A Generic Set-Formation Service

Amit Sehgal, Kenneth L. Calvert, James Griffioen
Laboratory for Advanced Networking
University of Kentucky
Lexington, KY 40506
amit,calvert,griff@netlab.uky.edu

Abstract— **Dynamic group formation and discovery is an important component of several emerging end system applications and network services including peer-to-peer and overlay systems, network storage/caching systems, and reliable multicast/distribution services. In our previous work, we proposed a programmable, scalable grouping service based on multicast and concast. In this paper, we present a generalized abstraction that subsumes the earlier service and can be implemented using other forms of network support in addition to (or apart from) multicast/concast. In designing the new grouping abstraction, we have attempted to separate the policies governing set formation from the network mechanisms (multicast, concast, distributed hash tables, etc.) used to implement those policies. In addition, we allow policies to be specified in terms of both application-specific criteria and network-based criteria. We describe how the service can be used to solve grouping problems from different application areas, and give an example showing how it can leverage different network support mechanisms, including ephemeral state processing.**

I. Introduction

Distributed applications and services frequently need to assign participants to sets or groups for reasons of efficiency and scalability; examples range from peer-to-peer applications, to overlay networks, to basic network services like reliable multicast. Common to many of these are the problems of (1) assigning nodes to groups, and (2) deciding how should the resulting group be structured or organized.

For example, consider a hierarchical reliable multicast protocol employing retransmission servers that *subcast* (i.e. transmit to a subset of receivers) repair packets to minimize network load. A key requirement for such a system is the ability to place receivers who experience similar patterns of packet loss in the same subcast group [15]. In some cases, a "group leader" may also need to be selected to act as the retransmission server. This information must then be conveyed to all receivers in the subgroup so they know where to send ACK or NACK messages.

This example highlights some important aspects of the problem of assigning nodes to groups or sets. First, the *policies* governing assignment of nodes to groups along with leader selection are determined by the application. Second, the *input* to the grouping process can come from the nodes participating in the application, from the network, or from both. An example of the former is a "lossprint" indicating which packets were received at a particular node—say, a fixed-size bitmap with a bit set for every lost packet. An example of the latter is the average round-trip delay between every pair of members of a subgroup, which could be used in selecting an optimal leader. Third, the outputs of the grouping process may take different forms for different participants. For example, it may suffice for subgroup members to know only their leader's identity, while the leader may need to know the entire subgroup membership.

In this paper we present a generic set-formation abstraction that can be used as a building block for various kinds of distributed applications. We use the term "set formation," rather than "grouping," to emphasize that the service does *not* construct multicast groups or routing trees; it simply defines sets of nodes, based on policies defined by the application. These policies may be specified in terms of application-supplied criteria, or network-derived performance and topology information, or some combination of application and network criteria. We propose three different implementation platforms for the abstract service and show how active approaches can both enhance and simplify the task of incorporating network information into the set-formation policy.

The remainder of the paper is organized as follows. Section II describes our set-formation abstraction. We then describe three architectures for the service implementation in Section III. We show how the service can be used to solve some common set-formation problems in Section IV. Section VI describes existing approaches to forming sets based on application-specific or network-specific criteria. Section V discusses simulation results; we conclude in Section VII.

II. A Generic Set-Formation Service

We propose a *generic set-formation service* that allows nodes to be assigned to sets according to application-supplied policies, based on *both* application-level and network-level information. The general service abstraction is not bound to any specific implementation or underlying network architecture. Instead, it makes minimal assumptions about underlying network mechanisms, while taking advantage of underlying network features when they are available (e.g., in active networks). The general "shape" of our service is shown together with its architectural context in Figure 1.

Before any set-formation can occur, a *set-formation session* must be created by some end system called the session *creator* (not indicated in the figure). At any given time, multiple sessions may be active, each associated with a different application and/or group of participants. Each session is uniquely identified by a *sessionID*, which is a combination of the creator's network address and a (unique to the creator) session number. We assume that session participants discover information about the session using some out-of-band protocol (e.g., a protocol similar to SAP/SDP [11], [10]).

The set-formation service itself is implemented at end systems. It takes two kinds of inputs: application-specific information, obtained from the application program running at each end system; and information obtained from the network layer. The nature of the information available from the network layer depends on the services present in the network, but at a minimum would include, for example, round-trip-time measurements to specific nodes and path enumerations obtained via `traceroute`. If more sophisticated active/programmable services are available in the network layer (e.g. concast, multicast, distributed hash tables, ephemeral state processing), then more sophisticated information may also be obtained. The output of the service at node A is a set S_A of node identifiers. Outputs at different nodes may be different; for example, in Figure 1, node A's output is the singleton set $\{B\}$, while node B's output is $\{A, C\}$.

At the time of session establishment, the creator specifies the *set-formation criteria* that determine—based on the input information supplied at each participating end system and obtained from the network—the output returned at each node participating in the session. The set-formation criteria can be thought of as a metric on (node, set) pairs: the set S_A returned at node A is such that the value of the metric for the pair (A, S_A) is "at least as good" as the metric for (A, S) for any other set S of participating nodes in the session, given the input information supplied by those nodes. In general, the metric has two components: an application-specific component, which operates on the input from the application, and a network component, which operates on the network-related input.

