LARK: A Light-weight, Resilient Application-Level Multicast Protocol

Srikanth Kandula, Jong-Kwon Lee, and Jennifer C. Hou
Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, IL 61801
Email: {kandula, jklee72, jhou}@cs.uiuc.edu

***Abstract*— Application-level multicasting (ALM) has attracted a significant amount of attention, as it is a convincing alternative over traditional IP multicasting. While recent work has been focused, initially toward deploying an ALM alternative [8], [1], and later towards scalability in terms of sustaining the protocol for a much larger number of nodes [4], this has come at the expense of having to maintain sophisticated state at end-hosts. Furthermore, the use of leader-election mechanisms and rendezvous points creates concentrated points of failure in the overlay network and seems to be alien to the objective of fully distributed overlay creation and maintenance. Deterministic selection of overlay edges localizes the domain of exposure of an overlay node causing the overlay to be susceptible to clustered failures.**

In this paper, we present a simple, light-weight, yet scalable ALM protocol, called *LARK*, that allows the formation and maintenance of overlay topologies in a completely distributed fashion while maintaining only O(1) state at each node and ensuring robustness in the presence of a large number of node failures. Conceptually, members self-organize into cliques, where a clique is a cluster of end-hosts in which each end-host is aware of, and exchanges state with, every other end-host in the cluster. No control message is exchanged for clique maintenance beyond the necessary state update among members belonging to the same clique. In addition, members are allowed to peer with randomly selected members belonging to other distinct cliques. This ensures that in the event that one or more members leave or fail, the other members can re-join the group at other peers they are aware of. We elaborate on the components of LARK and derive certain theoretical bounds on its performance. We also validate our design through simulations.

I. Introduction

Traditionally, many-to-many communication over the Internet has been realized using IP multicast [3]. The IP multicast primitive provides the option for any receiver with an IP unicast address to subscribe to a common datastream relayed by a source by specifying, as destination, a unique address (referred to as the IP multicast address). Among the several innovations IP multicast offers in comparison to multiple IP unicasts to all the receivers, is the association of the IP multicast address[1] with the entire multicast. A host that is interested in receiving packets from a multicast group notifies its local router via the Internet Group Management Protocol (IGMP [5]). The local router then attempts to join the multicast tree that is rooted at the source and grows with the use of reverse path forwarding. Both the core and local routers must "intercept" multicast packets and determine (via consulting to the forwarding cache updated by a multicast routing protocol) on which interfaces these packets should be forwarded.

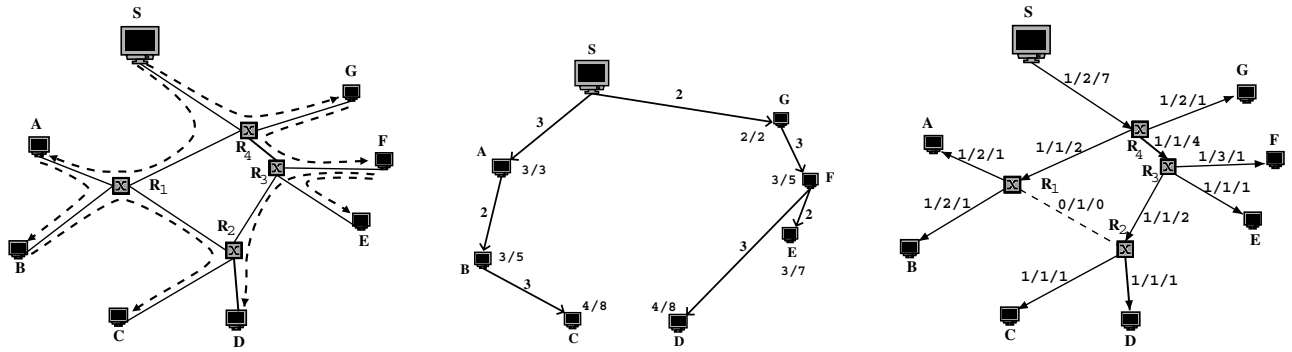
From the perspective of end hosts, the semantics of IP multicast are merely an extension to the existing IP unicast service, and the added complexity is minimal. However this is not the case for core routers. The functionality needed to realize IP multicast (and hence the complexity) is actually shifted over to the network layer. This is one of the primary reasons why IP multicast has not delivered.[2] One can also argue that such an imposition is in violation of the time-honored end-to-end principles [6] which discourage adding complexity on the communication substrate. In particular, the scalability issue arises from necessitating intermediate routers to maintain per-group state information for downstream receivers.

Recent attention has shifted to supporting multicast over the application layer, and the notion of application-level multicasting (ALM) was proposed. ALM refers to deploying the functionality of group communication at the application layer over end hosts rather than at the network layer. This implies that data is not sent over the inherent network topology but rather on an overlay topology that is constructed on-the-fly and continually optimized. There are many flavors of ALM depending on how much of the functionality is actually implemented outside the network layer. For instance, YOID [8] continues to employ IP multicast, albeit in a marginalized role, within a cluster of members that are close to each other under a shared-tree topology. At the other extreme, Narada [1] adopts a completely end-system approach, with the members maintaining all the requisite state and optimizing paths continuously.

ALM trades a less optimal use of router topology for the reduced overhead at the network layer. This is because the network layer abstracts away path selection choices (that it makes at the time of routing packets) from the application

[1]IP multicast addresses are in fact allocated from a separate class of IP addresses (class D).

[2]It must be said in its defense, however, that with the construction of the MBone overlay network, IP multicast has met with some albeit limited success.



