rule as:

$$\text{if } ctm_{i,d} - ctc_{i,d} > 0 \text{ then clone else migrate.} \tag{13}$$

The costs of each configuration are given by equations 11 and 10, therefore we have:

$$\begin{aligned} ctm_{i,d} - ctc_{i,d} &= cvp_d(nc) - cvp_d(1) - cvp_i(nc) \\ &\quad + cvl_d(nc) - cvl_d(1) - cf_i \end{aligned} \tag{14}$$

Note that the fixed costs at n_d, as well as the link costs at n_i have disappeared since they are the same in both configurations. With symmetric unicast paths and no multipath, all the traffic from n_d to n_i will go through a single link. Thus, assuming linear costs for link resources, we can use equation 9 to calculate the terms $cvl_d(nc)$ and $cvl_d(1)$. Assuming linear costs also for processing resources, equation 7 can be used to calculate $cvp_d(nc)$, $cvp_d(1)$, and $cvp_i(nc)$. After these operations we obtain:

$$ctm_{i,d} - ctc_{i,d} = SR \cdot ((nc-1) \cdot (pp_d + pl_{d,i}) - nc \cdot pp_i) - cf_i \tag{15}$$

Which can also be written as:

$$\begin{aligned} ctm_{i,d} - ctc_{i,d} &= SR \cdot nc \cdot (pp_d + pl_{d,i} - pp_i) - cf_i \\ &\quad - SR \cdot (pp_d + pl_{d,i}) \end{aligned} \tag{16}$$

The first line of the right side of the equation 16 represents the increase in costs associated with the migration configuration, while the last line corresponds to the increase associated with the cloning configuration. When nc is high, the migration configuration tends to become more expensive than the cloning configuration. Therefore the cloning configuration will generally be preferred for high nc, unless the fixed or processing costs at n_i are prohibitive.

With this result we obtain an easy way to make a decision, by using equation 15 to choose the cheapest configuration. As discussed earlier, an estimation on the upper bound of SR is considered available before the session starts. The number of clients, nc, is known at n_i, as well as the local cost cf_i and price pp_i. Consequently, before making a decision, the agent needs to obtain the following information from its neighbour n_d:
pp_d: processing price per unit
$pl_{d,i}$: link price per unit for the outgoing interface from n_d to n_i.

The information above is collected by a Prospecting capsule that is sent to the destination node before the cloning or migration action actually takes place. The costs for the intermediate links on the direct and reverse paths between nodes i and d have to be taken into account as well. The Prospecting capsule partially does this by accumulating into $pl_{d,i}$ the sum of the link costs for the active nodes on the path from node d to node i. When not all the nodes are active, an approach similar to the equivalent link abstraction (Sivakumar et al., 2000) could be used to estimate the transmission costs of a non-active network cloud.

This is the strategy adopted to obtain the simulation results shown in this paper. Although it seems a bit too simplistic, this strategy already takes into account an important criterion which is the delay penalty for the user which is imposed by the use of the reflectors instead of native IP multicast. As discussed earlier, this delay penalty is implied within the processing costs.

In classical multicast algorithms such decision dilemma usually doesn't apply, because only link resources are typically taken into account. In this case we can make $pp_d = pp_i = cf_i = 0$, and our calculations reduce to:

$$ctm_{i,d} - ctc_{i,d} = SR \cdot pl_{d,i} \cdot (nc-1) \tag{17}$$

In the above we have:
For $nc > 1 : ctm_{i,d} - ctc_{i,d} > 0$ and we choose to clone.
For $nc \leq 1 : ctm_{i,d} - ctc_{i,d} \leq 0$ and we choose to migrate.

These observations confirm that when bandwidth is the only scarce resource, cloning is the default choice except in the trivial case ($nc \leq 1$), since migration always implies duplicating packets on the link from n_d to n_i when $nc > 1$, and therefore causes the total costs to increase.

4.5 Merging

When two reflectors belonging to the same session meet at the same active node, they merge into a single reflector. This operation involves the union of both client lists and any other necessary information. Since this might result in resource overload, a preliminary negotiation between both agents is desirable to achieve favourable configurations. For instance, when sending the Prospecting capsule to an upstream neighbour to find out about costs, the capsule could also be programmed to look for the presence of another reflector for the same session, and check its current resource consumption. An outcome of the negotiation could be that server delegates some of its own clients to the prospecting reflector, in order to balance the load and reduce costs.