Let *appInfo*$_A$ denote the information supplied by the application at node A, and let *netInfo*$_A$ denote the network information measured or otherwise obtained at/for node A. For a given A and set S, let $M_{app}(A, S)$ denote a number that measures the "application-related goodness" of the set S with respect to the node A, and similarly let $M_{net}A(A, S)$ indicates the "network-related goodness" of the set S with respect to A. By specifying a set-formation criterion as a function $F(M_{app}, M_{net})$, the application can combine its own information—such as keywords describing available files, packet loss patterns, etc.—with network-based metrics, such as distance between nodes in hops. The service needs to provide a way for the creator to specify F, M_{app}, and M_{net}. Since the service is responsible for collecting network-based metrics, it must provide some elementary measures, in terms of which M_{net} can be defined. One possibility is to define a simple language for these specifications and have the creator provide a description of F in that language. Another approach would be to have the creator supply mobile code (e.g. a Java class file) that implements a service-defined interface specification.

We can now describe the set-formation process in more detail. Once the session is established, each participating end system A provides its application information (*appInfo*$_A$) to the set-formation service. The set-formation service is responsible for obtaining *netInfo*$_A$ for A as well as all other participating end systems. Using all of the input values available from participating nodes, the set-formation service determines the set S_A that maximizes the value of

$$F(M_{app}(A, S_A), M_{net}(A, S_A))$$

The resulting set S_A is then returned to node A.[1] (The examples considered in this paper use the sum of M_{app} and M_{net} for F.)

As an example, consider the reliable multicast example described in the Introduction. In this case, each

[1]The list of nodes in S_A need not be returned explicitly; an identifier or some other encoding of S_A might be returned.

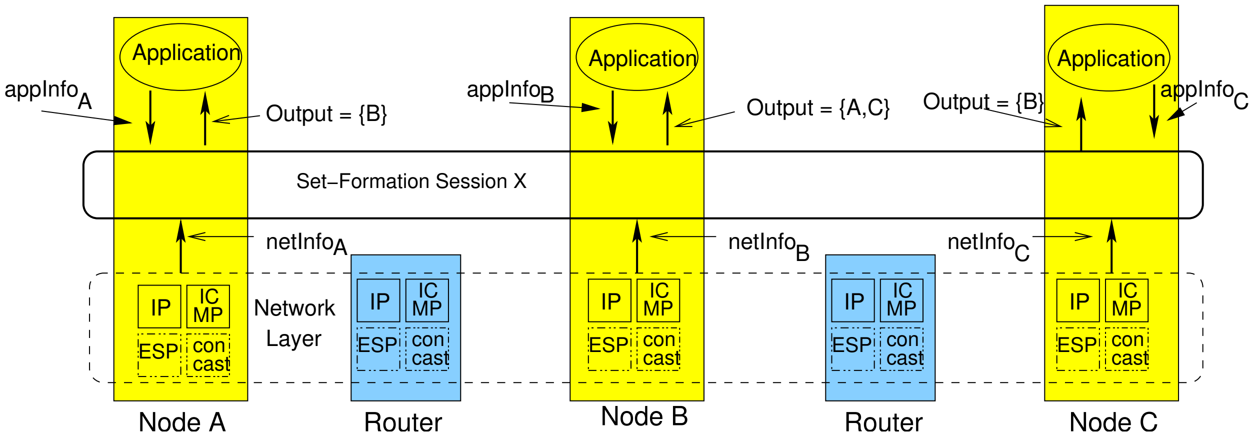


Fig. 1. Set-formation Abstraction

participant (multicast receiver) R would contribute its lossprint as $appInfo_R$. The application metric would compare lossprints to find the largest set of receivers with lossprints within some Hamming distance threshold of each other:

$M_{app}(R, S)$
    $v = 0;$
    for each R' in S {
        if (HD($appInfo_R, appInfo_{R'}$) $<$ *threshold*)
            $v+ = 1;$
        else return 0;
    }
    return v;

To avoid the case where receivers in different parts of the multicast tree (behind different bottleneck links) are grouped together, we can define the network metric to impose a maximum hop-distance between receivers, beyond which they are not assigned to the same set:

$M_{net}(R, S)$
    $v = 0;$
    for each R' in S {
        if (D(R, R') $<$ *threshold*)
            $v+ = 1;$
        else return 0;
    }
    return v;

where $D(R, R')$ is some measure of the network distance between R and R'. (The distance might be measured directly, or inferred by comparing the paths from R and R' to a common destination, counting the number of hops in which they differ, and using that difference as an upper bound on the distance between them.)

Defining a generic set-formation service in this way has several advantages. First, because the application and network metric are supplied by the application, the service can be adapted to meet the application's needs. Second, how the service is implemented is hidden from the application/user, allowing it to be offered across a wide range of environments (possibly with different levels of efficiency). For example, the service may be implemented using a centralized set-formation server (at the expense of scalability), or it might be implemented in a distributed fashion, or it might be programmed right into the network (e.g., active network approaches). In addition, the details of how the network information is obtained are hidden from the application writer. Section III proposes several possible implementations of the service. Third, by creating multiple sessions and using them sequentially with metrics that take into account prior results, applications can create "structured" sets. For example, the initial session might group all nodes with similar lossprints together. Then for each resulting set a new session would be created that returns to each node its neighbors in a spanning tree linking nodes of that set.

