Bandwidth-Sharing Schemes for Multiple Multi-Party Sessions

Zhen Liu[a], Naceur Malouch[b], Vishal Misra[c], Dan Rubenstein[c] and Sambit Sahu[a]

[a]IBM T.J. Watson Research Center, P.O.Box 704,
Yorktown Heights, NY 10598, USA

[b]INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93,
06902 Sophia Antipolis, France

[c]Columbia University, 500 W. 120th Street
New York, NY 10027, USA

A recent means for enabling multicast in the Internet involves deploying network overlays where end-systems participate in the forwarding of data to other end-systems. The use of overlays not only permits individual sessions to simultaneously structure their communication trees atop unicast-only networks, but also gives the session greater flexibility when forming the topology of the forwarding tree. Even though multiple sessions are often expected to compete for the same network overlay resources, most work to date assumes that overlay protocols operate as though each session has isolated access to the available overlay resources. We consider two algorithms that build depth-bounded overlay trees where each node's outgoing bandwidth constrains the number of nodes to which it can directly forward data. One algorithm tries to cluster a node's available bandwidth within a single tree, the other tries to disperse the available bandwidth among multiple trees. When node capacities are identical and session requirements are identical, a clustering approach will increase the number of sessions that can co-exist [6]. However, we show via simulation that in heterogeneous networking environments or in environments where session participants vary with time, the dispersing algorithm outperforms the clustering algorithm. These results can be used to guide future development of overlay protocols that must partition resources among multiple sessions.

1. INTRODUCTION

A viable approach today to provide multipoint communication over the IP network is via a *multicast overlay* [5,4,3]: a set of end-systems that connect the source of the transmission to all receivers in the form of a tree. Each intermediate node on the tree takes transmissions sent from its parent node and forwards these transmissions to its children nodes within the tree. The forwarding itself is often handled via network and transport level unicast protocols such as TCP, and UDP. While there have been several recent works [5,4,3] that develop efficient algorithms for building overlays for group communication, few have explored how to construct session topologies in environments where multiple sessions, whose memberships vary dynamically with time, compete for the same network resources.

Bounded-fanout multicast has been explored in the context of building (non-source-specific) minimum spanning trees that minimize aggregate edge costs instead of minimizing the delay from a specific source [2,1]. [11] presents a survey of previous work in the area of QoS multicast routing. However, there is no discussion in the survey of work that addresses the problem we consider here.

In this paper, we evaluate overlay construction algorithms that construct overlay trees in environments where each node bounds its fanout: the number of neighbors to which it can forward transmissions. Fanout bounds are often necessary in such overlay networks, since the end-systems that act as forwarding agents utilize modem, DSL or cable links and can only offer limited bandwidth to the sessions they support. Our optimization criteria is to build *depth-bounded* trees such that the number of hops through the overlay that are taken to reach a receiver within a certain bound. An algorithm that minimizes a tree's depth does not necessarily minimize the end-to-end propagation delay to a receiver as the delays between different pairs of overlay nodes can differ. Nonetheless, there are several reasons why depth-bounded trees are of interest. First, it is a non-trivial task to obtain accurate estimates of propagation delays between overlay participants. Second, delays that occur in transmission between such end-systems is often dominated by a combination of the delay across this last-mile technology and the lack of priority given to processing of arriving and departing packet transmissions.

We consider two baseline algorithms, CLUSTER and DISPERSE, whose approaches toward selecting the nodes that support transmission for a session lie at two extremes. CLUSTER attempts to apply a node's outgoing bandwidth toward a single session, thereby *minimizing* the number of sessions for which a node acts as a forwarding agent. In contrast, DISPERSE splits a node's outgoing bandwidth across as many sessions as possible while maintaining the depth-bound in the constructed tree, thereby *maximizing* the number of sessions for which a node acts as a forwarding agent.

