**RC23290 (W0408-028) August 6, 2004**
**Computer Science**

# IBM Research Report

## Stateless Application-Level Multicast for Dynamic Group Comunication

**George Popescu, Zhen Liu**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Stateless application-level multicast for dynamic group communication

George Popescu and Zhen Liu
*IBM T.J. Watson Research Center*
*{popescu, zhenl}@ us.ibm.com*

## Abstract

*Group communication in large-scale interactive applications such as real-time conferencing and collaboration, networked virtual environments (e.g. massively multi-player games) requires efficient, low-overhead group communication mechanisms. In this paper we consider the application-level (or end-system) multicast and propose a stateless group communication mechanism together with its tree building algorithms. Stateless multicast reduces the control signaling of dynamic multicast groups linearly with the group size. To support interactive applications involving a large number of dynamic multicast groups, the application level multicast uses stateless forwarding within clusters of network nodes and hierarchical aggregation of multicast group membership. We show that dynamic tree construction achieve low computation overhead with a controlled degradation of the end-to-end data path performance.*

## 1. Introduction

Group communication services have been successfully implemented in virtual private networks and local area networks for several decades. However lack of IP-supported multicast mechanisms and quality of service (QoS) support has prevented large-scale group communication applications from being deployed on the Internet. Recent work addresses these limitations by using application-level multicast on network overlays [5] composed of end-systems and possibly forwarding proxies. A review of recent work on group communication services using application-level multicast is presented in [7].

Among network group communication applications, distributed interactive applications (DIA), such as network virtual environments (massive multiplayer online games) [9], real-time conferencing, are the most sensitive to the quality of service provided by the network infrastructure [10]. Interactivity - a measure of application quality - is dictated by the end-to-end latency between participants [13].

In addition, control overhead due to the dynamics of group membership is an important design consideration in large DIA's. Indeed, the participants of a DIA act as senders and receivers in several multicast groups [11]. The tradeoff between efficiency of (application-level) multicast schemes and the control of these dynamic groups has big impact in the overall group communication performance. One common approach to cope with this issue is to group participants according to the communication needs (which will also be referred to as participant's communication interest in this paper) and to construct distribution trees for each multicast group. Scaling to a large number of dynamic groups dispersed in a large network overlay requires multicast solutions with low control overhead.

The proto-typical network architecture of a large-scale distributed interactive system consists of distributed servers that control data forwarding to dynamic groups of end-system (clients) [9]. These are in fact network overlays where multicast groups use simple star topology while group management is distributed among a fixed set of control nodes. Such architectures have limited scalability and restrict group membership dynamics. Data distribution for interactive applications has multiple (conflicting) goals: dynamic grouping of participants according to their communication interest, efficient data path construction to guarantee end-to-end network latency and reduction of signaling overhead generated by changes in participant's group membership. The multicast-grouping problem [4] [15] addresses the clustering of overlay participants according to the commonality of their communication interest to reduce the amount of unwanted data (wasted communication bandwidth) received at each participant.

Application-level multicast solutions proposed in [1][5][8][12] are relevant to data path construction in distributed interactive applications. Application level multicast algorithms presented in [5] are designed for static group membership, where the overhead of control signaling is small. Distributed data path construction

using Delaunay triangulation [8] and delay based clustering [12] increases network overlay scalability, but do not optimize data path quality (end-to-end delay). Real-time issues in the design of overlay networks for large-scale distributed systems are addressed in [1]. However the network overlay construction is optimized for streaming applications and do not consider the large signaling overhead specific to distributed interactive applications involving a large number of multicast groups.

We propose a stateless group communication mechanism along with algorithms for efficient data path control in large-scale distributed interactive applications. Participants contribute resources (CPU, bandwidth) to form an overlay network. The overlay is composed of end-systems (peer nodes) and proxies. We shall designate some of these as control nodes that perform the control functions of the group membership. Many of them are proxies practically speaking. Network overlay clustering can achieve scalability at the expense of a reduced multicast routing efficiency. Network nodes are clustered based on network proximity (round trip time) using distributed algorithms [8]. Each cluster is assigned a controller – an overlay node with higher communication capacity. Controllers monitor the network state information of network nodes whithin the cluster. The controllers may be organized in a hierarchy [6] or using distributed hashing [14].