A significant challenge in implementing our service (or indeed any mechanism for forming subsets of arbitrary sets of nodes) is the need to identify the participating nodes that are potential members of a node's set, and to bring together the input information of such nodes *in a scalable way*. The overhead required to achieve this, in terms of communication, state, and computation, depends on the amount of support available in the network and on the set-formation criteria specified by the application. For example, if some processing can be done while information is *en route* through the network, it may be possible to support larger sessions than are feasible without such support. On the other hand, if the set-formation criteria require that measurements for every possible pair of nodes be compared, the quadratic cost of that comparison generally cannot be hidden by the framework (although there may be special cases where the asymptotic cost is lower, for example due to vagaries of the topology). In the next section we describe how the service can be instantiated with different degrees of network programmability and support.

III. Implementing the Service

Like IP, our generic set-formation service is not tied to any particular network architecture and can be implemented and supported across a wide range of underlying network technologies. Although the service makes minimal assumptions about the underlying network, it is able to leverage and benefit from enhanced network services when they exist. This is particularly true for the network metrics used by the service. If the underlying network offers support for discovery of information such as distance (hop-counts) between nodes, such support can be used directly without the need to invoke end-system-based mechanisms (ping, traceroute, packet pairs, etc.).

In the following sections we discuss three different implementations of the generic set-formation service. Each is based on a different underlying network model ranging from standard IP where all higher-level services must be implemented entirely at end systems, to networks with programmable services such as concast. In the models that offer active/programmable network support, set-formation can be performed in the network itself and network characteristics can frequently be inferred in-band, resulting in better scalability and accuracy.

Although the set-formation service can be supported across a wide range of platforms, it should be noted that its performance, scalability, and accuracy will vary depending on the underlying network, just as IP's performance varies depending on the underlying physical network being used. In particular, the set of available metrics will generally depend heavily on the mechanisms available at the network level. However, the correctness of the set-formation abstraction is not compromised. That is, while the allowed specifications may be weaker than if greater network support is present, the sets that are returned will indeed satisfy the given specifications. In that sense, the service, like IP, is a best-effort service, offering the best possible sets given the underlying network service's limitations. For example, if the application wants to create sets such that all nodes in the set are within five network hops of one another, the service will not return sets where the nodes are more than five hops from one another. However, if the network does not offer complete network hop measurements, the service may omit some nodes that could have been part of the set. This may lead to a non-optimal set that reduces the application's performance in some way. In all cases, it is the application's responsibility to formulate the metric specification in such a way that unacceptable sets are ruled out.

A. Centralized Service Implementation

The obvious way to implement the set-formation service is to deploy a centralized "set-formation server". The creator of the session contacts the set-formation server to enable the session and install the application-specific set-formation functions, M_{app} and M_{net}. Participants in the session then send their data values, $appInfo_i$, to the server to be processed and grouped. The server then responds to each participant with information about the set to which the participant has been assigned.[2]

The centralized approach has some advantages. It can be implemented on any network since it does not require any special support from the network. Because it is an application-level service, it is completely end system driven. The only requirement is that a single end system (possibly located inside the network at a data center) must be selected as the set-formation server. In addition, from the standpoint of applying the application-specific metric M_{app}, the centralized approach is very easy to implement, because all the end system values are available at the set-formation server.

However, as with most centralized approaches, scalability is an issue. Clearly, as the number of set-formation session increases, the server can become a bottleneck in terms of both communication and computation. Distributing the service across servers and assigning sessions randomly—say, according to session ID—to different servers (cf. DNS root servers) provides at best a linear improvement in the number of sessions that can be served, and does not address the issue of very large sessions with tens or hundreds of thousands of participants.

Perhaps the biggest challenge of a centralized implementation is to automatically obtain network charac-

[2]Alternatively, the set-formation server may take a lazy approach to the distribution of the results, waiting until an end system explicity requests the results before sending them.

teristics on behalf of the session participants. Network measurements such as the distance between two participants are not yet widely available. In the absence of such support, the server must use conventional network measurement approaches, possibly relying on a specialized measurement infrastructure [3], [5]. Third party measurements done by a set-formation server using generic mechanisms like ping, traceroute, packet pairs, or directed SNMP probes impose an increased load on the network and are typically less accurate than measurements obtained with the assistance of the network infrastructure itself. In the next subsection we consider how such network support can improve the scalability of a centralized approach.

B. *Concast-based Service Implementation*

The centralized approach assumed no support from the network and thus was limited in its scalability as well as its ability to obtain network properties and performance characteristics. However, if the network offers limited forms of support, the service can be implemented more efficiently with better accuracy. One example of such a network is a *Concast-enabled* [6] network environment.

