# Collaborative Networking in an Uncooperative Internet[*]

Robbert van Renesse and Dan Dumitriu

Department of Computer Science, Cornell University
E-mail: {rvr,dumitriu}@cs.cornell.edu

## Abstract

*Collaborative applications often require peer-to-peer interaction and peer discovery mechanisms. In today's Internet, Firewall and NAT technology, and a lack of support of IP multicast, have made it very difficult to support such applications. Application Level Gateways and Directory Services can solve these problems to some extent, but have scalability problems and should be used as a last resort. This paper describes our experience with implementing a service called Astrolabe which uses a peer-to-peer epidemic protocol. We show how we solved peer-to-peer communication, auto-configuration, and peer discovery. The resulting Astrolabe service can be used to support the development of other peer-to-peer protocols and applications.*

## 1 Introduction

The original Internet was designed so that any host (or rather, network interface) had a unique address, and any two hosts could exchange messages using their respective addresses. As such, it was a network ideally suited as a platform for running collaborative, peer-to-peer applications. Unfortunately, driven by a shortage of addresses and commercial pressures, today's Internet is quite different from its original conception [5]. Firewalls [8], Network Address Translation [7], and DHCP [6] solve many problems of scale and security, but make direct peer-to-peer interactions in many cases impossible [11]. For example, many IP addresses in current use are not unique, nor are they routable except in their direct vicinity. IPv6 may eventually solve this situation, but not any time soon.

One now common solution is to route messages across HTTP over an Application Level Gateway (ALG) such as used in JXTA [9] and Groove [14]. The ALG (see Figure 1) is a more-or-less standard Web server that maintains a message queue for client hosts. Hosts can poll their own message queue, or queue messages intended for other hosts, simply by sending HTTP requests to the ALG. These requests and responses can be routed through web proxy servers, or NAT boxes, without problems. If enough ALGs are deployed on the Internet, this solution may scale quite well, particularly if an auto-configuration protocol like WPAD (Web Proxy Auto-discovery Protocol) were developed so that hosts use nearby ALGs to receive messages.

Nevertheless, this solution is unsatisfactory for at least three reasons. First, each host requires a persistent TCP connection to an ALG, just in case messages arrive for it, potentially resulting in many unused TCP connections that waste resources. Second, if two hosts can communicate directly, going through an ALG increases latency and wastes more resources. Finally, an ALG is a single point of failure, and its disruption may severely affect a large part of the peer-to-peer network. Thus it is desirable to use an ALG only as a last resort.

Another problem that face collaborative applications is peer discovery. When a new participant starts up, it either has to find the other participants, or be found. A centralized directory service may not scale or be sufficiently robust, and so IP multicast is often suggested as a basis for discovery (see, for example, [1]). Unfortunately, IP multicast in the current Internet is very poorly supported.

This paper investigates the problem of efficient peer-to-peer communication and peer discovery in the context of the Astrolabe service. Astrolabe provides, among other functions, peer-to-peer aggregation of information in the form

[*]This work was funded in part by DARPA/AFRL-IFGA grant F30602-99-1-0532, a grant under NASA's REE program administered by JPL, NSF-CISE grant 9703470, and by the AFRL/Cornell Information Assurance Institute.

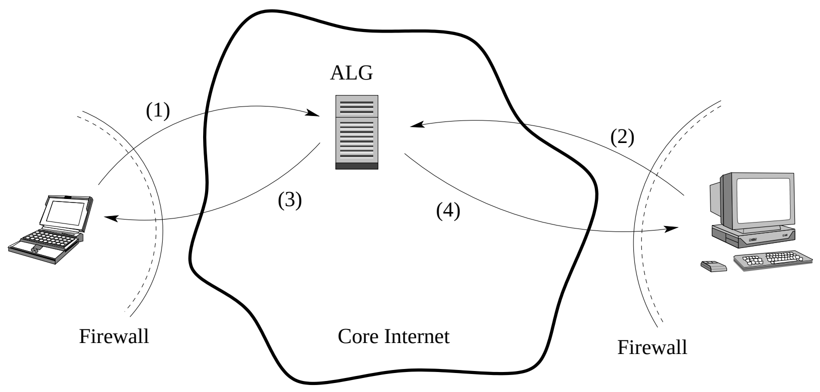

**Figure 1. Application Level Gateway. (1) Receiver sends a RECEIVE request using an HTTP POST request; (2) Sender sends the message using a SEND request using an HTTP POST request; (3) ALG forwards the message to the receiver using an HTTP 200 response; (4) ALG sends an empty HTTP 200 response back to the sender.**

of a DNS-like directory hierarchy. Astrolabe uses epidemic protocols for internal data dissemination. In addition to solving the problems outlined above, Astrolabe provides for automatic membership discovery and configuration of collaborative applications. Many of the problems and solutions described in this paper came up when deploying a *demo* version of Astrolabe, downloadable from the web. The demo had to install and run without any further configuration by the user who downloaded the demo.

