Orta - an Overlay for Real Time Applications

Stephen Strowes
Department of Computing Science
University of Glasgow
Email: strowesd@dcs.gla.ac.uk

Colin Perkins
Department of Computing Science
University of Glasgow
Email: csp@csperkins.org

***Abstract*— This paper presents Orta, a new peer-to-peer network overlay protocol intended for use with interactive real-time conferencing applications. The implementation is presented as a reusable software library, that is not tied to any existing application. One application, the UCL Robust-Audio Tool, is modified to use this library rather than IP multicast as a proof-of-concept implementation. We present the protocol design, along with evaluation results describing the performance of the overlay, with focus on its usefulness for real-time applications.**

I. Introduction

Given the peer-to-peer nature of the Internet, and increases in network and host capacity in recent years, it might be expected that group conferencing applications would become widespread. There are, however, many problems in designing real-time group conferencing applications, not least of which is the issue of scalable many-to-many data dissemination between members of the group.

The original design for the IP layer saw it as a stateless unicast packet delivery mechanism, with no direct support for multiparty applications. The IP multicast extensions [1] added direct support for many-to-many applications, with the network replicating packets as required. These extensions add complexity, in the form of per-group state, to the network, but allow peer-to-peer conferencing applications to be built with ease. Indeed, multicast was initially targeted at applications such as audio conferencing, video conferencing, electronic whiteboards, or indeed any application which involves participation between many/all participants.

Unfortunately, IP multicast is currently not as widespread as anticipated. Aside from economic factors preventing its widespread use, there are also concerns about the scalability due to the router state needed to support many small groups [2], and security of content/authorisation of users. Due to these concerns, various approaches to providing application level multicast have been developed [2]–[18], each of which tackles a slightly different intended application. These systems build some form of overlay multicast distribution network, which is almost always much more efficient than using IP unicast alone, but less efficient than IP multicast; the overlay multicast incurs duplication of packets at hosts where IP multicast would duplicate at the routers between the source and the recipients, but requires that far less than the one packet per recipient be sent per cycle when using IP unicast.

While some previous work attempts to directly address multi-source multicast using an overlay structure [2], [4], [8], [15], there is no published data relating to the performance of the systems when running in the multi-sender case, as they would be when running a real-time interactive videoconferencing application. Given the lack of numerical results in the area of group communication applications over overlay multicast, it is difficult to discuss the performance of overlay multicast in the setting for which IP multicast was originally designed.

We know, however, that such systems are possible to build. Indeed, the popular Skype[1] group communication tool builds some form of overlay multicast to send/receive data. Skype is, unfortunately, closed-source commercial software and no technical discussion on what techniques were used to build the underlying data structures is provided (what little is known about the Skype protocol [19] was obtained by reverse engineering of packet traces, and is not sufficient to understand the overlay multicast algorithms used).

In this paper, we present Orta, an overlay multicast protocol designed to cater for interactive applications, such as real-time multiparty audio conferencing. We describe the Orta protocol, and evaluate the performance of an initial implementation on a range of scenarios.

The contributions of our work are: (1) An improvement of an existing protocol designed to cater for interactive applications with many participants; (2) An open implementation of the Orta protocol, which can be freely used in applications as an alternative to IP multicast.

The remainder of this paper is organised as follows: Section II discusses certain constraints which have to be met to support interactive applications and, in particular, audio conferencing applications. Section III covers related work in the area, while Section IV covers the existing work on which Orta was based. Section V provides detailed information on the Orta protocol, with evaluation results and discussion of the behaviour of the protocol in Section VI. We conclude the paper with some possible future directions in Section VII, and conclude in Section VIII.

II. Group Conferencing & Constraints

Interactive applications such as group conferences place some requirements on any overlay over which they might run. In the following, we briefly review these constraints, to motivate the development of the Orta protocol.

[1] `http://www.skype.com/`

An interactive application requires swift reorganisation of the overlay in the face of changing network conditions to ensure all participants are fully aware of each other at all times. For example, members departing the group; members who rely on the leaving group member for packet delivery require that the overlay restructure as quickly as possible, with as little disruption to the packet stream as possible. Likewise, a new member must be able to interact with the rest of a group upon joining that group. In particular, a break in a distribution tree means that some participants may be disconnected to others leading to a break in conversation and loss of information.

Similar to that above, low packet loss rates would be desirable, so reorganising the overlay structure must be performed in a timely manner. Zero packet loss due to reconfiguration of the network is preferred, though occasional packet loss is acceptable when compared to longer bursts of packet loss.

Further, for conversational audio to be feasible, 400ms is considered to be an approximate upper bound on the round trip time of audio packets in a communication [20]. The 400ms 'limit' should not only consider raw RTT values, however. If we are to consider real-time audio as the data the overlay shall be carrying, there are additional constraints to meeting that 400ms upper bound. The sender waits for around 20ms to grab an audio frame, and may take a few milliseconds to encode that frame. The receiver may buffer this packet for a few milliseconds to take into account jitter on the input stream of packets, and again for the decoding. With all these factors taken into account, the additional delay may be anywhere between approximately 50ms and 70ms, and therefore the raw RTT to be met must be less than 350ms or 330ms respectively. Clearly the overlay must be able to organise itself to try and minimise latencies between hosts, and therefore must react quickly to new information regarding shorter routes between peers.

For a network overlay to be able to cater for an audio conferencing application, these factors must be considered in the design of the protocol.

III. Related Work

The most interesting and closely related work to multi-source multicast using peer-to-peer overlays is generally split between two approaches for achieving this end-goal: tree-based approaches, and mesh-based approaches. Tree-based approaches (e.g. [4], [9], [15]) involve directly constructing distribution trees from any source to all other group members, with members explicitly choosing their parents from the other hosts in the system of which they are aware. Mesh-based approaches instead have nodes build up a richer mesh structure consisting of many connections between different nodes, on top of which distribution trees are constructed. The use of multiple connections allows some level of redundancy when nodes fail or leave the group; further, redundant connections require that a routing protocol be run in the application to construct loop-free forwarding paths between group members [21]. This robustness makes mesh-based overlays suitable for interactive conferencing applications, whereas tree-based overlays cannot meet the constraints for reconfiguration time and robustness.

Of the mesh-based overlay multicast protcols published, perhaps the most developed is the End System Multicast (ESM) architecture using the Narada protocol [8], [22], [23], developed at Carnegie Mellon University. Narada was the primary inspiration for Orta, and is discussed in detail in section IV.

In addition to Narada, there are other approaches to forming a mesh structure over which to route data. Of note is the protocol described in [2], in that it creates the basic mesh structure, but does not go on to generate distribution trees within that mesh. This presents the possibility of a more naïve, brute-force implementation of connecting multiple participants in a conference, and is limited to group sizes of around 10 members.

Scattercast, [7], uses a protocol called Gossamer, based on the ESM work, which attempts to cut down control overhead, to expand to larger multicast groups. The purpose of Scattercast is to provide a large multicast infrastructure which uses SCXs (ScatterCast proXies) at known locations; these SCXs are application-level software components, often replicated to allow load balancing and redundancy, which form an overlay network over IP unicast links. Clients can then connect to SCXs using IP multicast if it's locally available, or by using normal unicast links if not.

