An Architecture for Internet Content Distribution as an Infrastructure Service

Yatin Chawathe, Steven McCanne, Eric Brewer

Abstract

The IP Multicast service model extends the traditional best effort Internet datagram delivery service for efficient multi-point packet delivery. However, in spite of a decade of research on multicast protocols and applications, a globally deployed multicast service is nowhere in sight, hindered by multitudes of problems such as manageability, lack of a robust inter-domain multicast routing protocol, scalability, and heterogeneity. In this work, we propose a new model for Internet multicast where we view multi-point delivery not as a network primitive but rather as an application-level infrastructure service. Our architecture relies on a collection of strategically placed network *agents* that collaboratively provides the multicast service for a session. Clients locate a nearby agent and tap into the session via that agent. Agents organize themselves into an overlay network of unicast connections and build data distribution trees on top of this overlay structure. This model effectively partitions the client set into a number of small data groups interconnected by robust unicast links. We call this communication model *scattercast* and the network agents that are central to this model *ScatterCast proXies or SCXs*. We present a protocol called *Gossamer* for grouping clients with SCXs and building an overlay mesh of unicast connections across SCXs. We demonstrate the efficacy of our architecture via a set of simulation experiments that show that the latencies incurred and redundant packet duplication in transmitting data over the scattercast mesh are low.

1 Introduction

The Internet multicast backbone, or MBone [7, 8] has been the research community's vehicle for efficient multi-point communication since its introduction ten years ago. IP multicast is an extension of the traditional best-effort Internet datagram model for efficient group-oriented communication where each source's data flow is delivered efficiently to all interested receivers according to a multicast distribution tree.

However, in spite of a decade of research on multicast protocols and applications, IP multicast is yet to take off. Although it has been available for research through the experimental MBone network, and has recently been implemented in many commercial routers, most ISPs are still reluctant to enable it in their domains. A number of crucial problems have impeded the global deployment of IP multicast. We summarize some of them below:

- In [9] and [21], the authors cite a number of problems that are inherent in the current IP multicast service model. These problems, including group management, lack of access control, absence of a good inter-domain multicast routing protocol, and distributed multicast address allocation, have proved to be a significant barrier to wide-spread commercial deployment of IP multicast.
- Moreover, the heterogeneity in the Internet makes it difficult to build multicast applications that can simultaneously satisfy the conflicting requirements of the wide range of client devices and networks that span the entire Internet.
- Finally, like IP unicast, the multicast service model provides only best-effort packet delivery. Richer services such as reliable, sequenced delivery and congestion control are relegated to higher transport or application layers. However, unlike in the unicast world where TCP addresses these issues for most applications, in the multicast domain, these problems are far more complex and much harder to address in the context of a single generic transport protocol

Recently, protocols such as IPv6, BGMP/MASC [22], and GLOP addressing [28] have attempted to address some of these issues. Researchers have also proposed changing the underlying multicast service model itself (EXPRESS [21] and Simple Multicast [31]) to better manage some of the above problems. However, none of these solutions address the crucial issues of heterogeneity, reliability, and congestion control, which remain a stumbling block for the success of multicast services. Moreover, as new protocols are invented to patch problems inherent in the multicast service model, the underlying network layer gets more and more complex.

One of the reasons for the success of the Internet is its simplicity and consequent robustness. In keeping with the principles of end-to-end design [34], the Internet was explicitly designed to leave the core network layer technology simple, robust, and easy to understand, and to migrate all complex services to higher layers. The unicast datagram forwarding service is easily amenable to this separation. On the other hand, we believe that the IP multicast service model is too complex to be implemented satisfactorily entirely as a network primitive. In this work, we thus distinguish between the notion of IP multicast as a network layer primitive and multi-point data delivery as a higher level network service. Rather than assume the existence of a global multicast "dial-tone," we view IP multicast as an efficient protocol building block that need not be available everywhere. We instead build multi-point delivery as an infrastructure service that leverages well-understood and robust *unicast* transport protocols and couples them with IP multicast for efficient multi-point data delivery. Separating multi-point delivery into a higher-level infrastructure service allows us to keep the network layer primitives simple and easy to manage.

Our architecture for Internet multicast partitions a heterogeneous set of session participants into disjoint data groups. Each data group is serviced by a strategically located network agent. A collection of network agents collaboratively provides the multicast service for a session. Clients locate a nearby agent and tap into the multicast session via that agent. Agents organize themselves into an overlay network of unicast connections and build data distribution trees on top of this overlay structure. We call this communication model *scattercast*[1] and the network agents that are central to this model ScatterCast proXies (SCXs). Figure 1 depicts the various components of the architecture.

Recently, researchers have proposed migrating the multi-point delivery functionality entirely to the end-clients that participate in the multicast session without any support from the network [13, 42]. Although the motivation for that work is similar to ours, we believe that without explicit support from the infrastructure, it is not possible to build practically deployable multi-point distribution systems that can scale well beyond a few hundred to a few thousand clients. In scattercast, each SCX can support many simultaneous clients, so even a session consisting of a hundred SCXs, each servicing a hundred clients, will result in a total session size of ten thousand. We believe that such infrastructure support is vital to the success of a multi-point delivery architecture and it needs to be an integral part of the architecture, rather than something that is patched in at a later time.

Just as the network layer Internet architecture provides a well-defined structure for IP routing and for peering of IP networks, so also this new scattercast service requires an infrastructure architecture that imposes structure on the peering model for SCXs and the interaction across SCXs, and between SCXs and clients. At the core of scattercast is a topology management protocol called *Gossamer* that SCXs use to locate each other in a decentralized manner and to self-configure themselves into an adaptive and efficient overlay mesh of unicast interconnections. SCXs run a variant of a distance-vector routing protocol on top of this mesh structure and effectively build reverse-shortest-path distribution trees.

By migrating the multicast service to higher layers, scattercast keeps the underlying network model simple and straightforward. Moreover, the problems that plague IP multicast are either eliminated or mitigated due to application-level intelligence. For example, there is no need for a global distributed IP-level multicast addressing scheme. Scattercast sessions have application-level names that are independent of the underlying network routing protocols. With the scattercast model, routers do not need to maintain complex group management state; this state is migrated to higher layer SCXs. Additionally, SCXs can impose application-specific access control restrictions to determine who is allowed to send or receive data in the session.

By explicitly using application-level agents in the network, scattercast also allows for a scenario where SCXs can use application semantics to adaptively modify the content in order to suit the needs of the clients. This property of scattercast is very useful to tackle the heterogeneity that plagues IP multicast applications and to build complex services such as reliability and congestion control on top of this architecture. In [5], the authors leverage the scattercast architecture to provide such application-specific reliable multicast service in the face of extreme heterogeneity. In Section 4, we provide an outline of how our architecture allows us to build such complex services and applications.

We note that Gossamer is certainly not the only self-configuration protocol that is possible for scattercast. It is the result of our initial experimentation with building the various components of the scattercast architecture. Although scattercast simplifies the network model by migrating complex multicast protocols to higher layers, it suffers from the drawback that its data distribution trees are not as efficient as native router-supported multicast. Yet, our architecture strives to build an efficient overlay mesh so that the resulting performance hit for the data distribution trees is minimal. Our simulation experiments demonstrate that the average delay from the source to receivers in scattercast is typically within twice that for native multicast or direct unicast from the source to the receivers.

In the rest of this paper we describe the service model for our architecture and discuss the design of the various components of the architecture. Section 2 describes the details of our architecture. Section 3 describes the Gossamer protocol. In Section 4, we discuss how our architecture allows us to build more complex services such as reliable delivery and congestion control on top of this framework, and describe example applications. Section 5 presents an evaluation of the Gossamer protocol and the status of our implementation. Finally, we summarize some related work and present future work and our conclusions.

2 The Scattercast Architecture

The scattercast architecture embeds in the network a collection of agents that together provide the scattercast service. Figure 1 illustrates the various components of the architecture. Clients (sources or receivers) wishing to participate in a scattercast session communicate with a nearby SCX and tap into the session via that SCX. SCXs self-configure themselves into an overlay structure for data distribution across the wide area.

2.1 The Service Model

Each scattercast session has an explicit URL-like unique name. The name is used to identify the session and to distinguish between sessions. Session names are of the form `scattercast://creator-identity/session-name`. The creator identity is used to avoid collisions in the session name-space. The simplest form of creator identity is the domain name of the agency that creates the session. For example, a multimedia seminar announcement may have the name `scattercast://cs.mydomain.edu/multimedia-seminar/`.

[1]The term *scattercast* is borrowed from prior work by Ratnasamy et al.[32] on a delivery-based model for multicast communication.