In [6], we show via a mathematical analysis that CLUSTER is optimal in homogeneous environments where all nodes have the same bandwidth limitations and all multicast sessions have the same set of requirements. In this paper we focus on heterogeneous networking environments where nodes have variable capacity, and overlay participants can join and leave the network. We begin by modeling the heterogeneous environment as a queuing system and use this system to derive a theoretical lower bound on the blocking probability of algorithms in these settings. This lower bound translates into an upper bound on the rate at which sessions can be admitted into the network. Since no algorithm can perform better than this theoretical bound, it provides an additional benchmark to evaluate algorithm performance. Last, we use simulation to evaluate these algorithms in settings where session membership varies with time. We find that, although CLUSTER is optimal in homogeneous settings where session participation is static, DISPERSE either performs as well or outperforms CLUSTER, yielding lower blocking rates than CLUSTER in heterogeneous settings or where session membership varies.

Recent investigation into a similar problem has been considered in [9,10]. Our work not only presents a contrasting solution, but our solution and evaluation goes further by also capturing the important dynamics of participants joining and leaving a session when the session is already and continues to be active. Furthermore, our theoretical results provide performance bounds that apply to any solution in the area.

The rest of the paper proceeds as follows. In Section 2, we introduce the general model of the network and multicast sessions for our analysis. Section 3 describes the various algorithms that we consider. In Section 4, we introduce a Stochastic Knapsack queuing model that we use to compute lower bounds on blocking probabilities in heterogeneous networking environments. In Section 5, we compare the performance of our various proposed algorithms to one another and to the lower bound. In Section 6 we compare the performance of the algorithms when session members dynamically leave and join, and Section 7 concludes the paper.

2. NETWORK MODEL

In this section, we present the network and session models that will be used to evaluate our algorithms that form session trees for multiple sessions. Let $\mathcal{N}$ be the set of receivers, i.e., the set of nodes that wish to participate in at least one multicast overlay session. We define $\mathcal{S} \subset \mathcal{N}$ to be a sequence of these nodes that wish to act as transmitters for a session. Note that this set can contain repetitions, i.e., a node appears k times in $\mathcal{S}$ if it is the sender for k different sessions. We denote the ith entry in the sequence by s_i.

Let $\mathcal{R}_i \subset \mathcal{N}$ be a set of nodes that wish to receive the transmissions from sender s_i. By convention, we require that $s_i \in \mathcal{R}_i$. We assume that a node n is only willing to participate in overlay multicast session when it is interested in receiving the transmitted data of that session. In other words, $\mathcal{R}_i$ is exactly the set of nodes that form the multicast tree that will carry data for the session for which s is the sender. Note that there will be a set of leaf nodes in this multicast tree that will be receiving, but will not be forwarding these transmissions.

The ith session forwards data to all participants at an arbitrary rate, ρ_{s_i}. β_n is the aggregate outgoing bandwidth of node n, such that if n forwards directly to c_j nodes for session j, then it must be the case that $\beta_n \geq \sum_{j \in A} c_j \rho_{s_j}$ where A is the set of active sessions. This requirement is applicable to several last-mile technologies such as cable and DSL that provide asymmetric bandwidths in the two directions, where incoming bandwidth is significantly larger (and essentially unbounded) in comparison to the outgoing bandwidth.

We define Δ_i to be a bound on the depth of the tree for the session whose transmissions emanate from s_i, i.e., no node should be more than Δ_i overlay hops from the source node. For simplicity of presentation, we assume that $\Delta = \Delta_i$ is constant for all sessions.

3. ALGORITHMS

We now describe the algorithms that we use to determine whether or not to admit an incoming session, and, when admitted, the topology structure of the tree.

3.1. Algorithm CLUSTER

CLUSTER forms trees by utilizing a set of nodes for forwarding such that the nodes' remaining available bandwidth is minimized. The algorithm distinguishes between *partial nodes*: nodes whose bandwidth has already been applied within other sessions, and *full nodes*: nodes that do not currently forward transmissions on behalf of other sessions. CLUSTER tries to incorporate as many already-existing partial nodes as possible into the middle (i.e., not as leaves) in the tree before incorporating full nodes into the tree. The goal is to keep the number of sessions small for which a node forwards packets for.