Work presented in [24] offers another method of deploying a mesh-based overlay multicast solution, by generating a hierarchy of meshes. The aim of the work was to allow for self-organising overlay networks to scale to the order of tens of thousands of hosts, but does not look at the possibility of massive conferencing applications where all of those thousands are entitled to be sources of data. The hierarchy formed generates a mesh of lead nodes, those lead nodes being elected within a cluster of nodes, also organised into a mesh. This form of organisation reduces the amount of control overhead required to maintain connectivity across all group members from that of the case where we have one single mesh encompassing all nodes.

HyperCast, [14], is one other system which tackles overlay multicast groups with many-to-many semantics. HyperCast forms a hypercube from group members, group members becoming vertices in the hypercube. Spanning trees are embedded into the hypercube easily, while control traffic is transmitted along the edges of the hypercube.

Skype on the other hand, is an example of a peer-to-peer conferencing and messaging system currently in use on the Internet, which also offers lookup capabilities on users within the system. An analysis of the protocols Skype uses are presented in [19]. One interesting point to note on the analysis of the Skype system is that hosts forwarding separate data streams onto other hosts further down the overlay structure will mix together data streams, thus reducing packets which have to be sent to nodes further from sources of a distribution tree. Much of the previous overlay multicast work discussed already works on the idea of replicating a packet as and when

necessary for it to reach all endpoints on the overlay, but audio applications do offer this opportunity to combine data streams together, thus combining two or more packets of data into one.

Of these, it is the Narada protocol which we feel closest suits our goal of providing an overlay to be used by interactive applications, and so what follows largely centres on what improvements can be made to Narada to suit our needs.

IV. The Narada Protocol

In the following we introduce the Narada protocol, from which Orta is derived. A more detailed description of Narada is available in [8].

Narada is a fully distributed peer-to-peer data distribution protocol, designed to be self-organising and self-improving. It relies on peers in the network observing certain performance metrics, which are used to gradually improve the state of the overlay. The metrics observed are application specific: for real-time communications, latency is typically used as the metric, due to the timing constraints mentioned in Section II. Available bandwidth is also an issue once transmission quality, the potential audience (and their potentially varied connection types), and the size of the group are taken into consideration.

Narada employs a two step process to building distribution trees. Firstly, a rich graph, termed a *mesh*, is constructed between members of the group (more fully connected than a tree, less so than a fully connected graph); then a routing protocol is run over the mesh to construct spanning trees for distributing data, each tree rooted at a source within the group, thus allowing for the possibility of group communication. The following observations motivate this approach:

- The construction of one tree is prone to failure, as it only takes one node failure to disrupt the tree structure.
- A single tree cannot be optimised for all participants.

The two-step approach allows the mesh layer to handle group membership, along with problems such as how to optimise the mesh or how to repair a partitioned mesh structure, with the routing protocol running independently on top of this layer. Multiple links between peers allows for alternate routes on members leaving the group. The additional links between nodes also allow for each distribution tree to potentially be of a higher quality for the host to which it belongs than a single, shared spanning tree would be.

Narada peers constantly probe existing links to ensure that they remain suitable for the application after addition to the mesh. This is done by sending regular ping packets to neighbours to measure latency between peers. Key to the protocol is its ability to probe existing and potential links in the mesh structure, and the mechanisms to add or remove links from that structure, all with the goal of improving the set of links included in the mesh.

Of note is that the mechanisms for adding or removing links from the mesh are different, and evaluate the utility of a link differently.

Narada runs a distance vector algorithm over the mesh, and calculates the distribution tree for each source using reverse shortest path between each recipient for each source, as is done in DVMRP [25]. The distance metric advertised via the distance vector algorithm is that of the weight of the link derived by whatever application specifics are required (e.g. latency), rather than simply the number of hops from source to destination. While conceptually the routing algorithm could be viewed and implemented as an entirely separate entity from the mesh structure upon which it runs, it makes sense for the distance vector information required to be passed between peers to be sent as part of the regular refresh packets Narada defines. Thus, the process of propagating routing information, as per the distance vector algorithm, is also the process by which members are able to monitor the liveliness of the other members.

V. The Orta Protocol

While Narada was designed to support real-time applications such as video conferencing, it has primarily been used with one-to-many non-interactive streaming applications. Streaming media applications have relatively long media playout buffers (in the order of seconds) and can tolerate brief disruptions in delivery; the relatively slow convergence time of Narada after a change in group membership is not a problem for these applications. When considering interactive conferencing applications, however, it becomes clear that rapid convergence is important. Interactive applications buffer only a small amount of data, and need a protocol that can rapidly adapt to changes in network conditions and group membership. In the following we describe a new protocol, *Orta*, which we have developed to improve the performance of interactive real-time applications.

Like Narada, Orta is a fully distributed, self-organising peer-to-peer data distribution protocol which attempts to improve overlay quality during the lifetime of a session. Orta is based on Narada, but distributes the control information in a different manner, to inform the entire group of state changes as quickly as possible and improve convergence times after a change in group memberhsip.

As discussed in Section IV, a key feature of the Narada protocol is that links can be added and removed over the lifetime of a group, so that performance of the mesh is improved. Narada uses a distance vector routing algorithm to embed distribution trees within the mesh. While this is simple to implement and not computationally expensive, such algorithms do not allow for quick propagation of information relating to network state changes and are slow to converge. Orta replaces the distance vector algorithm with link-state routing to maintain link state at each peer, over which Dijkstra's shortest path algorithm can be run. By changing to a link-state routing mechanism, state changes at any group member are flooded to the entire group, so all group members are informed as quickly as possible about the state change. This improves the convergence time of Orta, an important feature for a protocol designed for use with interactive teleconferences.

While the change to link state routing requires more computation at each peer compared to what is required in Narada, the rate of propagation of information provides much more

recent information for each node which can then be used to provide a more robust network structure for the carrying of multicast data. The flooding nature of the protocol means that peers will be less likely to arrive at differing states capable of creating loops in the network.

This rather major modification to how peers interact also allows for the algorithm responsible for dropping links to be changed, to mirror that of the algorithm designed to allow Narada to add links. Measuring the utility of a link by the same mechanism on both adding and dropping a link allows for the same threshold calculation to be used, and should offer more reliable decisions made by the link dropping mechanism.

In the following, we now describe the operation of the Orta protocol in detail.

A. Overlay Construction and Group Membership

Much as specified for Narada, Orta uses a two-stage process to build the overlay structures required to route data from any source in the group to all receivers: the initial stage builds a control mesh, then a link-state routing algorithm is run to construct spanning trees from each source for data distribution purposes.

No single Orta peer is solely responsible for maintaining group membership data; this responsibility falls onto all peers in the group. Sharing this burden limits group sizes by requiring that membership state be maintained and distributed to all peers, but allows for a high level of redundancy, and enables a new member to join by contacting any peer. Link state data information flooded on arrival of a new member, on a member leaving, on the addition of a new link, and on the removal of a link.

