Impact of Simple Cheating in Application-Level Multicast

Laurent Mathy Nick Blundell
Computing Department
Lancaster University, UK
{laurent, n.blundell}@comp.lancs.ac.uk

Vincent Roca Ayman El-Sayed
Planète Project
INRIA Rhones-Aples, France
{vincent.roca, ayman.elsayed}@inrialpes.fr

***Abstract*— We study the impact of cheating nodes in application-level multicast overlay trees. We focus on selfish nodes acting independently, cheating about their distance measurements during the control phase building or maintaining the tree. More precisely, we study, through simulations, the impact of simple cheating strategies in four protocols, representatives of different application-level multicast protocol "families": HBM (a protocol based on a centralized approach), TBCP (a distributed, tree first protocol), NICE (a distributed, tree first protocol based on clustering) and NARADA (a mesh first protocol). We evaluate the impact of cheats on the performance of the overlay trees as perceived by their nodes and the underlying network.**

I. Introduction

Application-level multicast [7], a technique whereby hosts or end-nodes are organized into an overlay distribution tree without requiring any specific support from the network (i.e. based on normal unicast routing and forwarding), has been proposed mainly as a way to palliate to the lack of deployment of native IP multicast in production networks [6]. Application-level multicast represents a trade-off between the efficiency of IP multicast and the ease of deployment of group communications as a single source replicating the data sequentially, using unicast, to a group of receivers.

Although there is no question about the superiority of IP multicast for data distribution to large groups, application-level multicast may still prove a cheaper solution for communications within small groups (groups whose membership is in the order of tens of receivers). Also, some systems and applications need to establish relations and communications requiring a semantics richer than that provided by IP. Levels of control of the communication patterns and reliability of the communication structure may be needed, that cannot be provided by IP multicast. For example, because the group members are the nodes of the overlay application-level multicast tree, these nodes can interpret and modify the distributed content "en route", something not possible in native multicast where application nodes are always leaf nodes of the tree.

For these reasons, we believe that application-level multicast is complementary to IP multicast and will remain a useful group communication tool if and when IP multicast is deployed ubiquitously.

Application-level multicast is based on the collaboration of group members with each other. Indeed, as group members (i.e. receivers) are the nodes of the overlay tree, they rely on each other to distribute the data. However, there is an intrinsic imbalance of roles in an overlay tree: non-leaf nodes must take part in the burden of replicating data along the tree, while nodes which are, on the tree, closer to the source (the source is often the root of the tree) observe lower propagation delays. Also, the closer to the source a node is, the lower the loss rate observed (since in normal data transfer from root to leaf nodes, losses "accumulate" as data travels down the tree branches). The collaboration is also extended to the control of the overlay tree which is often built based on distance measurements taken by the receivers amongst themselves. Whether a full measurement matrix is required prior to the construction of the tree, or the matrix can be populated overtime while improving the tree, or partial group and measurement knowledge is enough, is entirely dependent on the protocol used. As a rule of thumb, the better the knowledge of the group membership and distances between members, the better the performance of the tree can be tuned, but the least scalable the corresponding application-level multicast protocol is.

The important point here is that there is an opportunity for receivers to try and improve their position on the overlay tree by "manipulating" distance measurements, in order to be positioned closer to the data source while limiting, to a minimum, their replication burden. In the rest of the paper, we will refer to such receivers as "cheats", and the consequences of their actions is the focus of this paper.

If we consider the very popular round-trip time (RTT) distance measurements used in many application-level multicast protocols, a cheat could delay a probe received from another receiver to artificially increase their measured distance in order to try and reduce its replication burden (since the further away another receiver is, the more likely that receiver will be connected to another (closer) node). For scalability reasons, most of the existing application-level multicast protocols require that each node measures its distance to other nodes in the overlay tree and reports these distance measurements to other nodes and/or uses these for decision making. A cheat can therefore lie outright about its distance measurements, in order to try and improve its position in the tree.

It is important to note that the cheats considered in this paper do not attempt to disrupt the flow of data along the overlay tree or even to break the protocol used to build the tree, they simply try and improve their position in the tree.

In other words, we are not interested in disruptive behaviour such as denial of service: once in the tree, although the cheats can keep lying about measurements, they otherwise follow all other protocol rules.

In some controlled environments, cheating is almost impossible. For instance, this is the case when application-level multicast is used to provide a group communication service in an IPSec virtual private network (VPN) environment [1]. Here the gateways of the various sites connected through the VPN are fully secured and remotely controlled by the VPN operator. However, such a situation belongs to a very specific application domain.

Nevertheless, although the problems of cheats in application-level structures (and overlay trees in particular) has often been mentioned, we are not aware of any quantitative study of their effects on application-level multicast protocols, as well as on the underlying network. We believe that understanding such effects is critical if the benefits offered by application-level multicast are to be reaped in application domains where the receivers do not pertain to the same, tightly controlled, administrative domain (e.g. corporation), as is the case in gaming, video distribution/webcasting, etc.

Therefore, in this paper, we will study the effects of simple cheating strategies on four application-level multicast protocols. These cheating strategies will be simple, but targeted to the respective protocols: the cheating will be slightly different depending on the protocol considered. For this reason, it is important to note that our goal is not to compare the relative ability of the protocols considered to deal with cheats, but rather we seek to extract possible common consequences and trends created by the presence of cheats in application-level multicast overlay trees. Also, the protocols studied in this paper were chosen as being representatives of different application-level multicast protocol "families", and because simulators were readily available for these.

Furthermore, although there may exist more sophisticated cheating strategies, in this paper we deliberately look at simple ones, where selfish cheats act independently of each other and make no attempt to evade possible detection.

In section II, we briefly review the different families of application-level multicast protocols. In section III, we describe in more details the workings of the protocols chosen for this study, and we describe the simple cheating strategies used. Section IV presents our simulation study, while section V concludes with a summary of our observations and some recommendations.

II. Application-Level Multicast

In this section, we give a brief overview of application-level multicast protocol families.

A. Centralized Algorithms

The ALMI protocol [10] and HBM [12] are examples of a centralized approach to application-level multicast. They have a session controller node which gathers distance information from all of the group nodes and calculates the overlay tree which it uses to inform each node of its neighbours.

B. Distributed Algorithms

1) Mesh-First Algorithms: Narada [4] is an example of a mesh-first application-level multicast protocol where nodes arrange themselves into a well connected mesh on top of which a routing protocol similar to DVMRP is run, to build per-source overlay trees. The quality of the mesh, and therefore the overlay trees are improved incrementally over time by nodes adding and dropping mesh links based on a decentralised utility function. SCATTERCAST [3] is another protocol taking the same approach.