Multicast groups are constructed by grouping participants according to their communication interest. Efficient application level multicast tree construction uses network information (communication capacity, RTT between network nodes) monitored by the overlay controllers. Such information is easily available in a distributed interactive application which uses timestamps for each transmitted network packet [11]. Frequent interest changes of participating nodes require regrouping and rebuilding of multicast communication trees dynamically. Thus, state-based group communication will result in large control overhead in distributed interactive applications. This motivates our use of a stateless multicast protocol, in which data is forwarded to next hop nodes encoded in an application level multicast header. The reduction in control signaling is obtained at the expense of increased header processing at overlay peer nodes. The stateless protocol uses efficient multicast tree encoding to reduce the overall bandwidth and CPU utilization at forwarding peer nodes. In addition, resilience to node failures can be achieved using on-the-fly tree repair at forwarding nodes in the multicast tree. Each control node indexes a subset of multicast groups and controls only the peer nodes in its network proximity. The interest registration is symmetric in the sense that senders and receivers need only to register their communication interest to a parent node in the hierarchy. The control nodes aggregate the communication interest of network cluster nodes to reduce the amount of control signaling for dynamic group membership changes.

The paper is organized as follows. In Section 2 we present the mechanism and various implementations of stateless peer forwarding multicast. In Section 3, we propose a hierarchical approach for group membership management and the corresponding multicast tree construction algorithms. The evaluation of data distribution using stateless multicast is presented in Section 4. In Section 5 we discuss related work in the literature. Finally, in Section 6 we provide concluding remarks.

## 2. Stateless peer forwarding multicast

The stateless peer-forwarding multicast we propose here uses an application level header containing an encoding of IP addresses of all multicast group members. The source constructs the multicast tree and transmits the encoded multicast header with each data packet. Peer nodes process (decode) the header to determine the next hops and construct new headers for each data packet. The encoding of the multicast tree requires little additional overhead (compared to simply listing node addresses) but gives forwarding nodes control of the data path. Forwarding peer nodes can modify part of the multicast tree to achieve resilience to node failure and to adapt to forwarding capacity fluctuations of peer nodes.

Two header encoding/processing methods are described in the next section. The first method constructs the multicast header by sequentially entering the addresses of nodes visited during per level traversal of the multicast tree. The second method uses preorder traversal and reduction of the multicast header at each forwarding node.

## 2.1 Multicast header encoding/decoding

**2.1.1 Per level header encoding.** The per-level encoding method constructs the multicast headers by concatenating the sequence of node out-degree (number of children) with the sequence of receiver addresses, both obtained during a per level traversal of the multicast tree. The first entry of the encoded header is the position in the tree of the current node, counted from the root during per level tree traversal. The node out-degree sequence for the tree shown in Figure 1 is: 1,2,1,2,2,1,3 (the sender forward to n1 which is the root of the multicast tree).

The forwarding peer node find the next hop addresses by reading the number of next hop nodes from the fanout

sequence and locating the position of the first child of the current node in the encoded header. The number of next hop nodes is indicated by the current node entry in the fanout sequence. The position of the first next hop node is obtained by summing the fanout of all nodes prior to current node entry in the fanout sequence. At node n[k] (k indicate node position in a per level tree traversal) the multicast header is <k><1,2,1,2,2,1,3,…> <address[n1], ...>. Let s[k] be the k-th entry in the fanout sequence of the tree encoding. The addresses of the next hop nodes are the s[k+1]successive IP addresses in the address sequence, starting from the $\sum_{i=1}^{k} s(i)$ entry. The header construction algorithm using per-level header encoding is shown Table I.

New multicast headers are computed for each next hop of the current forwarding node. The header size decreases at each forwarding node in the multicast tree, reducing the overhead of the application level forwarding. The first entry of each new header is the position of the next hop node in the address list. The rest of the header is copied from the current node starting from the address of the first child node. Next the new header insert the addresses of all nodes following the current one in the per level tree traversal sequence.