Each member regularly signals to its neighbours within the mesh that it is still alive by means of a refresh packet, which contains a sequence number. These refresh packets are also used to carry the routing information needed for data delivery, as described in Section V-D, with each peer flooding information regarding significant link weight variation (i.e. delay variation) between it and its neighbours on a regular basis (every 30 seconds, say).

On receipt of any of these flood packets, a peer runs Dijkstra's shortest path algorithm using the link state it has to generate a shortest path spanning tree from each group member to generate the routing table at each peer. This can clearly lead to a lot of computation being performed at each peer, though ideally network conditions would not change all too frequently; if no state has changed, no computation is required.

B. Overlay Maintenance

Orta requires that each member monitor its own links, and flood information to the rest of the group regarding those links. Each group member collects information about the state of each link in the mesh on receipt of floods from other members. As in Narada, the burden of maintaining group membership data falls on each member of the group; in addition Orta peers must store link-state information.

Orta peers send information relating to links to local neighbours to the entire group by flooding the information (as opposed to Narada peers, who send known information about the whole group only to neighbours). Thus, the peer responsible for probing an existing link in the mesh is always able to provide the group with the current information regarding the state of that link. This method of propagating control information allows for: 1) Dijkstra's shortest path algorithm to be calculated over the link state to derive routing tables at each member, covered in Section V-D; and 2) faster reactions to changes in network conditions, owing to the nature of the propagation mechanism.

If the weight of a link has not changed since the last cycle, a peer may opt to not send information regarding that link. Members must still send a refresh packet containing a sequence number as normal, and so it might be the case that on many refresh cycles, little more than the sequence number is flooded for any given group member. Each member must still forward information on each link it owns, even if at a reduced rate, for the same reasons as mentioned above: to allow for a gradual repair to take place if the mesh becomes partitioned and link state changes at any point during that partition.

Looping of flooded packets is avoided by having said packets carry the incremented sequence number from its source. This simplifies the handling of flooded packets somewhat, even though all operations over control state should be idempotent; for example, receiving two copies of a packet to add a link from member A to member B should not result in two link entries for $A \rightarrow B$.

To reduce the amount of information flooded relating to each link in the group, the Orta protocol requires that a distinction be made between the actual weight, and the advertised weight, of a link. The actual weight of a link is the current latency observed over a link, while the advertised weight is a recently observed latency on that link. The advertised latency need only change when the difference between the actual latency and the advertised latency is sufficiently large. This should also help reduce computational load at each peer, assuming network conditions are stable enough so as to not force an update on the advertised weight.

1) Members Joining: Orta, like Narada, does not concern itself with lookup mechanisms to locate an existing group member. Location of a group member is assumed to take place via some other mechanism. Any Orta peer is capable of admitting entry of a new peer to the group.

On entry to the group, information on the new member and new link is flooded to the entire group, thus allowing for distribution trees to immediately take into account the new member, even though the initial link chosen might not be the best possible for the group. Once a new member is admitted, mechanisms for improving the quality of the mesh take over.

On a successful join the existing member should send the new host both member information and link-state information; this information should allow for quick integration into the mesh network structure.

These methods allow for a substantially faster integration

into the group than the distance-vector mechanism Narada employs, which will require some time for the group to learn about the new member, and also for the member to learn about the rest of the group.

2) Members Leaving: On leaving, a member informs the group of their departure and also of any links which will be dropped by this. Again, this allows for the peers to immediately be able to restructure their distribution trees such that the new mesh topology immediately comes into effect. Orta peers may leave immediately after informing the group of their departure, in contrast to Narada peers which are required to forward data until such a time as the routing protocol has routed around the leaving host (by means of introducing a "transient forward" value, guaranteed to be higher than any weight achievable by a real link yet also lower than the infinite cost which signifies no link is in place).

Unlike Narada, Orta does not specify a transient forward state for a link, nor is a peer required to continue to forward packets along a link for some reasonable length of time. Provided the mechanisms which deal with adding and removing links have created a well-connected mesh, the departure of a peer will not cause problems.

3) Member Failure: In the event of a member failure, state relating to that member will linger in each other member's local state. This can lead to data loss due to incorrect routing tables. Orta employs the same mechanism as Narada for detecting and repairing partitions in the mesh structure, as detailed in Algorithm 1, and relies on peers regularly sending sequence numbers to the rest of the group.

Orta requires that a peer who discovers a failed member flood to the rest of the group that both the failed member and any links to or from that member are to be removed, by making use of the link-state information held locally. This should naturally happen on both sides of a partition, thus clearing any state relating to the failed member at all remaining peers.

Orta peers must be able to incorporate into their local state information relating to unknown members or links carried in a refresh packet, as state changes on one side of a partition will not be observed by members on the other.

C. Improving the Quality of the Mesh

Key to the Orta protocol is that links are added and removed in a bid to improve the quality of the mesh based on some metric. The mechanism employed in Orta for dropping links is entirely different to that of Narada, and is shown in Section V-C.2.

1) Adding Links: Orta peers add mesh links in exactly the same way as defined by Narada, and require that peers attempt to seek out links between each other which offer some significant improvement for the data trees, or alternatively add links to help ensure that a partition in the mesh structure is less likely to occur.

Group members randomly select peers within the mesh to which they are not connected, and probes those peers to determine the properties of the links between them. If latency is the metric the protocol is concerned with, this is achieved by sending them a ping packet. An Orta peer, by holding all link state for the group, and receiving a ping packet for a potential neighbour, uses Algorithm 2 to determine whether a link to another group member should be added. The utility of a link is essentially a measure of how much that link improves the quality of the mesh; the utility of a link lies in the range $0..1$, with 1 being the highest attainable utility.

On adding a link after receiving a favourable turnaround time to a ping packet, the round trip time on that packet is advertised as the latency for the new link, which the normal mechanisms used to monitor the weights of links to neighbours can then modify over time.

Algorithm 1 Algorithm used by peer i to detect and repair partitions in the mesh structure.

```
Let Q be a queue of members for which i has stopped receiving sequence number updates for at least T_min time.
Let T_max be maximum time an entry may remain in Q.
while true do
  Update Q
  while !Empty(Q) and Head(Q) is present in Q for ≥ T_max time do
    j = Dequeue(Q)
    Initiate probe cycle to determine if j is dead, or to add a link to it.
  end while
  if !Empty(Q) then
    prob = Length(Q) / GroupSize
    with probability prob do
      j = Dequeue(Q);
      Initiate probe cycle to determine if j is dead, or to add a link to it.
    end
  end if
  sleep(P) // Sleep for time P seconds.
end while
```

Algorithm 2 Evaluate Utility of link L

```
utility = 0
for each member, M, such that M ≠ localhost do
  L_n = new latency to M, with L in place
  L_c = current latency to M, without L in place
  if L_n < L_c then
    utility+= (L_c − L_n) / L_c
  end if
end for
if utility > threshold then
  add link L
end if
```

2) Removing Links: Orta benefits from having all link state available at each peer by being able to mirror Algorithm 2 to evaluate a link by the same measure when dropping links.