2) Tree-First: The NICE [2] protocol uses hierarchical clustering techniques to build overlay trees whereby group members arrange themselves into clusters with nodes closest to themselves.

TBCP [9] and HMTP [15] build an overlay tree by having receiver nodes recursively choose better parents to connect to, in a distributed fashion. These protocols are said to use a "limited scope approach", because, at each step of the recursion, a node only measures its distance to the children of its current parent.

3) Coordinate Systems: The protocols presented in this section use the notion of coordinates in various virtual geometrical spaces.

In the Delaunay triangulation method [8], each receiver is assigned coordinates in a Euclidian plane and the tree is computed via a distributed application of the geometric process known as Delaunay triangulation.

Application-level multicast based on CAN [11] splits a mutli-dimensional virtual torus into adjacent regions and uses a sort of broadcast method to flood a data packet to all the regions in a controlled way.

Finally, SCRIBE [13] exploits the properties of a peer-to-peer network system to build application-level multicast trees, by merging peer-to-peer "search" paths to form a tree.

III. The Protocols in our Study

In this paper, we chose to concentrate on four protocols: HBM as a representative of the centralized approach; TBCP as a representative of distributed, tree first, limited scope approach; NICE as a representative of the method based on clustering techniques; and NARADA as a representative of the mesh-first approach.

A. HBM

1) Principles: In HBM, the construction and maintenance of the overlay tree is under the control of a single host, the rendez-vous point (RP) or controller. Periodically and asynchronously, each group member measures its distance to all the others (or a subset of them) and reports these to the RP which thus knows the identity of each group member and the communication costs between them. The RP is then responsible for the overlay topology calculation and its dissemination among the group members.

Although HBM is a general protocol that does not restrict the properties of its overlay topology, the topology used in

this study is a degree-bounded shared tree of minimum cost, based on RTT distance metrics.

2) Simple Cheating Method: An HBM cheat always reports a distance of zero to the source, and adds 10 seconds to the RTT distances it measured to the rest of the group. An HBM cheat also delays by 10 seconds any measurement probes it receives from any other group member. This probe delaying action is mandatory since otherwise the RP could easily detect cheats by comparing the $A \hookleftarrow B$ and $B \hookleftarrow A$ RTT measurements. If they differ significantly, the RP could easily conclude that one of A and B has a suspect behaviour. Then, after cross-checking with other metrics evaluations where A and B are implicated, the RP could easily determine which node is cheating.

A cheat is thus aiming to become one of the source's children, while having no children at all.

B. TBCP

1) Principles: In TBCP, each node chooses individually the maximum number of children (i.e. the fanout) that it will accept. This fanout is strictly enforced and must have a value of at least one. TBCP has been designed to operate with minimum knowledge of the group membership and associated measurement matrix. It is a recursive algorithm where, starting at the tree root (which is considered to be the source) as a potential parent, a newcomer measures the distance between itself and the potential parent, along with the distance between itself and all of its potential siblings (i.e. the potential parent's current children). These distances are reported to the potential parent who, thanks to the measurements previously reported by its existing children, has complete knowledge of the measurement matrix for the "local" full mesh comprising itself, its children and the newcomer. The potential parent then considers all the local configurations for the acceptance of the newcomer in the tree (i.e. considers the newcomer as a child if there is room, considers sending the newcomer as a child of one of its current children, considers keeping the newcomer as a child while sending one of its existing children as a child of the newcomer, and considers keeping the newcomer as child while sending one of its existing children as a child of one of its existing children), evaluating the "goodness" of each local configuration with a score function. The best local configuration (according to the score function) is chosen and the appropriate node directed to its "next" potential parent where the algorithm starts again. It is important to note that when choosing amongst several equivalent local configurations, TBCP always favours those resulting in the newcomer "moving", to provide stability for already joined receivers.

TBCP has a maintenance method where nodes periodically "re-join" one of its known ancestors chosen at random, but for the purpose of this study, all nodes will always "re-join" at the root, as we expect this to be the behaviour chosen by a cheat who is trying to get as close as possible to the root.

2) Simple Cheating Method: A TBCP cheat will always report a distance of zero to its potential parent. As all receivers start joining the tree at the root, this provides the cheats with an opportunity to try and stay as close as possible to the root.

Because cheats want to minimize the work they do for the rest of the group, a cheat will choose the minimum allowed fanout value (i.e. 1). However, to try and avoid having a child, cheats also lie about their distance to other receivers: a cheat always delays a received probe by a fixed amount of time (10 seconds) and always adds a fixed amount of time (10 seconds) to the distance it reports from other receivers.

C. NICE

1) Principles: In NICE, nodes arrange themselves into a hierarchy of clusters whereby clusters belong to layers and nodes belonging to a cluster are close to each other in relation to some given cost metric[2]. At the highest layer of the hierarchy is a single cluster whose cluster members are each the leader of a single cluster in each of the subsequent lower layers. All nodes belong to a cluster in the lowest layer of the hierarchy but cluster leaders are also members of a cluster in their next-higher layer.

A node joins the group by first contacting a Rendez-vous Point (RP) to discover members belonging to the highest-layer cluster. The joining node then probes each of these cluster members to discover the closest to itself with whom it makes a request to join. The closest, highest-layer cluster member replies to the joining node with a list of cluster members in the next-lower layer to who it is the cluster leader. The joining node then probes each of the cluster members in the next-lower layer and the algorithm continues recursively until the new node joins the cluster closest to itself in the lowest-layer of the hierarchy.

Members of a cluster periodically exchange *heartbeat* messages with each other containing an estimate of the distance from themselves to each of the other cluster members. Whenever membership of a cluster changes (i.e. if a new node joins or leaves) the cluster leader, using this cluster member distance information, checks if it is still the center of the cluster and thus the most appropriate leader, transferring leadership to another cluster member if necessary.

The cluster leader periodically checks the size of its cluster and splits the cluster if its membership exceeds an upper bound. Likewise, if the cluster size falls below a lower bound the leader merges its cluster with the closest cluster belonging to the next-higher layer.

