Short paper session, IWAN 2001, 3rd. International Working Conference on Active Networks
Philadelphia, Pennsylvania, USA, September-October 2001.

Building Bidirectional Multicast Trees Using Autonomous Reflectors

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

Abstract

We show an active application to build bidirectional multicast trees, based on the dynamic deployment of agents on top of an active network infrastructure, and interconnected by unicast. Such trees are used to build a repair service that seeks to maintain application-level connectivity in the presence of network-level multicast failures. Reflector trees are built on demand in a decentralized way, using three elementary agent operations: clone, migrate, and merge. They terminate automatically when no longer needed. In this paper we show how agents make decisions on which set of operations to use in order to produce a low cost tree configuration, taking node and link resources into account.

1 Introduction

Several years after its initial standardization efforts, the current level of deployment of IP multicast is still unsatisfactory. Active networking (AN) is therefore a promising technology in this context [10, 17]. We have addressed this issue by proposing an autonomous reflector service that seeks to maintain application-level connectivity in the presence of network-level multicast failures [18, 19]. This paper is a short synthesis of the current state of our work on this subject.

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 a set of unicast clients, and from every unicast client to the multicast group and to all other unicast clients. This enables session connectivity when some or all participants have no access to IP multicast, or when multicast failures occur. Existing reflector software must typically be manually installed [6, 8, 9, 12], or have limited autoconfiguration capabilities [11]. Besides that, they can generate a significant amount of redundant traffic that contributes to network congestion, and therefore do not scale to large sessions.

Our reflectors are based on mobile code, and run on top of active network [16] or active server (AS) nodes [2, 6]. They are able to 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 on local knowledge available at the terminals or active nodes where they run, and a minimum amount of knowledge about their neighbor 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.

2 Autonomous Reflectors

The scheme relies on a default unicast routing service interconnecting all active nodes, such that at any moment it is possible for an active application to obtain the next hop to a given destination. This service can

take the form of default IP routing or an active routing service [7].

The reflectors start at leaf nodes and then progressively move, clone and merge with other reflectors along their respective unicast paths towards a root node or rendez-vous point (RP). When repairing multicast trees, the RP is the main source of content in the session (e.g. lecturer's site). Figure 1 shows a tree construction example for a session consisting of an RP or main source node S and the session members R1, R2 and R3.

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.

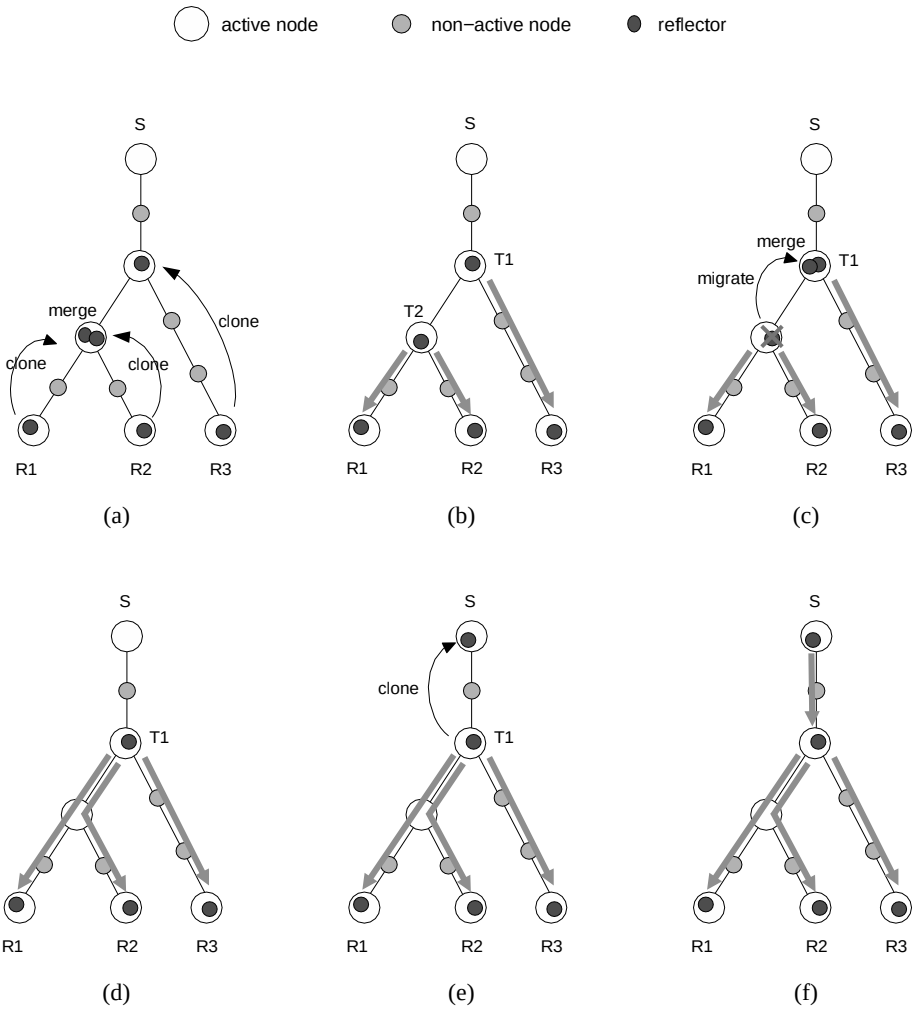

Figure 1: Tree construction example.