Algorithm CLUSTER(s_i):

(1) Let P be the sequence of partial nodes in $\mathcal{R}_i$ that are sorted in the decreasing order of available bandwidth capacity. Let U be the set of full nodes in $\mathcal{R}_i$, sorted by decreasing available bandwidth capacity.

(2) Determine the minimum m for which a tree T can be constructed connecting all nodes in $\mathcal{R}_i$ where m nodes in U are non-leaf nodes and a possibly empty subset of nodes $P' \subseteq P$ as non-leaf nodes in which

- $D_{\max}(T) \leq \Delta$
- nodes with higher available bandwidth capacity appear at a lesser depth (closer to the source).
- any set of partial nodes P'' that satisfies $P' \subset P'' \subseteq P$ cannot be used as non-leaf nodes in a tree T' that satisfies $D_{\max}(T') \leq \Delta$.

(3) If such an m exists, build the tree from m nodes drawn from U and the remaining nodes drawn from the set P'. Otherwise, return that no such tree can be built.

We note that determining the minimum m for a given set P' can be done in time $O(|\mathcal{N}| \log |\mathcal{N}|)$. This follows from a result in work by Malouch *et al* [7] that proves that the minimum depth tree is the one where nodes with greater bandwidth availability are placed closer to the source of the tree. Hence, it is sufficient to build the tree by first sorting nodes in order of decreasing fanout, and then attaching the nodes to the tree in this order to the node of minimal depth that contains available edges.

3.2. Algorithm DISPERSE

Algorithm DISPERSE forms trees by utilizing as many nodes as possible for forwarding without exceeding the depth constraint. Barring a depth constraint, a chain of nodes is the preferred tree for DISPERSE.

Algorithm DISPERSE(s_i):

(1) Let S be the sequence of nodes in $\mathcal{R}_i$ sorted in order of decreasing available bandwidth capacity. Let c be the number of children that the first member of S can support for the session.

(2) Let $d_{\min}(c)$ be the depth of a minimum-depth tree, $T(c)$ that can be built using no more than bandwidth c from any node in S. Again, we use results in [7] to form this tree in time $O(|\mathcal{R}_i| \log |\mathcal{R}_i|)$. If $d_{\min}(c) > \Delta$, return that a tree cannot be constructed.

(3) Form the tree $T(c-1)$ with depth $d_{\min}(c-1)$. If $d_{\min}(c-1) > \Delta$, return tree $T(c)$. Otherwise, decrement c and go to step **(2)**.

Note that DISPERSE does more than merely "dispersing" the bandwidth in that it also tries to find the adequate fanout (c) that should be used to satisfy the depth constraint.

3.3. Algorithm Extensions

The algorithms described above have two shortcomings. The first shortcoming addresses the utilization of the node that will transmit data. This node may not have sufficient bandwidth to forward its own transmissions because its bandwidth is being utilized to forward transmissions of other sessions. We consider three possible actions that we refer to as *variants* that the network can take when faced with such a situation.

- no-swap: If s_i does not have enough outgoing capacity to initiate a session, the session is rejected.
- swap: s_i shifts some of its forwarding responsibility to another node that is participating in the same sessions as s_i that has the bandwidth capacity to take on the additional

bandwidth load. If no other node can be located that has sufficient spare capacity, the session to originate from s_i is dropped.

• reservation: A fixed amount of outgoing capacity is reserved at every node specifically for the transmission of a session that is initiated at that node. At other times, the bandwidth must remain unused.

The second shortcoming addresses the ***dynamics*** of membership within sessions. In particular, it is likely that there will be applications in which session members arrive late and leave early. Late arrivers must be added to the multicast tree within the depth bounds. By leaving a session or all sessions, a node that forwards transmissions for the sessions will disconnect the tree for each session it leaves. This disconnection must be repaired.