2) Simple Cheating Method: A NICE cheat sets out to join a cluster in the highest layer possible in order to minimise its distance to the data source. Note that, although NICE supports any-source, application-level multicast routing through a bi-directional overlay tree, in this study we consider only the optimal case of a single source at the root. For a cheat to join a cluster in the next-higher layer it must become the leader of its highest-layer cluster and so tries to achieve leadership through quoting, in its *heartbeat* messages, only a fraction of the actual distances to the other cluster members. On recalculating which node is closest to all of the other nodes the current cluster

leader will likely transfer its leadership to the cheat which, in effect, gets pushed up to the next-higher layer.

Once a cheat has gained leadership of a cluster it will make sure never to transfer leadership from itself to any other cluster members, by reporting a distance of zero to all other cluster members in its regular heartbeat messages.

In an attempt to preserve its resources, a cheat will never merge its clusters in the lower layers if their size falls below a lower bound and will also delay cluster join requests from other nodes by 10 seconds to reduce the likelihood of these joining the clusters. Note that, whilst a cheat will try to avoid having large clusters in all of the layers it occupies, the cheat will be required to forward data to members of its clusters in each of the lower layers which could potentially result in a high node fanout for the cheat as described in [2][1].

D. NARADA

1) Principles: Narada is a mesh-first, application-level multicast protocol whereby nodes organise themselves into a well connected mesh through the addition of links to other group members, termed their *mesh-neighbours*[4]. Nodes exchange routing tables with their *mesh-neighbours* allowing per-source data delivery trees to be constructed on the mesh using well-known reverse-shortest-path routing techniques as in DVMRP [5]. Narada therefore supports multi-source, application-level multicast but group size is limited by the need for nodes to have complete knowledge of all other group members.

On joining the mesh, a node selects, at random, a handful of nodes to add as *mesh-neighbours* from a subset of currently active mesh members obtained using some out-of-band bootstrap mechanism. As a result of adding these random mesh links, the recently-joined node's position in the mesh is likely to be sub-optimal in relation to the given cost metric. However, once connected a node is able to improve its position in the mesh by periodically probing random members, learned of through gossipping membership update messages with its *mesh-neighbours*. When a node is probed, it returns a copy of its routing table to the probing node who then calculates the utility of adding a mesh link to the probed node. A mesh link is deemed to be good if it improves the cost of a number of paths in the probing node's routing table such that the number of improved paths is greater than some threshold parameter.

In order for nodes to keep adding better mesh links, it is necessary for them to 'drop' their least useful links, where usefulness when considering to drop a mesh link is approximated by how many other members can be reached on shortest-paths through the mesh link.

Consequently, the overall mesh quality improves over time with respect to the given cost metric, resulting in more efficient data routing paths on the per-source overlay trees.

2) Simple Cheating Method: On discovering the identity of a data source in the mesh, through either receiving a data packet or through out-of-band mechanisms, a Narada cheat will set out to add the source as a *mesh-neighbour* and so receive data directly from the source.

However, to reduce the likelihood of a cheat being dropped as a *mesh-neighbour* to the source when the source eventually discovers that it is not very useful, a cheat makes sure to establish at least one mesh link to another node through which, by lying to the source about the cost of shortest-paths in routing update messages, it misleads the source to believe that it can deliver data to all of the other group members at a fraction of the actual costs (this is achieved by the cheat reporting route costs to other nodes that are a small fraction of their actual values). Once attached to the source, a cheat is likely to be dropped by its other *mesh-neighbours* and so in the same way, as with the source, misleads them as to its benefit for reaching other members of the group.

Narada is susceptible to partitioning when the degree of mesh nodes is small, so in order not to break the protocol whilst preserving resources, a cheat maintains three mesh links and no more (by setting its maximum out-degree – fanout – to the appropriate value).

IV. Effects of Cheating in Application-level Multicast

A. Performance Indicators

Several indicators are widely used to evaluate the performance of application-level multicast protocols. Two such classical performance indicators are the *link stress* and the *stretch*.

The link stress (or stress, in short) is a measure of the network efficiency of the application-level multicast protocols and is defined as the number of redundant copies of a data packet carried on a network link. The maximum stress is therefore the maximum number of duplicates seen by any single network link, while the average stress is the sum of duplicates divided by the total number of network links making up the branches of the tree. A major goal of all application-level multicast protocols is, of course, to keep the value of these stress indicators as small as possible, since higher network stress levels (especially maximum stress) indicate higher risks of network congestion.

The stretch, or relative delay penalty (RDP), is a measure of the penalty paid by a receiver for receiving data on an application-level tree rather than directly from the source. It is defined as the ratio TD/UD, where TD is the tree delay, that is the latency from the source to the receiver observed along the tree; and UD is the unicast delay, that is the networked delay resulting from direct communication from the source to the receiver. The average stretch (over all the receivers) and maximum stretch (i.e. worse penalty) are therefore good indicator of the tree efficiency of the application-level multicast protocols.

The above mentioned performance indicators are used to characterize the intrinsic performance of application-level multicast protocols. However, in this paper, our focus is not on benchmarking the performance of the protocols, but rather to study the impact of cheats on their performance. In our

[1]Note that, in this study we do not consider the NICE protocol extention whereby cluster leader's delegate some of their data forwarding responsibility to their highest-layer cluster members.

study, we therefore use the following performance indicators which are the above mentioned metrics normalized to the performance of the protocol without cheat, used as a reference:

- $\mathrm{stress\_ratio} = \mathrm{stress}/\mathrm{stress}_{\mathrm{ref}}$, where $\mathrm{stress}_{\mathrm{ref}}$ is the corresponding stress observed when all receivers behave in an honest way.
  We will be interested in the maximum stress ratio as a measurement of the impact of cheats on the underlying physical network. Indeed, maximum stress represents the highest load created by an application-level overlay tree on any network link, and thus the maximum stress ratio gives a good idea about the way risks of congestion evolve in the presence of cheats. Note that a stress ratio smaller (resp. greater) than 1 represents an improvement (resp. deterioration) compared with the case without any cheat.
- $\mathrm{stretch\_ratio} = \mathrm{stretch}/\mathrm{stretch}_{\mathrm{ref}}$, where $\mathrm{stretch}_{\mathrm{ref}}$ is the stretch of a receiver observed when all receivers behave in an honest way. Note that since the unicast delay is dictated by the physical topology and routing in the underlying network, it is independent of whether a receiver cheats or not, and we therefore have $\mathrm{stretch\_ratio} = (\mathrm{TD}/\mathrm{UD})/(\mathrm{TD}_{\mathrm{ref}}/\mathrm{UD}_{\mathrm{ref}}) = \mathrm{TD}/\mathrm{TD}_{\mathrm{ref}}$, since $\mathrm{UD} = \mathrm{UD}_{\mathrm{ref}}$.
  To have a better view of the influence of cheating in application-level multicast, we will segregate the receivers in a group of cheats and a group of honest receivers, and measure average, minimum and maximum stretch ratios in each group. This will allow us to not only study the impact of cheats on the performance observed by honest nodes, but also study the effects independent, selfish cheats have on each others. Note that as the overall goal is always to try and minimize stretch, a stretch ratio smaller (resp. greater) than 1 represents an improvement (resp. a deterioration), with a minimum ratio therefore representing the best improvement and a maximum ratio representing the worst deterioration.