**Table I: Per level header encoding/decoding**

```
1) Start at the root node;
root.levet=1; seqeunce_header=null;
addr_header=null; current_node = root;
2) enqueue the current_node
3) While (queue not empty)
{dequeue current_node
        Insert (current_node.IP_addr) in addr_header;
        Insert(current_node.fanout) in
        sequence_header;
        count=0;
   while (count < current_node.fanout)
        {enqueue  current_node.child[count];
        count++;}
}
4) concatenate (root.level, sequence_header,
addr_header)
```

a) per-level header encoding algorithm

```
1)   k=header[0]; nb_nodes=header[k+1]; m=0;
  for(j=0;j<nb_nodes; j++) new_header[j]=null;

2) while(header[k] != separator)
{ m+= header[k+1]; k++;}
 i=0;  pos = k+addr_size;
3) while (i<nb_nodes)
{for(p=0;p<addr_size;p++)
new_header[i].addr[p]
=header[pos+addr_size*i+p]
new_header[i].header[0]=m+i; i++;
Copy the encoding sequence from position
corresponding to current node+1 to
new_header[i].header
Copy from the current group to end of the header to
new_header[i].header
}
```

b) header processing algorithm

**2.1.2 Header encoding using preorder tree traversal.** This method constructs the multicast header by encoding the pre-order traversal of the multicast tree. The header contains the address of the node preceded by a code indicating its tree-level (the root is at level 0). Tree-level codes followed by node addresses are entered in the multicast header during the recursive traversal of the tree.

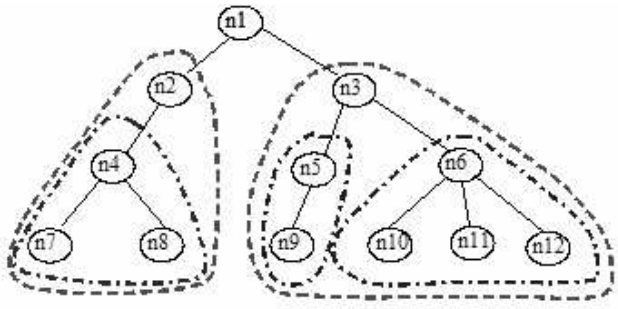


Figure 1: Tree encoding/ processing: the sub-trees of the pre-order method, marked in red, are contained in the header transmitted at n2 and n3 and in green at n4, n5 and n6

The header is processed at each forwarding node such that only the nodes in the sub-tree rooted at the current node are kept, with the tree-level code adjusted correspondingly (see Figure 1). The next hop nodes of the current peer are identified by the value of their tree-level code. Finding the next hops requires a complete search of the multicast header. The sub-trees corresponding to the next hops are the sequence of nodes between two occurrences of tree-level code corresponding to the next hop nodes. The length of the header and the amount of processing per hop decreases

as the packet reaches nodes closer to tree leaves. The header encoding and processing algorithms are shown in Table II.

**Table II: Pre-order encoding/decoding**

```
Start at root node:  node = root; root.label=0;
Traverse_node(node)
{Visit_node(node);
If (no children) return;
while(node has children)
{child.label=node.label+1;
traverse_node(child)}
}
Visit_node (node)
{ insert node. label;
insert node.addr;}
```

a) Pre-order tree encoding

```
1.  k=0; p=0;
2. while (Node[p].label != terminator)
    { if (Node[p].label = = 1)
        {next_hop[k] = Node[p].addr;
        Insert Node[p] into header[k];
        k++;}
        while(Node[p].label != 1)
            { Node[p].label--;
            Insert Node[p] into header[k];
             p++;}
    }
```

b) header processing

## 2.2 Analysis of header processing algorithms

The header encoding/decoding methods presented in the previous section offer different trade-offs between the header overhead (the node forwarding capacity required for transmission of application level header) and the computation effort required at forwarding nodes. The encoding overhead is in both cases less than 25% of the space required for multicast node addresses. The overhead is smaller for per-level traversal (~N/2 where N is the number of peer nodes in the group) than for the preorder method (~N). The preorder traversal achieves the smallest total overhead by transmitting only the sub-tree rooted at the each forwarding node.

The analysis of stateless multicast for an overlay with N nodes, each with fanout m gives ($l = [\log_m N]$):

$$\sum_{k=1}^{l-1} 5m^k m^{l-k} = 5(l-1)m^l < Overhead(preorder) < \sum_{k=1}^{l-1} 5m^k (m^{l-k+1} - 1) = 5((l-1)m^{l+1} - (m^l - m)/(m-1)) \quad (1)$$

for the total overhead of the preorder method and:

$$5m^{2l}/(m-1) < Overhead(per-level) < 5(m^{2l+1})/(m-1) \quad (2)$$

for the per level traversal.