It is possible to show that assuming linear costs and symmetric paths, the same rule (Rule 13) with equation 15 can still be applied to take a joint merge decision involving two reflectors. Due to lack of space the analysis is not shown here but can be found in a separate report (Yamamoto and Leduc, 2001b). The resulting merge procedure is to consider as if all clients of the reflector at node n_d that share the outgoing interface towards n_i ($nc_{d,i}$) were attached to node n_i so that we can make $nc \leftarrow nc + nc_{d,i}$ in Equation 15; and then apply Rule 13. If the rule says “migrate” then the reflector at node n_i migrates to n_d. Otherwise (“clone”) n_i attaches itself to n_d as a client, and n_d transfers its $nc_{d,i}$ clients to n_i.

To implement this mechanism, a Prospecting capsule is first sent from node n_i to n_d. The capsule goes back to n_i with the values of pp_d and $pl_{d,i}$ and $nc_{d,i}$ (which is zero when no reflector for the group is running at n_d). The reflector at n_i then uses these values in Rule 13 to take a local decision to clone or to migrate. If the rule advises a clone decision and $nc_{d,i} > 0$, then the reflector at d will take action to transfer its $nc_{d,i}$ clients to n_i, after the cloning operation has been successfully completed. Otherwise everything occurs as described in Section 4.4.

4.6 Terminating

Since reflectors must “pay” for resource usage in the active nodes, and their clients are their sole source of income, they will be automatically eliminated by the active platform when there are no further clients. However, there is a risk that sudden changes in load make prices increase in unpredictable ways, causing fully operational reflectors to die out prematurely.

In our current implementation this problem is still not solved, and in our view it can only be solved with the help of load control operating at shorter time scales than the ones in which the reflectors operate. This requires adaptive (elastic) flows or transcoding, and here we are assuming that the reflectors merely repair connectivity failures, and don’t interfere with the session data contents.

5 Simulation results

We have performed some simulation experiments using ns-2, in order to visualise the tree construction and destruction mechanisms. The topology for the simulations is illustrated in Figure 2 (Left). All links have a fixed capacity of 10Mbit/s and a propagation delay of 10ms. The main session source is located at node S. The leaves of the tree contain terminal reflectors that join the session at random times from t=0s to t=10s. All nodes are active and have the same prices for resources: $pp = 1, pl = 1$, and $cf = 1 \cdot 8 \cdot C$ where C is the size of the mobile code in bytes, and is currently set to 50000, which is the current approximate size of the bytecode in our Java prototype.

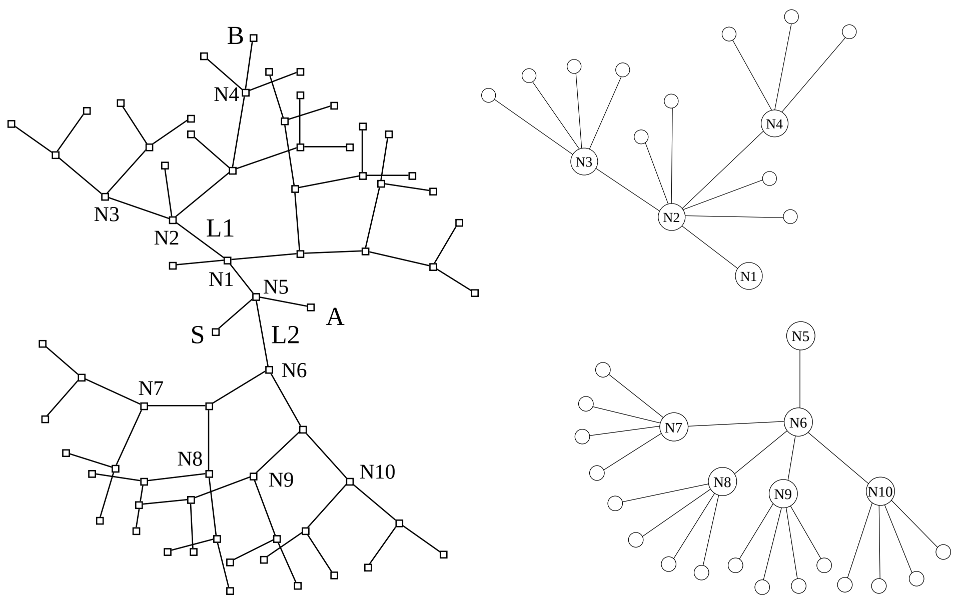

Figure 2: Left: Topology used in the simulations. Right: Two sample reflector trees over the topology on the left. Top Right: tree that results from the failure of L1, rooted at N1. Bottom Right: tree that results from the failure of L2, rooted at N5.