(a) Multicast group comprising of a source and seven nodes

(b) Overlay topology. Nodes are marked by Ideal Hops/ALM hops

(c) Ideal multicast topology weighted by Ideal/ALM/Unicast stress

Fig. 1. Multicast on an overlay topology, Average node path stretch = 1.66 hops, percent increase in path usage $= \frac{1}{11} = 9\%$, ALM stress = $\frac{18}{12} = 1.50$, unicast stress = $\frac{22}{11} = 2.00$.

layer. Consequently, with the use of ALM, some paths are prone to be used multiple times to relay the same packet. Fig. 1 gives an illustrative example in which an underlying network comprising a source, seven receiver hosts and four routers is shown. In Fig. 1(a), undirected edges represent actual host-router/router-router connections, while directed edges depict the actual overlay edges. The abstracted graph representation of the overlay network is given in Fig. 1(b). As shown in Fig. 1(a), the link (F, R_3) is used multiple times to relay packets.

In spite of the many advantages that are accrued by employing an overlay network, several issues have to be addressed. In particular, Narada [1] considered how to form a mesh-like overlay network and continuously optimize its performance. NICE [4] addressed the scalability issue and proposed a hierarchical clustering structure among end hosts to support large multicast groups. However, the issue of robustness against widespread faults in the network layer has been largely left untackled. (As a matter of fact, as will be explained in Section V, hierarchical layering is more susceptible to failures[3] and incurs significantly more overhead in the course of failure recovery.) A light-weight solution that suffers little from failures and/or recovers quickly from failures is the need of the hour.

In this paper, we propose a completely decentralized, light-weight ALM protocol, called *LARK*, that aims at boosting the robustness of ALM while at the same time ensuring scalability. A member that is interested in receiving multicast packets joins at a random host in the group. Members self-organize into cliques, where a clique is a cluster of end-hosts in which each end-host is aware of, and exchanges state with, every other end-host in the cluster. The overlay network structure is thus restricted to one level of hierarchy. No control message is exchanged for clique maintenance beyond the necessary state update among members belonging to the same clique. In addition, members also peer with randomly selected members belonging to other distinct cliques. This ensures that in the event that one or more members leave or fail, the other members can re-join the group at other peers they are aware of. Optimization in terms of path quality is achieved with periodic state exchange between members in the clique as well as between peering members belonging to other cliques.[4]

What distinguishes LARK from other existing approaches is the almost complete absence of constraints on the behavior of the topology, thereby achieving the requisite optimization with regard to control traffic overhead. LARK allows any member to recover from failure or to better optimize its path by choosing to peer with other members without any restriction of maintaining any specific invariant property for the entire multicast group. LARK extends easily to work with a multiple source scenario as the tree structure is not coupled with the overlay graph structure. The simulation results indicate that both the control overhead incurred in overlay formation and maintenance and the time it takes to recover from failure are smaller. This is achieved by trading moderately the performance of LARK with respect to the quality of overlay networks.

The rest of the paper is organized as follows. In Section II, we describe the metrics used to evaluate the performance of ALM protocols, and summarize our design objectives. Then we present in Section III an overview of LARK, and elaborate on its components in Section IV. In Section V, we discuss the fault recovery mechanisms in LARK. Following that, we present in Section VI our simulation results. Finally we conclude the paper in Section VII.

[3]We make no distinction between the member leave and the node failure other than the fact that in the former case, the departing member may send out an explicit "leave" message that allows other members to recognize its absence immediately. In the case of the latter, the absence is detected by means of a timeout.

[4]We shall expound on the delicate tension between optimizing for latency and optimizing for reduced physical link stress in a later section.

II. Performance Metrics and Design Objectives

In order to assess the overhead of ALM in relying on an overlay network to relay packets, we use the metrics originally defined in [1]: *stretch* and *stress*. In what follows, we first give the definition of these performance metrics and then outline our design objectives.

A. Performance Metrics

The *stretch* of a path between a pair of hosts is defined as the ratio of the delay along a path connecting two end hosts on the overlay network to that along the point-to-point unicast path. The delay metrics typically used are the latency and the hop count. For example, the hop-count value for the ALM path and the IP multicast path between the source and a receiver is marked at the receiver in Fig. 1(b). Since the stretch is defined relative to IP unicast, the IP unicast path has itself an ideal stretch of unity. In a single source multicast tree, the average stretch is the average of the stretch values of individual paths between receivers and the source. The objective is therefore to minimize the average stretch.

The *stress* on a link is defined as the number of multiple copies of the same packet traversing a physical link when data is forwarded along the overlay network. The stress of each link for the IP multicast, ALM, and unicast cases is marked alongside each link in Fig. 1(c). Since the stress of a link is measured relative to IP multicast, IP multicast has a stress of unity by definition. To compare the stress across topologies, recent work [1], [4] computes

$$\text{Stress} = \frac{\sum_{l \in E} c_l}{|E|} \tag{1}$$

where c_l is the number of data packet copies transmitted on link l and E is the set of all links in the topology. Note that this value is one for IP multicast, and the closer the value is to one, the better the quality of the ALM protocol is.

Similar to the stress of a link, the stress of a router can be defined as the number of multiple copies of the same packet traversing the router per individual packet transmission using multicast.

B. Design Goals

In this section, we describe the guiding principles behind the design of LARK. We believe that these principles should be followed by any ALM alternative to IP multicast.