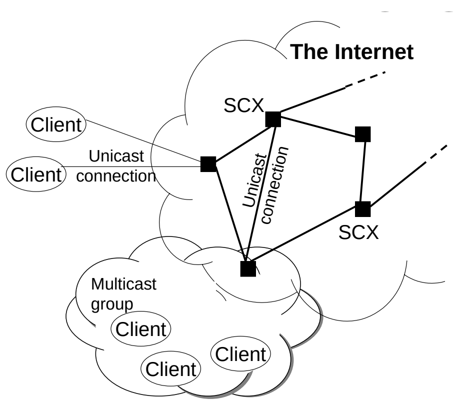

Figure 1: **The scattercast architecture**. Clients communicate with SCXs either via locally scoped multicast groups or via unicast. SCXs form a mesh of unicast interconnections between themselves.

A single scattercast session may consist of multiple independent data streams, each with its own transport requirements. For example, some streams may require reliable data delivery while others may be satisfied with best-effort performance. Applications may also wish to split different media types into separate data streams. For example, a real-time Internet broadcast may be composed of a video and an audio stream. Scattercast uses the notion of *data channels* to separate such independent streams. Each data channel has an associated well-known numeric identifier. When sources transmit data packets, they include the channel identifier in the packet header. Scattercast uses this identifier to route the data over the appropriate channel. This notion of multiple channels within the same scattercast session allows us to reuse the same scattercast overlay network for different types of data streams.

The scattercast service model requires both sources and receivers to explicitly join the session. Moreover, sources must explicitly announce their intent to send data. The underlying scattercast protocols use this information to build efficient source-rooted data distribution trees.

Each client (source or receiver) attaches to a nearby SCX and interacts with the rest of the session via that SCX. Each SCX, in turn, simultaneously serves many clients. As shown in Figure 1, clients communicate with their SCX using multicast if possible, otherwise they revert to unicast connections to the SCX. Across SCXs, data is transmitted using an overlay structure of unicast connections. These data transport connections may be UDP, TCP, or some other unicast transport protocol depending upon the requirements of the channels for that session.

Typically, SCX lifetimes are limited to those of their clients. An SCX is created on demand for a specific session and it dies when all of its clients leave the session. We note that although clients may join and leave a scattercast session at a rapid rate, since an SCX serves a number of clients, it remains part of the session as long as it has at least one client to serve. We assume that, in general, SCXs join and leave a scattercast session at a relatively slow rate.

As described above, the scattercast service model results in a two-tiered communication model—communication between clients and SCXs, and inter-SCX communication. In the rest of this section, we address some of the issues raised by this model. In particular, we look at the following questions:

- How do clients discover scattercast sessions?
- What is the environment that SCXs run in?
- How do clients locate a "nearby" SCX?
- How do clients attach to the scattercast session via the SCX?

We leave the discussion of the details of the inter-SCX communication to Section 3.

2.2 Scattercast Announcements

```
<SCATTERCAST
     name="scattercast://creator-identity/session-name">
  <DESCRIPTION>
       An optional textual description of the session
       goes here
  </DESCRIPTION>

  <CHANNEL id="numeric-identifier">
       <DESCRIPTION>
           An optional textual description for this channel
       </DESCRIPTION>

       <TRANSPORT unicast="unicast-protocol-name"
              multicast="multicast-protocol-name"/>
  </CHANNEL>

  <CHANNEL id="numeric-identifier">
       ...
  </CHANNEL>

  <RENDEZVOUS>
       rendezvous-point-location
       ...
  </RENDEZVOUS>

</SCATTERCAST>
```

Figure 2: **Format of a scattercast announcement**.

Scattercast sessions can be advertised either on the web or using a special well-known scattercast session in a manner similar to the Session Announcement Protocol (SAP) [19] used on the MBone. Scattercast announcements are represented using the Extended Markup Language (XML) [3]. The announcement contains all the necessary parameters that pertain to the session. Figure 2 shows the format of a scattercast announcement. Each announcement must include the name of the session, one or more <CHANNEL> sections, and a <RENDEZVOUS> section. As described in Section 2.1, each <CHANNEL> section includes the numeric identifier associated with the channel and the types of transport protocols that the channel should use for communication across SCXs and between clients and SCXs. Unicast protocol descriptors may be UDP, TCP, or some other unicast transport protocol, while multicast protocol descriptors may be UDP or some reliable multicast protocol such as SRM [11]. The <RENDEZVOUS> section lists one or more *rendezvous points* that SCXs use to find each other. We discuss the details of the rendezvous mechanisms in Section 3.4.

2.3 SCX Environment

In order for the scattercast service to be viable, it is crucial to address the question of where SCXs reside and what conditions they operate under. As an infrastructure service, SCXs must remain highly available and robust against failures. To address these issues, we rely on strategically located service *clusters* for hosting SCXs. These clusters consist of commodity workstations typically housed at ISP points of presence. Clusters are an efficient and cost-effective way of providing robustness and availability to the scattercast service. We assume the existence of a cluster management platform that provides the function of creating SCXs when required and ensuring that the SCXs remain available and recover from faults. Various such cluster service platforms have been proposed in the research community [1, 4, 12, 16]. We rely on the Active Service platform for hosting SCXs. The details of this cluster platform can be found in [1]. For the purpose of this discussion, it is sufficient to note that the cluster platform deals with the actual details of launching SCXs when required, monitoring them for faults, and recovering from failures when necessary.

2.4 Locating an SCX

In order for clients to receive data from the scattercast session in an efficient manner, it is imperative that they attach themselves to an SCX that is close to them. With a potentially large number of SCX-capable cluster platforms spread across the Internet, clients need a way to locate the closest SCX. This is a well-studied research problem, and we identify a few solutions:

Static configuration: Clients may be statically configured with the location of their closest cluster platform. This mechanism is simple to implement, but does not permit automatic discovery of new nearby cluster platforms.

Auto configuration: A modification to the static configuration scheme is to use a statically configured DNS name to identify the local cluster platform (e.g. `scattercast.mydomain.edu`), or a script akin to web-proxy auto-configuration scripts [26]. The WPAD (Web Proxy Auto Discovery) Draft [15] describes a number of mechanisms for discovery of network services based on DHCP [10], SLP [38], or DNS queries [18, 17]. These mechanisms do not require the client to know the exact names of the cluster platform machines, but still do not account for dynamic network changes.

Transparent DNS redirection: A more sophisticated approach relies on using special DNS names that are resolved differently for different clients based on the clients' location. This approach is used by the Sandpiper Networks' Footprint web caching service [35].

The basic idea is to construct a *redirection framework* that manages a special DNS domain, say `redirect.scattercast.net`, and resolves client queries for that domain into an address for a scattercast cluster that is closest to the client. The redirection framework is composed of an elaborate network of probes that monitor the Internet building a real-time network map that identifies the delays between different parts of the network. Using this map, the redirection framework can easily identify the closest scattercast cluster for any client. Thus the client always manages to find a nearby cluster without any pre-configuration.

Explicit application-level redirection: Using DNS resolution for redirecting clients to appropriate clusters can be plagued by problems due to clients caching stale addresses. Old clusters may no longer be offering the scattercast service, new clusters may have cropped up that are closer to the client, or network conditions might have changed. This can be addressed by using an explicit application-level redirection mechanism such as that used by HTTP.

Although the redirection framework approach for locating scattercast clusters is superior, our prototype scattercast implementation relies on static client configuration. The Sandpiper Footprint service [35] has implemented a proprietary system that includes a well-designed redirection sub-system, and a practical deployed scattercast architecture should utilize that work.

2.5 Client Attachment

Once a client has discovered the nearest scattercast service platform, it contacts the cluster and makes a request for an SCX. The request includes the session announcement for the session that the client is interested in and an indication of whether the client is a source of data or not. The cluster creates a new SCX if needed and returns the location of the SCX, including a unicast IP address and port number as well as a locally scoped IP multicast group that can be used if multicast connectivity is available between the client and the SCX. The client initially attempts to communicate with the SCX over the IP multicast group, but reverts to unicast communication if that fails. This allows us to leverage the efficiency of IP multicast in the local domain when it is available.

As long as the client is part of the session, it sends periodic KEEP_ALIVE messages to the SCX. It announces its imminent departure via an AM_LEAVING message. The SCX uses this message (or the loss of KEEP_ALIVE messages) as an indication of the client's death. When all clients of the SCX have left the session, the SCX too leaves the session.