When a participant detects a multicast failure, its terminal reflector sends another reflector to the next active 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 restore application connectivity when native multicast fails, such that at least the data coming from the main session source reaches all session members.

The intermediate reflector listens to the multicast group for a while, and if no multicast activity is detected it migrates to the next active hop towards the RP (*upstream migration*, Fig. 1(c)). When two reflectors

belonging to the same session meet at the same active node, they merge into a single reflector (Fig. 1(a)(c)). A reflector that has more than one child might decide to clone upstream instead of migrating, therefore adding one more hierarchy level to the tree (Fig. 1(e)). A reflector might decide to merge with an upstream reflector in order to reduce costs.

This tree building process is completed when a reflector reaches a node where data from the main source is received, either via multicast or via another reflector (Fig. 1(f)). Since each agent reflects every packet received to every other directly attached reflector, the result is a bidirectional shared tree.

A repair tree of reflectors progressively grows in this way towards the RP, until failure points are successfully bypassed. A reflector that succeeds bypassing the failure point becomes the root of a reflector tree. The tree is organized in a client-server hierarchy, where a parent reflector serves a number of child reflectors (clients).

This bottom-up tree construction algorithm has two consequences: First, not only one tree but several ones might arise in response to a failure. Second, these repair trees are located as close as possible to the participants affected by the failure, and do not interfere with the rest of the session running in native multicast.

A reflector that runs out of clients automatically terminates itself. When multicast connectivity is 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.

AN Capsules are used to implement a signaling mechanism among neighboring 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 the next section.

3 Reflector decisions

In order to take a decision to either clone or to migrate to an upstream node, a reflector first needs to estimate the costs that would result from choosing either option. It also has to take into account whether another reflector is already present in the upstream node, such that a merging operation must take place. Such a decision process may be complex. At present we adopt a simplified approach to the problem that relies on an estimation of the session bandwidth (that must be available at the beginning of the session) in order to select the configuration with the lowest cost.

Three cost components are taken into account, corresponding to the three main types of resources consumed: processing, link and memory storage. Since network packets constitute the bulk of the data treated by a reflector, the processing costs during a given interval increase with the number and size of the packets treated, which in turn increase with the number of connected clients. The link usage costs are due to the bandwidth used for the transmission of packets, and also increase with the number of packets treated. Since reflectors do not maintain internal packet buffers, the amount of memory storage used varies little with the amount of clients. Thus the storage costs correspond essentially to the fixed costs of using the mobile code platform.

With the above assumptions in mind, in [18] we showed that if we adopt a simple decision strategy which is to choose the configuration with the lowest cost, the following rule can be applied when a reflector located at node i has to decide whether to send a clone to node d or to migrate there:

$$\text{if } ctm_{i,d} - ctc_{i,d} > 0 \text{ then clone else migrate.} \tag{1}$$

where:
$ctm_{i,d}$ is the cost of the resulting configuration after migration
$ctc_{i,d}$ is the cost of the resulting configuration after cloning
and:

$$ctm_{i,d} - ctc_{i,d} = SR \cdot nc_i \cdot (pp_d + pl_{d,i} - pp_i) - SR \cdot (pp_d + pl_{d,i}) - cf_i \tag{2}$$

where:
SR is total rate of the session (session bandwidth)
nc_i is the current number of clients of the reflector that runs at node i
pp_d and pp_i are the (constant) processing prices per bit per second at nodes d and i, respectively
$pl_{d,i}$ is the price per unit of bandwidth on node d's outgoing link towards node i
cf_i is the fixed (storage) cost at node i

When nc_i 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_i, unless the fixed or processing costs at node i are prohibitive. Equation 2 also applies to the case of non-active multicast algorithms, when only link resources are typically taken into account. In this case we can make $pp_d = pp_i = cf_i = 0$, and cloning becomes the default choice except in the trivial case ($nc_i \leq 1$).

In a subsequent work [19] we have enhanced the analysis by putting special emphasis on the merging mechanism. We have shown that under a few simplifying assumptions, the same rule can still be applied to take a joint merge decision involving two reflectors. The resulting merge procedure consists in considering as if all clients of the reflector at node d that share the outgoing interface towards i ($nc_{d,i}$) were already attached to node i so that we can make $nc_i \leftarrow nc_i + nc_{d,i}$ in Equation 2; and then apply Rule 1. If the rule says "migrate" then the reflector at node i migrates to d. Otherwise ("clone") i attaches itself to d as a client, and d transfers its $nc_{d,i}$ clients to i.

To implement such decision mechanism, a Prospecting capsule is first sent from node i to d. If no reflector for the group is running at d, the capsule comes back with the values of pp_d and $pl_{d,i}$. The reflector at i then uses these values in Rule 1 to take a local decision to clone or to migrate.