- **Support for large group sizes:** In order to support the plethora of target applications ranging from video-conferencing to real-time stock tickers, a multicast solution needs to scale well with the size of the group. This translates to the requirement that the control overhead, i.e. overhead involved in creating and maintaining the overlay, and the amount of state information stored at each node have to be bounded or allowed to grow at most logarithmically [4] with the group size.
- **Resilience to moderate volume of node leave/failure:** The multicast solution must perform well under transient behaviors that arise from occasional network outages or host leave events. A direct requirement of this objective is functional homogeneity among the participants of the overlay network. In particular, the data delivery and fault recovery strategies of the protocol should be independent of individual members' attributes. This is largely due to the fact that dependence on leaders would provide points of failure in the network that is impossible or prohibitively expensive to recover from.
- **Performance and Resource Utilization:** An ALM solution needs to approximate its multicast counterpart closely in terms of both performance and resource utilization. The trade-off between achieving good performance and using minimal resources needs to be carefully balanced. We will use the average stretch and the average link stress as defined in Section II-A to measure performance and resource utilization respectively.
- **Deployability and Backward Compatibility:** Finally, a virtue of any proposed solution is its ability to be deployed with little additional effort in existing networks, to co-exist with solutions that are already in place and to anticipate future modifications to adapt to varying requirements of the problem. In this regard, ALM scores heavily in the added abstraction of programming at the application layer, ease of re-programming and ease of deployment due to non-interference with the underlying routing layer.

III. Overview of LARK

We assume that the addresses of a limited set of already existing group members are publicly available. One possible method is to have early members register DNS entries that map a multicast group DNS name to their own network addresses. A new member that intends to join the multicast group need only to perform a DNS lookup on the well-known DNS group name to obtain network addresses of existing members. The new member then issues a join request to an arbitrarily chosen existing member (called the parent of the new member). Functional homogeneity of nodes in the overlay network allows each existing member to accept join requests. The new member then joins the *clique* which contains its parent, where a clique is a subset of overlay nodes that possess and maintain control information about each other.

Constraining the size of cliques is essential to bound the amount of per-node state and control overhead. To this end, we have designed a conservative distributed algorithm with which every overlay node uses to accept join requests, and to slice, if necessary, the clique with designated bridge nodes. (We will elaborate on the operations in Section IV-A.) Conceptually, LARK uses two design parameters: the *per-clique degree constraint* and the *per-node degree constraint*. The *per-clique degree constraint* D_c bounds the number of nodes that belong to a clique. Maintenance of state information is limited to members internal to a clique, and is $O(1)$ regardless of the number of nodes in the overlay. D_c is an overlay network parameter, and is universal for all the nodes. On the other hand, we allow each node in the overlay network to forward (receive)

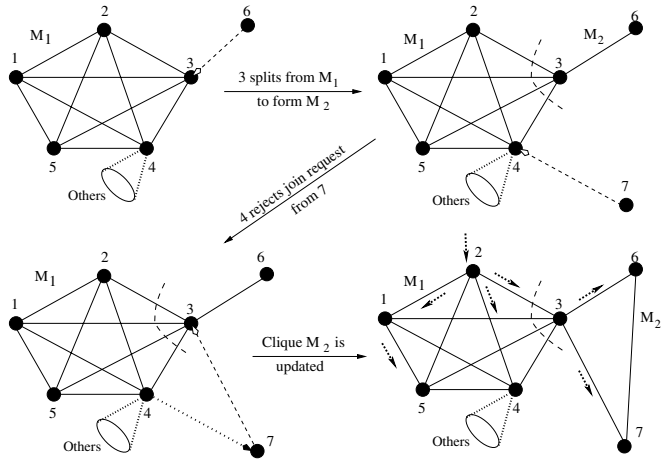


Fig. 2. Member join and clique slicing. Dashed arrows, long dotted arrows, and short dotted arrows, denote, respectively, join requests, join replies, and sample data flows in the clique.

data to (from) any of its neighbors, not all of which need to belong to the same clique as this node. This necessitates the presence of bridge nodes that transmit data received in one clique to neighbors belonging to another clique. In order to constrain the data forwarding and state maintenance load at each node, we define the *per-node degree constraint* D_n as the number of neighbors each node may have. Each node can choose its own value of D_n, as long as the following necessary condition for the construction of a connected overlay network holds:

$$D_n \geq 2 \cdot D_c \quad \forall n. \tag{2}$$

When a join request arrives from node m at a node ℓ in a full clique ($D_c + 1$ nodes), if node ℓ has fewer neighbors than its constraint D_n, then node ℓ slices itself from the clique and forms a new clique consisting of nodes ℓ and m. The bridge node ℓ continues to retain the state regarding its earlier neighbors. Such a bridge node is counted as belonging to each clique that it was a member of, for the purposes of enforcing clique size constraints. We will elaborate on clique slicing in Section IV-A.

Example 1: Consider the scenario depicted in Fig. 2 in which a clique has already been formed between nodes 1 to 5. Assume that the per-clique degree constraint, D_c, is four. Suppose a join request is initiated from node 6 and arrives at node 3. Node 3 has to split from its original clique to form a new clique, as it already has 4 in-clique neighbors. After joining the multicast group, node 6 exchanges state information with node 3, while nodes 1 to 5, including 3, exchange information among one another. Not all the join requests will be successful. For example, a join request by node 7 at node 4 may be rejected because node 4 has reached its neighbor limit. In this case, node 4 can recommend to node 7 a set of alternate network addresses that node 7 may contact. This set of nodes usually consists of neighbors and peers of the replier (node 4). Node 7 randomly chooses a node from this candidate-set (say node 3) and sends its new join request.

As noted above, members within a clique and neighboring nodes exchange state information periodically for several purposes. First, update messages serve as an indicator to the receiver that the sender is alive. Second, since every update message is acknowledged, sending an update message and receiving the corresponding acknowledgment allows the sender to estimate various metrics of quality on the round-trip path, such as latency and error rates. Finally, the topological information provided in the update message is used to populate a peer-pool of node addresses. In Section IV-C, we will describe in detail the nature of state messages that are exchanged.