3 Gossamer: Inter-SCX Communication

In addition to communication between clients and SCXs, a crucial part of the scattercast architecture is the set of mechanisms that SCXs use to construct an application-level overlay distribution network, and to transmit data on top of this overlay structure. We now present Gossamer, our protocol for constructing and maintaining this overlay topology. The goal of Gossamer is to build an efficient data distribution tree from the source of data. The simplest way of distributing data across SCXs is to construct a unicast star topology rooted at the source SCX (i.e. the SCX to which the source of data is attached). This simplistic approach however has the danger of resulting in excessive network load near the source. With a star topology, the source SCX will simply perform an n-way unicast transmission of the data to all destination SCXs. Since each packet is duplicated multiple times at the source SCX, the bandwidth requirements on the physical Internet links near the source SCX can be excessive. In order to limit the amount of duplicate packets traversing any physical link across the network, Gossamer should build smarter distribution trees where the source SCX transmits data only to a handful of nearby SCXs which in turn forward the data towards the rest of the session. In other words, Gossamer distribution trees should restrict the degree of any single SCX node depending upon its bandwidth capabilities. We note however that such a degree-restricted tree will result in longer delays for certain SCXs than the corresponding delays in the original unicast star. The goal of Gossamer then is to build a degree-restricted spanning tree across SCXs while at the same time keeping the average delay between the source and all destinations at a minimum.

We can define the problem more formally as follows:

GIVEN: A set of Internet nodes V that represent SCXs participating in a scattercast session, a source SCX $s \in V$, and node degree constraints $k_i (\forall v_i \in V) \geq 2$. We can build an *abstract distance graph* $G = (V, E)$ that is the complete graph over the set of SCX nodes. The cost of edge $\{v_i, v_j\} \in E$ is set to the unicast distance between nodes v_i and v_j (assuming shortest path symmetric Internet routing).

FIND: A *distribution tree* T, which is a spanning tree of the graph G such that d_i, the degree of node $v_i \in V$ in T is at most k_i, and T's total cost C is the minimum[2] among all possible such trees, where C is defined as the sum of path lengths in T between the source s and all other nodes.

The problem of constructing minimal degree-constrained spanning trees of graphs is known to be NP-hard [14]. Moreover, the problem remains NP-hard even in the specific case of complete graphs. Hence we need to rely on heuristics to solve the above problem. However, it is difficult to compare the performance of the heuristic approach to the optimal solution, since computing the optimal tree is prohibitively expensive. But, we do know that the cost of the optimal tree is bounded by the cost, C_{star}, of the corresponding unicast star topology, T_{star}, rooted at the source s. We can use this bound as a metric for evaluating the performance of the heuristic approach. We also note that, in terms of path lengths, this cost C_{star} is equivalent to the cost incurred for a source-rooted IP multicast routing tree (assuming shortest-path and symmetric internet routing).

3.1 A Practical Topology Construction and Management Algorithm

In order to be practically deployable, any heuristic that we develop must be entirely decentralized and must be able to cope with a dynamically changing membership of the set V. The ultimate goal of Gossamer is to construct spanning trees for data distribution. Although it is possible to construct such a tree across SCXs directly, Gossamer instead first builds a richer mesh structure made up of unicast connections across SCXs, and on top of this mesh, runs a routing protocol to compute source-rooted reverse shortest path distribution trees. The reasons for this are two-fold. First, the mesh provides redundancy to the scattercast topology, making it more resilient to failures than a simple fragile tree structure. If an edge or node in the topology fails, the routing algorithm automatically constructs new paths by routing around the failure. Second, routing algorithms have built-in mechanisms to deal with detection and avoidance of loops in the distribution paths. This makes construction of loop-free distribution trees much simpler.

[2]Note that minimizing the total cost C is equivalent to minimizing the average delay.

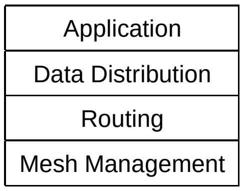

Figure 3: **Gossamer Protocol Layers**.

Since the data distribution trees have degree constraints, we impose similar constraints while constructing the mesh. This ensures that the shortest path trees that are built by the routing and data distribution protocols on top of this mesh automatically satisfy the degree constraints.

Figure 3 shows the different layers involved in the Gossamer protocol. At the bottom is a mesh management layer that deals with the basic topology construction and maintenance. The routing layer runs a distance vector routing protocol on top of the mesh and provides input to the mesh management layer in order to assist in optimizing the mesh, and as a consequence, the paths between the sources and the receivers. The data distribution layer constructs distribution trees based on routing information extracted from the routing layer and deals with the forwarding algorithms that are used to disseminate the data. Finally, any application-specific computation may be performed on top of the Gossamer layers.

We now look at the details of the Gossamer protocol starting with a brief overview of the entire protocol.

3.2 Protocol Overview

When an SCX joins a session, it uses a variant of a network resource discovery protocol proposed by Harchol et al. [20] to discover other mesh members. It uses the set of *rendezvous points* listed in the session announcement to bootstrap the discover process. As the SCX encounters new nodes, it selects some of them to be its neighbors in the mesh. As defined by the degree constraints, each SCX has a target number of neighbors that it attempts to connect to in the mesh. In order to ensure that nodes can insert edges in the mesh without requiring any explicit coordination across nodes, we split the degree constraint at each node into two: a maximum number (k_1) of edges that the node is allowed to insert from it to other nodes, and a maximum number (k_2) of edges that it is willing to accept from other nodes. As long as we ensure that $k_2 > k_1$, any new SCX node joining the mesh will eventually find some k_1 nodes that have room to accept connections from it. We use the notation $<k_1,k_2>$ to represent these degree constraints, effectively resulting in a total degree constraint of $k_1 + k_2$.

Rather than pick a random set of k_1 neighbors, each node uses a local optimization algorithm to choose neighbors that will result in better distribution trees. The trees themselves are constructed by running a distance vector routing protocol on top of the mesh topology. Each node maintains a routing table with an entry for each source SCX. The data distribution layer uses this routing information to construct source-rooted reverse shortest path distribution trees.

We now look at some of the details of the mesh construction and tree building algorithms.

3.3 Node Discovery: Name Dropper

When an SCX joins a session, it uses a variant of the Name Dropper protocol proposed by Harchol et al. [20] to discover other SCX nodes. We note that it is not required for SCXs to have complete and accurate information of mesh membership at all times. An SCX (say X_i) initiates the Name Dropper algorithm by discovering a small set of mesh members through some startup rendezvous mechanism. Let us denote by $\Gamma(X_i)$ the set of other SCXs that X_i knows of. Periodically, X_i performs a discovery round. During this round, it picks a random node $X_j \in \Gamma(X_i)$ and sends a DISCOVERY message to it. This message includes a bounded random list $\gamma(X_i) \subseteq \Gamma(X_i)$. When X_j receives the message, it merges this list into its own set $\Gamma(X_j)$ of known nodes. In addition, it returns a DISCOVERY_RESPONSE message that includes its own list $\gamma(X_j) \subseteq \Gamma(X_j)$. X_i in turn merges this list into its own set $\Gamma(X_i)$, and thus gradually learns of all or most of the other nodes in the system.

Our discovery algorithm described above differs from the original Name Dropper proposal in two ways. The original algorithm transmits membership information in only one direction ($\gamma(X_i)$ sent to X_j) during a round. By incorporating an exchange in both directions, we allow for a newly joining X_i to quickly discover a number of other SCXs. This, however, comes at the cost of increased communication cost. The second difference is that, to minimize communication overhead, we limit the size of the lists exchanged at each round. We have not analyzed the effects of this bounded list size on the performance of the Name Dropper algorithm. However, in practice, our simulation results described in Section 5.2 demonstrate the practical usability of the algorithm.

3.4 Rendezvous

The Name Dropper algorithm assumes the existence of a bootstrapping rendezvous mechanism to initiate the discovery process. We rely on well-known *rendezvous points* for this purpose. Each scattercast session has associated with it one or more rendezvous points that are advertised in the session announcement. These rendezvous points are SCXs that remain alive and are part of the session for the entire duration of the session. When a new SCX joins the session, it initializes its mesh membership set $\Gamma(X)$ to the list of rendezvous points for the session. Using the Name Dropper algorithm described in the previous section, it can eventually discover all the other nodes in the mesh.

The redundancy introduced by multiple rendezvous points ensures that new SCXs can join the mesh even in the face of rendezvous point failure. We note, however, that even if all rendezvous points in the session fail, existing mesh members can continue to operate; the only functionality that is lost is the ability for new SCXs to join the session.