Our attachment and disconnect procedures for the two algorithms attempt to mimic the traits exhibited by the original algorithm. For CLUSTER, a late joiner is attached to the node with minimum available bandwidth that is sufficient to support an additional transmission such that the joiner is added at or above the depth bound. If no such attachment point can be provided, then the late joiner is attached to a node beyond the depth bound that has sufficient bandwidth to support such a node, when one exists. The node is otherwise rejected. The reason why we proceed to attach nodes in situations where the depth bound is violated will be explained below. When a node leaves the session, its children become the roots of detached subtrees, where the subtrees can have variable depth. These subtrees are then attached at the highest points at which there is available capacity to perform the attachment, with the subtrees of greatest depth being attached at the highest point. When there are several nodes at the same height to which an attachment can be made, the node with minimum available bandwidth is selected. A subtree is discarded when there is insufficient available bandwidth to connect the root of the subtree.

For DISPERSE, a late joiner is attached to the node with maximum available bandwidth that lies within the depth bound. If no such node exists, then a node with maximum available bandwidth that lies outside the depth bound is used. If no node has sufficient available bandwidth to forward transmissions to the late joiner, the late joiner is rejected. The algorithm to re-attach detached subtrees after a node leaves the session is the same as that for CLUSTER, except that the node with maximum available bandwidth is selected when there are several nodes at the same height to which an attachment can be made.

The tree formed as a result of nodes joining and leaving the session may be able to admit fewer nodes into the session beyond the depth bound than one constructed in a static setting. For this reason, the tree is periodically reconfigured by applying the CLUSTER algorithm or the DISPERSE algorithm to the set of nodes that are currently participating in the session. Hence, nodes that are attached beyond the depth bound can be brought within the depth bound during this reconfiguration. Those nodes that lie outside the depth bound after a reconfiguration are dropped from the session.

4. QUEUING MODEL: A STOCHASTIC KNAPSACK

We derive performance bounds on our algorithms by considering equivalent queuing systems. We map our system into a stochastic knapsack framework [8], and compute lower bounds on blocking probabilities based on the equivalent system.

Let $\mathcal{M}$ be the set of all possible bandwidth requirements of Sessions in the system (i.e.

product of transmission rate and session size-1). Let M be $|\mathcal{M}|$. The stochastic knapsack consists of C resource units to which objects from M classes arrive. Objects from class m are distinguished by their arrival rate, λ_m, mean holding time, $1/\mu_m$ and their size b_m.

Let n_m denote the number of class-m objects in the knapsack. Then the total amount of resource utilized by the objects in the knapsack is $\mathbf{b} \cdot \mathbf{n}$, where $\mathbf{b} := (b_1, b_2, \ldots, b_M)$ and $\mathbf{n} := (n_1, n_2, \ldots, n_M)$. We define the process in terms of the state space of the different class-m objects, i.e. let $\mathcal{K} := \{\mathbf{n} \in \mathcal{I}^m : \mathbf{b} \cdot \mathbf{n} \leq C\}$.
The knapsack always admits an arriving object when there is sufficient room. More specifically, it admits an arriving class-m object if $b_m \leq C - \mathbf{b} \cdot \mathbf{n}$. Let $\mathcal{K}_m$ be the subset of such states, i.e. $\mathcal{K}_m := \{\mathbf{n} \in \mathcal{K} : \mathbf{b} \cdot \mathbf{n} \leq C - b_m\}$.
The blocking probability B_m for a class-m call under Poisson arrival assumption is then given by [8]

$$B_m = 1 - \frac{\sum_{\mathbf{n} \in \mathcal{K}_m} \prod_{j=1}^{M} \rho_j^{n_j} / n_j!}{\sum_{\mathbf{n} \in \mathcal{K}} \prod_{j=1}^{M} \rho_j^{n_j} / n_j!} \tag{1}$$