The requirement on such an algorithm is that the utility lost on dropping a link is the same as if the link were to be

added again immediately, given all other network conditions remaining stable.

Given that Orta peers store complete link-state information for the peer group, there is enough information available locally at each peer to calculate the utility of a link by running a modified version of the algorithm used to add links.

Algorithm 3 Drop Links

$utility \Leftarrow 0$
$L \Leftarrow$ link L to randomly selected neighbour
for each member, M, such that $M \neq localhost$ **do**
  $L_n \Leftarrow$ new latency to M, without L in place
  $L_c \Leftarrow$ current latency to M, with L in place
  if $L_n = \infty$ **then**
    return
  else if $L_n > L_c$ **then**
    $utility \Leftarrow utility + \frac{(L_n - L_c)}{L_n}$
  end if
end for
if $utility < threshold$ **then**
  drop link L
end if

Algorithm 3 outlines the actions Orta takes to determine the usefulness of a link. Given unchanging network conditions, the utility of a dropped link would be exactly the same as if it were to be added again. For this to work, the threshold for dropping a link must be calculated as if that link were not in place (as the utility if the link were being added would be compared to the threshold before the addition of the link).

Little has to be changed from the Algorithm 2 for adding links; it is simply a reverse of the link addition. New overlay distances are those without the link being evaluated while current overlays distances are those with the link in place; if the utility of a link is below a given threshold – rather than above – the link will be dropped.

In normal circumstances, the dropping of a link will not create a partition in the mesh, provided that when checking the latencies, the observation of any infinite length links is enough to determine that the link should not be dropped.

By introducing this altered mechanism for dropping links, an Orta peer can judge the adding or removal of links against the same threshold calculation.

3) Calculation of the Threshold: The threshold on which the adding and removing links in both Narada and Orta is difficult to specify for all network conditions. The threshold must be dependent on the size of the group, the number of neighbours a peer has, and also the number of neighbours the peer at the other end of the link has. Multiplying by these numbers alone would give a threshold value far too high to be able to add any links, though the whole lot can be multiplied by some small constant to deliver a useful threshold value.

It makes sense for the threshold to increase sharply once a peer has achieved a handful of links; the idea behind the threshold is that a peer should be able to achieve some links relatively easily, and after attaining those it shouldn't be able to add further poor quality links, only higher quality links.

Given the factors that the threshold must rely on, and a peer A with its neighbours B, the threshold might be easily calculated as:

$$\begin{aligned} n = &\quad \text{no. of neighbours(A)} * \text{no. of neighbours(B)} \\ m = &\quad \text{number of members} \\ threshold = &\quad const * n * m \end{aligned}$$

The calculation of this threshold value is covered further in Section VII.

D. Data Delivery

Routing tables are re-calculated on any link-state change using Dijkstra's shortest path algorithm. The routing tables stored reflect the nature of the peer group being one-to-many: rather than having a lookup table of *destination* against *next hop* as might be seen in a conventional IP router, here it makes more sense to store *source* against *next hop(s)*.

To calculate the routing table then, Dijkstra's algorithm is run for each source in the peer group. The local peer can then simply trace its own location in the spanning tree created, and store the outgoing links on this tree, if any, in the routing table against the source. If the local peer is a leaf-node on the tree, no entry need be added to the routing table for this source.

Using link-state flooding, however, more computation is required to arrive at the same result, though the outcome of this computation should be more up-to-date, and therefore precise. The worst case complexity of Dijkstra's in $O(m\log n)$, where m is the number of links in the network, and n is the number of nodes. Consider that at each peer, the algorithm will run once for each member in the group; with this in mind, the computational complexity of recalculating routing tables using this scheme is actually $O(mn\log n)$.

Once the routing tables have been calculated, they can simply be used for lookup purposes on the receipt of any data packet. The routing code must then both send data up toward the application layer, while also duplicating the packet for any outgoing links dictated by the routing table.

The routing table can only be affected by control traffic when link-state changes, at which point the routing table must be recomputed. Given that transmission times on control information packets are bound primarily by latency between hosts in the system, there will be short periods of time whereby the system is yet to converge on the same solution. Due to this fact, and that the fact that the system will constantly attempt to improve the quality of the links in the mesh, it is entirely likely that some packets may be lost or duplicated during transition periods. Further, routing tables at peers might allow looping of data packets during a transition period. These transition periods should not last long, however, due to the nature of the flooding mechanism used to distribute the control traffic. Data packets carry a time-to-live field, which would prevent looping packets from flooding the overlay until the group is destroyed.

The increased frequency of state changes with a larger group precludes this protocol from being used for larger groups beyond the order of a few dozen members, due to the amount of computation taking place. The combination of increased computation, and more frequent state changes suggests that as group size increases, the time taken to reconfigure all routing tables in the group will take longer.

Orta uses UDP for data transport, leaving congestion control to the protocol being carried through the Orta links. This simplifies the design of Orta, and allows for a great deal of flexibility. Conferencing applications would typically use RTP as their transport; RTP profiles for different types of data provide their own method of offering congestion control, so to force an assumption at a lower level would no doubt affect the performance of the RTP transport.

E. Summary

As in Narada, Orta uses a two-step process to construct the spanning trees it uses for multicasting of data, the first step involving the construction of a richer graph between nodes called a mesh, and the second being to create spanning trees rooted at each source in the group via some routing protocol.

Orta utilises link state routing rather than distance vector routing, which offers some very important benefits:

- Members are brought up to date with all state changes much faster.
- Owing to the storage of link state, link removal is more accurate.
- Also owing to the storage of link state, Dijkstra's shortest path algorithm can be used at each peer to calculate spanning trees from each source, and can do so to have the group arrive at a set of distribution trees which all 'agree', due to the flooding process.

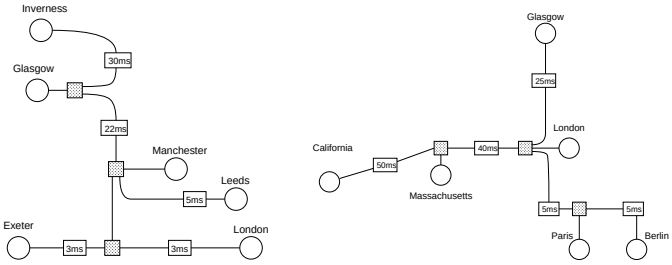
These changes alone should suit Orta to real-time applications such as audio conferencing.

VI. Evaluation

The goals of our evaluation are to demonstrate that the Orta protocol generates robust, well connected, overlay networks and reacts rapidly to changes in membership. As described in section II, these properties are required for effective interactive conferencing applications. In the following, we show that Orta works as specified, providing good performance characteristics for interactive real-time applications. Full details of the performance evaluation are available in [26].

One key aspect of the evaluation is that as this overlay is designed for multi-source multicasting of data between all group members, all experiments dealing with application layer data will send data from each source simultaneously, unless otherwise stated. This differs from much of the other work in the area of network overlays to provide multicast to applications, which generally deal with one source of data.