Peering with nodes in remote cliques, apart from serving to optimize topology, also adds resilience, and provides fast fault recovery to, failures. By exchanging state information within a clique, node n can know cliques that are previously *unknown* to it and/or those that are connected by high quality links. Peering requests are sent randomly to nodes in the peer pool. A peering relationship is formed if both the participating nodes agree. The bound, D_p, on the number of peers that each node can maintain is primarily a function of the resources available at the node, and should be small enough not to induce significant congestion on the underlying links. We will elaborate on the peering process in detail in Section V.

With peers formed, the topology of the overlay network can be optimized by having peers that are mutually connected by high quality links to group together in a clique, with the rationale that such a design uses fewer network resources and provides better connectivity. On the other hand, in the case that a number of nodes within a clique simultaneously fail due to outages of the underlying network, links with peer nodes in other cliques can serve as alternate paths for temporary data delivery and eventual forming of new cliques.

One key advantage of LARK is that members can reorganize their local topological structure in the case of node failures/leaves, in a completely decentralized manner without having to incur additional overheads such as those associated with leader election. We will describe the failure recovery mechanism in Section V.

IV. Detailed Description of LARK Components

A. Member Join Mechanism

The pseudo-code for sending and replying join requests is given in Fig. 3. A node sends a join request to a randomly chosen node in the overlay network. If the receiver of the request can accommodate the new node, it responds with a join reply consisting of the network addresses of other members in the clique. Upon receipt of a join reply, the new member pings each of the other neighbors advertised in the join reply and starts its update and peer timers. When a node cannot accept a join request, it issues a join rejection consisting of a set of alternate hosts that the joining member may contact. Upon receipt of a join rejection, a node adds the set of alternates into its candidate list and chooses randomly from the candidate list the next candidate to send a join request. We will present in Section VI simulation results of the average number of join

request_join(*G, new_id, candidate_list*)
1. Randomly choose a node from candidate_list and issue a join request to that node
2. **if** (successful join) *//Ready for data transfer*
   a. store the list of neighbors and peers that is returned
   b. ping all the in-clique neighbors
   c. start the update timer and peer timer
3. **else** *//Try again with a node in the new candidate list*
   a. add to candidate_list list of alternate nodes returned in the join reply
   b. **goto** 1

join_timer (*t*)
//t: timeout interval of the join timer
//t_{last}: last occurrence of the timeout
//n_x: number of neighbors accepted since last timeout
1. t_{last} = now
2. n_x = 0

receive_join (*G, join_requester_info*)
//l: the receiver
//D_c: per-clique degree constraint
//D_n: per-node degree constraint
//d_c: number of neighbors in the current clique
//d_n: total number of neighbors in all the cliques of l
//latency(l,nbr_i): latency between node l and its i_{th} neighbor
//x: k× max(latency(l, nbr_i)), i = 1 … d_c, k>2
//t: last time the join timer was reset
//n_x: number of neighbors accepted by l in [t, t+x]
1. **if** ($n_x \leq \frac{D_c - d_c}{d_c+1} - 1$ **or**
   ($n_x \in [\frac{D_c - d_c}{d_c+1} - 1, \frac{D_c - d_c}{d_c+1}]$ **and** $y \in rand[0,1], y \leq n_x$))
   //Accept the join request
   a. Prepare a join reply and send
2. **elif** ($D_n \geq d_n + 2 \cdot D_c - d_c$)
   //Slice the clique and accept join request
   a. Generate new clique id, prepare join reply and send
3. **else** *//Reject join request*
   a. Generate a list of alternates, prepare a join rejection and send

Fig. 3. Pseudo-code for the distributed join algorithm

failures, which a node incurs before it successfully joins the group.

The key obstacle to a completely distributed join algorithm is the problem of ensuring that the clique size constraint continues to be met. Let d_c denote the number of neighbors of a node in the current clique, and d_n the total number of neighbors of the node in the current overlay network. In a naive approach, each member of the clique can accept $D_c - d_c$ new members while continuing to satisfy the constraint. However, if join requests arrive at clique members simultaneously and are accepted independently, the degree constraint of the clique will be violated. The solution given in Fig. 3 (receive_join, line 1) is motivated by the observation that since new members have to "ping" other members in the clique that they just joined, the time delay between the acceptance of a new member into a clique and the arrival of its ping message at every member of the clique is bounded by twice of the maximum latency between any two members of the clique. To account for varying network delays, end-system packet processing jitter and other means, we multiply the maximum latency by a constant $k, k > 2$. Intuitively, each end host maintains a join timer with a timeout value that is k times the current maximum latency. Within each timer interval, a node conservatively accepts only as many requests as it can under the assumption that every other active member in the group would do the same. If the number of nodes that can be accepted in one timer interval is less than one, a coin toss on the [0,1] real line makes the decision.

When a node determines that accepting a join request would violate its per-clique degree constraint, it attempts to accept the request by slicing itself off its existing clique(s) and forming another new clique, as long as the per-node degree constraint continues to be satisfied. When a node forms a new clique, it is necessary to ensure that the node can accommodate at least D_c more neighbors because accepting one more neighbor into the clique is equivalent to accepting all the nodes that this new neighbor can accept, the number of which is bounded by D_c. Hence a naive approach would be to allow a node to slice itself off as long as its per-node degree allows it to accept a clique-load of neighbors, i.e.,

$$D_n \geq d_n + D_c. \tag{3}$$

However, this does not account for the members that could have been already accepted by other members in the clique, the number of which is bounded by $D_c - d_c$. Hence, the correct condition is $D_n \geq d_n + 2D_c - d_c$ as given in Fig. 3 (receive_join, line 2).