The total overhead was defined as the sum of multicast headers length at all forwarding peer nodes in the tree. The ratio between the total overhead of preorder and per-level traversal method varies as $\approx (l-1)/m^{l-1}$ ($\approx m([\log_m N]-1)/N$) with the number of nodes in the multicast tree - overhead savings for the preorder grows exponential with tree depth. The average overhead reduction per forwarding node between preorder traversal and per level traversal is proportional to the tree depth.

The computation effort at forwarding nodes is composed of header search and next hops header creation. On a given application level multicast tree, the difference in computation effort between two header processing methods is dominated by the search for the next hop addresses. The pre-order method scans the entire application level header at each node in the distribution tree, while the per-level traversal stops at the next level after the current forwarding node. The difference in computation effort: $m^{l-k-1}$ (where l is the depth of the entire tree and k is the depth of the current node), decreases exponentially with the depth of the node in the tree. Per-level method requires less computation effort at nodes closer to the source. Since pre-order reduces the overhead exponentially with the depth of the tree, it is preferred for large multicast groups composed of nodes with small fanout. For small groups with high fanout nodes the per-level method results in a smaller header computation load. This encoding is more advantageous when nodes act as forwarding proxies for several small multicast groups.

## 2.3 Signaling overhead reduction for stateless multicast

We estimated the overhead of leaving and joining a group of N nodes when using a stateless group communication protocol vs. a statefull multicast method (where the state of the multicast group is kept at each node in the overlay). The overhead of leave/join for the stateless group communication is assumed constant and is used as a reference. The leave overhead of a statefull method depends on the tree construction method; the least expensive – promoting a leaf node in place of the leaving node requires three times the signaling load of the stateless multicast. Another method, which produces better trees with a small amount of signaling, promotes the children on the longest branch from the leaving node.

The worst case for the signaling – a complete tree – requires moving up all longest branch nodes from the current leaving node to the leaf level. The signaling cost for a node at level k in the tree is $l-k+1$ where l is the depth of the tree; in average the leave requires $1+O(1+\varepsilon(m,l))$ assuming that a node leaves with the same probability from any position in the tree. The join overhead however is much larger since it may require a complete re-building of the tree. At worst the insertion of the node (for the algorithms that uses a sorted list of nodes) at parent k will require signaling all the remaining nodes starting from the k+1. Assuming the node can be inserted with the same probability in any position in the tree, the average signaling overhead is linear with the number of nodes in the tree. The join and leave overhead above was computed for a single multicast group; the overhead grows linearly with the number of multicast groups.

## 3. Dynamic group membership control

Distributed interactive applications are characterized by high dynamics of client group membership, which translates into high control overhead of multicast groups. Stateless forwarding multicast presented in the previous section limits the size of the multicast group to tens of members in order to reduce the impact of the application header overhead. In addition, the efficient construction of distribution trees requires that control information (client's communication interest, forwarding capacity and network delay information) is available at the tree-building node.

We propose in this section group membership management and multicast tree-building algorithms for data distribution in large-scale distributed interactive applications. Multicast groups are managed by a hierarchy of overlay control nodes, which also control data path quality. The network overlay is clustered in cardinality bounded network clusters managed by cluster leader/control nodes, which constructs local multicast trees and perform multicast header encoding. The cluster leader nodes keep control information (multicast group membership) and per node network data (node forwarding capacity, delay measurement) for all end-systems in a cluster. Per node network information is registered at cluster leader/control nodes in the bootstrap phase, when nodes join the overlay. We present in the following the hierarchical communication interest aggregation and the tree construction methods for group communication using stateless multicast.

The stateles forwarding multicast is particularly suited for distributed interactive applications. A proto-typical distributed interactive application has an average multicast group size of tens of participants [9]; for groups smaller than 50 participants, the overhead of stateless forwarding is less than 20%. In addition, the size of the multicast groups can be controlled by the application; larger size multicast groups can be controlled by clustering participants with in cardinality bounded groups.

Node communication interest is signaled to the control nodes using group join/leave messages. Control nodes aggregate the group membership information by substituting local cluster group membership lists with their own network identifier (e.g. IP address). The multicast group information indexed at overlay control nodes is dynamic: control nodes insert (when receiving joins on new topics) and remove (when there is no receiver left in the group) group states. The leave messages are propagated in the control hierarchy only when the node is the last, among the set of nodes indexed at the control node, leaving the group; similarly, the join messages are propagated in the control hierarchy only when the control node does not have other receivers for the group indicated in the join message.