The multicast communication via links L1 and L2 is interrupted at t=20s. As a result two reflector trees appear. Both trees starts at around t=24s, but the tree on the upper side of the topology is ready at t=29s, while the second one is only ready at t=37s. After this construction phase all terminal reflectors affected by failures are served by an intermediate reflector. The resulting trees are shown in Figure 2 (Right).

At t=70s the multicast communication via L1 and L2 is restored. Most reflectors detect this a couple of seconds later, and disconnect from their parent reflectors, which die out between t=77s and t=80s.

Figures 3(A), (B), and (C) show the aggregate session data rate received by three sample session participants: A, B, and C, respectively, whose location on the tree can be observed in Figure 2 (Left). The main source rate is 500kbps while all the other session members send around 10kbps each.

Participant A happened to join the group at around t=10s, while B joined right at the beginning t=0s. During the failure period, although the multicast feed to node A is up and running, it receives less aggregate traffic until the reflector tree is fully operational, since during this period it doesn't receive the multicast packets coming from the participants that have stayed on the other side of the failure point. Participants B and C suffer from the failures until about t=30s and t=38s, respectively. After that, their respective level of reception becomes about the same as the one of A, as if they were also unaffected by the failure. When the multicast feed is restored, sudden peaks of traffic arrive at B and C, due to duplicate packets sent once via multicast and again via the reflector. These packets are eliminated by the terminal reflectors before being sent to the applications. We can verify this by looking at the sequence numbers received by the decoder connected to C, on Figure 3.

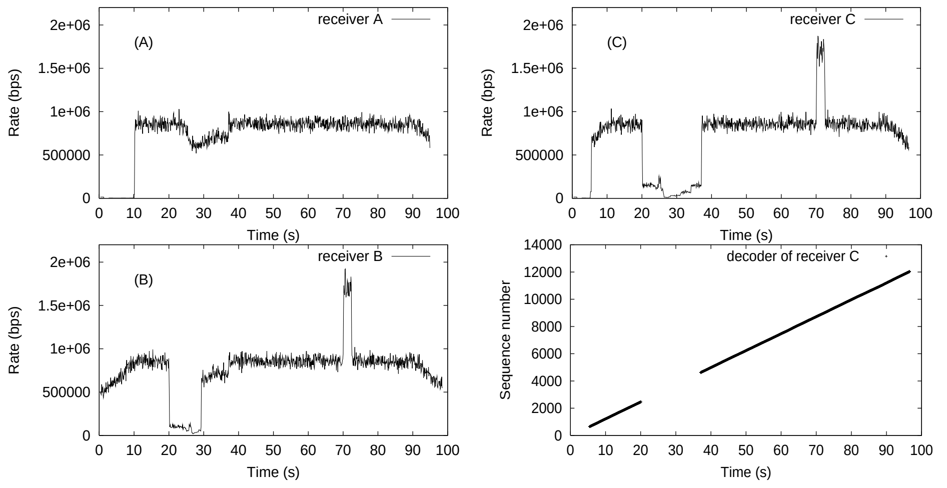

Figure 3: (A), (B), and (C): Data rate of three sample participants. (A): unaffected by multicast failure. (B) and (C): affected by failures of L1 and L2 respectively. (D): Sequence numbers received by the decoder of participant C.

In order to visualise the dynamics of the mobile code operations of migrating, moving, merging and terminating, we describe them in Figure 4 for the upper reflector tree on the topology. For compactness, we number events in time with integer numbers starting from 1, followed by the code of the operation performed. From t=24s to t=26s all terminal reflectors spawn upstream clones. Since the order in which each terminal reflector sends a clone won't have any influence on the subsequent operations, we assign event number 1 to all. This is indicated as "1C" in the figure. The next event is event 2, and it's a cloning operation from node N4 towards its upstream neighbour. This is indicated as "2C". By following the sequence of events in this way, it is possible to track the main actions that lead to the tree configuration shown in Figure 2 (Top Right), and to its subsequent destruction ("T" operation). Although the merging operations are not indicated, they can be deduced as well, since they occur whenever a reflector arrives at a node where another one is already present.