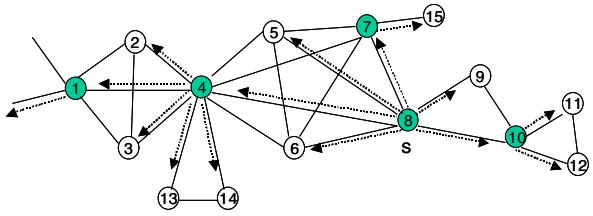
The node that sliced itself off its old clique(s) to form a new clique is called a bridge node as it belongs to both the old and new cliques. It exchanges state information with neighbors in each clique it belongs to. Control overheads are bounded by enforcing that the state information exchange between this bridge node and other nodes in the new clique is transparent to members in the older cliques. In summary, a bridge node can belong to multiple cliques and plays a role of connecting those cliques together. The clique identifier is assigned and maintained locally at each node for the purpose of data delivery. For example, in Fig. 2, node 1, 2, 3, 4, and 5 comprise one clique, and node 3, 6, and 7 comprise another. Node 3 belongs to both clique M_1 and M_2. The actual clique identifier, however, is managed at each node, e.g., $C_1 = \{2, 3, 4, 5\}$ at node 1 and $C_1 = \{1, 2, 4, 5\}$, $C_2 = \{6, 7\}$ at node 3.

B. Data Delivery

The clique-based overlay structure of LARK allows a simple and scalable data delivery mechanism. The mechanism is given in Fig. 4 and does not require any global topological information other than the limited state information on a node's neighbors. Given a data source, the data delivery path is built in the form of a source-specific tree on the clustered overlay topology. Each node forwards data packets received from a neighbor in a clique to other neighbors in different

multicast_data_forward (*s, p*)
// Data forwarding operation at a host r for a data packet p received from a host s
// r: this host
// s: the node that sends p to r
// m: # of cliques which r belongs to
// C_i^r: a set of host r's neighbors in clique i
1. **for** (each $i \in [1, 2, ..., m]$)
  1.1 **if** ($s \notin C_i^r$)
    a. Forward the data packet p to hosts in C_i^r

Fig. 4. Pseudo-code for the data forwarding mechanism.



At node 8: **c1** = {4,5,6,7} : **fwd**
(source) **c2** = {9,10} : **fwd**

At node 7: c1 = {4,5,6,**8**}
c2 = {15} : **fwd**

At node 6: c1 = {4,5,7,**8**}

At node 4: **c1** = {1,2,3} : **fwd**
c2 = {5,6,7,**8**}
c3 = {13,14} : **fwd**

At node 10: c1 = {**8**,9}
c2 = {11,12} : **fwd**

Fig. 5. An example that demonstrates how multicast packets are delivered. Node 8 is the source.

cliques it also belongs to. Fig. 5 depicts an example scenario that demonstrate how multicast packets are delivered.

C. Exchange of Update Messages

Each update message from node n is sent to all of its neighbors and peers. Update messages can be used in several ways. First, update messages and associated acknowledgments contain timestamps allowing the sender to obtain accurate estimates of path quality such as round trip latency and path loss rate. The path loss rate can be estimated by keeping a per-neighbor(peer) counter that keeps track of the fraction of un-acknowledged packets within the last T seconds. To accommodate measurement variations, a low-pass filter with an exponentially weighted moving average can be used in the latency calculation to obtain a weighted average of a few recent path loss estimates [11].

Second, update messages can be used to detect neighbor (peer) failures. A node maintains a counter for each neighbor (peer) that it directly communicates with. The counter value is incremented when the current node sends an update to the peer and is reset when an update is received from the peer. A counter value, say k, means that the current node has not heard from the peer in the last k update intervals. When the value of k reaches an upper bound k_{max}, its neighbor(peer) is no longer alive and/or active.

Finally, an update message consists of partial link-state information about its neighbors and peers which can be used, if desirable, by other neighbors to populate their peer pool for both the purpose of topology optimization and random peering. We will elaborate on how update messages are used for topology optimization and random peering in Section V.

Each node in the overlay network maintains a *constant* amount of state that is bounded by $O(D_n + D_p)$, where D_p is the degree constraint on the number of peers. The control overhead at each node is expected to mainly come from exchanging update messages because the update messages are generated periodically at each node and sent to all its neighbors and peers.

V. Random Peering and Its Use in Failure Recovery

An essential component of LARK is random peering performed by the members in the multicast group. The technical motivation for random peering is best illustrated by considering the failure-handling technique used in NICE [4]. NICE builds a hierarchy of members within the multicast group. In the case of member leave/failure, members within the cluster elect a leader based on who they perceive the "center" of the cluster to be. This election is reconciled with multiple messages being exchanged between each of the candidate leaders and finally, a unique leader is elected. There are two potential disadvantages with such an approach. First, member leave/failure incurs $O(k^2)$ messages among the remaining members in the election process, where k is the number of members in the cluster. During this period, all members do not have parents and hence do not receive any multicast packets. A second disadvantage is the inability to handle a large number of failures across multiple levels of the hierarchy. Specifically, let n denote the number of members in the multicast group. Given a particular cluster under the NICE infrastructure, suppose $O(k \log n)$ member leaves/failures are induced in such a way that there are $O(k)$ leaves/failures in every layer-*i* cluster, then the complete path information from any recipient to the root is lost with high probability. This is because, while each cluster is able to recover from one leave/failure by means of electing a new leader, the new leader has no means of recovering from the partition in all the other clusters its belongs to. Moreover, once the new leader is elected, it may no longer remain a member of all the clusters the previous leader was, as the criteria for choosing the cluster head depend upon the "centerness" of the member in the cluster. Consequently, this can lead to a breakdown in the entire hierarchy.