Each controller constructs application level multicast trees for a segment of the data path that contain its siblings and peer (end-system) child nodes. The end-to-end path is composed of multiple segments from several application level multicast trees. Controllers keep the state information needed for construction of application level multicast trees, while the rest of overlay nodes (end-system, forwarding proxies) act as simple forwarders. Each forwarding node has a message processor, which implements the forwarding and tree processing algorithms presented in section two.

### 3.1 Tree construction algorithms for stateless multicast

In the previous section we argued that group membership dynamics requires distributed control of the data path; each control node builds per group sub-trees containing only the subset of nodes in a network cluster. The overlay network is first partitioned by clustering nodes based on network distance such that available forwarding capacity is optimally distributed among the clusters. Data path between multicast group members is delay-bounded within each cluster using efficient tree construction algorithms; cluster leader (control) nodes perform data forwarding on source-based trees constructed with sibling control nodes only. End-to-end data path between multicast group participants is therefore concatenated from several delay bounded overlay segments. With this restriction, the end-to-end

delay constraints require bounding the maximum delay of each sub-tree rooted at control peer nodes.

We propose in the following algorithms for construction of overlay multicast trees composed of cluster leader/control nodes with high forwarding capacity and peer nodes (end-systems) with small forwarding capacity. Network information necessary for tree building is available at cluster leader nodes. Tree construction uses a weighted distance metric that combines the node distance from the source and node fanout. The distance is normalized to the maximum distance from the source while the fanout is normalized to the maximum fanout among the receiver nodes. The distance from the source to the current node C is:

$$d(C,S) = w1 * \Delta(C,S) / \max(\Delta(C,S)) + w2 * (F - f(C)) / F) \quad (3)$$

where $\Delta(C,S)$ is the delay between the current node and the source and F is the maximum fanout among the nodes in the multicast group.

We consider two cases: network distance relative to cluster leader nodes and forwarding capacity of the peer nodes is known; the network delays between any pair of nodes (or an estimate of network delay) within the cluster and node forwarding capacity is available at cluster leader/control nodes.

In the first method of constructing the multicast trees - closest node - peer nodes forward to the "closest" node (computed using a combined delay/capacity metric) in the same multicast group. The algorithm first orders the nodes according to the weighted distance metric and subsequently add nodes to the last inserted parent in the order of decreasing distance; the weighted distance metric favor nodes with high capacity over closer nodes (using delay only) with low capacity.

The second and third algorithms use network delay information is available using either direct measurement – $O(n^2)$ with number of nodes in the network cluster - or through positioning network nodes in a high-dimensional network plane – O(n) with the number of cluster nodes. The distance on the overlay from the source is computed with:

$$\Delta(Tr(k)) = \left( \sum_{i=0}^{card(Tr(k))-2} d(n_i^k, n_{i+1}^k) \right.$$
$$Tr(k) = [n_0^k, \ldots n_{card(Tr(k))-1}^k] \text{ path of traversal} \quad (4)$$
$$\text{of } T_C \text{ tree from the root to node } k$$

A measure for evaluating tree quality is the maximum delay from the source:

$$M\Delta(Tc) = \max_k \left( \sum_{i=0}^{card(Tr(k))-2} d(n_i^k, n_{i+1}^k) \right) \quad (5)$$

Denote the tree built with n nodes Tc(n), f(k): fanout of node k and c(k): current number of children of node k.

The first strategy is to look for a parent node among the nodes already in the tree that is closest to the source on the delay path from the source. The *closest_parent* algorithm is presented in Table III.a. The complexity of the algorithm is O(NLogN) for node ordering and $O(N^2)$ for the tree construction (worst case analysis: N(N+1)/2).

A *minimum_distance_link* algorithm that selects the minimum distance between the nodes with available fanout currently inserted in the tree and the unattached nodes will result in optimal tree construction in case all nodes have identical capacity. In the case nodes have different forwarding capacities, the algorithm uses the combined distance metric, promoting nodes with higher capacity at the top of the tree. The algorithm is shown in the figure Table III.b.

The *minimum_distance_link* algorithm builds trees with maximum delay smaller than the *closest_parent* algorithm when the nodes in the overlay have similar fanout. The complexity of the algorithm is $O(N^3)$; the worst case analysis gives :

$$N^2(N-1)/2 - N(N-1)(2N-1)/6 \quad (6)$$

for an overlay of N nodes.