The results above are intended to illustrate the basic behaviour of our autonomous reflectors in ideal conditions. They are not intended to show a realistic picture of a real network. The topology is regular, the network is unloaded, all nodes are active, the delays are short and the paths for unicast and multicast traffic coincide.

In our simulations we have noticed little impact of increased network latencies or moderate load on the results, even when the propagation delay on each link is increased to the order hundreds of milliseconds. However, we have often observed much larger latencies for joining live MBone sessions, as well as variable loss patterns. Thus we can

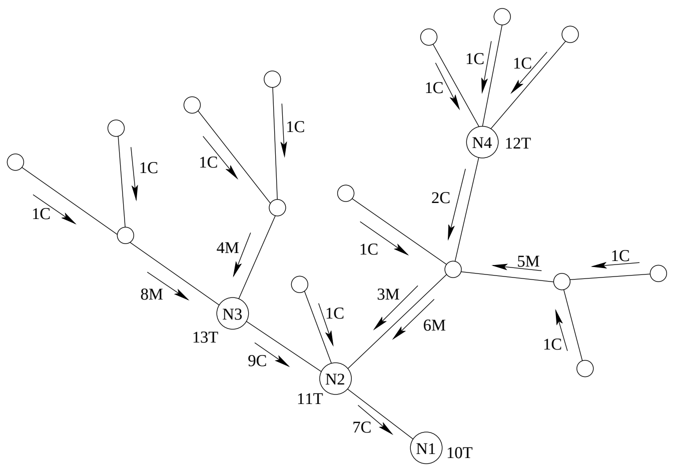

Figure 4: Dynamics of mobile code operations. The event numbers are shown beside the arrows or the node names, followed by the code of the operation: C (clone), M (migrate), T (terminate).

expect higher reaction times for our reflectors in such a situation.

Further results including the revised merge configuration, the impact of varying node and link prices, and of the amount of non-active nodes can be found in (Yamamoto and Leduc, 2001b). We are currently working on random topologies with concurrent sessions to assess the loading sharing capabilities and the distance to the global optimum.

6 Implementation

We are currently implementing a prototype of the mobile reflectors in Java using an architecture that allows the code to be easily ported to any EE that supports active extensions with minor modifications. The architecture is organised in three planes: data plane, monitoring plane, and control plane. This structure roughly follows the one suggested by (Blair et al., 1999).

The data plane is responsible for the blind forwarding of multicast and unicast data. Its core is inspired by the Mug reflector (Highfield, 1998): in Mug, a node that sends a RTP/UDP packet to the reflector is added to its client list; a client that stays idle (i.e. sends no more packets) for some time is removed from the client list. A selector is attached to the Mug-like core in order to switch between the multicast and the unicast upstream channels. The selector is controlled by the control plane. The data plane treats packets seamlessly whether they come from another reflector or from a multicast application such as the MBone media tools `vic` or `rat`.

The monitoring plane keeps track of the current resource usage and performance parameters of the data plane. To monitor CPU usage, we are currently integrating the CPU accounting facilities provided by J-Seal2 portable resource control framework (Binder et al., 2001). This requires some adaptations to the calculations in Section 4 to take into account measured resource consumption values besides estimated ones.

The control plane uses the data available in the monitoring plane to make decisions. Its core is a state machine with transitions triggered by events generated at the monitoring plane.

This architecture allows new strategies to be easily added to the control plane without affecting the other planes. It also maps naturally to the Bond platform (Bölöni and Marinescu, 1999), which opens up future possibilities for dynamic updates to the state machine through "agent surgery" (Bölöni and Marinescu, 1999). The communication mechanism among neighbouring reflectors takes the form of capsules such as in ANTS (Wetherall et al., 1998). Alternatively, the communication could be made via an existing agent message passing mechanism (e.g. Bond, see (Bölöni and Marinescu, 1999)). Both offer extra flexibility for enhancements and preclude the need to specify application-specific message formats and develop the corresponding parsers.

7 Conclusions and Future Work

We have described a decentralised scheme based on mobile code, to build a loosely connected network of autonomous reflectors that seeks to maintain session connectivity in the presence of multicast failures. The self-organising nature of the scheme ensures its robustness, scalability and autonomy properties, which make it suitable for sessions of any size, while minimising the necessary amount of human intervention.