To solve this issue, we propose to decouple failure recovery from any underlying structure constraints. By this, we mean that any attempt to recover from failure/member leave should not depend upon constraints imposed by the solution upon the participating entities. For instance, in the case of NICE, failure recovery is difficult, and sometimes impossible, to achieve since members are still required to remain in the same cluster as they were, and do not instead capitalize on the local nature of the failure by leaving the cluster and joining other clusters. The protocol is forced to necessarily work around a single point of failure, i.e. the cluster leader, thereby causing unbounded delay in repairing the partition.

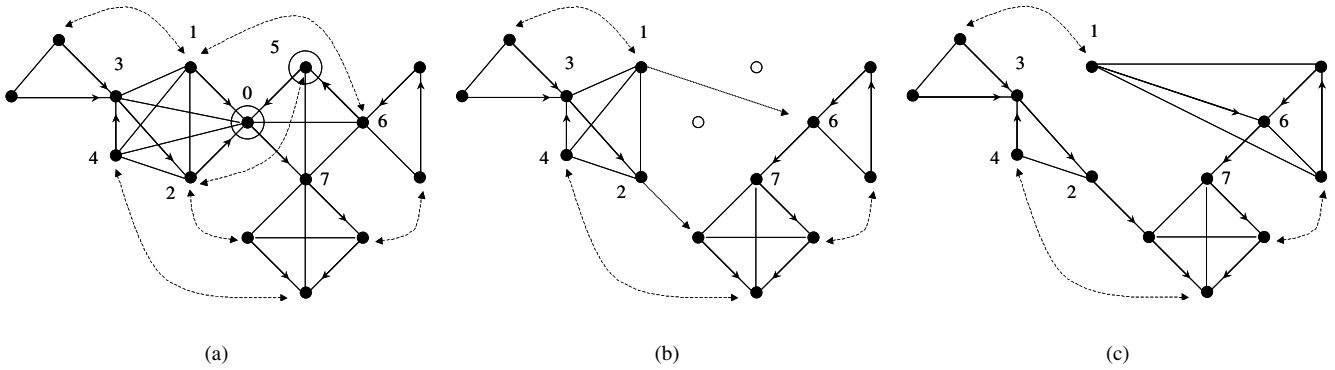


Fig. 6. Random peering and failure recovery. Dashed edges represent peering relations. (a) A sample LARK topology. The circled nodes 0 and 5 are about to leave the group. (b) Nodes 1 and 2 are trying to recover from the partition by joining one of their peers. (c) Node 1 moved to some other clique, and node 2 accompanied its children in the previous clique to another clique.

We tackle this problem by requiring that members continuously attempt to peer with as many random members from other cliques as possible (Fig. 6(a)). This can be achieved separately in the background and with a low period so as not to impose on the control channel bandwidth. The number of peering relations that a member wishes to have has no bearing on its actual degree. Subsequently, a member can choose to peer with as many members from distinct cliques that it can find. This approach differs markedly from NICE where a degree bound is enforced upon members in the same cluster and no requirement that they peer with members from other clusters is made.

To obtain the information on random peer candidates, nodes include members that belong to different cliques as part of their update message. As the group grows in size, more cliques are formed by members splitting from the existing cliques. However, since these bridge nodes continue to remain neighbors of members in the old cliques, they can provide members belonging to their different cliques with information about each other.

Fig. 6 gives an example of partition detection and failure recovery. With the clustered overlay structure of LARK, a partition occurs when a bridge node leaves the multicast group or fails. In this case, each remaining neighbor still retains the capability to recover from the partition, as each member has already peered with a number of members in other distinct cliques. We stress here that no member needs to renew acquaintance with members in the current clique since every member has peered with another member in a distinct clique. One requirement here is that each member responsible for failure recovery should contact only peers over the dead bridge node to recover from the partition correctly.

With the leave/failure of a bridge node in a clique, the clique is more or less "disbanded" with each member requesting to join other cliques they knew of. However, if every members in this clique joins other cliques, repeated partitions may occur at adjacent cliques. To prevent further partition from taking place, the failure recovery algorithm was devised to minimize the number of members attempting to move to other cliques. To this end, each member maintains the address of its parent node, i.e., the node at which it joined. If a member leaves a clique, whether it leaves the multicast group or moves to some other clique, each child member replaces it with the living ancestor member in the same clique, if exists, as the new parent. For example, the parent of node 6 is changed from node 5 to node 7 in Figs. 6(a) and 6(b). Only those child members any of whose living ancestor members are not in the same clique will join other cliques (node 1 and 2 in Fig. 6(b)). Moreover, they leave the current clique only if they do not have their own descendants in that clique (Fig. 6(c)).

VI. Performance Evaluation

We have conducted an event-driven simulation study to evaluate the performance of LARK. Our simulation is carried out on a transit-stub topology of approximately 10,000 nodes generated using the GT-ITM topology generator [13]. The end-hosts for the multicast group are chosen randomly from nodes in the stub domain, and the group sizes were varied from 50 to 1000 hosts. All the members join the multicast group randomly between simulation time 0 and 200 seconds. To rule out second-order effects, we assume there is no data loss and/or excessive queuing delay due to congestion.