**Table III: Tree construction algorithms**

```
1.  Sort nodes n(k), k=1:N in the order of increasing
weighted distance d(n(k), S) ;
 i=1; list=NULL;  ΔTr(n(parent)) = 0 ;
2.  while(i<N)
        { Δmin = max_val; parent=0;
        insert(parent, list); position=Head(list);
        while(position=advance(list, position))
        {k=retrieve(list, position);
        ΔTr = ΔTr(n(k)) + d(n(k), n(i)) ;
                if ( ΔTr < Δmin )
                { Δmin =  ΔTr ; parent = k;}
        }
        insert n(i) as child of  n(parent);
        insert(i, list); c(parent)++;
        if ( c(parent)==f(parent))
        remove(parent, list);
   ΔTr(n(i)) = ΔTr(n(parent)) + d(n(i), n(parent))
   i++;
        }
```

a) Closest parent algorithm

```
1.  Construct parent and unattached node lists
plist=null; clist=null; j=1; parent=0;
insert(parent, list);
      for (i=1; i<N; i++) insert(i,clist);
2.  while (j<N)
{ Δmin = max_val;
while (position=advance(plist,position))
k=retrieve(list, position)
      while(pos=advance(pos, clist))
          i=retrieve(pos, clist);
{ ΔTr = ΔTr(n(k)) + d(n(k),n(i));
                if ( ΔTr < Δmin )
                { Δmin = ΔTr ;
                parent = k;child=i;}
}
      insert n(child) as child of  n(parent);
      insert(child,plist);remove(child,clist);
      c(parent)++;
      if ( c(parent)==f(parent))
      remove(parent,plist);
ΔTr(n(child)) = ΔTr(n(parent)) + d(n(child),n(parent))
      j++;
 }
```

b) Minimum distance link algorithm

## 4. Simulation results

We used the GT-ITM topology simulator [3] to evaluate the performance of stateless multicast on an overlay composed of high fanout cluster leader nodes and end-system (peer) nodes. The multicast group contained a total of 100 nodes selected from a transit-stub topology. The topology has one transit domain of 20 nodes with 10 stub domains per transit node. Each stub domain contains an average of 10 nodes. Cluster leader nodes with an average forward capacity five to ten times larger than that of peer nodes were placed in the transit domain. Peer nodes with identical forwarding capacity were selected at random (uniformly distributed) from the stub domains. Simulation results were averaged over ten overlays obtained by fixing the source node and the cluster leader nodes and randomly selecting the peer nodes. The shortest path distance between any pair of overlay nodes was computed for each overlay.

To analyze the performance of stateless multicast on a hierarchical overlay we evaluated the following metrics:

1. *multicast tree delay stretch*: the ratio of maximum delay between the source and a receivers computed on the multicast tree and the delay from the source to furthest node on the direct path, and
2. cluster *leader nodes utilization ratio*: the ratio between the forward capacity of cluster leader nodes used for forwarding in the multicast group and the capacity required to unicast data from the source.

The first experiment recorded the delay stretch when the number of cluster leader nodes (with capacity five times larger than the end-system nodes) increases between 1% and 8% of the total number of nodes. The results in Figure 2 show the improvement in delay stretch for closest parent and minimum distance link vs. closest node method.