B. Simulation Setup

We have studied the effects of cheats in application-level multicast on an Internet topology of 600 routers generated by GT-ITM [14]. We have tested 25 groups of 20 receivers and 25 groups of 100 receivers. Each group was tried with respectively 5%, 10%, 20%, 30%, 40%, 50%, 75% and 100% of cheating receivers. It is worth noting that all the protocols were studied with the same groups and the same cheats within these groups, while nodes hosting application-level multicast agents were always connected to edge routers of the topology[2], to achieve a realistic set-up. Trees of maximum fanout of 2, 3, 4 and 5 were built for each of these groups[3].

[2]For each trial, the total number of simulated nodes was therefore either 621 or 701, comprising the routers, the source and the receivers.

[3]More precisely, the parameter controlling the maximum number of children was set to these values. It is important to note that some protocols, NICE in particular in our study, do not enforce the fanout value at all times, but rather use it as a target value for stable trees.

For protocols that improve the quality of their overlay trees over time (e.g. NARADA and NICE), simulations run for 1000 seconds in order to ensure stabilisation of these trees.

As a comparison point, beside the case where the protocols were run with all receivers behaving in an honest way, we also built random trees for each of the groups, where the receivers joined in random order and simply randomly connected to one of the nodes in the tree (i.e. the source or an already existing receiver), whose maximum fanout had not been attained yet. These random trees are used as representatives of "bad" application-level multicast trees.

The cheating techniques described in section III were implemented in the HBM simulator that was used in [12], the TBCP simulator used in [9], and the NARADA/NICE simulator[4] used in [2].

C. Network Stress Ratios

We study the stress ratio in order to assess the overall impact of cheats in application-level multicast on the network. Figures 1 to 4 depict the maximum stress ratio observed during the simulations.

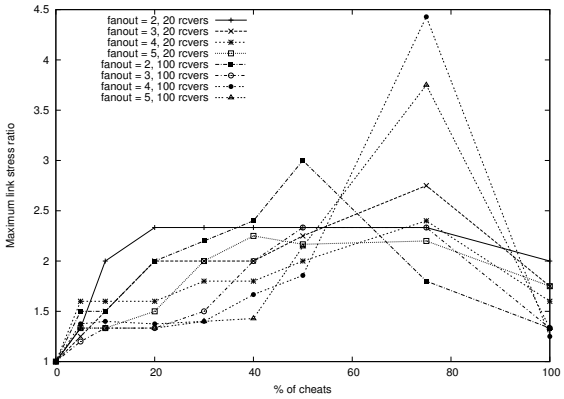

Fig. 1. Maximum link stress ratios in HBM

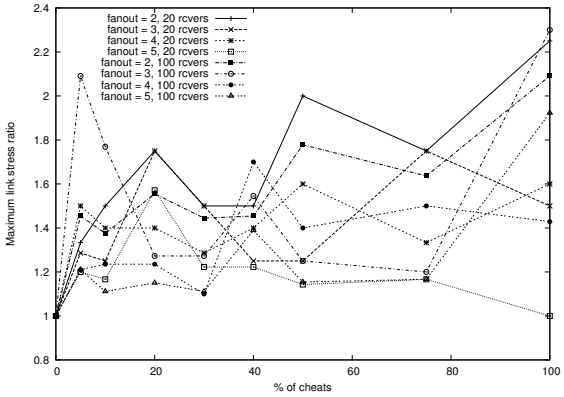

Fig. 2. Maximum link stress ratios in TBCP

[4]myns simulator – http://www.cs.umd.edu/∼suman/research/myns/

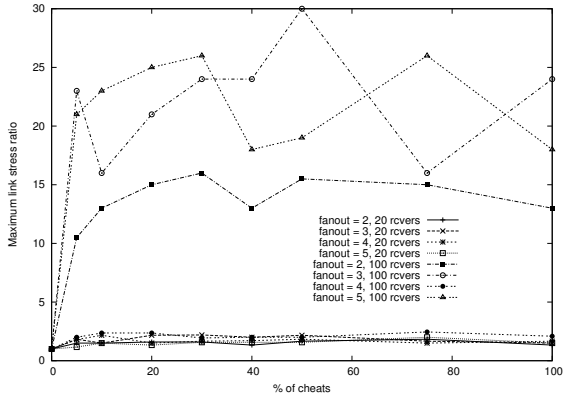

Fig. 3. Maximum link stress ratios in NICE

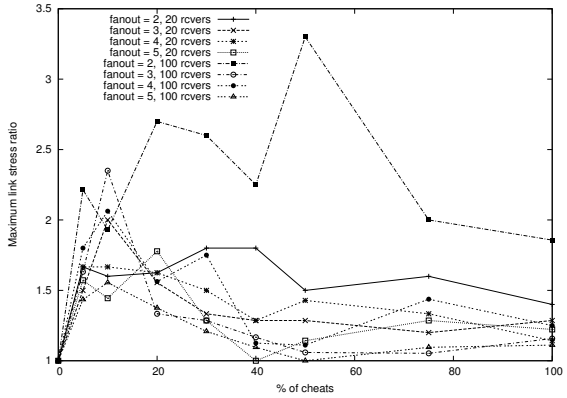

Fig. 4. Maximum link stress ratios in NARADA

We see that the effects of cheats are quite different in the various protocols. In HBM (figure 1) the maximum stress ratio progressively increases with the percentage of cheats because the centralized algorithm creates a shared tree using a distance database that is less and less related to the reality. As a consequence the resulting overlay topology is less and less efficient. However, we also observe that the maximum link stress ratio is better when most or all of the receivers cheat than when the cheats are in smaller numbers. This is because of the complete distance knowledge at the RP: when most of the receivers cheat, the vast majority of distances advertised to the RP are merely the real values shifted by a constant (except of course for the distances to the source which are advertised as zero, and therefore introduce some degree of randomness at the top of the tree). We have also observed that HBM almost always produces an overlay tree whose maximum stress is smaller than that of a random overlay tree. This is again due to the fact that HBM always strives to make the best usage out of its complete distance knowledge.