3.5 SCX Leaves

When an SCX leaves the session, it floods a time-stamped notification to the rest of the mesh. This allows the remaining SCXs to remove this node from their membership set $\Gamma(X)$. The departing SCX sends the notification to its immediate mesh neighbors who in turn propagate it to the rest of the session. Since the mesh is not loop-free, SCXs use the time-stamp in the notification to detect duplicate copies of the notification and stop forwarding the copies. In addition to leaving a session explicitly, an SCX may fail without any warning. In such a situation, its neighbors in the mesh will detect the failure and notify the rest of the session. To detect neighbor failure, neighboring nodes in the mesh exchange periodic KEEP_ALIVE messages. Loss of these messages is an indication of failure. Upon receiving an SCX leave/failure notification, other SCXs mark that SCX as dead in their membership list, and trigger updates in the routing layer.

It is possible that the death of an SCX causes the mesh to be partitioned. Although such an occurrence will be rare, it must be dealt with and the mesh repaired. In order to detect mesh partitions, we rely on a periodic HEARTBEAT that is generated by one of the rendezvous points and propagated over the mesh. The rendezvous points run a simple distributed election algorithm and pick one of themselves as the heartbeat generator. As long as every SCX in the session continues to receive this heartbeat, the entire mesh is connected. Loss of heartbeat messages indicate a potential mesh partition, and the SCX that detects the loss attempts to heal the partition by re-contacting the heartbeat generator. It is possible that a large number of SCXs that are partitioned from the heartbeat generator detect the partition at the same time. To prevent all of them from contacting the heartbeat generator simultaneously, we use a randomized damping interval before the SCX attempts to heal the partition. In the event that the heartbeat generator itself has died, the remaining rendezvous points elect a new heartbeat generator and the healing process continues.

3.6 Mesh Optimization

So far we have not discussed any mechanisms for ensuring that the mesh constructed by Gossamer actually results in efficient distribution trees. Let us now look at some of the algorithms that Gossamer employs to optimize the mesh over time. In this discussion, we assume that the routing layer runs a limited form of distance vector routing, where the routing table contains only a small number of entries: one for each source of data.

The goal of the mesh optimization algorithm should be to improve the quality of the mesh. Since the eventual goal of Gossamer is to build efficient data distribution trees, the optimization algorithm should attempt to add edges that will result in an effective improvement of the routes towards the sources of data and remove edges that are not as useful.

SCXs periodically probe other mesh nodes to evaluate the usefulness of adding new edges. A node X_i probes another node X_j using a REQUEST_STATUS message. The STATUS response

```
optimize($X_j$) {
    Let $\Lambda$ = (set of neighbors of $X_i$) $\cup$ $X_j$
    For each $X \in \Lambda$ {
        Let C.F.[$X$] = compute_cost_function($X$)
    }
    Let $Y (\in \Lambda)$ = node with maximum C.F.
    Let $H$ = hysteresis value
    If ($Y == X_j$) then
        reject $X_j$
    Else if (C.F.[$Y$] $-$ C.F.[$X_j$] $> H$) then {
        accept $X_j$
        reject $Y$
    }
    Else reject $X_j$
}
```

Figure 4: **Algorithm used by X_i to determine whether to accept X_j as a neighbor**.

```
compute_cost_function($X$) {
    Let C.F.[$X$] = 0.0
    For each source $S$ in $X_i$'s routing table {
        Let C.F.[$X$] $+=$ normalized cost$^*$ of routing to $S$
                    via $X$
    }
    If ($X_i$'s routing table is empty) then {
        Let C.F.[$X$] = normalized cost$^*$ of the edge
                    between $X_i$ and $X$
    }
    Return C.F.[$X$]
}
```

***Note:** Normalized routing cost is defined as the cost of the route to S via X divided by the maximum of the corresponding such costs for all $X' \in \Lambda$ (see Figure 4 for definition of Λ). Similarly the normalized edge cost is defined as the ratio of the cost of the edge between X_i and X to the maximum of the corresponding edge costs for all $X' \in \Lambda$. We note that the normalized cost is always a value between 0.0 and 1.0.

Figure 5: **Algorithm used by X_i to compute the cost function for node X.**

from X_j contains a copy of the current routing table of X_j and a CAN_ACCEPT flag that indicates whether X_j has room to accept a connection from X_i. The state of this flag depends upon whether X_j has reached its limit, k_2, of connections it is willing to accept from other nodes. X_i uses this status information to evaluate the usefulness of X_j as a neighbor over its current set of neighbors.

If X_i has not yet filled its limit k_1 of edges it is allowed to add, it will accept X_j as a neighbor. But, if X_i already has k_1 neighbors, then in order to accept X_j, it will have to remove one of its existing neighbors. X_i runs an optimization algorithm that evaluates the "cost" of all of its neighbors and X_j. In order to realize efficient data distribution trees, the mesh needs to be optimized for efficient routes from receivers to source SCXs. The cost function takes this into account and computes the cost of routing to the various sources via the individual neighbors. Figure 5 shows the cost function used as input to the optimization algorithm which itself is described in Figure 4. The SCX will accept X_j as a neighbor only if X_j's cost function is less than that of one of its existing neighbors by at least H. H is a hysteresis value that allows us to trade off the stability of the mesh versus the level of optimization. A higher value of H will result in fewer changes to the mesh structure, but may result in a less efficient mesh. After preliminary experiments, we have set the hysteresis value to 0.15 times the number of known source SCXs.

3.7 Routing Layer

On top of the mesh, the routing layer runs a variant of a distance vector routing protocol. Sources announce their intent to send data to their SCX. The source SCX immediately creates a zero-length routing table entry to itself in its local routing table. This entry gets advertised to the rest of the session via periodic routing update messages that neighboring SCXs exchange with each other. Each SCX maintains a routing table that contains one entry per source SCX. In order to detect routing loops and avoid the counting-to-infinity problem [6], each SCX stores in its routing table entries the complete path from it to the source SCX. When a node attempts to select a better route based on a routing update received from a neighboring SCX, it first checks to ensure that it is not already part of the neighbor's path, thus avoiding routing loops.

The routing protocol relies on unicast distances between nodes as the metric for the routing protocol. Each node in the mesh runs a simple *ping* experiment to determine its distance to its mesh neighbors. A ping experiment consists of a small sequence of time-stamped packets that the node sends to its neighbor. When a neighbor receives the ping packets, it simply reflects them back to the sender. The sender uses the average time difference between sending the packets and receiving the responses to compute the round-trip times and thus the one-way distances.

3.8 Data Distribution

Gossamer uses the routing tables generated by the routing layer to construct source-rooted reverse shortest path data distribution trees. The trees are built out of an independent set of transport connections that are separate from the control connections used by the mesh construction and routing protocols. A separate tree is constructed for each channel in the session. The session announcement specifies the form of transport connections that each channel uses.

Data forwarding occurs as follows. Data is forwarded at the Application Data Unit (ADU) level. Applications define their own notions of packet boundaries, and all data forwarding in scattercast respects these ADU boundaries. Each ADU consists of a Gossamer header that identifies the source of data, the source SCX, the numeric channel identifier, and the length of the ADU payload. At each node X_i, when an ADU is received on a channel from a neighbor X_j, it is forwarded only if X_j is the next hop in X_i's route towards the source SCX. Every ADU that passes this reverse-path check is forwarded to all those neighbors that use X_i as their next hop for routing towards the source SCX.

Transient changes in the distribution tree due to routing updates may result in temporary disruption of data flow. To minimize any data loss during a route change, data continues to be forwarded along the old route for a short while until the downstream SCX starts receiving data along the new route.

4 Building Services on Top of Scattercast

Scattercast provides the basic mechanisms to enable multi-point communication in the wide area. Our architecture alleviates many of the problems associated with IP multicast. By using a URL-like naming scheme for scattercast, we eliminate the need for a globally distributed network layer multicast addressing scheme. Since scattercast sessions include the identity of the creator as part of the session name, name collisions are trivially avoided. Scattercast also eliminates any need for complex per-group state maintenance at routers, and instead migrates this state to application-level SCXs where it can be more easily handled. Additionally, with application-level intelligence in SCXs, the scattercast model allows SCXs to implement application-specific access control restrictions to determine who is allowed to participate in the session and to send data within the session.

Additionally, the presence of application-level agents makes it possible to build more complex higher-level services such as reliability, congestion control, and heterogeneous communication. Although a number of protocols such as RMTP [23], SRM [11] and PGM [36] have been proposed to build reliability on top of the best-effort IP multicast service, they are all fundamentally challenged by the heterogeneity that exists across the Internet. In the multicast domain, a communication source is potentially confronted with a wide range of path characteristics to each receiver, for example, different delays, link rates, packet losses, and competing congestion on the paths to the different receivers. This multiplicity of data paths and the possibility of multiple congestion points along independent sections of the paths imposes great difficulty on the design of an end-to-end scheme for reliable multicast. TCP-friendly multicast congestion control schemes [39, 40] typically only work with single-source sessions, and do not satisfactorily accommodate bandwidth heterogeneity across the multicast distribution tree.