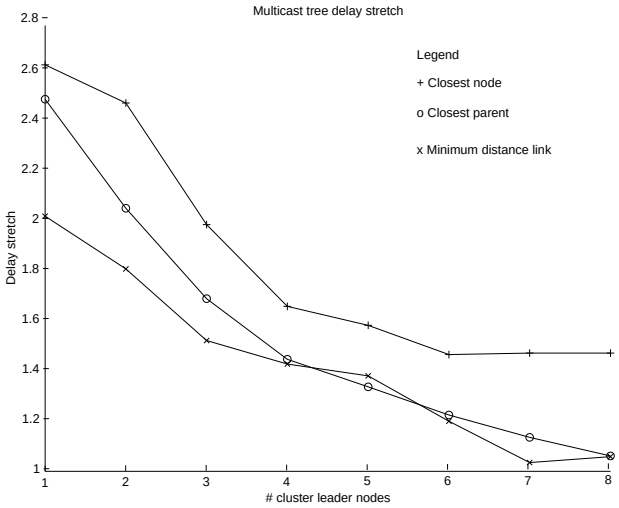


Figure 2: The delay stretch vs number of cluster leader nodes

The delay stretch gain of *closest_parent* and *minimum_distance_link* over *closest_node* method increases with the number clusters; however it is less than 50%, even when the cluster leader nodes have enough forwarding capacity to unicast to all peer nodes. The *minimum_distance_link* outperforms the other algorithms especially when the number of clusters is small. However its performance is close to that of the *closest_parent* when the number of clusters is large. As expected, the delay stretch decreases with the number of cluster leader nodes; with enough forwarding capacity at cluster leader nodes to unicast to all peer nodes, the added overlay delay stretch decreases to zero (data is forwarded on the shortest paths from the source to every node in the multicast tree) and the choice of tree building method has less impact on the quality of the data path.

Figure 3 shows the variation of delay stretch with the utilization of cluster leader nodes. Without the information between any two nodes the delay between parent and child node can be as large as the diameter of the cluster they belong to, increasing the delays between the nodes on the longest path in the multicast tree.

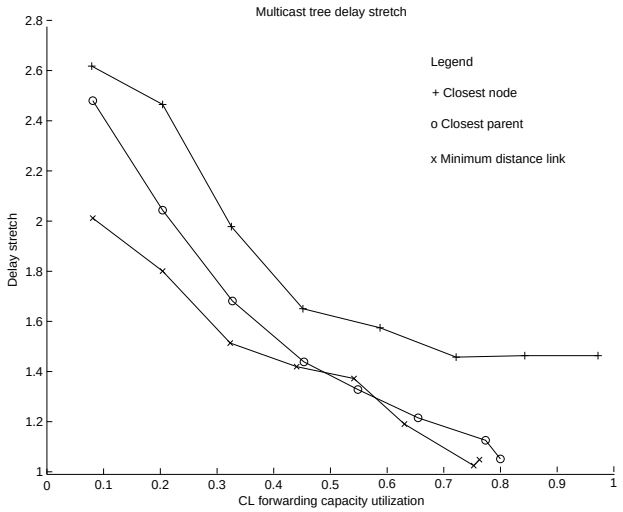


Figure 3: The delay stretch – aggregate cluster leader node utilization

Closest parent and minimum distance link are more efficient methods for constructing delay minimized multicast trees; using only 80% cluster leader forwarding capacity they achieve virtually no end-to-end maximum delay increase. The minimum distance link and closest parent can use only to 60% aggregate cluster leader forwarding capacity and degrade the delay stretch with less than 40%. When forwarding is concentrated at few cluster leader nodes, the closest parent method achieves comparable end-to-end performance with minimum_distance_link with smaller – $O(N^2)$- tree building complexity.

The second experiment measured the delay stretch with the increase of forwarding capacity available at cluster leader nodes (Figure 4).

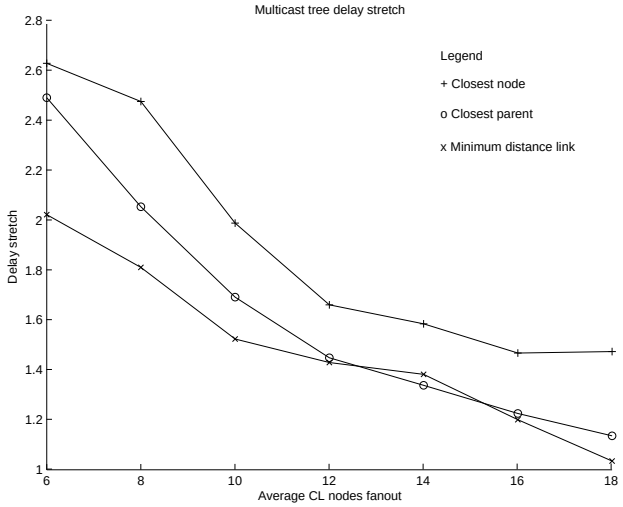


Figure 4: The delay stretch vs. average forward capacity of cluster leader nodes

The overlay consists of five cluster leader nodes each with forwarding capacity between three and nine times the forwarding capacity of end-system nodes. Both closest parent and minimum distance link achieve 15% improvement over the closest node method. Also when comparing to the first experiment, we conclude that the number of clusters (cluster leader nodes) has a larger impact on the maximum end-to-end delay than an increase in cluster leader node forwarding capacity

## 5. Related work

Several multicast protocols for networked group communication have been proposed recently. The small group multicast protocol [2] (and other Xcast protocols) also encapsulates the receiver addresses in the packet header. However these protocols relay on network support to route data to the destination, their efficiency being dependent on the underlying network support. The stateless multicast protocol we propose allows more flexibility in the design of control and routing path at the expense of small degradation in routing efficiency. The quality of the end-to-end data path can be optimized using network and application level metrics; in contrast the small group multicast protocols rely on the quality of service mechanisms provided by the underlying network.