We begin with a discussion of the testing environment, then demonstrate that the protocol produces appropriate overlays, and evaluate the worst case link stress, adaptability of the protocol, and volume of control traffic generated. For all tests, the threshold for addition or removal of links was calculated as described in Section V-C.3, with the constant value being set to 0.02. The importance of the threshold is discussed further in Section VII.



(a) UK Dummynet. (b) Cross-Atlantic Dummynet.

Fig. 1. Testing environments. Boxes containing numbers represent the one-way delay on links, shaded boxes represent switches, empty circles hosts.

A. Testing Environments

The Orta protocol was tested in the following environments:

1) Varying sized groups of machines on a LAN, with group sizes between 4 and 36 peers. These machines were all 1GHz Pentium IIIs running Linux 2.4.21.
2) A small network of systems linked together by transparent bridges running FreeBSD with Dummynet functionality enabled to impose latencies on links. This network consists six 450MHz Pentium IIIs running Linux 2.6.11, to act as end-hosts on the network, and five machines running FreeBSD 4.11 capable of functioning as transparent bridges, or perhaps act as additional end-hosts.

The first environment is useful for observing how the protocol behaves at different group sizes; the second simulates a more realistic setting where peers are sited at different geographic locations, imposing real latencies on packet transmission.

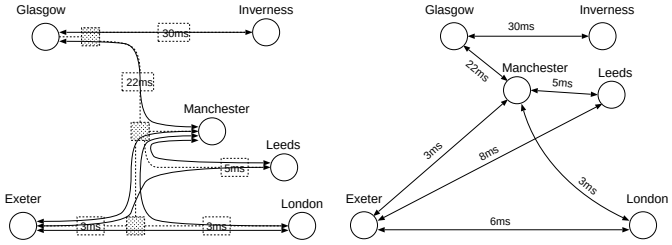
Details of the dummynet networks constructed can be seen in Figures 1(a)[2] and 1(b)[3]. The topologies were chosen to offer real-world latencies given group members at different locations around the world, as might be expected in typical group conferencing applications; place names are simply useful to get a rough idea of the physical geography emulated.

B. Overview of Protocol Behaviour

Figures 2 and 3 show the typical overlay structure and distribution trees for the networks presented in Figures 1(a) and 1(b), using the threshold defined earlier. In each, subfigure (a) shows the mesh superimposed over the physical topology, while subfigure (b) shows the logical network structure.

[2]Measured from Glasgow; Inverness: uhi.ac.uk (60ms); Manchester: www.mbs.ac.uk (45ms); Leeds: www.leeds.ac.uk (45ms); London: scary.cs.ucl.ac.uk (50ms); Exeter: www.ex.ac.uk (50ms). Ping times are approximate to average ping time logged at start of evaluation.

[3]Measured from Glasgow; California: kame.isi.edu (175ms); Massachusetts: mit.edu (110ms); London: scary.cs.ucl.ac.uk (50ms); Paris: www.univ-paris3.fr (50ms); Berlin: ping www.tu-berlin.de (60ms). Ping times are approximate to average ping time logged at start of evaluation.



(a) Resulting mesh structure from the UK Dummynet, shown over the physical network structure.

(b) Resulting mesh from the UK Dummynet, logical structure.

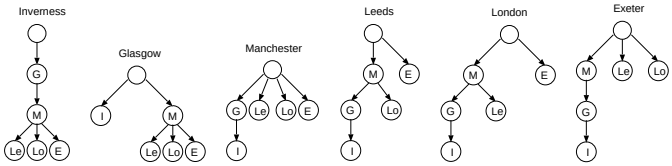
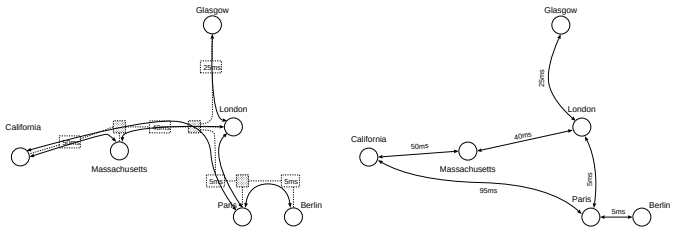


Fig. 2. The resulting mesh over the UK dummynet, and the distribution trees rooted at each source.



(a) Resulting mesh structure from the Cross-Atlantic Dummynet, shown over the physical network structure.

(b) Resulting mesh from the Cross-Atlantic Dummynet, logical structure.

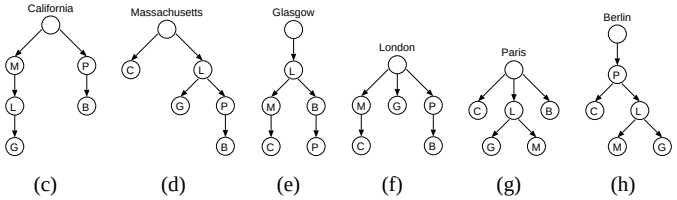


Fig. 3. The resulting mesh over the Cross-Atlantic dummynet, and the distribution trees rooted at each source.

Subfigures (c) through (h) show the distribution trees rooted at each source, identifying recipients by abbreviated location names.

Figure 2 shows a typical mesh structure formed on the UK network. The mesh is well connected with a relatively low link stress, and is generally robust to failures. It exhibits a higher degree of dependence on centrally located and well connected hosts than might be desired (e.g. a failure in Manchester would partition the mesh), but this largely matches the topology of the underlying network.

Orta has created few connections over the longer physical links, instead creating many connections over the shorter links. This is expected, since Algorithm 2 will favour shorter links due to the desire to minimise communication latency. Connectivity to distant peers could be increased by lowering the threshold for adding links. This would make the network more robust to failure, at the expense of additional link stress and (potentially) end-to-end latency.

The mesh structure Orta forms over the Cross-Atlantic network, as seen in Figure 3, has similar properties to the UK network. Again, connectivity of the mesh could be improved by lowering the threshold required for adding links. The worry is that a departing group member can partition the mesh, which should be avoided if at all possible by the Orta peers.

These results demonstrate that Orta generates appropriate topologies for the intended applications. Real-time interactive conferences are expected to be formed from members interested in the topic of conversation, who can be expected to remain in the group over its lifetime. Unlike other classes of peer-to-peer application, group membership dynamics are not expected to be relatively limited, and the key point is to deal with network failures and topology changes. Accordingly, we do not view the dependence on some well connected central peers to be a problem.

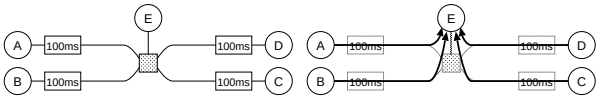
C. Worst Case Stress

The very nature of the overlay means that individual hosts are sending and receiving more data packets than the application layer is aware of. The number of packets duplicated is determined by how many links the protocol creates to or from each host for a given network setup. This duplication raises two concerns:

1) Additional bandwidth usage over links to end-hosts. While the discussion of the protocol has not considered bandwidth, it would require serious consideration if connection types varied considerably. Clearly, many connections to be carrying data over a modem link is not as desirable as the same number of connections through an ADSL link, or an Ethernet link.
2) The increase in packets to be processed at each host leads to increased processing costs in the overlay code executing at the application layer, and puts additional pressure on the networking subsystem of the host operating system.