Roughly speaking, *concast* is the inverse of multicast. In multicast, data is sent from a single source to multiple receivers by *duplicating* packets inside the network. Concast, on the other hand, allows multiple senders to transmit packets to a single receiver that are *merged* enroute to the receiver, producing a single packet that is delivered to the receiver. A *merge specification*, tailored to the application and provided by the receiver, defines which packets belong to the concast *flow*, and how they are merged together in the network. A signaling protocol deploys the merge specification to concast-capable routers along the path from the flow's senders to the flow's receiver. When a concast packet en route from sender to receiver encounters one of these concast-capable routers, it is processed according to the merge specification. Typically this processing results in some flow-associated state information being updated; when the state satisfies some predicate, a packet is constructed and forwarded toward the receiver.

In prior work [17] we described a way to group nodes based on concast merge specifications. The service, unlike the generic service described here, was tightly-coupled to the concast abstraction, but did demonstrate the ability to form sets using concast. The concast set-formation abstraction also was designed to support set-formation primarily based on application-specific information, with topology-dependent information being added implicitly, through the network-based processing provided by the concast service.

The service works by selecting one of the participants to act as a concast receiver. This participant provides a concast merge specification to the concast set-formation service that consisted of three functions: *process_value*, *can_combine* and *combine*. The three functions are used to transform data into a format appropriate for comparison, test whether nodes should be grouped together, and ultimately combine the nodes together, possibly merging the data from the combined nodes. Participants then concast their values toward the receiver who receives a single packet containing the set information. The receiver then multicasts the results to all participants. The approach described in that earlier work [17] needs to be modified only slightly to support the proposed *generic* set-formation service. Either the service must be modified to support more general network metrics (i.e. outside concast), or the set of allowable metrics must be constrained to match what can be obtained using concast.

Unlike a centralized approach without network support, concast is able to obtain certain network characteristics while performing the merge, without additional network probes or measurements. For example, because concast builds a "merge tree", the service can identify nodes that are near one another (i.e., there exists a common router that can be reached in a certain number of hops), or the service can identify nodes that share a bottleneck link. Also, the service must be modified to support the notion of a creator that may or may not act as the receiver. The receiver may be identified by the creator, or it may be selected by the generic set-formation service.

An important feature of the concast-based set-formation policy is that the merging takes place within the network. As messages from multiple senders arrive at intermediate network nodes, the set-formation metrics are applied and a single outgoing message is created. This helps in two ways. First the processing load is distributed across the network. Second, the number of messages traversing the network is significantly reduced, thus reducing the likelihood of *implosion* at the receiver.

C. *ESP-based Set Formation*

Another network-level service that can be used to implement the generic set-formation service is *Ephemeral State Processing (ESP).*

Ephemeral State Processing [8] is a general purpose network-level building-block service that supports end-to-end services by allowing packets to create limited amounts of temporary state at routers and to invoke simple predefined computations on that state. The three main components of ESP are: the *ephemeral state store (ESS)* which stores data associated with a user-selected tag for a fixed duration of time; the *instruction set* that defines the computations packets can invoke to modify values in the store and/or the packet; and finally the *wire protocol*, which defines the ESP packet format and way in which packet are processed and forwarded. Each ESP packet carries the identifier of a single predefined instruction to be executed at ESP-capable routers, together with a small number of tags referring to values stored in the ESS, and a small number of values to be used in the computation. As an ESP packet traverses by an ESP-capable router, it is recognized as such and the specified instruction is executed. The execution of the instruction may update the values in the ESS and/or fields in the packet; upon completion the packet is either replaced in the normal forwarding path or silently discarded. End systems construct global computations by sending ESP packets to each other in a coordinated manner. The sequence of packets "seen" by an ESP-capable node forms a simple program, the output of which can be collected by one or more final ESP packets.

ESP can be used for the purpose of set-formation by having the nodes send their application-specific data in ESP packets. As the ESP packets sent by participants traverse the network, they gather information about the network topology and they deposit application-specific input at routers so that it can be used to form sets. Thus, both the M_{app} and the M_{net} can be mapped onto ESP instructions (or sequences of ESP instructions). The resulting ESP computation derives the needed network information and processes it inside the network; the end systems receive what they need to complete set formation. Section IV provides an example of how network/application metrics can be mapped onto a sequence of ESP instructions.

Like the concast approach, ESP allows information to be processed inside the network, thereby avoiding the problems of overload and implosion that can occur with a centralized set-formation server. Also, because ESP is a lightweight service, there is no network-level session setup overhead, especially compared to concast. ESP also allows network measurements to be easily obtained, both as the participant's values travel through the network, and also via the use of the ESP messages that employ the reflector bit [8], which allows an end system to determine information about the path from any node back to itself. In Section IV, we show how an ESP-based generic set-formation service can be used in forming overlay networks.

IV. Using the service

Following the success of napster [2] and gnutella [9], there has been an explosion in the research directed towards building efficient overlay and peer-to-peer networks. In this section, we show how our generic set-formation service can be used as a building block on which these types of services can be constructed.