TBCP (figure 2) shows a smaller maximum stress ratio, that is somewhat independent of the group size and the number of cheats. This is explained by the facts that, in TBCP, the cheats have the luxury to ensure their own maximum fanout is reduced to 1, thus shifting the stress from the edge of the network (i.e. access links) towards links inside the network, as the tree growths in "length" (with long branches of cheats dangling from the root) rather than in "width". In other words, in TBCP, the effects of cheats is to shuttle data packets several times across the network, while cheats do the minimum data replication they can.

Because of the way TBCP overlay trees grow in "length" in the presence of cheats, the maximum stress ratio was always worse for random overlay trees than for TBCP. This is because random overlay trees tend to concentrate more traffic towards the edge of the network (as receivers limit their fanout to the same value as honest receivers in TBCP – and random trees therefore grow "wider" than TBCP trees with cheats), while still producing the shuttling effects of data packets across the network.

In NICE (figure 3), the situation is opposed to the one observed for TBCP. Indeed, NICE does not strictly enforce its fanout at all times (see section III-C.1) and therefore allows clusters to form whose membership is greater than dictated by the fanout value (in particular near the tree root), resulting in some cluster heads serving more children than "they should". As a result, we see that for small groups, the maximum stress ratio is small as most cheats would have occupied a position close to the root if they had not cheated anyway, thus resulting in an overlay tree very similar to a tree of honest receivers. For larger groups, the maximum stress ratios can show the greatest values we observed, because cheats create large clusters near the root, thus increasing the stress on the access links of nodes near the top of the overlay tree (including the root). We therefore see that NICE has actually a tendency to grow its overlay "too wide".

No surprisingly, the maximum stress ratio was almost always better for random trees than for NICE, because the fanout is strictly enforced in the random trees.

In NARADA (figure 4), we make the interesting observation that fewer cheats have a greater effect on the network efficiency of the protocol than when cheats are present in larger numbers. This is because the more cheats are present, the fewer the opportunities that exist for a cheat to create "forced" mesh links to nodes near the source, as the utility (see section III-D.1) associated with each cheat decreases. This decrease in utility is caused by the fact that, as the number of cheats increases, honest receivers and cheats alike are fooled into thinking that they are close to more and more members of the group, in essence choosing cheats as mesh-neighbours with equal probability. In other words, through the combination of enforcing a strict degree for all nodes in the mesh and the use of the notion of utility for the construction of the overlay tree, NARADA forces the cheats to share the profits of cheating. As a result, because cheats also report distances to other receivers as a fraction of the real values (this strategy is actually chosen to "beat" the utility function (see section III-D.2), the more cheats in the group, the more NARADA brings the mesh "back" to what it is when all receivers are honest (to the exception of the "lucky" cheats who connect directly to the root). This behaviour, although also observed

in HBM, is much more prominent with NARADA.

Figure 4 also shows that NARADA just does not cope very well with small tree fanout (or mesh degree) values, as reported in [4].

Not surprisingly given the discussion above, we found that NARADA produces stress ratios that are worse than those produced by a random overlay tree, when there is a small number of cheats. However, as the number of cheats increases, NARADA outperforms random trees as far as stress is concerned.

D. Stretch Ratios

There are basically two ways to grow an overlay tree: in length and in width. Growing an overlay tree in length results in lower network stress levels at the expense of higher stretch levels (as receivers are pushed away from the source/root and therefore observe longer delays along the tree). The opposite holds true for an overlay tree grown in width. Therefore, when considering average stretch levels over all receivers in the tree, we would observe that the application-level protocols that were showing the smaller stress ratios show the higher stretch ratios, and vice versa. To have a better understanding of the effects of cheats on the stretch levels of receivers in an application-level overlay tree, we will therefore focus separately on the effects of cheats on honest receivers and amongst the cheats themselves.

Figures 5 to 8 show the average stretch ratio for honest receivers in the presence of cheats.

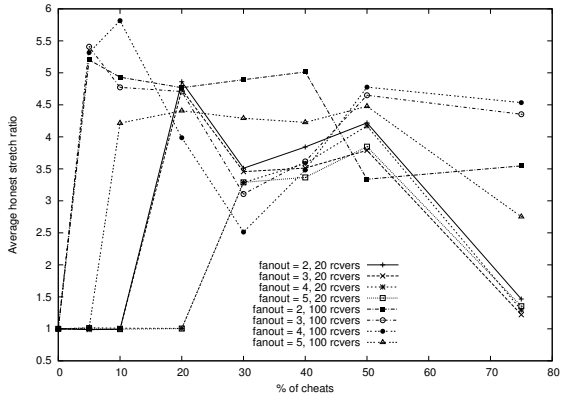

Fig. 5. Average stretch ratios for honest receivers in HBM

We see that in HBM (figure 5), the average stretch ratio for honest receivers rapidly increases, even with a small percentage of cheats. This is due to the fact that honest receivers are immediately moved away from the source, and this is particularly true with trees having a small fanout. We also observe that the average stretch ratio for honest receivers tend to stabilize as the number of cheats increases, because, as explained in section IV-C, the more cheats are present, the more accurate the RP's neighbour selection becomes (and therefore honest receivers end up in the "correct" part of the tree).