For the moment each reflector treats only one media stream (e.g. either audio, or video, or whiteboard). In order to deal with several media, we plan to group multiple physical reflectors (each treating one media type) into a single logical reflector for cloning and migration purposes. In a near future, experiments over the MBone can be envisaged in the framework of the European COST Action 264, and with the help of existing active network overlays such as the ABone.

We plan to integrate the work presented in this paper with previous work on congestion control (Yamamoto and Leduc, 2000b; Yamamoto and Leduc, 2000a) such that reflectors also perform application-oriented filtering and/or transcoding of data in the presence of congestion, in a network which is likely to be only sparsely populated by active nodes. Other possible extensions include: exploring alternative paths, supporting strong route asymmetries and non-linear costs, multiple or changing centres of interest, QoS guarantees. It would also be interesting to generalise the technique for other group applications that require self-organisation.

Acknowledgements

This work has been carried out within the TINTIN project funded by the Walloon region in the framework of the programme *"Du numérique au multimédia"*. Part of this work was performed while the main author was a visiting researcher at Lancaster University. We would like to thank David Hutchison, Steven Simpson, Mark Banfield, Laurent Mathy, Stefan Schmid, and Katia Saikoski for their helpful support. We would also like to thank Allex Villazón (University of Geneva), Sandrine Calomme (University of Liège), and the anonymous reviewers for their insightful comments.

References

Akamine, H. et al. (2000). An Approach for Heterogeneous Video Multicast Using Active Networking. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 157–170, Tokyo, Japan.

Amir, E. (1998). *An Agent-based Approach to Real-time Multimedia Transmission over Heterogeneous Environments*. Ph.D. dissertation, University of California at Berkeley.

Baldi, M., Picco, G. P., and Risso, F. (1998). Designing a Videoconference System for Active Networks. In *Mobile Agents'98*.

Ballardie, A. (1997). Core Based Trees (CBT) Multicast Routing Architecture. Internet rfc 2201 (experimental), IETF.

Binder, W., Hulaas, J. G., Villazón, A., and Vidal, R. (2001). Portable Resource Control in Java: The J-SEAL2 Approach. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA-2001)*, Tampa Bay, Florida, USA.

Blair, G. S., Andersen, A., Blair, L., and Coulson, G. (1999). The Role of Reflection in Supporting Dynamic QoS Management Functions. In *IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, London, UK.

Bölöni, L. and Marinescu, D. C. (1999). A Multi-Plane State Machine Agent Model. Technical Report CSD-TR 99-027, Purdue University. Also a poster at the Fourth International Conference on AUTONOMOUS AGENTS (Agents 2000) Barcelona, Spain, June 2000.

Choi, S. Y., Turner, J., and Wolf, T. (2001). Configuring Sessions in Programmable Networks. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska.

Chu, Y., Rao, S. G., and Zhang, H. (2000). A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, pages 1–12, Santa Clara, CA, USA.

Clearwater, S. H., editor (1996). *Market-based Control: A Paradigm for Distributed Resource Allocation*. World Scientific Publishing.

Costa, L. H. M. K., Fdida, S., and Duarte, O. C. M. B. (2001). Hop by Hop Multicast Routing Protocol. In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, USA.

Duysburgh, B. et al. (2000). Data Transcoding in Multicast Sessions in Active Networks. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 130–144, Tokyo, Japan.

Fry, M. and Ghosh, A. (1999). Application level active networking. *Computer Networks*, 31(7):655–667.

Ghosh, A., Fry, M., and Crowcroft, J. (2000). An Architecture for Application Layer Routing. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 71–86, Tokyo, Japan.

Gibney, M., Jennings, N., Vriend, N., and Griffiths, J. (1999). Market-based call routing in telecommunications networks using adaptive pricing and real bidding. In *Proceedings of the IATA'99 Workshop*, Springer LNAI 1699, Stockholm, Sweden.

Handley, M., Perkins, C., and Whelan, E. (2000). Session Announcement Protocol. Internet rfc 2974 (experimental), IETF.

Handley, M., Schulzrinne, H., Schooler, E., and Rosenberg, J. (1999). SIP: Session Initiation Protocol. Internet rfc 2543 (standards track), IETF.

Highfield, J. (1998). Mug multicast packet reflector. URL http://www.stile.lboro.ac.uk/ cojch/mug/mug.html.