Overlay and peer-to-peer networks consist of end systems connected by virtual links (often implemented as tunnels). In many cases, end systems function both as "hosts"—participants on the distribute application—and as "routers" forwarding packets on the sender's behalf. Thus, overlay paths between hosts may not (and often do not) correspond to the unicast path in the underlying network topology. A poorly constructed overlay may result in packets traversing unnecessarily long paths that may cross the same physical link multiple times. Ideally, the overlay topology will be based (to some extent) on the underlying network topology. Several researchers have proposed techniques that address these problems by attempting to connect nearby nodes together [14], [18], [12], [16], [19], [13]. In the following section we show how our general-purpose grouping service can aid in achieving this. Section IV-B then briefly describes other common grouping problems that can be solved using our service.

A. Overlay Formation

In this section we show how an ESP-based implementation of the grouping criteria can be used to group nodes based on their topological proximity to one another. In this case we are solely interested in optimizing $M_{net}(A, S_A)$, where S_A is the set of nearby nodes—e.g. nodes whose distance to A is $< threshold$ (as described in Section II), or the k nodes closest to A. In both cases, we derive $M_{net}(A, S_A)$ from a sequence of ESP instructions that determines the distance from node A to surrounding nodes and then selects the closest neighbors.

Given a large N (number of participants), probing/measuring O(N^2) path distances is clearly too expensive—not to mention that nodes must already know about each other in order to take the measurements. Therefore we employ an approximation technique that strictly limits the scope of a node's search as well as the network traffic generated during the search. Each node only learns about nodes that lie near some path through the network. The distance to these nodes is approximated via triangulation–which, in the worst case, overestimates the distance. Having computed the distances, the nearest nodes are chosen and returned as the group members.

This can all be accomplished with the three ESP instructions shown in Figures 2–4. Initially all participants transmit an *sf-deposit* instruction toward a common destination;[3] We assume the instruction is executed at every router along the path to the destination. (We relax the assumption that all routers are ESP capable at the end of this subsection.) The purpose of the sf-deposit instruction is to record, at each router, the nearest participant—measured in hop counts. A *sf-range* instruction is then sent (by all participants) to the same destination to discover how far away the other nodes are. After the sf-range returns the list of distances to other nodes, an *sf-gather* ESP message is sent to discover the identity of the nearest nodes.

```
sf-deposit () {
    p.hops ++
    if (*p.t1 is ⊥)
        put (p.t1, p.sender_id)
        put (p.t2, p.hops)
    else if (*p.t2 ≥ p.hops)
        put (p.t1, p.sender_id)
        put (p.t2, p.hops)
    if (p.hops ≥ p.hops_thresh)
        drop p
    else
        forward p
}
```

Fig. 2. ESP instruction to deposit state

```
sf-range () {
    p.hops ++
    if (*p.t1 != p.sender_id)
        total ← *p.t2 + p.hops
        if (total < p.max_index)
            p.count[total] ++
    forward p
}
```

Fig. 3. ESP instruction to find range of hop values

The sf-deposit instruction (Figure 2) creates state at each router to record the closest known participating node to that router, as measured by hop count. Two tags, t_1 and t_2, specify the ESS entries that store the identity of the closest participating node and its distance (in hops) to the router, respectively. The *hops_thresh* parameter in the packet controls the scope of the packet, and causes packets that traverse more than *hops_thresh* routers to be dropped. The sf-deposit instruction checks to see if the sender of the packet is closer to the router than the best known node (if any). If so, the sender's IP address, which is carried in the sender_id parameter, is stored in t_1, and the number of hops it has traversed so far is stored in t_2. Otherwise the ESS is left unchanged and the packet is discarded. (It need not be forwarded because if an sf-deposit packet from a sender closer to the path has already passed through that node, the current packet will not cause a state change at any downstream node along the path.)

[3]The destination can be any arbitrary node. Its only role is to echo ESP packets back to their originators.

```
sf-gather () {
    p.hops ++
    if (*p.t1 != p.sender_id)
        total ← *p.t2 + p.hops
        if ((p.min_hops ≤ total) && (total ≤ p.max_hops))
            p.index ++
            if (p.index < p.index_thresh)
                p.nn[p.index] ← *p.t1
                p.nnhops[p.index] ← *p.t2 + p.hops
    forward p
}
```

Fig. 4. ESP inst to gather state

The sf-range instruction (Figure 3) carries a histogram (array) of neighbor distances. At every router, the instruction computes the triangulated hop distance between the packet's sender and the closest known node and increments the appropriate entry in the histogram. Upon reaching the destination, the packet (containing the resulting histogram) is returned to its origin, enabling the sender to know how many neighbors are at each (triangulated) distance from it.

The sf-gather instruction (see Figure 4) is sent third, to gather the identities of nodes whose hop-distance is within a specified range At each router, the sf-gather instruction again computes the triangle hop-distance as in the previous instruction. If that hop distance is between *min_hops* and *max_hops*, it is selected as "nearby" and its identity (address) is stored in the packet's *nn* array and the triangular distance is stored in the *nnhops* array.