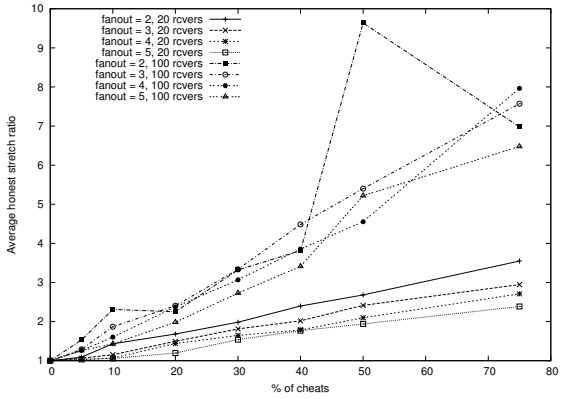

Fig. 6. Average stretch ratios for honest receivers in TBCP

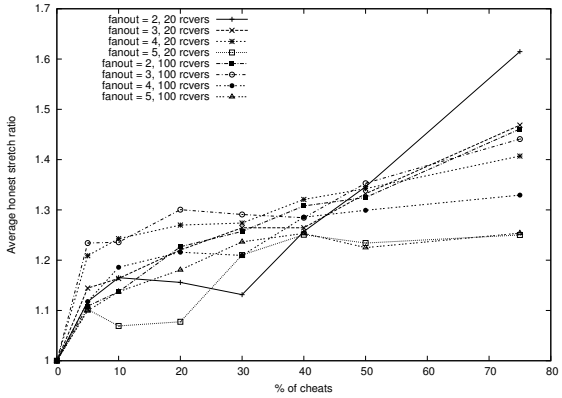

Fig. 7. Average stretch ratios for honest receivers in NICE

We see that in TBCP (figure 6), the average stretch ratio for honest receivers increases steadily, with the rate of increase proportional to the group size. This is because, as cheats occupy higher positions in the overlay tree, the remaining honest receivers get pushed towards the bottom of the tree and thus see an increasing stretch. Also note that even a small increase in stretch, when more cheats (with a fanout of 1) are added, can result in a more substantial increase in average stretch ratio for honest receivers, as this number of honest receivers decreases steadily (i.e. the increase in stretch is shared amongst fewer honest receivers).

Expectedly, NICE (figure 7) shows little average stretch ratio for honest receivers, almost independently of the group size. This is again due to the fact that, during normal operations of the protocol, clusters are allowed to grow bigger than dictated by the maximum fanout, leaving more unaffected honest receivers by the presence of cheats than in the other protocols.

In NARADA (figure 8), we see that the effects of cheats on the stretch of honest receivers stabilizes as the number of cheats increases, especially in small groups. This is because, as explained in section IV-C, NARADA operates in such a way that the actions of individual cheats tend to balance each other as the number of cheats increases. This effect is less

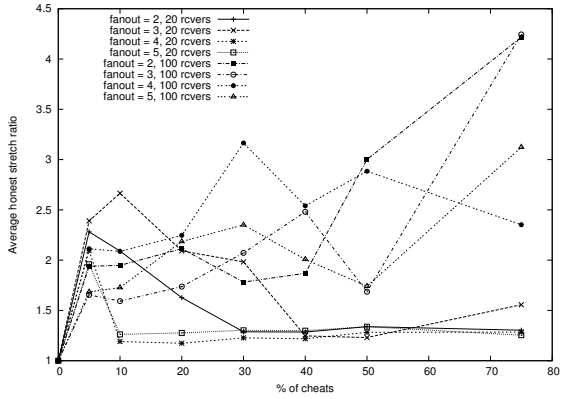

Fig. 8. Average stretch ratios for honest receivers in NARADA

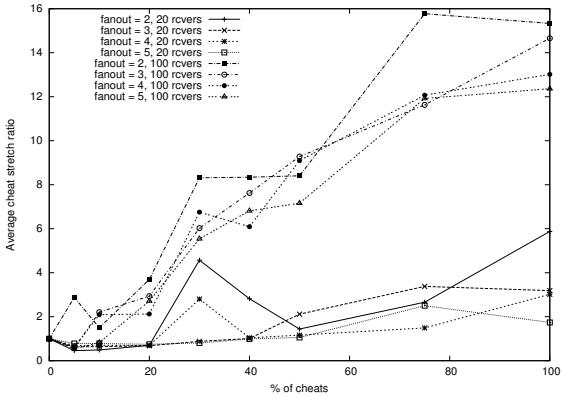

Fig. 10. Average stretch ratios for cheats in TBCP

noticeable for larger groups, though, as the overlay tree gets longer, and thus the effects on the stretch (of honest receivers) accumulate faster.

Figures 9 to 12 show the average ratio in stretch for the cheats themselves.

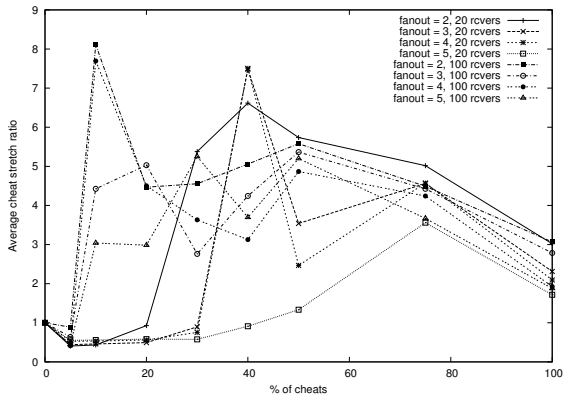

Fig. 9. Average stretch ratios for cheats in HBM

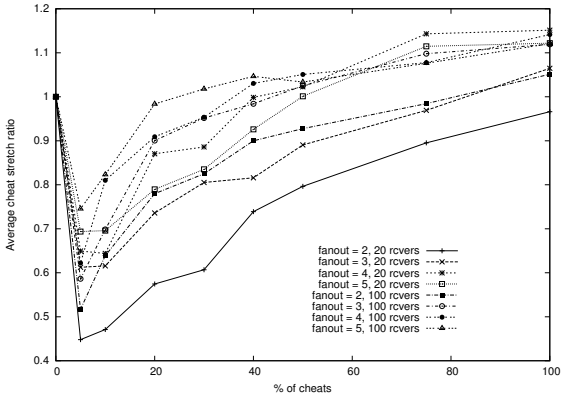

Fig. 11. Average stretch ratios for cheats in NICE

In all the protocols, we see a decrease in average stretch ratios for cheats when the number of cheats are small. This indicates that the cheats indeed get a better position near the source when cheating. However, as more and more cheats operate in an overlay tree, these compete with each other, and the rate of increase of the average stretch ratio for cheats give an indication of the ferocity of the competition.

This is particularly visible with HBM (figure 9). Cheats experience an average benefit only if the fanout enables them to be close to the source. Thereafter, the situation rapidly deteriorates because the additional cheats are located lower in the tree, beneath honest receivers. So when the number of cheats increases, honest receivers experience on average a slightly better stretch ratio than cheats. Yet this average hides a major discrepancy between cheats who succeeded to be directly attached to the source and others.

Figure 10 shows that competition between cheats is the fiercest in TBCP. This is because cheats that have already found a place in the tree, do not relinquish their position to new cheats joining, the latter getting pushed down the tree. Further more, the rate of "descent" towards the bottom of the tree is exacerbated by the fact that cheats use a fanout of 1, leaving other cheats far further away from the source than in the reference overlay tree where all receivers are honest.