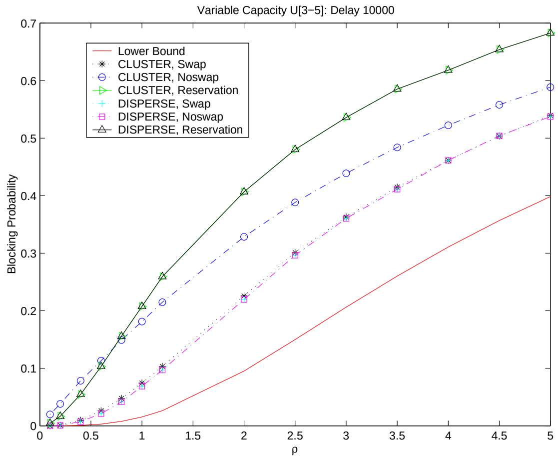
As a parsimonious metric to compare performance, we look at the weighted blocking probability w_b for each Stochastic Knapsack, where we weigh the blocking probability for a class$-m$ call with the resource request of the call b_m, i.e. if $\mathbf{B} := (B_1, B_2, \ldots, B_M)$, then $w_b = \frac{\mathbf{B} \cdot \mathbf{b}}{\sum_{m=1}^{M} b_m}$.

To compute the upper bound on the number of admitted sessions in our system (which correspondingly yields a lower bound on the blocking probability), we ignore the delay constraints and map the capacity C of the knapsack into $\sum_{n \in \mathcal{N}} \beta_n$, i.e. the aggregated bandwidth in the set of nodes. Thus, whenever there is available capacity in the knapsack (available bandwidth in the nodes), we'll assume the session is admitted irrespective of the delay constraints. The size of the object b_m is then simply the product of the size of the session (minus one) and the sending rate of the session. We also assume that the bandwidth requirements b_m are integers.

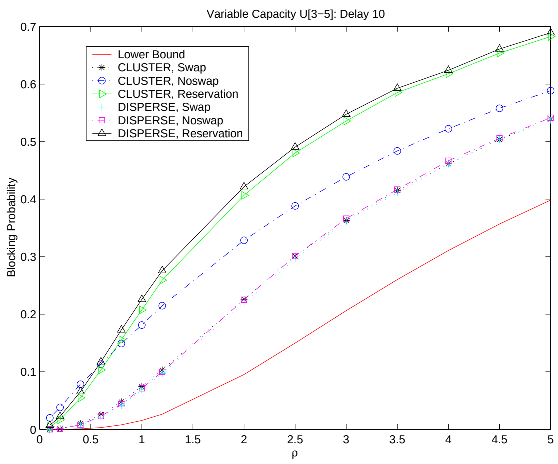
5. SIMULATION RESULTS WITH STATIC GROUP MEMBERSHIP

In this section, we compare the performance of the algorithms proposed in Section 3 via an evaluation of their blocking probability, i.e., the fraction of sessions that must be turned away because the participating receivers do not have sufficient bandwidth capabilities to support the arriving session. To perform this evaluation, we use simulation. The number of nodes, $|\mathcal{N}|$ equals 100 in all simulation runs. Sessions arrive at random points at time, where the arrival times are described by a Poisson process with rate $\lambda = 1$. These sessions last for a time that is exponentially distributed with rate ρ, which we vary over the simulation runs. For each arriving session we randomly select one node to be the source. Here, we restrict each node to being the source of at most one session at any instant in time.

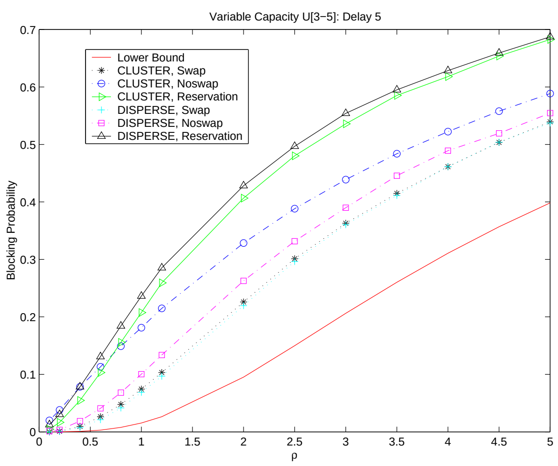
We now describe two experiments and discuss the conclusions that we draw from each set of experiments. In the first experiment we simulate variable node capacities while in the second one we simulate variable session rates. Other scenarios are also simulated in [6]. In the figures, we vary ρ, the expected lifetime of a session, along the x-axis. The y-axis indicates the blocking probability, and the various curves plot results for the various versions of the CLUSTER and DISPERSE algorithms we consider. For each experiment,