Figure 5 illustrates the process graphically. After the sf-deposit message is sent, each router knows about the node closest to it. In the figure this information is indicated by the notation N_*,x where N_* is the identity of the node closest to the router and 'x' is N_*'s hop distance from the router. In the second phase, the sf-range packet generates a histogram of triangulated distances. For the example in Figure 5 we get the following histogram of node-count distances (note that N_s at distance 1 does not need to be recorded).

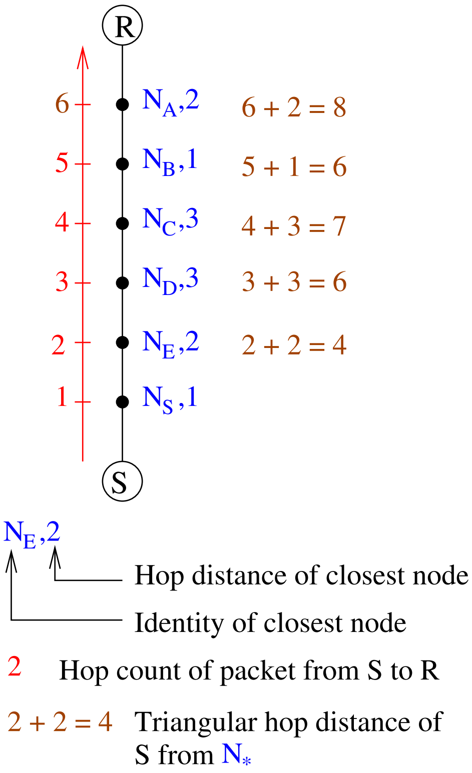


Fig. 5. Example of using ESP to calculate set of closest nodes

Hop-count	1	2	3	4	5	6	7	8	...
Node-count	0	0	0	1	0	2	1	1	...

If the sender is interested in determining the identity of three of its closest neighbors then it is easy to determine the range of hop-counts that an sf-gather ESP packet should look for. From the table we see that our three nearest neighbors are between four and six hops away (using triangulation). Thus in phase three, the sf-gather instruction collects the identity of nodes N_E, N_D and N_B which is then conveyed back to N_S.

One shortcoming of the ESP approach presented above is that each router only records information about the *closest* node; if multiple nodes are at the same distance from the router, only one will be recorded, and the others are effectively "hidden". As a result, the source may be grouped with nodes that are farther away than the "hidden" nodes. There are several ways to address this issue. One straightforward possibility is to modify the sf-deposit instruction so that routers record all nodes and their distances up to some maximum. The sf-range (and sf-gather) should also be modified to count (collect) multiple nodes at each distance. Another alternative is to modify the sf-deposit instruction to allow packets from nodes that are not the closest to be forwarded until they reach a router where their distance exceeds some maximum value. Still another optimization is to iterate the algorithm, each time using a different destination—preferably, topologically far away from the previous destination. In fact, if the sf-deposit instruction is modified to record the farthest as well as the closest sender, the destination to use on the next iteration can be automatically identified. By iterating the algorithm across a set of destinations, we improve the probability that a node discovers all of its nearby neighbors.

In the foregoing discussion we assumed that all routers supported ESP. However, a slight modification to the sf-deposit instruction allows the metric to be used when ESP is only partially deployed. In particular, the sf-deposit instruction can use the TTL field in the IP header[4] to accurately count the IP hops through non-ESP routers. This can be achieved at ESP-capable nodes by copying the TTL from the IP header into the ESP header before forwarding the packet. Thus the next ESP-capable node can compare the last known TTL (in the ESP header) with the current TTL (in the IP header).

B. Other Uses

We now briefly discuss the use of the set formation abstraction in determining solutions to other common set-formation problems.

Reliable Multicast: The basic idea is to identify the set of receivers behind the same lossy link and place them in a "retransmission group". The application level input is a lossprint—i.e., a bitmap representing packets where the bit is set if that packet was lost. The criteria for a set is based on the similarity of the lossprints. The network-level input is node proximity to one another (i.e., receivers that share links on the path to the multicast source) which is used to avoid forming sets containing nodes that do not share a common bottleneck. As described earlier, a triangulation approach can be used to estimate the distance between two nodes. Thus the network criteria would restrict the set to contain members within a certain distance of one another.

Overlay Node Selection: Overlay networks are often constructed from a set of "potential" overlay nodes. That is, the set of potential overlay nodes is larger than the set actually used. For example, the Abone[1] offers a large number of potential nodes, but not all Abone nodes offer all execution environments. As another example, Planetlab [4] offers a large number of potential overlay

[4]ESP instructions are permitted to read but not write fields in the IP header.

nodes, yet some nodes are colocated and only one of the colocated nodes is needed for the overlay.

In this case, our set-formation service can be used initially on application-supplied values (provided by the potential overlay nodes) to determine which nodes should be used in the overlay (i.e., have the desired characteristics). Having selected the nodes to participate in the overlay, the service described above can then be used to identify nearby nodes and construct the overlay topology.

Local/Global Overlays: Another approach to the design of overlays makes use of two types of connections: connections to nearby nodes and connections to far away nodes [14]. The goal is for half the connections to be made to nearby hosts and half to be made to randomly selected distant nodes. The motivation is based on the idea that short connections will help create pockets of nearby nodes and long connections will help achieve global connectivity between the pockets.