There is very little competition amongst cheats in NICE (figure 11) and NARADA (figure 12). NICE, however, offers better opportunities for cheats to better their positions in the tree, thus resulting in lower average stretch ratio values for cheats.

Tables I to IV, indicate when the average stretch ratios for cheats become worse than in random trees. These tables show that, from the point of view of cheats, the *collective* benefit of cheating always eventually disappears, leaving the "average" cheat in a worse position than if the tree was random. However, this observation should be contrasted by the fact that cheats are selfish and that some of the cheats always see a dramatic improvement in their positions in the overlay trees, as illustrated in figures 13 to 16.

These tables show that NICE and NARADA provide more favourable conditions for cheats to collectively gain an advantage over the honest receivers, in term of stretch. Table II also confirms that competition amongst cheats is very high

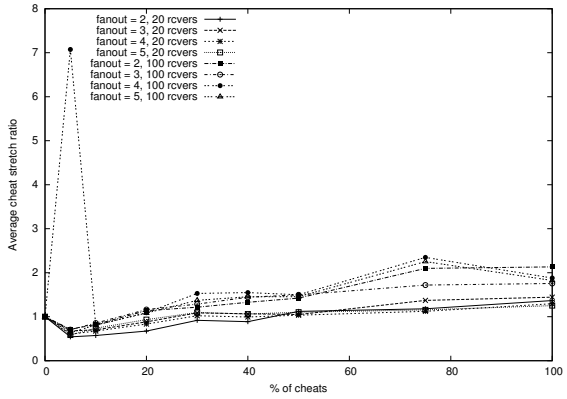

Fig. 12. Average stretch ratios for cheats in NARADA

group size	fanout	% of cheats
20	2	20 – 30
20	3	30 – 40
20	4	30 – 40
20	5	40 – 50
100	2	5 – 10
100	3	5 – 10
100	4	5 – 10
100	5	5 – 10

TABLE I

HBM CHEATS VS RANDOM TREE: % OF CHEATS WHEN CHEATS ARE BETTER OFF BEING IN RANDOM TREE.

in TBCP as a small number of cheats makes these cheats collectively better off in a random tree.

Figures 13 to 16 show that cheating in NICE and NARADA is potentially very rewarding for individual cheats, while the benefits of cheating for an individual are similar in HBM and TBCP. It should be noted that NICE gives more chances to more individual cheats to improve dramatically their position in the tree, then any other of the protocols studied in this paper.

Finally, for HBM, TBCP and NARADA, we have observed that the maximum stretch ratio for honest receivers can be in the order of several hundreds (i.e. one of the honest receivers

group size	fanout	% of cheats
20	2	20 – 30
20	3	30 – 40
20	4	20 – 30
20	5	40 – 50
100	2	0 – 5
100	3	5 – 10
100	4	5 – 10
100	5	10 – 20

TABLE II

TBCP CHEATS VS RANDOM TREE: % OF CHEATS WHEN CHEATS ARE BETTER OFF BEING IN RANDOM TREE.

group size	fanout	% of cheats
20	2	never
20	3	75 – 100
20	4	40 – 50
20	5	40 – 50
100	2	75 – 100
100	3	40 – 50
100	4	30 – 40
100	5	20 – 30

TABLE III

NICE CHEATS VS RANDOM TREE: % OF CHEATS WHEN CHEATS ARE BETTER OFF BEING IN RANDOM TREE.

group size	fanout	% of cheats
20	2	40 – 50
20	3	20 – 30
20	4	20 – 30
20	5	20 – 30
100	2	10 – 20
100	3	10 – 20
100	4	0 – 5
100	5	10 – 20

TABLE IV

NARADA CHEATS VS RANDOM TREE: % OF CHEATS WHEN CHEATS ARE BETTER OFF BEING IN RANDOM TREE.

is several hundred times further away from the source than it was in the reference tree). These situations occur when the honest receiver is actually physically close to the source and therefore gets displaced relatively very far by the cheats. The noticeable exception is of course that in NICE, the honest receivers are relatively undisturbed by the cheats and the observed maximum stretch ratio for honest receivers are small (usually less than 10).

V. DISCUSSIONS

In this paper, we have studied the impact of simple cheating on the performance of application-level multicast overlay trees.

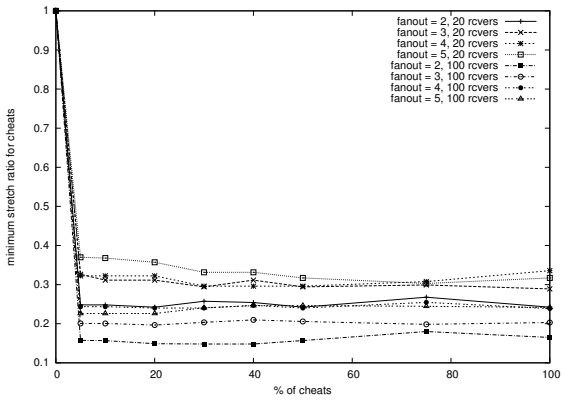

Fig. 13. Minimum stretch ratios for cheats in HBM

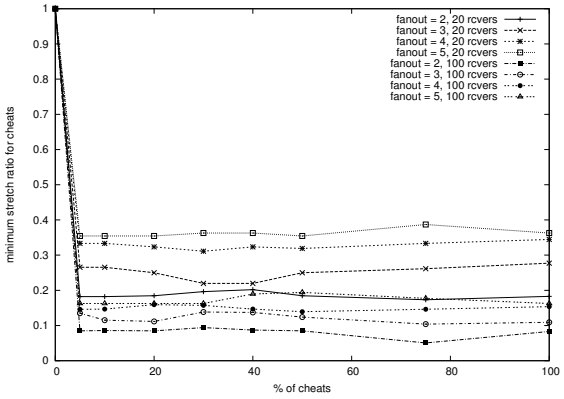

Fig. 14. Minimum stretch ratio for cheats in TBCP

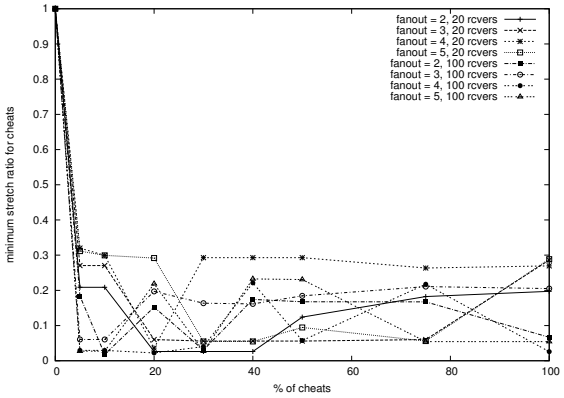

Fig. 15. Minimum stretch ratio for cheats in NICE

We have shown that simple cheating strategies always have negative impact, either on the performance of the tree as perceived by its nodes (both cheats and honest receivers), or on the underlying physical network, or on both.