The stress of a link is simply defined as the number of identical copies of a packet carried by a physical link to deliver that packet to the rest of the peer group. Worst Case Stress is then the maximum stress value observed on any physical link in the group. In comparison, all links in a properly configured IP multicast group have a stress of 1, while a naïve conferencing overlay which created a connection between every pair of members over which the members copy packets directly to all recipients would have a physical link stress of n on access links (n = number of recipients). Orta should attempt to minimise the stress of links throughout the group by spreading the duplication of packets throughout the distribution trees created from each source in the group.

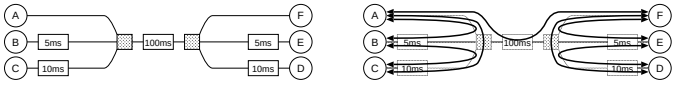
A goal of Orta is to keep link stress low if possible, spreading the packet replication load across the group. This



(a) Physical topology of the network.

(b) Resulting mesh structure over physical topology.

Fig. 4. Test network, designed to attract connections to the centre peer, shown with resulting mesh structure.



(a) Physical topology of the network.

(b) Resulting mesh structure over physical topology.

Fig. 5. Test network, designed to attract connections to the centre peer, shown with resulting mesh structure.

is countered by the desire for a well connected mesh that is robust to failures and prefers low latency links, both factors that may lead to an increase in the worse case link stress.

The physical topologies designed specifically for testing the worst case stress of a link are shown in Figures 4(a) and 5(a), and were designed to observe how Orta behaves in smaller, more artificial, environments. The resulting mesh structures are shown in Figures 4(b) and 5(b).

The quality of the overlay in Figure 4(b) suffers by not creating enough connections between peers. This is, in part, due to the highly artificial nature of the overlay; if node E was behind an access link with an additional latency of a few milliseconds, the peers on the periphery of the network would be more likely to create connections between each other, since routing data through E would be more expensive. The distribution trees for this network all route through E, and E duplicates packets to all other group members. The worst case stress for the network is 4.

In Figure 5(b), however, Orta has created many more links between peers. Peers A and F bridge the two halves of the group because they are the closest together of the two halves. The worst case stress in this example is 3, and occurs on the links from A to its connecting switch, and from F to its connecting switch. The stress of the high latency link is 1. Orta, having partitioned this group into the two distant groups with a connection in the middle, has avoided the potential worst case stress of 5, in this case.

These network are highly artificial, but demonstrate that Orta is behaving in an appropriate manner, producing an overlay with acceptable worst case stress whilst avoiding links that would excessively induce latency.

Considering the UK network in Figure 2 for less artificial results, we can see that the worst case stress on any physical link is 4 at Manchester. A link that appears heavily loaded is that from Exeter, however despite carrying three TCP connections, this physical link is not used as much as it may seem. While the highest stress level it sees is 3, this stress level is only actually met by considering the distribution tree rooted at itself; by considering the other distribution trees, Exeter's stress level is only 1.

The Cross-Atlantic network in Figure 3 exhibits a worst case stress of 3 at both London and Paris, but in this scenario, those stress levels are met for numerous distribution trees. The worst case stress is clearly affected by the number of links that the protocol creates for the peer group.

In summary, we believe the worst case stress results for Orta are acceptable. It is clear that overlay topologies could be generated that would cause less link stress, but these come at the expense of additional latency and are not appropriate for the application.

D. Adaptability of Mesh to Changing Network Conditions

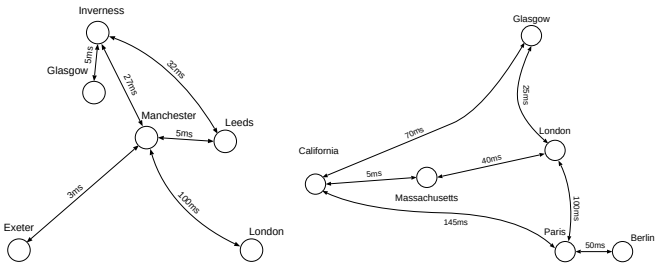
Since the intention of the mesh approach to building distribution trees for carrying data is that the quality of the mesh gradually improves over time, it's worthwhile observing how Orta behaves when the network conditions change during the lifetime of a peer group.

The following alterations to the dummynets were chosen to observe the behaviour of the protocol under different conditions during the operation of the overlay; some 'long' links have been shortened, and some 'short' links have been lengthened.

- On the UK dummynet network, the latency on the bridge between Inverness and Glasgow was reduced from 30ms to 5ms, thus removing that long link to try and prompt more links to be formed up to Inverness. Further, the latency on the bridge separating London and the switch to which is is attached was increased from 3ms to 100ms, providing a longer link which Orta should try to avoid.
- On the Cross-Atlantic dummynet network, the latency on the bridge between California and Massachusetts was reduced from 50ms to 5ms; the latency on the bridge separating Paris and Berlin from the rest of the network was increased from 5ms to 100ms; finally, the latency on the final bridge before Berlin was increased from 5ms to 50ms.

The behaviour of Orta with these changing conditions over the UK Dummynet can be seen in Figures 2(b) and 6(a). The reduction of latency to Inverness has allowed more links to be added, though the removal of links to Glasgow is surprising. The assumption is, again, that without additional weights on access links, the difference in distance from Manchester to Inverness (via Glasgow or not), for example, is negligible. Orta has reduced the number of links to the London host, having had the latency to it increased considerably. On reconfiguration, Orta is still placing too few links to some nodes.

The variation observed in the Cross-Atlantic dummynet, Figures 3(b) and 6(b), is minimal. Reducing the latency to California has allowed Orta to create one further link in the mesh, connecting California to Glasgow. No addition of links has occurred elsewhere. For the London/Paris/Berlin peers, this is understandble, due to the similar increase in the latencies observed on those links; by reducing the latency to California, it should be expected that a new link be formed to that peer.



(a) Logical topology of mesh after link weights were changed.

(b) Logical topology of mesh after link weights were changed.

Fig. 6. Variability of mesh under changing conditions on the UK Dummynet.

Observation of the round trip time (RTT) between peers over the lifetime of overlays on both networks, and in particular while the overlays are reorganising, show that delays between pairs of hosts peak directly after the latencies on links have been altered, after which the overlay adds and removes links. The overlays manages to reduce the modified RTT between any pair of hosts (as described in Section II) to beneath 400ms within 30 seconds, without disrupting data flows.

The behaviour of the mesh in these situations shows that Orta can provide reasonable overlays for use in carrying real-time data, and also that the mesh is capable of reacting to changes in the network topology quickly enough to avoid unacceptable delays for extended periods of time between peers when routing through the overlay.