The scattercast architecture explicitly allows applications to address these problems of heterogeneity and congestion control that cripple traditional reliable multicast protocols. Rather than rely on traditional notions of bit-level reliability, scattercast allows for the notion of semantic reliability, that is, reliability of information rather than that of the representation of the information. SCXs can use application-level knowledge to alter the content dynamically or to adapt the rate and ordering of data objects. For example, an SCX that feeds data down a bandwidth-constrained link may convert bandwidth-intensive data such as images or video to lower bit-rate versions before transmitting them down the constrained link. Moreover, scattercast can leverage the robust and congestion-friendly behavior of well-known unicast transport protocols such as TCP to assist in wide-area inter-SCX communication. SCXs can provide intelligent congestion management via techniques such as buffering, on-the-fly transcoding to a lower bit rate, or explicit congestion notifications to upstream SCXs to slow down their transmission rates.

In [5], the authors describe a framework for providing reliable multi-point communication based on the scattercast architecture. As described above, they rely on application-specific customization of SCXs to assist in the reliability protocol. They refer to the scattercast proxies used for reliable communication as *Reliable Multicast proXies* or *RMXs*. The details of how SCXs/RMXs are customized on a per-application basis to provide application-specific reliability are described in [5]. RMXs implement end-to-end reliability on top of the scattercast framework using a PGM-like mechanism [36]. Loss recovery is initiated by sending a retransmission request upstream towards the source. Intermediate RMXs aggregate retransmission requests. They first attempt to recover the data themselves, and if that fails forward the request towards the source. The authors describe the details in [5].

4.1 Scattercast Applications

To illustrate the viability of the scattercast architecture, we are investigating a range of applications. We now look at two specific applications that we are building on top of the scattercast architecture: a reliable shared electronic whiteboard, and an Internet audio broadcast application.

The shared electronic whiteboard allows a diverse set of media to be created and displayed interactively by a group of users. Our whiteboard application is based on similar previous tools such as *wb* [27] and *mediaboard* [37]. A whiteboard session consists of a shared presentation space that is divided into a number of canvas pages. It supports data types such as line drawings, text, images, and postscript files. Each data object on the whiteboard is encoded and transmitted as an independent ADU. The application uses the Scalable Reliable Multicast (SRM) protocol [11] to achieve reliability in the local multicast groups between multicast-capable clients and their SCXs. It relies on TCP for unicast data transmission across SCXs and between multicast-incapable clients and their SCXs. As the ADU flows through the SCX network, it may be transformed on the fly in order to mitigate the effects of heterogeneity across the range of participating clients. For example, an image may be transcoded to a lower bit-rate representation for faster transmission across a low bandwidth link. We convert images to a progressive JPEG representation; this allows SCXs to transmit each scan of the progressive image independently, and to decide how many scans to transmit and how fast to transmit each of them. The details of the whiteboard application and the data transformations that can be applied to the whiteboard ADUs are described in [5]; we port their work on RMXs for whiteboards to our architecture.

The second application that we are building is an Internet audio broadcast tool. We use MP3 (MPEG 1 or 2 Layer III Audio) as the underlying audio format. The source broadcasts MP3 frames to the entire session through its SCX. Users wishing to "tune" into the broadcast use standard MP3 clients such as mpg123 [29] or WinAmp [30]. SCX cluster platforms export an HTTP interface to the MP3 clients. A client tunes to a specific broadcast by contacting its local cluster platform and including the web address of the broadcast's session announcement. The cluster platform in turn redirects the client to the appropriate SCX for that broadcast. SCXs too export an HTTP interface through which the MP3 frames are streamed to the clients. If the client is capable of using multicast to receive MP3 broadcasts (e.g. WinAmp with a multicast plug-in [25]), it may directly communicate with its SCX using a locally scoped multicast channel.

5 Evaluation

In this section, we evaluate the behavior of our architecture with respect to the mesh structure and the data distribution trees that Gossamer produces. We rely on simulation experiments to evaluate the operation of our protocol. The main metric that we use for evaluation is the total cost C of routing from a source over the Gossamer distribution network. As described in Section 3, C is defined as the sum of path lengths in the distribution tree between the source SCX and all other SCXs. The length of each path in the tree is the sum of the unicast distances between the pairs of nodes that make up the path. We compare this cost to the cost C_{star} of routing over the unicast star topology from the source SCX to all other nodes, that is, the sum of the unicast distances between the source SCX and all other nodes. We use the *cost ratio* $\frac{C}{C_{star}}$ as our performance metric.

5.1 Simulation Setup

We implemented Gossamer in a simple protocol simulator. The simulator implements the Name Dropper discovery algorithm, the mesh optimization algorithms, and a distance vector routing protocol for building data distribution trees. The input to the simulator is an internet topology generated using the Georgia Tech Topology Generator [41]. We used the Transit-Stub model to generate our experimental topologies. Each topology consisted of 1000 nodes

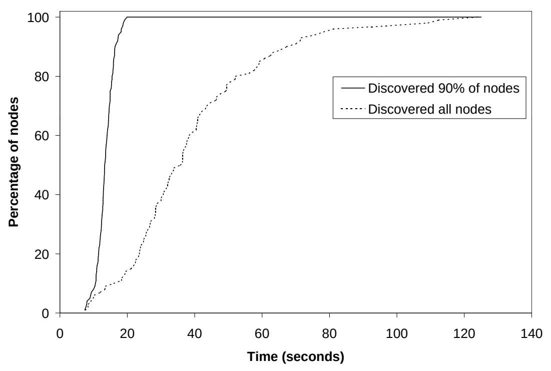

Figure 6: **Percentage of nodes that have discovered (a) at least 90% of the other nodes, and (b) all of the other nodes v/s time**.

The simulator assumes shortest path internet routing and accordingly computes unicast distances between nodes in the topology. Some of these nodes are selected at random as SCX nodes and the Gossamer protocol is run across these nodes. The simulator does not take into account the effects of any cross traffic and queueing delays on the behavior of the protocol. In the next few sections, we present the results of our experiments to evaluate the performance of Gossamer in a range of environments. In each of our experiments, there is a well known rendezvous SCX. All remaining SCXs join the session at a random instant within the first five seconds of the experiment. Unless mentioned otherwise, each session consists of a hundred SCXs and one randomly chosen source SCX, and each SCX has a node degree constraint of <3,4>. SCXs execute the Gossamer algorithms every 5 seconds.

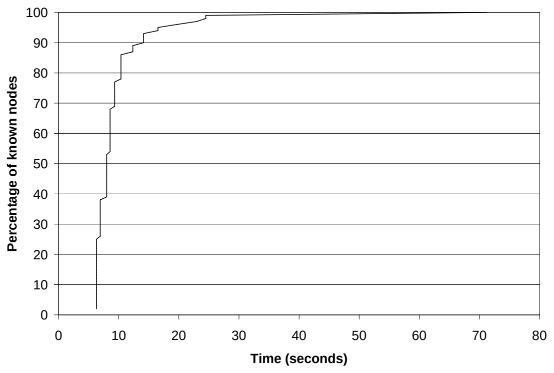

Figure 7: **Percentage of nodes discovered by a single SCX v/s time**.

The routing layer performs routing updates every 30 seconds. The experiment ends when there are no changes to the mesh structure for at least 100 seconds.

5.2 Name Dropper Performance

Figures 6 and 7 depict the performance of the Name Dropper resource discovery algorithm during an experiment consisting of 100 SCXs. In our experiment, we restricted the size of the membership set $\gamma(X)$ exchanged during each DISCOVERY round to 30. From Figure 6, we see that all nodes discover at least 90% of the rest of the nodes within the first 20 seconds of the experiment. Over 85% of the nodes discover everyone else within the first minute of the experiment. The rest of the nodes discover everyone else within the first 2 minutes. Figure 7 shows the behavior of the Name Dropper algorithm for a single randomly chosen SCX in the experiment. The SCX rapidly discovers most of the other nodes in the session. The rate of discovery tapers off for the last 5% of the nodes. This indicates that the session participants quickly discover each other and can start forming a mesh structure. We will see how the discovery time for Name Dropper scales with increasing session size in Section 5.4. Let us now see how the mesh construction algorithms behave.