(a) high maximum hop-depth

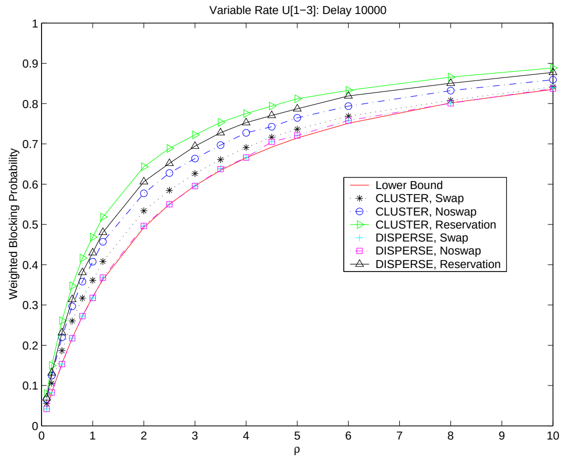


(b) medium maximum hop-depth

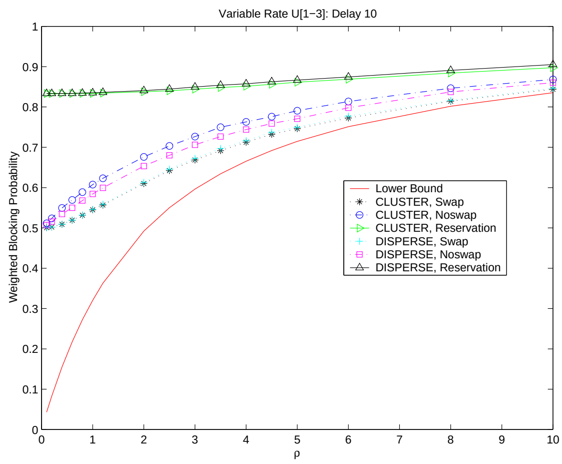


(c) low maximum hop-depth

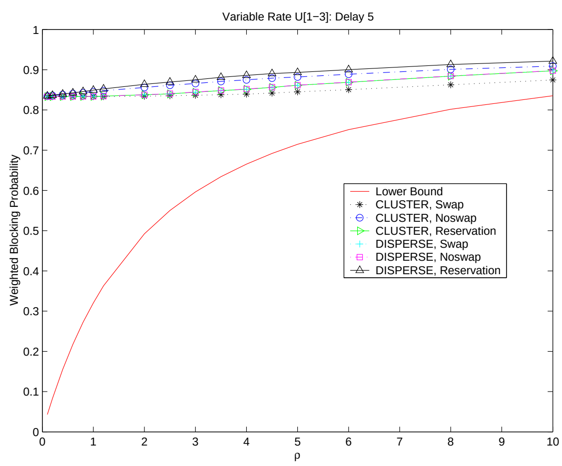
Figure 1. Case of variable node capacities



(a) high maximum hop-depth



(b) medium maximum hop-depth



(c) low maximum hop-depth

Figure 2. Case of variable session rates

we plot results from three sets with different maximum hop-depth constraints: (a) high, (b) medium, and (c) low. We also plot the theoretical lower bound on the weighted blocking probability computed from the Stochastic Knapsack framework. For the case where all sessions are offered at the same rate and the same size, this weighted blocking probability simply reduces to the straightforward blocking probability.