We have also witnessed a range of responses to the cheating strategies from the studied protocols. However, none of the studied protocols coped well, in the presence of cheats, for all the performance aspects described. Actually, none of these protocols were explicitly designed to deal with cheats, and all showed, at various point of the study, that their performance could quickly degrade to be worse than the performance exhibited by a random tree.

Although the simple cheating techniques used in this study were tailored to the specific protocols, it is worth noting that they all exploited the fact that the protocols relied on receivers to take their own distance measurements and either make independent decisions based upon these measurements or advertise these to other nodes. In the case of the distance metric used in this study (the RTT), detection of a cheat advertising dramatically reduced distances seems rather straightforward: the node which the cheat claims to be close to, can always check its distance to the cheat, with tamper-proof probes (e.g. probes that do not contain any timestamp information or that have undergone a cryptographic modification such as hashing).

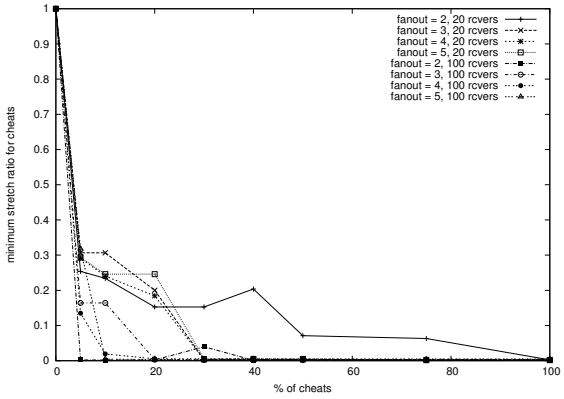

Fig. 16. Minimum stretch ratio for cheats in NARADA

However, in order not to jeopardize scalability by increasing the measurement overhead too much, such distance checks should probably be carried out as a periodic sampling process that eventually detects cheats. Note that such an approach is only applicable for metrics whose values are independent of the point of measurement (which is the case for the RTT between two points and measured at either point, but certainly not for the delay between these points). The sampling methods should probably also make use of statistical methods to help cope with natural variations in the measured values.

Nevertheless, it is not clear at all how effective such simple distance sampling methods would be in the presence of several cooperating cheats, since at the very least, the actual distance between these may never be reliably verified. In such a case, correlating the distance measurements taken among several application-level multicast nodes may be a way ahead, but doing so in a distributed fashion without requiring complete knowledge, at all nodes, of the group and its measurement matrix is a challenging problem.

On the other hand, it is probably impossible to ever detect, or prevent, a cheat which delays measurement probes systematically, in order to artificially increase measured distances. Such an "attack" can even easily be implemented by putting a proxy in front of the node which desires to cheat. As a result, cheat detection methods may just provide a "high-pass filter" on measurement matrices (i.e. a method to weed out artificially low distances but not artificially high ones). We will study the impact of such detection methods on the performance of the resulting overlay trees in the presence of cheats in our future work.

Designing general cheat detection and/or prevention techniques for various types of metrics is, we believe, an open research challenge. Indeed, there is always the danger that such techniques be designed on an ad-hoc basis, depending on the metric and cheating methods used, resulting in some kind of "cat and mouse" situation with cheats.

This paper was only concerned with selfish cheats operating independently. The case of cooperating cheats, maybe using evolutionary cheating strategies, is an open research challenge.

As future work, we will also study the effects of simple cheating on coordinate-based methods, and the application-level multicast protocols built on top of peer-to-peer systems (see section II). For such protocols, the cheats will not only have the opportunity to lie about their distance to other nodes, but they will also be able to carefully "choose" their place in the corresponding virtual geometric space.

REFERENCES

[1] L. Alchaal, V. Roca, A. El-Sayed, and M. Habert. A vprn solution for fully secure and efficient group communications. In *8th IEEE Symposium on Computers and Communications (ISCC'03), Kemer-Antalya, Turkey*, June 2003.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *ACM SIGCOMM*, Aug 2002.

[3] Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *IEEE Infocom*, Mar 2000.

[4] Y-H. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *ACM SIGMETRICS*, pages 1–12, Santa Clare, CA, USA, June 2000.

[5] S. Deering and D. Cheriton. Multicast Routing in Datagrams Internetworks and Extended LANs. *ACM Trans. Comp. Syst.*, 8:85–110, May 1990.

[6] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, Jan/Feb 2000.

[7] A. El-Sayed, V. Roca, and L. Mathy. A Survey of Proposals for an Alternative Group Communication Service. *IEEE Network*, 17(1):46–51, Jan/Feb 2003.

[8] J. Lieberherr, M. Nahas, and W. Si. Application-Layer Multicast with Delaunay Triangulations. In *IEEE GLOBECOM*, Nov 2001.

[9] L. Mathy, R. Canonico, and D. Hutchison. An Overlay Tree Building Control Protocol. In *Proc. of Intl. workshop on Networked Group Communication (NGC)*, pages 76–87, Nov 2001.

[10] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: an Application Level Multicast Infrastructure. In *3rd USENIX Symposium on Internet Technologies*, San Francisco, CA, USA, Mar 2001.

[11] S. Ratnasamy, M. Handley, and S. Shenker. Application-Level Multicast using Content Addressable Networks. In *Proc. of Intl. workshop on Networked Group Communication (NGC)*, Nov 2001.

[12] V. Roca and A. El-Sayed. A Host-Based Multicast Solution for Group Communications. In *IEEE Intl. Conf. Networking*, Jul 2001.

[13] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: the Design of a Large-Scale Event Notification Infrastructure. In *Proc. of Intl. workshop on Networked Group Communication (NGC)*, Nov 2001.

[14] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *IEEE Infocom*, pages 40–52, Mar 1996.

[15] B. Zhang, S. Jamin, and L. Zhang. Host Multicast: a Framework for Delivering Multicast to End Users. In *IEEE Infocom*, Jun 2002.