A slight variation of the sf-deposit instruction described in section IV-A can be used to find both nearby and far-away nodes. If the comparison operator is $\geq$, the instruction returns nearby nodes. If the comparison operator is $<$, the grouping mechanism returns the most distant nodes. Other possibilities to select nodes in between can also be created.

Hierarchical Overlays: TAG [12] uses traceroute measurements from all potential participants of the application to compute the path overlap and thereby determine a structured hierarchical order (a parent-child relationship) among the nodes. This structured information is passed on to the nodes which use it to construct an overlay tree.

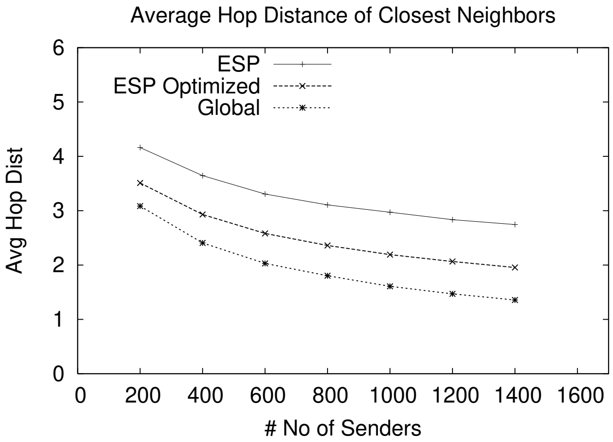
Our set-formation abstraction can be used to achieve a similar effect. ESP packets carrying the sf-deposit instruction can be used to establish state at each router indicating the closest node to that router. Each node then sends an ESP packet (similar to the gather probe described earlier) to identify the first router that it shares with other participants. The node listed as the closest node to that router is selected as the parent for all the other nodes, thereby creating a hierarchy similar to that of TAG. Thus with two simple operations this abstraction lets us achieve the formation of an overlay tree without having to make (costly) traceroute comparisons (i.e., each participant marks their route using a single ESP packet rather than the iterative probing required by traceroute).

V. Simulations

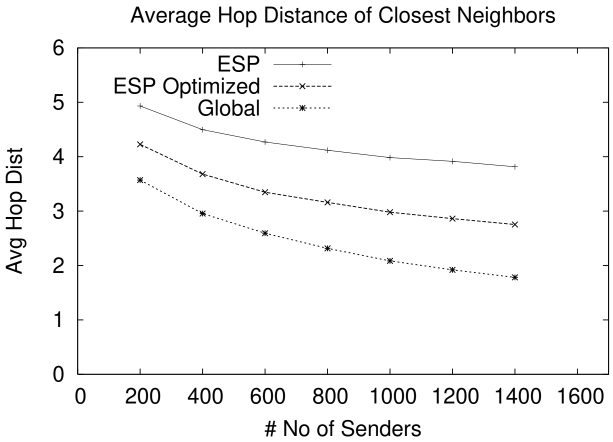
To measure the performance of our ESP-based implementation of the grouping criteria, we simulated the basic service described in the previous section (called "ESP" in the figures), along with an optimized version of the service, which allows discovery of all nodes at the minimum distance from each router, i.e. it avoids the "hidden node" problem described earlier (this version is called "ESP-Optimized" in the figures). We compared both of these approaches with an optimal solution derived from complete global knowledge (labelled "Global" in the figures). The figures also show how well each of the approaches scale as the number of participants increase.

In our experiments, we used a 2376 node GT-ITM [7] transit-stub graph with randomly-selected participant nodes from stub domains. The number of participants varied from 200 to 1400. The three solutions described above (ESP, ESP-optimized, and Global) were used to determine the set of closest neighbors for each node. We experimented with group sizes of 2, 4, and 8 nearest nodes. Each graph shows the average hop-distance a node is from the nodes it is grouped with. Each experiment was averaged over 10 runs using a different set of participating nodes in each run. Figure 6(a), Figure 6(b) and Figure 6(c) show the results of our simulation when searching for two, four and eight closest neighbors respectively.

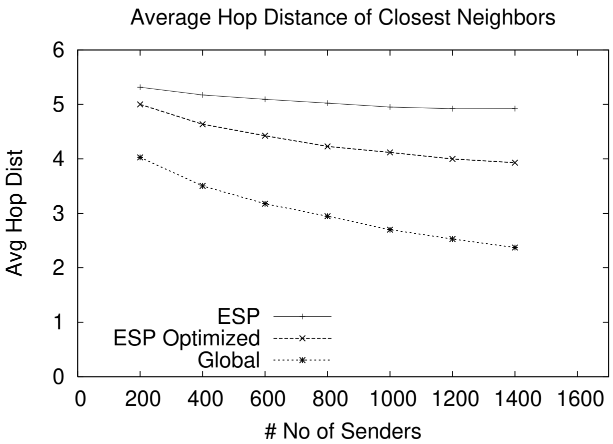
We observe that as the number of participants increased, the average hop-distance to the closest nodes decreases. This is consistent with the fact that the density of nodes within the stub and in the neighboring stubs increases as participation increases. We also observe that both ESP and ESP-Optimized are not unreasonably far from the optimal solution, and the general shape of the curves match that of the optimal solution. The graphs also show that the basic ESP implementation hides a substantial number of nodes that should be paired together. However, the ESP-Optimized solution does a reasonable job of finding these nodes. Because distances are approximated based on triangulation and some grouping combinations are never considered (due to the fact that decisions are made locally at each router), ESP-Optimized selects (on average) one non-optimal neighbor (i.e., a neighbor that was one hop farther away than the neighbor selected by the optimal solution). Given the large number of participants and potential neighbors, this is a relatively small error. Moreover, the number of messages used to compute the grouping is