E. Time Taken to Repair a Partitioned Mesh

While it is possible to have the mesh disrupted by a member leaving, the trickiest situation to deal with is that of a member failure. Group members must determine that this member has failed as per Algorithm 1, and attempt to clean up state.

In order that the overlay could be partitioned with ease, the code responsible for attempting to add links during the normal operation of the overlay was disabled, and timer variables chosen to demonstrate how the mesh repartitions itself. We assume that the mesh is partitioned on a peer dying unexpectedly, leaving all physical links in place.

To demonstrate the repairing of an overlay partition, a chain of seven peers was constructed, as per Figure 7(a). Group member D was stopped abruptly (‘kill -9’), the other peers left to repair the partition created as part of their normal running cycle.

In this test implementation, refresh packets are sent every 30 seconds, and sequence numbers checked at peers every 15 seconds. With these values, it took 43 seconds before the first connection was formed from one side of the partition to the other.

With new links in place, flooding of any control information could reach all group members, thus allowing normal protocol mechanisms to bring peers up to date. The implementation set the upper bound for silence from a peer to an arbitrary 70 seconds; after 72 seconds member D is declared to have failed, and removed from the local state of all peers.

(a) The initial logical network topology.

(b) The logical shape of the network after repairing the partition.

Fig. 7. Logical view of a peer group both before and after a partition.

In the time that the group is partitioned, state changes on one side of the partition are not observed on the other; in all, it took a further 118 seconds for the link-state stored at all peers to match again. The delay for this part of the process is borne out of peers waiting for refresh packets to arrive in order to update the link-state, and as defined in Section V-B, peers are not required to send information on all links at each refresh cycle.

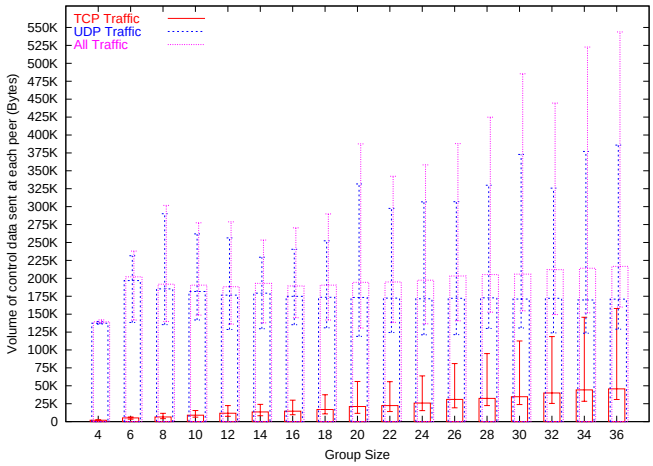
These delays are affected greatly by the implementation of the protocol; a more frequent refresh cycle would be used in an interactive conferencing environment. More frequent refresh packets or checking of stored sequence numbers from other peers would allow for a faster detection of silent peers; a reduction on the upper bound used to determine when a peer has failed would have seen group member D declared as having failed earlier than it was. These timer variables would naturally be reduced for real-world use, and serve as an example of where the real delay exists in not only partitioning a mesh, but also repairing link-state at all peers.

It is also worth noting that under normal operation a partition formed in the mesh, perhaps from a member failing, may be repaired before the code dedicated to fixing a partition is initiated. The mechanism to randomly ping peers for the purposes of evaluating new links is likely to ping a member on the other side of the partition. It is clear, however, that the algorithm provided by Narada for purpose of partition repair is not ideal for real-time applications.

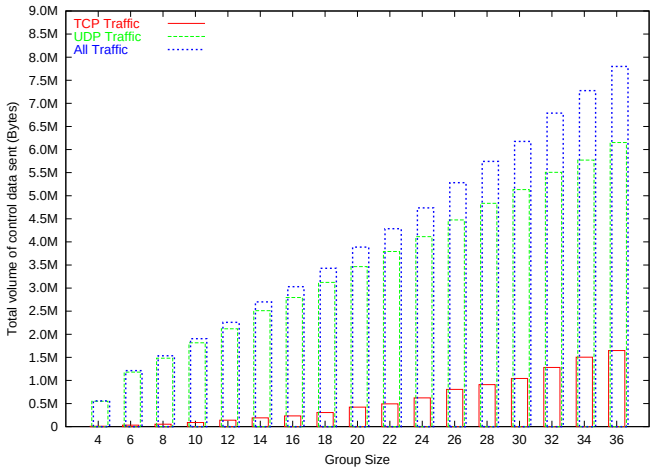
F. Volumes of Control Traffic Sent

This is an interesting metric due to the very different way that control traffic is handled in the Orta protocol compared to the Narada protocol. Values presented are from running each group size for 5 minutes.

In the implementation of Orta, most control traffic is carried over TCP connections; the only exception to this being the pinging mechanism, which uses UDP as a carrier. The average volume of control traffic sent by each group member over TCP, as shown in Figure 8(a), appears to rise linearly. This makes sense, in that provided link state remains reasonably constant, then most TCP traffic being sent is the regular refresh packets from each peer. The average volume of UDP traffic sent over the lifetime of the group appears to remain constant at different group sizes, presumably due to the number of neighbours each peer has in the mesh does not vary significantly. As group sizes increase, so too does variation in the maximum volume of data sent on each type of connection, suggesting that even though the average number of neighbours a peer has in a group remains constant, larger groups do see some peers with larger neighbour sets than others.



(a) Average volume of control traffic sent from each host, including minimum/maximum variation.



(b) Total volume of control traffic sent throughout the entire group during a session at each group size.

Fig. 8. Variation in volumes of control traffic sent during the lifetime of the mesh at various group sizes.

Plotting the total control traffic sent over the whole peer group in Figure 8(b), we see that the total volume of control data required to maintain the group rises linearly with group size. This is a useful property in that the limiting factor for larger group sizes is not the volume of control data, but may actually the amount of computation involved at each peer. Further testing would be required to determine if this was the case.

G. Summary

Orta provides many desirable properties for real-time applications. The volume of control traffic sent during the lifetime of a peer group is reasonably predictable, based on group size. Round trip times achieved on the test networks are within the limits required for conversational audio to take place, and the protocol proves to be capable of adapting to changing network conditions. Orta achieves worst case stress of substantially less than that of a naïve unicast application. The mechanism used to fix mesh partition is not ideal for the target application area, but there are potential improvements which can be made here, covered in brief in Section VII.

While more testing over a larger variety of networks would be required to boost confidence in the new protocol, these results are promising.

VII. Future Directions

Performance of the mechanism to deal with a partition in the mesh structure is highly dependent on the choice of parameter values in certain scenarios, and does not make full use of the available information in others. There are two situations where a partition would have to be repaired: on a member exiting, and failure of either physical links or of a group member.

For the former case, it is reasonable for peers processing leave notifications to spot members who are no longer reachable, and to preemptively attempt to add links to those members until all members can be reached once again. Our current implementation does not do this, and must therefore wait for the random probing to repair the mesh, causing additional latency when a departing peer partitions the mesh.