5.3 Mesh Construction

Figure 8 shows the variation of cost ratio $\frac{C}{C_{star}}$ for each distribution tree during the progress of an experiment involving 5 source nodes. We notice that over time, the cost ratio progressively decreases for each of the five distribution trees. Within about 300 seconds all of the cost ratios mostly stabilize to their final value. Initially, the SCXs attempt to locate other nodes and to determine their utility as neighbors. Slowly over time, as they discover their optimal neighbors, the mesh stabilizes into its final overlay structure.

Figure 9 demonstrates the distribution of cost ratios for stable meshes over a large number of experiments. We ran 100 experiments over 25 different topologies. As earlier, each experiment

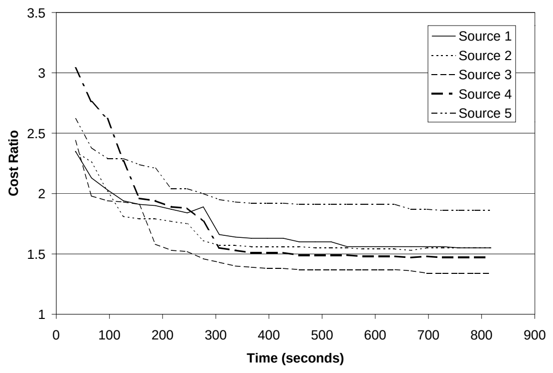

Figure 8: **Variation of cost ratio ($\frac{C}{C_{star}}$) v/s time**.

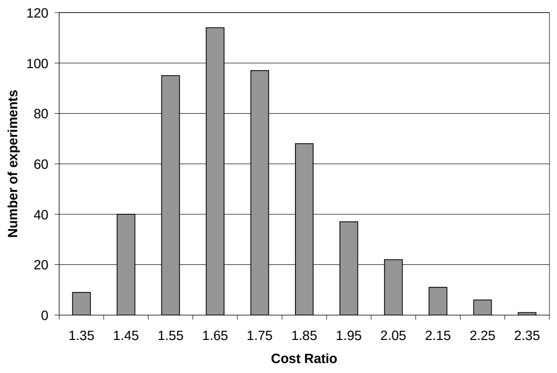

Figure 9: **Distribution of cost ratio ($\frac{C}{C_{star}}$) over a range of experiments**.

had 100 SCXs and 5 sources. The cost ratios are measured for each source once the mesh structure has stabilized. The y-axis represents the number of experiments that resulted in a cost ratio between ± 0.05 of the corresponding x-axis value. As seen from Figure 9, the distribution of the cost ratios is centered around 1.65, that is, the cost of routing data on the scattercast distribution tree is typically 1.65 times that of directly unicasting the data from the source SCX to the other nodes. For a small number of experiments, the cost ratio was as low as 1.35 or as high as 2.35.

In Figure 10, we study the stabilization properties of Gossamer. The figure shows the cumulative distribution of the average number of changes made to the mesh per node. Each edge that is added to or removed from the mesh is counted as two changes, one for each node in the edge. We notice that most of the changes to the mesh topology occur in the initial stages of the experiment. Within about 300 seconds, the mesh stabilizes to almost its final structure.

In Figure 11, we demonstrate the effectiveness of the Gossamer distribution trees in limiting the number of duplicate copies of data that any internet link needs to carry. We ran a single experiment and

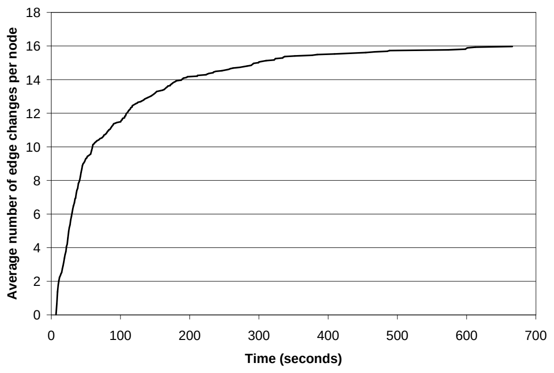

Figure 10: **Cumulative number of edge changes per node in the mesh structure over time. Each edge that is added or removed is counted as two changes, one for each node in the edge**.

computed the number of physical links in the underlying internet topology that carried duplicate data copies. The x-axis represents the number of data copies that any link may see, and the y-axis represents the number of links that carry a certain number of copies. We notice that most links carry only one copy of the data; in the Gossamer distribution tree, 153 of the physical links carry only one copy, and in the unicast star topology 177 links carry a single copy. However, in the unicast star topology, links near the source suffer from excessive packet duplication. As seen from Figure 11, at least two links carry over 95 copies of each data packet in the unicast star topology. On the other hand, with the Gossamer mesh structure, no link carries more than 14 copies of the data. Thus, we see that Gossamer is quite effective in limiting the packet duplication overhead in comparison to naive unicast.

5.4 Scaling Behavior

The above experiments depict the behavior of Gossamer for a fixed number of SCXs and a fixed node degree. We now look at how the protocol operates as we vary these parameters. For all of the following experiments, we compute each data point by running 25 independent simulations and computing the average and the 95% confidence interval.

Figure 12 shows the scaling behavior of the Name Dropper algorithm. The x-axis plots the session size in terms of the number of SCXs and the y-axis plots the time when all of the SCXs in the session have discovered at least 90% of the other SCXs. As expected, the time to completion of the Name Dropper algorithm increases with increasing session size. We note that our Name Dropper performance scales essentially linearly as opposed to the $O(log^2 n)$ performance for the original algorithm in [20]. This is due to the fact that we use a bounded list size during each DISCOVERY round unlike the original algorithm which exchanges the entire membership set $\Gamma(X)$ in each round. This penalty is incurred to limit the communication overhead in each round.

Figure 13 shows the variation in cost ratio across a range of session sizes. For a small number of SCXs, most of the SCXs are

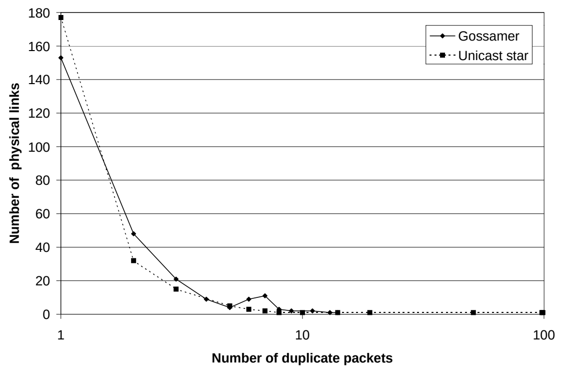

Figure 11: **Comparison of the number of physical links with packet duplication for the Gossamer topology v/s the unicast star topology**.

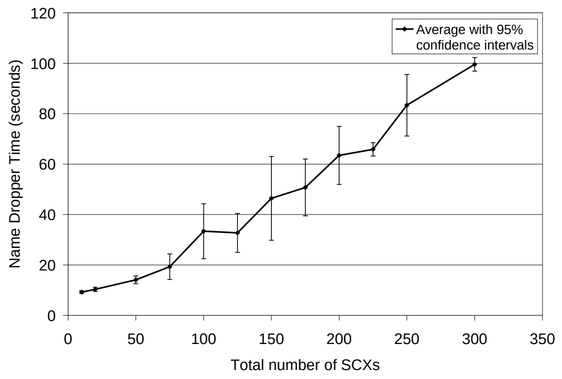

Figure 12: **Name Dropper scaling: Time for everyone to discover at least 90% of the other nodes v/s session size**.

directly connected to the source, and hence the cost ratio is low. As the number of SCXs increases, most SCXs receive data through other transit SCXs, thereby resulting in a higher cost ratio. We measured the cost ratio for sessions with up to 350 SCXs. For most fair-sized sessions, the cost ratio remains within 1.6 and 1.9. Figure 14 shows how the cost ratio varies with node degree. As expected, the cost ratio decreases with increasing node degrees. As the node degree increases, the depth of the distribution trees decreases, thereby decreasing the cost of routing over the tree.

Finally, Figure 15 shows the variation in the running time of the algorithm with respect to the total number of SCXs. We note that, as shown in Figure 10, although the mesh may not stabilize for a long time, most mesh changes occur early on and subside fairly quickly; only a small number of nodes continue to optimize their connections for a while. Figure 15 shows that the time it takes for the mesh to stabilize increases with increasing number of SCXs. This is expected since a larger session size implies more SCXs to discover and more SCXs to attempt to optimize for. In Section 7, we discuss some techniques that we are planning to investigate for improving the stabilization time of the Gossamer protocol.