(a) 2 Closest Nodes



(b) 4 Closest Nodes



(c) 8 Closest Nodes

Fig. 6. Average Hop Distance of Closest Neighbors

relatively low, with most ESP packets being dropped at one of the first few routers (because the sending node is not the closest).

VI. Related Work

The problem of assigning nodes to sets arises in a number of contexts and various problem-specific solutions have been developed. The recent popularity of self-organizing P2P network spurred much of this work, focusing on the problem of identifying sets of nodes that are close to each other in the network topology. To our knowledge, our work is the first to offer a general framework that combines topology-related criteria with application-specific criteria.

Ratnasamy et al have proposed a *landmark-based* scheme [14] that uses network delay measurements as the basis for determining proximity of clients. Each participant client measures the round-trip delay between itself and a set of landmark servers. It then generates a rank-ordering of the servers based on their observed delays. The idea is that two clients with the same rank-ordering are very likely to reside in the same vicinity of the network. This service could easily be used as a source of network-related information in implementing our abstraction.

The *beacons* approach of Shankar et al [18] uses designated measurement points to collect network-distance information. Each measurement point reports back to a client a list of "probably" close nodes; the list obtained from every measurement point can be compared to obtain the overlapping set of nodes that have the highest probability of being close to the probing node.

Topologically Aware Grouping (TAG) [12] exploits network measurements made by its members to construct an overlay tree that minimizes the network delay penalty and also minimizes the number of overlay links over the same physical link. TAG clients use `traceroute` to find a path to a common destination, and the traceroute data from all clients is compared to build an efficient overlay tree.

VII. Conclusion

We have proposed a generalized abstraction that helps nodes achieve dynamic set-formation and discovery. The input to this set-formation process can be application-specific values, network-specific values, or both. In addition, the application supplies a grouping criteria that is used to determine the optimal grouping for that application. Because the abstraction separates the policies/criteria that govern set formation from the underlying implementation, it is possible to adapt the abstraction to various underlying technologies. We have shown how the abstraction can be mapped onto three underlying services: a centralized server, a concast-based service, and an ESP-based service. We have also shown how the set-formation abstraction could be used to solve the problem of determining a node's closest neighbors. Our results illustrate the utility of the abstraction, but they also show that efficient implementations are possible and that the mechanism can produce groupings that are not far from the optimal grouping.

VIII. Acknowledgements

This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-1-0514, and by NSF Grant number EIA-0101242. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, or the U.S. Government.

References

[1] Active network backbone (abone). http://www.isi.edu/abone/.

[2] Napster. http://www.naptser.com.

[3] National Internet Measurement Infrastructure. http://www.ncne.nlanr.net/nimi/.

[4] Planetlab. http://www.planet-lab.org/.

[5] traceroute.org. http://www.traceroute.org/.

[6] K. Calvert, J. Griffioen, A. Sehgal, and S. Wen. Concast: Design and implementation of a new network service. In *Proceedings of 1999 International Conference on Network Protocols, Toronto, Ontario*, November 1999.

[7] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997.

[8] Kenneth L. Calvert, James Griffioen, and Su Wen. Lightweight network support for scalable end-to-end services. In *Proceedings of ACM Sigcomm*, August 2002.

[9] Clip2.com. The gnutella protocol specification ver 0.4, 2000. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[10] M. Handley and V. Jacobson. Sdp: Session description protocol, April 1998. RFC 2327.

[11] M. Handley, C. Perkins, and E. Whelan. Session Announcement Protocol, October 2000. RFC 2974.

[12] Minseok Kwon and Sonia Fahmy. Topology-aware overlay networks for group communication. In *the Proceedings of ACM NOSSDAV*, May 2002.

[13] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM'02*, June 2002.

[15] Sylvia Ratnasamy and Steven McCanne. Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements. In *the Proceedings of the 1999 INFOCOM Conference*, March 1999.

[16] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *18th Conference on Distributed Systems Platforms*, 2001. Lecture Notes in Computer Science 2218.

[17] Amit Sehgal, Kenneth L. Calvert, and James Griffioen. A Flexible Concast-based Grouping Service. In *the Proceedings of International Working Conference on Active Networks*, December 2002.

[18] Narendar Shankar, Christopher Komareddy, and Bobby Bhattacharjee. Finding close friends on the internet. In *the Proceedings of the International Conference on Network Protocols*, November 2001.

[19] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.