Considering failures of links or group members, a peer could monitor ping packets not yet returned from neighbours and having a peer assign some large link weight (less than infinity) to a neighbour which has not yet returned a packet. This would allow for the mesh to route around a failure via normal means before it might even be declared as having failed by Algorithm 1. With a well-formed mesh, the routing changes would happen swiftly; if the peer formed the only route between two halves of the mesh, normal link discovery mechanisms would quickly allow for new links to route over. In either eventuality, it appears that it should be possible to repair the mesh in a matter of seconds, rather than minutes.

Neither issue is significant in the usual case: the mesh is generally well connected, and will not usually be partitioned by a departing member, and failures of members or links are assumed to be rare. The changes suggested above are useful for robustness in rare cases, not in the general case. A future implementation of Orta will incorporate these alterations to cater for rare cases.

Another issue is choice of the threshold used to determine whether or not a link should be included in the mesh. Threshold value calculation currently has to be almost hardcoded to the type of network that the overlay will be running across. This is certainly not ideal, and is an area that might require considerable effort to derive a proper solution.

Further difficulty in choosing the threshold value is that it causes the overlay to behave differently in different types of environments. It might be beneficial to have different thresholds ‘hardwired’ for different environments, so a threshold might be more appropriate for conferencing between home users over ADSL links, while another might be better for groups of users on the same LAN. Adaptation of the threshold calculation during the lifetime of the overlay could be achievable, perhaps based on group size, number of sources, variability of link types, latencies between group members, to

name a few. Evolutionary algorithms could perhaps be of use here, to allow the system itself to decide what threshold to use based on results from past attempts at threshold values. This approach would require substantial testing before real-world usage.

Other potential enhancements might be analysis of the benefit of mixing multiple audio streams to minimise the amount of duplicated traffic sent through the overlay, and analysis of the suitability of clustering techniques [24] to enable larger groups. These are outside the scope of the present protocol, but would form the basis of interesting extensions.

VIII. Conclusions

By altering the mechanism the Narada protocol used for the distribution of control state to group members, Orta allows forms a more responsive peer-to-peer overlay, geared toward the transport of real-time audio between many recipients. This change brings with it slightly increased computational loads at all peers, but allows a more accurate mechanism to be developed for the purpose of dropping links, and removes the requirement on peers to continue to forward data for some time after leaving the group. In addition, the immediate advertisement of a new member to the group allows the new member to participate with the group immediately.

As a proof of concept we have also integrated Orta with an existing multicast audio conferencing application [27], demonstrating that it can replace IP multicast in real-world conferencing applications. An implementation of the Orta protocol is available from `http://orta.sf.net/` under an open source license.

Orta is a new protocol, derived from the Narada protocol, appropriate for the carrying of real-time data to small or medium-sized conference groups in the order of 10s of members. The protocol offers distribution trees optimised for each source, and will reconfigure in light of changing network conditions, offering ideal conditions for the carrying of real-time data in the absence of IP multicast.

Acknowledgment

The authors would like to thank Ladan Gharai and Tom Lehman at ISI for providing access to a US-based host for testing purposes.

References

[1] S. E. Deering, "Multicast routing in internetworks and extended lans," in *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*. ACM Press, 1988, pp. 55–64.

[2] J. Lennox and H. Schulzrinne, "A protocol for reliable decentralized conferencing," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. ACM Press, 2003, pp. 72–81.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2002, pp. 205–217.

[4] S. Birrer, D. Lu, F. E. Bustamante, Y. Qiao, and P. A. Dinda, "Fatnemo: Building a resilient multi-source multicast fat-tree," in *WCW*, ser. Lecture Notes in Computer Science, vol. 3293. Springer, 2004, pp. 182–196.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment," in *IPTPS'03*, February 2003. [Online]. Available: citeseer.ist.psu.edu/castro03splitstream.html

[6] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, 2002, to appear. [Online]. Available: citeseer.ist.psu.edu/castro02scribe.html

[7] Y. Chawathe, "Scattercast: an adaptable broadcast distribution framework," *Multimedia Syst.*, vol. 9, no. 1, pp. 104–118, 2003.

[8] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12. [Online]. Available: citeseer.ist.psu.edu/chu01case.html

[9] P. Francis, "Yoid: Extending the internet multicast architecture," April 2000, http://www.isi.edu/div7/yoid/.

[10] M. Hefeeda, A. Habib, B. Boyan, D. Xu, and B. Bhargava, "PROMISE: peer-to-peer media streaming using collectcast," Tech. Rep., August 2003, cS-TR 03-016, Purdue University. Extended version. [Online]. Available: citeseer.ist.psu.edu/article/hefeeda03promise.html

[11] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev, "Collectcast: A peer-to-peer service for media streaming." [Online]. Available: citeseer.ist.psu.edu/707957.html

[12] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable multicasting with an overlay network," pp. 197–212. [Online]. Available: citeseer.ist.psu.edu/jannotti00overcast.html

[13] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "GnuStream: A P2P Media Streaming System Prototype," in *Proceedings of the International Conference on Multimedia and Expo (ICME)*, vol. 2, July 2003, pp. 325–328. [Online]. Available: citeseer.ist.psu.edu/xu03gnustream.html

[14] J. Liebeherr and T. K. Beam, "Hypercast: A protocol for maintaining multicast group members in a logical hypercube topology," in *Proceedings of 1st International Workshop on Networked Group Communication (NGC '99)*, July 1999, pp. 72–89. [Online]. Available: citeseer.ist.psu.edu/liebeherr99hypercast.html

[15] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS 2001)*, March 2001, pp. 49–60. [Online]. Available: citeseer.ist.psu.edu/pendarakis00almi.html

[16] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Proceedings of the Third International COST264 Workshop on Networked Group Communication*. Springer-Verlag, 2001, pp. 14–29.

[17] D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," 2003. [Online]. Available: citeseer.ist.psu.edu/tran03zigzag.html

[18] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. ACM Press, June 2001, pp. 11–20.

[19] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," September 2004.

[20] International Telecommunications Union, Recommendation G.114, "One-way transmission time," February 1996.

[21] W. Wang, D. A. Helder, S. Jamin, and L. Zhang, "Overlay optimizations for end-host multicast," in *Proceedings of the International Workshop on Networked Group Communication (NGC)*, October 2002.

[22] Y. Chu, A. Ganjam, T. S. E. Ng, and S. G. Rao, "Early experience with an internet broadcast system based on overlay multicast," 2004. [Online]. Available: citeseer.ist.psu.edu/689003.html

[23] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *ACM SIGCOMM 2001*. San Diago, CA: ACM, Aug. [Online]. Available: citeseer.ist.psu.edu/chu01enabling.html

[24] S. Jain, R. Mahajain, D. Wetherall, and G. Borriello, "Scalable self-organizing overlays," UW-CSE, Tech. Rep. 02-02-02, Feb 2002.

[25] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," http://www.ietf.org/rfc/rfc1075.txt.

[26] S. Strowes, "Peer-to-Peer Audio Conferencing." Master's thesis, University of Glasgow, 2005.

[27] "Robust-Audio Tool homepage," http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/.