If another reflector is running, the Prospecting capsule communicates its values of nc_i, pp_i, and cf_i to it and receives back an advice of type "clone" or "you decide". The advice is calculated by the reflector at d using Rule 1 with the values provided by the Prospecting capsule and $nc_i \leftarrow nc_i + nc_{d,i}$. If the rule advises a clone decision and $nc_{d,i} > 0$, then the resulting advice is "clone", and the reflector at d will take action to transfer its $nc_{d,i}$ clients to n_i. Otherwise no action is required from n_d and therefore the advice is "you decide", meaning that the decision can be taken locally by n_i.

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 [15] can be used to estimate the transmission costs of a non-active network cloud.

The decision rule shown is only valid under the assumptions of linear costs and symmetric paths, which do not always hold in practice. Asymmetric links and paths are becoming more and more common, routes may change, and prices may change in response to demand and other factors. If resources and routes are stable enough during the time scale of changes in the tree, reflectors can make quick decisions considering linear costs and then correct their decisions later by reshaping the tree. Further discussions on how to overcome such limitations can be found in [19].

4 Simulation results

We have simulated the tree construction algorithm for the repair of multicast trees using autonomous reflectors. The reflectors run on top of a simulated execution environment implemented for ns-2. The topology used in the simulations consists of a random tree of 64 nodes. All nodes are active and have the same prices for resources, and $cf = 1 \cdot 8 \cdot C$ where C is the size of the mobile code in bytes. The resulting shape of the tree is a trade-off between the number of reflectors placed and the number of copies of packets sent over the same link (link stress).

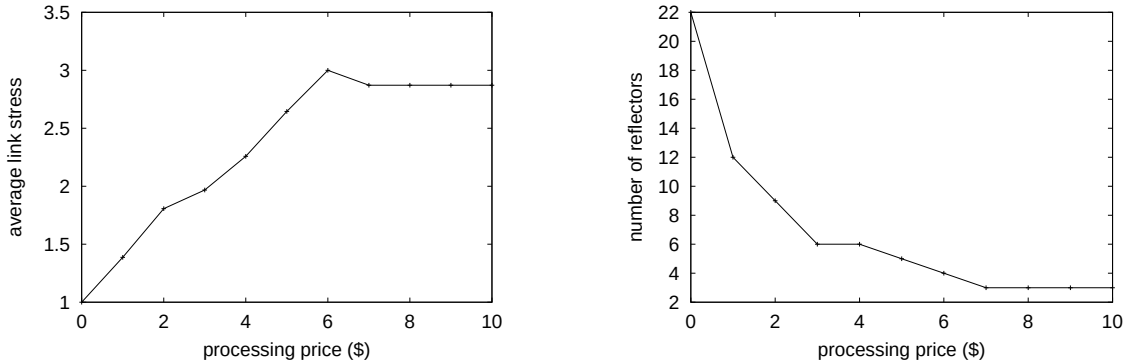

Figure 2: Impact of processing prices on the link stress (left) and number of reflectors on tree (right). All nodes are active and $ps = pl = 1$.

Figure 2 shows the effect of different processing prices on the tree construction behavior, for a constant link price $pl = 1$, and $ps = cf/(8 \cdot C)$. When the processing price pp increases the link stress also increases (Fig. 2 (left)), indicating that fewer reflectors are placed, and these reflectors need to serve a larger number of clients. This is confirmed by Fig. 2(right), that shows that the number of reflectors decreases with the increase in processing price. Further results can be found in [18, 19], including reaction times, the dynamic of code mobility operations, the impact of varying link prices and non-active nodes.

5 Related Work

Mobile reflectors that can clone or migrate appeared in [3], as part of a videoconference architecture for active networks. However, the authors focused on software design issues, and did not cover algorithms and criteria for placing reflectors.

The problem of choosing active nodes to place a given programmable functionality has been identified as a routing problem [4, 14]. The potential of active routing is broadly discussed in [13].

6 Conclusions and Future Work

The self-organizing nature of the autonomous reflector scheme ensures its robustness, scalability and autonomy properties, which make it suitable for sessions of any size. The approach exploits the trade-off between node and link resources to produce low cost trees that adapt to the available resources. The ubiquitous deployment of such a temporary connectivity repair service in a hardwired manner wouldn't make sense. This is a sample application where active networking can be useful in practice, including in the relative short term, if a few active nodes can be made available.

We are currently working on the load control and tree reshaping mechanisms. We are also implementing the mobile reflectors in Java. The implementation is briefly described in [18]. In a near future, experiments over the MBone can be envisaged in the framework of the European COST Action 264 [5] and with the help of existing active network overlays such as the ABONE [1].

Acknowledgments

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"*.

References

[1] Active Network Backbone (ABONE). URL http://www.isi.edu/abone/.

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

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

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

[5] COST Action 264, URL http://www.lip6.fr/COST264.

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

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

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

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

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

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

[12] Live Networks, Inc., URL http://www.live.com/.

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

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

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

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

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

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

[19] L. Yamamoto and G. Leduc. Reflectors for Seamless Group Communication. Technical Report RUN-TR 02-01 (submitted for publication), University of Liège, July 2001.