5.5 Implementation Status

We have implemented a preliminary prototype of our scattercast architecture. The SCX implementation is built on top of the Active Service [1] cluster platform framework. The cluster platform takes care of the details of launching a new SCX and providing robustness and fault tolerance to SCXs. We rely on web advertisements of scattercast sessions. Clients download session announcements from the web and join the session via their local SCX cluster platform. Although Section 2.4 outlines a number of mechanisms for finding the local SCX cluster platform, for simplicity, we have implemented static configuration. The cluster platform location is either read from a well-known location (e.g. a file in /etc on Unix or a registry property on Windows) or passed to the application on the command-line. SCXs implement most of the features of the Gossamer protocol, which they use to self-organize into an overlay

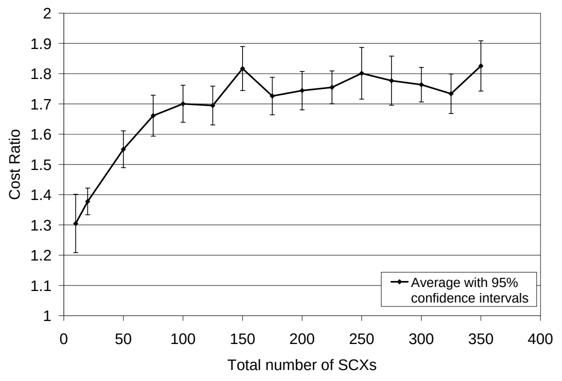

Figure 13: **Variation of cost ratio ($\frac{C}{C_{star}}$) v/s session size**.

structure. We are currently building applications on top of this architecture. We have implemented the shared electronic whiteboard tool and are currently implementing the Internet audio broadcast application described in Section 4.1.

6 Related Work

The Endsystem Multicast [42] and Yallcast [13] research projects have proposed similar multi-point data distribution frameworks that build distribution trees purely on an end-host basis. Like scattercast, both Endsystem Multicast and Yallcast address the ineffectiveness of IP multicast for content distribution. They rely on self-organizing protocols for constructing distribution trees out of unicast tunnels across end-hosts participating in the multicast session. Like scattercast, Endsystem Multicast builds a mesh structure across participating end-hosts and then constructs source-rooted trees by running a routing protocol. On the other hand, Yallcast directly builds a spanning tree structure across the end-hosts without any intermediate mesh structure. Although this approach avoids the

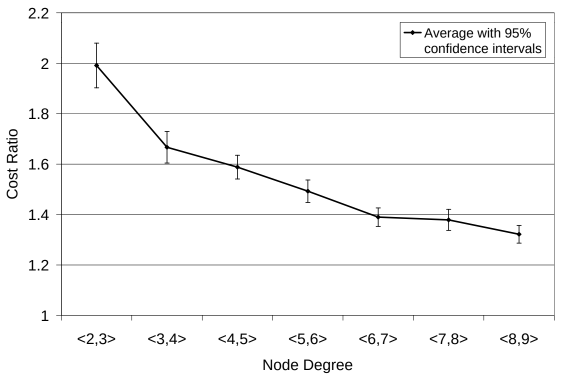

Figure 14: **Variation of cost ratio ($\frac{C}{C_{star}}$) v/s node degree ($<k_1,k_2>$ as defined in Section 3.2)**.

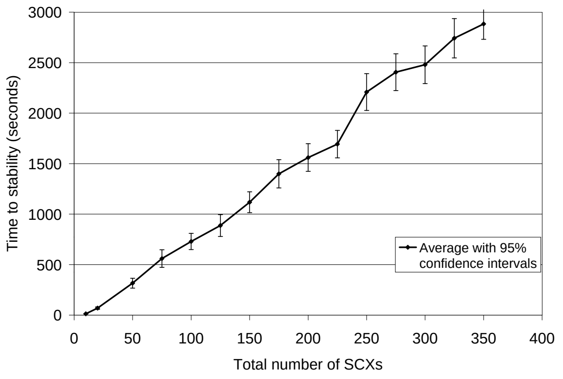

Figure 15: **Time to stability v/s session size**.

redundant edges that a mesh structure incurs, it requires expensive loop detection and avoidance mechanisms, and is also extremely fragile and susceptible to partitions.

The main difference between our approach and that of Yallcast and Endsystem Multicast is the explicit use of infrastructure service agents—SCXs—in our architecture. Although it is possible to incorporate proxies into Endsystem Multicast and Yallcast, SCXs are an integral aspect of the scattercast architecture. We believe that for such a framework to scale well beyond a few hundred clients, infrastructure support will be absolutely crucial. A fully decentralized end-host-only self-organization protocol will not scale beyond a few hundred or a few thousand participants. On the other hand, since scattercast proxies can simultaneously serve many clients, we believe that one or a small number of proxies per ISP will be sufficient to serve a large client population.

The scattercast architecture germinated from prior work by Ratnasamy et al. [32]. They defined a *delivery-based model* for reliable multicast communication where receivers organize themselves into a multilevel hierarchy of disjoint multicast delivery groups. Data transmission is achieved by unicasting data between group representatives which in turn multicast data to their delivery group. Such a delivery model enables the data delivery process to be tailored to match the homogeneous network characteristics within individual delivery groups. In [33], Ratnasamy et al. use a multicast-tree-inference algorithm to build a protocol building block—a distributed Group Formation Protocol (GFP)—that allows receivers to self-organize into a source-rooted hierarchy of disjoint multicast groups where the hierarchy is congruent with the native multicast tree topology. However, this protocol relies on the existence of a global multicast control channel, which scattercast explicitly intends to avoid.

The Adhoc Multicast Routing Protocol, AMRoute [24], is an approach for multicast in mobile adhoc networks that creates bidirectional shared trees for data distribution using only group senders and receivers as tree nodes. Unicast tunnels are used as tree links to connect neighbors on the user multicast tree. Thus, AMRoute does not need to be supported by network nodes that are not interested in or capable of multicast and group state cost is incurred only by group senders and receivers. However, unlike scattercast, AMRoute does not attempt to optimize the distribution tree in any form. Scattercast, on the other hand, explicitly relies on Gossamer to build an efficient overlay network for data transmission.

In [2], Bauer et al. compare a number of heuristics to find efficient degree-restricted multicast trees in the presence of constraints on the copying ability of the individual switch nodes in the network. Although some of these heuristics may be applied to construct distribution trees in scattercast, we believe that the mesh-first approach used by Gossamer is superior to directly building spanning trees.

Reliable multicast transport protocols such as RMTP [23] organize group members into a hierarchical tree structure for aggregating acknowledgments at midpoints in the network. Each branch in the tree has a designated receiver (DR) to receive acknowledgments from its children and aggregate them upwards to the sender. The scattercast architecture is similar to RMTP in that it groups clients around an SCX just as RMTP groups receivers around DRs. But RMTP uses its tree structure only for acknowledgments and recovery of lost data; all initial data transmission happens over a global multicast group. Scattercast, on the other hand, relies on tunneled distribution trees for data transmission as well.

7 Future Work

We have implemented a preliminary prototype of the scattercast architecture. Although our simulation experiments demonstrate the efficacy of the architecture, we plan to conduct experiments using a deployed system in a real network. We plan to deploy our prototype across the wide area to evaluate its performance and its scalability especially in the face of real world traffic.

We are currently investigating extending the Gossamer protocol to allow for multi-level Gossamer meshes in order to achieve better scaling properties. SCXs in the local area participate in a local Gossamer protocol independent of the rest of the session. A small number of representative SCXs from the local area also participate in a higher level Gossamer protocol with other SCXs across the entire Internet. This ensures that the topmost level consists of fewer SCXs and hence can stabilize to its final mesh structure relatively quickly. The lower-level SCXs only interact with their nearby SCXs and are not affected by the behavior of the rest of the session.

Although scattercast currently permits only source-rooted trees, it is possible to extend this to bidirectional shared trees. Sources that wish to use a shared tree can explicitly include the root SCX for the shared tree in their packet headers. Gossamer will then route the data over the explicitly specified shared tree. Such shared trees are especially useful for scenarios containing a large number of participants, all of whom are generating packets. For example, in order to implement a reliability protocol such as SRM, RMTP or PGM on top of scattercast, receivers in a scattercast session may intermittently transmit ACK or NACK packets, and may potentially respond to retransmission requests. In such a situation, it is advantageous to use the distribution tree rooted at the source as a bidirectional shared tree for the ACK, NACK, and retransmission traffic. In the future, we plan to investigate the use of Gossamer for constructing such bidirectional shared trees.

8 Summary

We have presented an architecture for Internet content distribution that relies on application-level intelligence embedded within the network infrastructure rather than on network layer multicast primitives to provide efficient multi-point data distribution. Our architecture, which we call scattercast, makes use of a collection of intelligent network agents (ScatterCast proXies or SCXs) that collaboratively provide the multicast service for a session. Clients participate in the session via a nearby SCX by either using locally scoped IP multicast groups or direct unicast connections to the local SCX. SCXs organize themselves into an overlay network of unicast interconnections and build data distribution trees on top of the overlay structure.