5.1. Experiment 1: Variable Node Capacities (Figure 1)

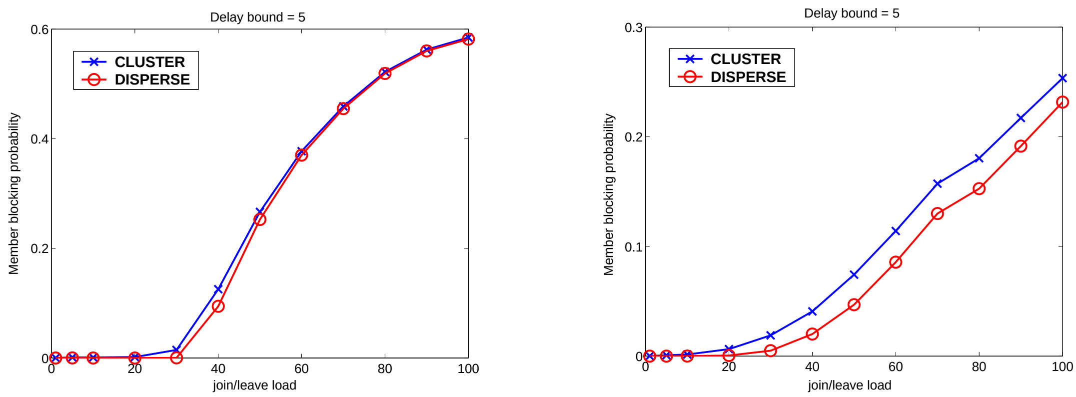
In this set of simulations, the bandwidth capacity units of each node is selected uniformly at random from the set $\{3, 4, 5\}$. Session size is fixed at 100 and each session's rate is fixed at 1. Since each session requires a set of 99 transmissions to reach all participants, a session consumes an aggregate bandwidth of 99 units. From these experiments (Figure 1), we first observe that reservation yields a higher blocking probability than the other variants. If swapping is enabled, CLUSTER and DISPERSE yield similar blocking probabilities. The one exception is the case where the maximum hop-depth is low. Here, using DISPERSE results in marginal improvements in blocking probability in comparison to CLUSTER. The conclusion to be drawn is that if swapping is not permitted, DISPERSE yields a lower blocking probability. Otherwise, the choice of CLUSTER or DISPERSE is arbitrary since the blocking probabilities of the two algorithms are virtually identical.

5.2. Experiment 2: Variable Session Rates (Figure 2)

In this set of simulations, the rates of arriving sessions are selected uniformly at random from the set of integers ranging from 1 to 3. Node capacity is fixed at 4 and session size is fixed at 100. Examination of the plots in Figure 2 reveals that The weighted blocking probability that results from using DISPERSE meets the theoretical lower bound for large (unbounded) maximum hop-depths. For tighter maximum hop-depths, CLUSTER with swapping produces a slightly lower weighted blocking probability than DISPERSE, but the weighted blocking probability without swapping is significantly higher. Here, we conclude that when session depths can be unbounded, or when swapping is not permitted, DISPERSE should be used. Otherwise, CLUSTER yields moderate improvements in weighted blocking probability.

6. SIMULATIONS WITH DYNAMIC GROUP MEMBERSHIP

In this section we present results of simulations conducted with dynamic node membership. Individual nodes join and leave the sessions, and we use the dynamic versions of CLUSTER and DISPERSE described in the end of Section 3. We simulate two scenarios: in one, the number of *sessions* is fixed, but individual nodes join to and leave from this static set of sessions. In the other, the set of active sessions themselves as well as the membership within those sessions vary over time. For all experiments, the maximum session size is 100 nodes, and each session's transmission utilizes one unit of bandwidth. For the static session case, the total number of sessions is fixed at 10, the join process is Poisson with rate 100 and the inter-update interval equal to 10. For the dynamic session case, the session arrival process is Poisson with rate 1, the session duration is exponentially distributed with a mean of 5 and the join rate within each session is again 100. The inter-update interval for the dynamic session case is much smaller, 0.05, as the topology in this scenario evolves much faster than in the static session case. For both scenarios, the member sojourn time is exponentially distributed, and we vary the mean along the x-axis, where the value indicated along the x-axis is the join rate divided by the leave rate. We plot the member blocking probability along the y-axis. Node capacities are uniformly distributed among $\{3, 4, 5\}$.