To compare fairly with NICE, we used the same network topology and unicast routing protocol for both NICE and LARK. Moreover, the same update period of 5 seconds is enforced. The period with which random peering requests are sent by each host in LARK is also set to 5 seconds. (Note that we observe the period with which random peering requests are made could be set to a larger value, without degrading the performance significantly.) We experiment with different per-clique constraint D_c and per-node degree constraint D_n. However, due to the page limit, we report below only results with $D_c = 5$ and $D_n = 25$. Each simulation run lasts for 1000 seconds so as to allow an appropriate overlay topology to stabilize.

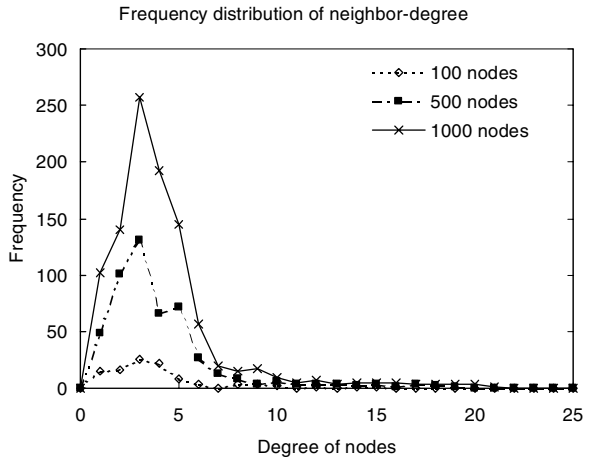


Fig. 7. The distribution of node degree under various group sizes in the overlay network.

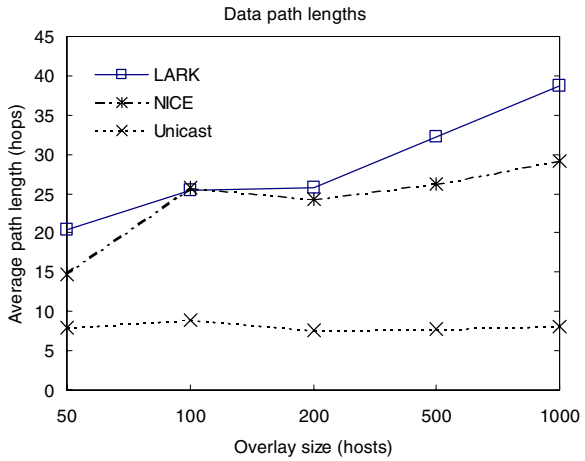


Fig. 8. Average path length under LARK, NICE, and multiple-unicast under different group sizes in a network of 100 end-hosts.

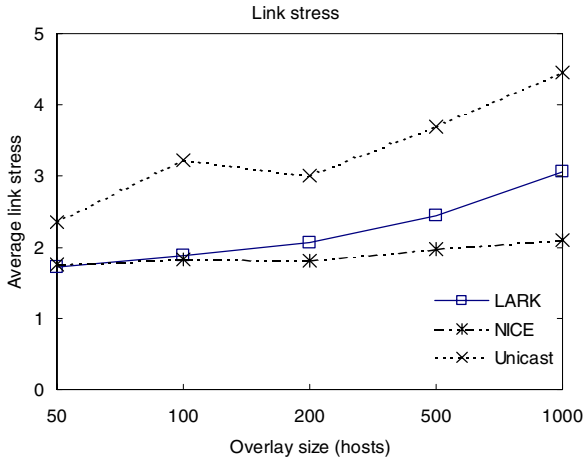


Fig. 9. Average link stress under LARK, NICE, and multiple-unicast under different group sizes in a network of 100 end-hosts.

A. Quality of Overlay Topology

Fig. 7 gives the distribution of the node degree (i.e., the number of neighbors each host has) in the overlay networks under different group sizes. The node degree is directly proportional to the amount of control overhead involved in the clique formation and maintenance. As illustrated in the figure, a significant proportion of hosts have less than 10 neighbors, and the average node degree is approximately 4 under all the cases.

Figs. 8 and 9 depict the average path length and the average link stress under LARK, NICE, and multiple unicast for the various multicast group sizes. The measurements were made at simulation time 1000 seconds. The performance of NICE with respect to path length and link stress is slightly better than that of LARK. This is, in part, because the join procedure in NICE aggressively finds good points of attachment for new members in the overlay topology, with the help of a rendezvous point and cluster leaders at each layer in the hierarchy. In contrast, in the current design of LARK, a new member joins the multicast group at a randomly chosen member, and topology optimization commences only after the join procedure is completed. Moreover, topology optimization is performed in a completely distributed manner by each host, and as a result is likely to render a suboptimal overlay topology. As will be discussed below, the fact that we deliberately do not assume specific roles on individual members is one of the reasons LARK is robust to, and can recover fast from, failures. In some sense, we are trading the mild degradation in the quality of overlay networks for better robustness.

B. Control Overhead

The control overhead is assessed by counting control messages (including messages for overlay formation and maintenance as well as join and leave operations) at the access links of end-hosts during simulation time between 500 seconds and 900 seconds. Fig. 10 gives the average control overhead at end-hosts under LARK and NICE. Both protocols are shown to incur low control overhead and scale well to large groups. Although the average control overhead for moderate group sizes is lower under NICE, it increases comparatively faster with the group size. In particular, most of the control overhead is concentrated at cluster leaders in NICE. According to the analysis of NICE [4], the control overhead at cluster leaders increase logarithmically with the increase in the group size. On the other hand, the control overhead is constant under LARK regardless of the group size. This is because the number of control message exchanges at each host is bounded by the number of its neighbors and peers (which is a constant even for a very large group size). As a result, LARK scales better than NICE in the case of extremely large groups.