Tree construction algorithms for real-time applications have been proposed in [1]. The tree construction is optimized for minimizing the average latency, not the maximum latency from the source. This formulation of the delay problem apply for video-distribution applications which minimize buffering at nodes in the multicast tree, but not for collaborative applications where the real-time constraints are related to the maximum end-to-end delay between nodes in the multicast group. While organizing the control overlay to reduce the amount of signaling required for node join and leave, OMNI protocol is designed for video on demand applications, where the amount of signaling is much smaller than in distributed interactive applications considered here.

## 6. Conclusion

We propose in this paper a stateless group communication mechanism along with algorithms for efficient data path control in distributed interactive applications. The overlay organization is hierarchical: each control node manages a cluster of peer nodes assigned based on network proximity.

The stateless protocol uses efficient multicast tree encoding to reduce the overall bandwidth and CPU utilization at forwarding peer nodes. We show that tree

construction algorithms achieve scalability and low overhead with a controlled degradation of the end-to-end data path performance on a hierarchical overlay composed of high forwarding capacity nodes and end-system nodes. In addition, stateless multicast reduces control signaling (node join and leave) proportional to the size of the multicast group. The reduction in control signaling is obtained at the expense of increased header processing at overlay peer nodes.

## References


[1] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," Proceedings of IEEE Infocom, April 2003.

[2] R. Boivie, N. Feldman, C. Metz: On the Wire - Small Group Multicast: A New Solution for Multicasting on the Internet. IEEE Internet Computing 4(3), 2000, pp. 75-79.

[3] K. Calvert, E. Zegura, and S. Bhattacharjee, "How to model an Internetwork", in Proc. of Infocom, 1996, pp. 594-602.

[4] T. Chang, G. Popescu, C. Codella , "Scalable and Efficient Update Dissemination for Interactive Distributed Applications," ICDCS 2002, pp. 143-152.

[5] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, "A case for end system multicast," in Proceedings of the ACM Sigmetrics 2000, June 2000.

[6] Y. Chawathe, S. McCanne, and E. Brewer, "RMX: Reliable Multicast for Heterogeneous Networks," in Proceedings of the IEEE Infocom '00, Tel-Aviv, Israel, March 2000, pp. 795-804.

[7] El-Sayed, A., V. Roca, L Mathy, "A survey of Proposals for an Alternative Group Communication Service, " IEEE Network, Jan/Feb 2003.

[8] J. Liebeherr, M Nahas, W. Si, "Application Layer Multicasting with Delaunay Triangulation Overlays", IEEE Journal on Selected Areas in Communications, Vol.20, No.8, October 2002.

[9] Macedonia, M., M. Zyda, D. Pratt, P. Barham, and S. Zeswitz. "NPSNET: A Network Software Architecture For Large Scale Virtual Environments." In Presence: Teleoperators and Virtual Environments, 3(4), 1994, pp. 265-287.

[10] Popescu, G., C. Codella, "An Architecture for QoS Data Replication in Network Virtual Environments", Proceedings of IEEE VR2002, March 2002, pp 41-48.

[11] Pullen, J.M. Wood, D.C. "Network technology and DIS", Proceedings of the IEEE , Volume: 83 , Issue 8 , Aug. 1995, pp.1156 – 1167.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "Application level multicast using Content-Addressable Network", Proc. of 3rd Int. Workshop on Network Group Communication (NGC 2001), London, UK, 2001, pp. 14-29.

[13] Singhal, S., and M. Zyda. Networked Virtual Environments, New York, ACM Press, Addison-Wesley, 1999.

[14] Stoica I. R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Content-Addressable Network", Proceedings of the ACM SIGCOMM 2001, San Diego, CA, USA, August 2001, pp. 149-160.

[15] Wong, T., R. Katz, S. McCanne, "An evaluation of preference clustering in large scale multicast applications," Proceedings of Infocom 2000, pp. 451-460.