(a) Static Sessions

(b) Dynamic Sessions

Figure 3. Dynamic join/leave experiments, Heterogeneous capacities

In Figure 3(a), we see that the blocking rates that result from applying CLUSTER and DISPERSE are similar, especially when the join rate divided by the leave rate is high. Intuitively, this is because under high loads, available bandwidth is scarce, so that the typical tree formed is often the same, whether formed via CLUSTER or DISPERSE. For dynamic sessions, the blocking rates that result from applying DISPERSE are lower than those from applying CLUSTER (Figure 3(b)). Intuitively, we see two reasons why this is so. First, since in CLUSTER a node is either a leaf node, or (with high probability) completely utilized, a leave operation leads to a bursty process of re-attachment of the subtrees. Either there is no subtree to be re-attached, or there are a high number of nodes that must be re-attached, the latter requiring more bandwidth units. In contrast, with DISPERSE the available bandwidth is spread out among nodes. Hence, after a leave operation subtrees are more easily accommodated. Besides, the number of subtrees to re-attach is smaller than the one produced by CLUSTER even if the number of nodes to re-attach is the same. Indeed, in the best case, DISPERSE has to re-attach only one chain which needs one bandwith unit.

Second, DISPERSE exploits better the fact that the depth is unbounded between reconfigurations. Since in the leave operation the subtrees are re-attached in order to balance the whole tree, a node that has joined the session beyond the depth bound, could be brought within the bound either when other nodes leave or when the reconfiguration is performed.

7. CONCLUSIONS

In this paper, we have explored the problem of building depth-bounded multicast trees for multiple sessions in networking systems where tree depth and a node's outgoing bandwidth constrain the permitted topology of the tree. We consider two algorithms that build overlay trees within these constraints, one tries to cluster a node's available bandwidth within a single tree, the other tries to disperse the available bandwidth among multiple trees. We derive a lower bound for the blocking probability of algorithms in this networking environment and compare the performance of our algorithms to this lower bound through simulation.

An interesting finding of our paper is that, although the clustering algorithm provably minimizes the session blocking rate in homogeneous network environments, simulation results reveal that the dispersing algorithm exhibits a lower blocking rate in heterogeneous networking environments where session sizes, session rates or node capacities can differ. Furthermore, the dispersing algorithm performs better in both homogeneous and heterogeneous settings in environments where session participants join and leave in the middle of a session.

In summary, our study indicates that it is more efficient to spread each node's forwarding capacity across sessions given the heterogeneous nature of real networking environments. Our on-going work consists in specifying the distributed algorithms for dynamically building the trees and determining how and when it is appropriate to perform the swapping operation.

REFERENCES

1. F. Bauer and A. Varma. Degree-constrained Multicasting in Point-to-point Networks. In *Proceedings of IEEE INFOCOM'95*, Boston, MA, March 1995.
2. Fred Bauer and Anujan Varma. Distributed Degree-Constrained Multicasting in Point-to-Point Networks. Technical Report UCSC-CRL-95-09, UCSC, 1995.
3. Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.
4. Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.
5. Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS'00*, Santa Clara, CA, May 2000.
6. Z. Liu, N. Malouch, V. Misra, D. Rubenstein, and S. Sahu. Bandwidth-sharing schemes for multiple multi-party sessions. *Research Report, INRIA Sophia-Antipolis, May 2003. Available at* `http://www.inria.fr/rrrt/rr-4821.html`.
7. N. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A Graph Theoretic Approach to Bounding Delay in Proxy-Assisted End-system Multicast. In *Proceedings of the International Workshop on Quality of Service (IWQoS)*, Miami Beach, FL, May 2002.
8. Keith W. Ross. *Multiservice Loss Models for Broadband Telecommunication Networks.* Springer-Verlag, 1995.
9. S. Shi and J. Turner. Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks. In *Proceedings of NOSSDAV'01*, Port Jefferson, NY, June 2001.
10. S. Shi and J. Turner. Routing in Overlay Multicast Networks . In *Proceedings of IEEE INFOCOM'02*, New York, NY, June 2002.
11. B. Wang and J. Hou. Multicast Routing and its QoS Extension. *IEEE Network*, Jan/Feb 2000.