C. Failure Recovery

To investigate the effect of member leave/failure, we carry out several simulation runs with a varying number of members that leave the multicast group. After the overlay network is stabilized with 100 members, a set of randomly chosen n members leave the group without notifying other members.

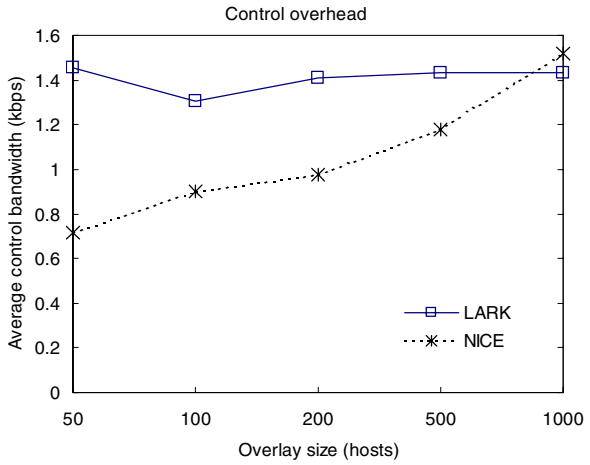


Fig. 10. Average control bandwidth at end-host access link of the LARK and NICE protocol with 100 end-hosts with varying overlay sizes

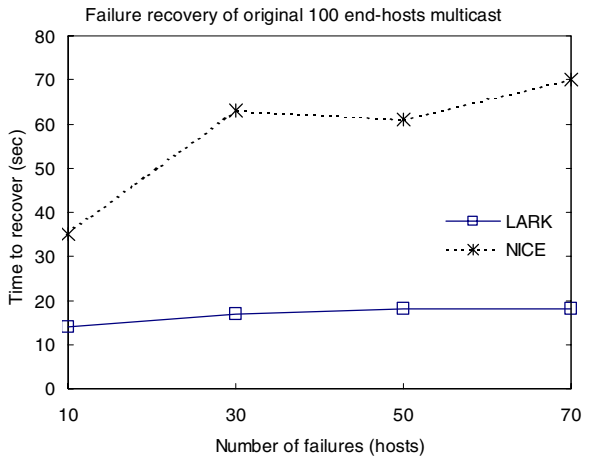


Fig. 11. Time to restore the data path from multiple host failures: originally 100 end-hosts multicast

Then the time it takes for receivers to restore their data paths (i.e., the time elapsed before all the remaining group members receives data packets) is recorded.

Fig. 11 depicts simulation results when 10, 30, 50, and 70 hosts out of 100 end-hosts fail. As shown in this figure, LARK recovers from the failures much faster than NICE under all the cases. Furthermore, the time to recover from failures is almost constant, regardless of the number of host failures. This results from the fact that failure recovery in LARK is performed by allowing members that detect the failure to independently join one of their peers in some other cliques (which have been known by random peering). As a result, the recovery time has no direct relation to the number of failure. In contrast, as previously described in Section V, under NICE single member leave/failure requires $O(k^2)$ message exchanges to elect a new leader, where k is the number of members in the cluster. In particular, member leave that occurs in the higher layer may have more severe effects on the time it takes for remaining members to restore data path.

VII. Conclusion

In this paper, we have presented a simple, light-weight, and yet scalable ALM protocol, called *LARK*, that allows the formation and maintenance of overlay topologies in a completely distributed fashion while only maintaining $O(1)$ state at each overlay node. In particular, LARK is engineered to work for a much larger problem space than existing solutions [2], [4], and to be resilient to high volume of node leave events as well as to clustered outages.

In spite of its significant improvement with respect to scalability and robustness, LARK can be further improved with respect to the quality of overlay networks (e.g., path length and link stress). The current design for optimizing the overlay topology in LARK is only based on simple decision made by each host by simply comparing the distances to its neighbors and peers and choosing the closest member to move into. How to optimize the overlay network topology without compromising robustness is an issue to be investigated.

References

[1] Y. -H. Chu, S. G. Rao, and H. Zhang "A case for end-system multicast," in *Proc. ACM SIGMETRICS*, Santa Clara, CA, 2000.

[2] Y. -H. Chu, S. G. Rao, S. Seshan and H. Zhang "Enabling conferencing applications on the Internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM*, San Diego, CA, 2001.

[3] S. Deering, "A scalable multicast routing protocol," Ph.D Dissertation, Stanford Univ., 1989.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast", in *Proc. ACM SIGCOMM*, Pittsburgh, PA, 2002.

[5] W. Fenner "Internet group management protocol, version 2," `ftp://ftp.isi.edu/in-notes/rfc2236.txt`

[6] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. on Comput. Syst.*, vol. 2, pp. 195-206, 1984.

[7] P. Francis, "Yallcast: extending the Internet multicast architecture, " `http://www.yallcast.com`, Sep. 1999.

[8] P. Francis, "YOID: extending the Internet multicast architecture," `http://www.aciri.org/yoid`, Apr. 2002.

[9] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O' Toole Jr. "Overcast: Reliable multicasting with an overlay network," in *Proc. 4th Symp. Operating System Design and Implementation (OSDI)*, Oct. 2000.

[10] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location routing," Tech. Rep., UCB/CSD-01-1141, Univ. of California, Berkeley, CA, Apr. 2001.

[11] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM Symp. Operating Systems Principles*, Banff, Canada, Oct. 2001.

[12] J. Jannotti, "Network-layer support for overlay networks," in *Proc. IEEE Conf. Open Architectures and Network Programming*, New York, June 2002.

[13] K. Calvert, E. Zegura, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, Mar. 1996.