By migrating the multi-point delivery functionality out of the network layer to a higher infrastructure service layer, scattercast maintains the simplicity of the underlying network model. Moreover, scattercast simplifies the design of complex reliability and congestion control protocols by allowing for application-specific adaptation to deal with the heterogeneity that typically cripples traditional reliable multicast protocols.

Scattercast relies on a protocol called Gossamer to build an efficient overlay structure. Our simulation results show that the latencies incurred by transmitting data over the scattercast mesh are typically within 1.6 to 1.9 times those associated with directly unicasting or multicasting the data from the source to the various destinations. At the same time, the mesh generated by Gossamer substantially limits the bandwidth usage of the physical Internet links in comparison to naive n-way unicast.

The scattercast architecture is a first step towards a new approach for content distribution that explicitly moves application intelligence into the network infrastructure, while at the same time maintaining compatibility with the existing IP architecture. We believe that as the Internet evolves, architectures similar to scattercast based on intelligent application-aware network components will become increasingly prevalent. Our experience with scattercast can provide valuable input for the design of such next-generation Internet architectures.

References

[1] AMIR, E., MCCANNE, S., AND KATZ, R. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM '98* (Vancouver, British Columbia, Canada, Sept. 1998).

[2] BAUER, F., AND VARMA, A. Degree-constrained Multicasting in Point-to-point Networks. In *Proceedings of IEEE Infocom '95* (Mar. 1995).

[3] BRAY, T., PAOLI, J., AND SPERBERG-MCQUEEN, C. M. *XML: Extensible Markup Language*, Dec. 1997. W3C Proposed Recommendation. `http://www.w3.org/TR/PR-xml-971208`.

[4] CHAWATHE, Y., AND BREWER, E. System Support for Scalable and Fault Tolerant Internet Services. In *Proceedings of Middleware '98* (Lake District, U.K., Sept. 1998).

[5] CHAWATHE, Y., MCCANNE, S., AND BREWER, E. A. RMX: Reliable Multicast for Heterogeneous Networks. In *Proceedings of INFOCOM 2000* (Tel Aviv, Israel, Mar. 2000).

[6] CHENG, C., RILEY, R., KUMAR, S. P. R., AND GARCIA-LUNA-ACEVES, J. J. A Loop-Free Extended Bellman-Ford Routing Protocol Without Bouncing Effect. In *Proceedings of SIGCOMM '89* (1989).

[7] DEERING, S., ESTRIN, D., FARINACCI, D., JACOBSON, V., LIU, C.-G., AND WEI, L. An Architecture for Wide-area Multicast Routing. *IEEE/ACM Transactions on Networking 4*, 2 (Apr. 1996).

[8] DEERING, S. E. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Dec. 1991.

[9] DIOT, C., LEVINE, B. N., LYLES, B., KASSAN, H., AND BALENSIEFEN, D. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Networks special issue on Multicasting* (2000).

[10] DROMS, R. *Dynamic Host Configuration Protocol*, Mar. 1997. RFC-2131.

[11] FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., AND ZHANG, L. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of ACM SIGCOMM '95* (Boston, MA, Aug. 1995), pp. 342–356.

[12] FOX, A., GRIBBLE, S., CHAWATHE, Y., BREWER, E., AND GAUTHIER, P. Cluster-based Scalable Network Services. In *Proceedings of SOSP '97* (St. Malo, France, Oct. 1997), pp. 78–91.

[13] FRANCIS, P. Yallcast: Extending the Internet Multicast Architecture. `http://www.yallcast.com/`.

[14] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[15] GAUTHIER, P., COHEN, J., DUNSMUIR, M., AND PERKINS, C. *Web Proxy Auto-Discovery Protocol*, Dec. 1999. Internet Draft.

[16] GRIBBLE, S. D., WELSH, M., BREWER, E. A., AND CULLER, D. The MultiSpace: An Evolutionary Platform for Infrastructural Services. In *Proceedings of the 1999 Usenix Annual Technical Conference* (Monterey, CA, June 1999).

[17] GULBRANDEN, A., AND VIXIE, P. *A DNS RR for specifying the location of services (DNS SRV)*, Oct. 1996. RFC-2052.

[18] HAMILTON, M., AND WRIGHT, R. *Use of DNS Aliases for Network Services*, Oct. 1997. RFC-2219.

[19] HANDLEY, M. *SAP: Session Announcement Protocol*, Mar. 1997. Internet Draft.

[20] HARCHOL-BALTER, M., LEIGHTON, T., AND LEWIN, D. Resource Discovery in Distributed Networks. In *Proceedings of the 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing* (Atlanta, GA, May 1999).

[21] HOLBROOKE, H. W., AND CHERITON, D. R. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *Proceedings of SIGCOMM '99* (Cambridge, MA, Sept. 1999).

[22] KUMAR, S., RADOSLAVOV, P., THALER, D., ALAETTINOGLU, C., ESTRIN, D., AND HANDLEY, M. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *Proceedings of SIGCOMM '98* (Vancouver, British Columbia, Canada, Sept. 1998).

[23] LIN, J. C., AND PAUL, S. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of IEEE Infocom '96* (San Francisco, CA, Mar. 1996), pp. 1414–1424.

[24] LIU, M., TALPADE, R., MCAULEY, A., AND BOMMAIAH, E. AM-Route: Adhoc Multicast Routing Protocol. Technical Report CSHCN T.R. 99-1/ISR T.R. 99-8, Institute for Systems Research, University of Maryland, College Park, MD, 1999.

[25] LIVE.COM. A Multicast Plugin for WinAmp. `http://www.live.com/multikit/winamp-plugin.html`.

[26] LUOTONEN, A. Proxy Auto Configuration. Netscape Corporation. `http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html`.

[27] MCCANNE, S. A Distributed Whiteboard for Network Conferencing. Class report, UC Berkeley, May 1992.

[28] MEYER, D., AND LOTHBERG, P. *Static Allocations in 233/8*. Internet Engineering Task Force, draft-ietf-mboned-static-allocation-*.txt, May 1999. Internet Draft.

[29] MPG123. A Real-time MPEG Audio Player for Layers 1, 2 and 3. `http://www.mpg123.de/`.

[30] NULLSOFT INC. The WinAmp MP3 Player. `http://www.winamp.com/`.

[31] PERLMAN, R., LEE, C., BALLARDIE, T., CROWCROFT, J., WANG, Z., MAUFER, T., DIOT, C., THOO, J., AND GREEN, M. *Simple Multicast: A Design for Simple, Low-overhead Multicast*. Internet Engineering Task Force, Mar. 1999. Internet Draft (work in progress).

[32] RATNASAMY, S., CHAWATHE, Y., AND MCCANNE, S. A Delivery-based Model for Multicast Protocols using Scattercast. *Unpublished*, July 1999.

[33] RATNASAMY, S., AND MCCANNE, S. Scaling End-to-end Multicast Transports with a Topologically-sensitive Group Formation Protocol. In *Proceedings of the 7th International Conference on Network Protocols* (Toronto, Canada, Oct. 1999).

[34] SALTZER, J., REED, D., AND CLARK, D. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems 2*, 4 (1984), 195–206.

[35] SANDPIPER/DIGITAL ISLAND INC. The Sandpiper Networks Footprint Service. `http://www.digisle.net/services/cd/footprint.shtml`.

[36] SPEAKMAN, T., FARINACCI, D., LIN, S., AND TWEEDLY, A. *Pragmatic General Multicast (PGM) Reliable Transport Protocol*. CISCO Systems, 1998. Internet Draft.

[37] TUNG, T.-L. Mediaboard: A Shared Whiteboard Application for the MBone. Master's thesis, University of California, Berkeley, Dec. 1997.

[38] VEIZADES, J., GUTTMAN, E., PERKINS, C., AND DAY, M. *Service Location Protocol*, Oct. 1997. Internet Draft.

[39] VICISANO, L., RIZZO, L., AND CROWCROFT, J. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proceedings of INFOCOM '98* (Mar. 1998).

[40] WHETTEN, B., AND CONLAN, J. A Rate-based Congestion Control Scheme for Reliable Multicast. GlobalCast Communications, Oct. 1998.

[41] ZEGURA, E. W., CALVERT, K. L., AND BHATTACHARJEE, S. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96* (San Francisco, CA, Mar. 1996).

[42] ZHANG, H. A Case for EndSystem Multicast, June 1999. Presentation at the Internet End-to-end Research Meeting.