Kirstein, P. T. and Bennett, R. (2000). RE 4007 MECCANO Project Final Report. URL http://www-mice.cs.ucl.ac.uk/multimedia/projects/meccano/ deliverables/.

Kiwior, D. and Zabele, S. (2001). Active Resource Allocation in Active Networks. *IEEE JSAC*, 19(3):452–459.

Kon, F., Campbell, R., and Nahrsted, K. (2000). Using Dynamic Configuration to Manage A Scalable Multimedia Distribution System. *Computer Communication Journal*. Elsevier Science, Fall 2000.

Live Networks, Inc. (2000). URL http://www.live.com/.

Maxemchuck, N. F. and Low, S. H. (2001). Active Routing. *IEEE JSAC*, 19(3):552–565.

Najafi, K. (2001). *Modelling, Routing and Architecture in Active Networks*. Ph.D. dissertation, University of Toronto, Canada.

Partridge, C., Snoeren, A. C., Strayer, W. T., et al. (2001). FIRE: Flexible Intra-AS Routing Environment. *IEEE JSAC*, 19(3):410–425.

Roadknight, C. and Marshall, I. W. (2000). Differentiated Quality of Service in Application Layer Active Networks. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 358–370, Tokyo, Japan.

Safaei, F., Ouveysi, I., Zukerman, M., and Pattie, R. (2001). Carrier-Scale Programmable Networks: Wholesaler Platform and Resource Optimization. *IEEE JSAC*, 19(3):566–573.

Schulzrinne, H., Casner, S. L., Frederick, R., and Jacobson, V. (1996). RTP: A Transport Protocol for Real-Time Applications. Internet RFC 1889 (update in progress).

Shehory, O., Sycara, K., Chalasani, P., and Jha, S. (1998). Agent Cloning: An Approach to Agent Mobility and Resource Allocation. *IEEE Communications Magazine*, pages 58–67.

Sivakumar, R., Han, S., and Bharghavan, V. (2000). A Scalable Architecture for Active Networks. In *Proceedings of IEEE OPENARCH 2000*, Tel-Aviv, Israel.

Stoica, I., Ng, T. S. E., and Zhang, H. (2000). REUNITE: A Recursive Unicast Approach to Multicast. In *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel.

Tennenhouse, D. L. et al. (1997). A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86.

Tschudin, C. (1997). Open resource allocation for mobile code. In *Proceedings of the Mobile Agent'97 Workshop*, Berlin, Germany.

Tschudin, C. F. (1999a). A Self-Deploying Election Service for Active Networks. In *Proc. 3rd International Conference on Coordination Models and Languages (COORDINATION'99)*, Springer LNCS 1594, pages 183–195, Amsterdam, The Netherlands.

Tschudin, C. F. (1999b). Apoptosis - The Programmed Death of Distributed Services. In Vitek, J. and Jensen, C., editors, *Secure Internet Programming - Security Issues for Mobile and Distributed Objects*, Springer LNCS 1603, pages 253–260.

Wen, S., Griffioen, J., and Calvert, K. L. (2001). Building Multicast Services from Unicast Forwarding and Ephemeral State. In *Proceedings of IEEE OPENARCH 2001*, Anchorage, Alaska, USA.

Wetherall, D. J., Guttag, J. V., and Tennenhouse, D. L. (1998). ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of IEEE OPENARCH'98*, San Francisco, CA, USA.

Wittmann, R., Krasnodembski, K., and Zitterbart, M. (1998). Heterogeneous Multicasting based on RSVP and QoS Filters. In *SYBEN'98*, Zürich, Switzerland.

Yamamoto, L. and Leduc, G. (2000a). An Active Layered Multicast Adaptation Protocol. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 180–194, Tokyo, Japan.

Yamamoto, L. and Leduc, G. (2000b). An Agent-Inspired Active Network Resource Trading Model Applied to Congestion Control. In *Proceedings of the MATA 2000 Workshop*, Springer LNCS 1931, pages 151–169, Paris, France.

Yamamoto, L. and Leduc, G. (2001a). Autonomous Multicast Reflectors over Active Networks. In *AISB'01 Symposium on Software Mobility and Adaptive Behaviour*, pages 40–49, York, UK.

Yamamoto, L. and Leduc, G. (2001b). Autonomous Reflectors: Note on the Merge Operation. Technical report, University of Liège.