We will start with reviewing various related work in Section 2. Next we describe, in short, the function and implementation of the Astrolabe service in Section 3. In Section 4, we present how addressing is performed in Astrolabe. Auto-configuration of Astrolabe is discussed in 5. Communication using ALGs is the topic of Section 6. Section 7 presents Astrolabe's peer discovery mechanisms. We conclude in Section 8.

## 2 Related Work

Groove Networks, Inc. (groove.net) is the provider of a peer-to-peer communications technology. Although in theory peers in Groove can communicate directly with one another (assuming they are not separated by firewalls or NAT), they heavily rely on their proprietary ALG called *Groove Relay Server* [14]. Unless peers are explicitly configured to communicate directly with one another, they will use the Relay Server for communication. They also use the Relay Server for other functions. This includes message queuing for off-line peers and resource discovery. This makes most Groove applications heavily dependent on the Relay Server, and direct peer-to-peer interactions are rarely used. Groove Networks has deployed dozens of Relay Servers, and the Server will be available for sale for use in enterprise settings as well.

JXTA [9] (jxta.org) is an open platform for peer-to-peer interactions in the network, intended for pervasive use from servers to workstations to PDAs and cell phones. JXTA offers a variety of services such as Peer Discovery and Peer Membership. In order to allow peer discovery and peer-to-peer communication, the notion of an ALG (*Rendez-Vous Server* in JXTA terminology) has been proposed, but this is still an ongoing research effort.

Activision (activision.com) has a proprietary UDP-based broadcast protocol for use in the face of NAT [11]. Game players send their local UDP/IP addresses to a centrally located server. The server stores this information along with the UDP/IP address generated by the NAT box, and informs each player about each other player. The players then send *hello* messages to each other. On receipt of an *hello* message, a player process returns a reply. If a reply is received, the corresponding address becomes the preferred address for the peer player. This only works for NAT boxes that pass UDP packets and set up return paths. Many NAT boxes and firewalls do not pass UDP packets, however.

An Overlay Network (ON) is a network abstraction layered on top of the Internet (or possibly another ON). An ON can offer special capabilities to its users, such as multicast or security. ONs deploy application-level routers in the edges of the Internet, without directly affecting routing in the Internet itself.

A Virtual Private Network (VPN) is an example of an ON. A VPN can span multiple firewalls, and provides as such a solution to peer-to-peer communication. Current VPN solutions, such as PPTP [10], do not scale well, and will not support a global VPN. For small scale collaborative applications, however, VPNs have been shown to work adequately.

Another example of an ON is MIT's Resilient Overlay Network (RON) [2]. RON is intended to overcome long-term link outages due to IP routing misbehavior, and does so by re-routing messages through its application-level routers. Although RON currently does not support routing through NAT, it could be adapted to do so [2]. Unfortunately, RON uses an $O(n^2)$ pinging protocol in order to monitor the conditions of links, and therefore does not scale beyond a few dozen RON routers. Detour [15] is another ON project similar to RON.

NAT (and similarly IP Masquerading) has several well-known problems [11]. Realm Specific IP (RSIP) [3] is a proposal for dealing with address shortage as an alternative to NAT. Unlike NAT, RSIP is not transparent to TCP clients, and requires a protocol implementation upgrade on end-hosts. An important advantage of RSIP is that clients would be able to accept incoming connections.

Finally, we note that e-mail is perhaps the most ubiquitous peer-to-peer protocol available. It supports, through the use of DNS, SMTP, POP, and/or IMAP, uninhibited peer-to-peer communication, message queuing, and peer discovery. Although we have considered this option, the potential latencies involved are too large to be practical for Astrolabe.

## 3 Astrolabe

Astrolabe provides a scalable aggregation service to its clients. Astrolabe organizes hosts into a domain hierarchy, in which the hosts themselves form the leaf domains. Each domain has a set of attributes. Unlike the leaf domains' attributes, which may be directly updated by their corresponding hosts, the attributes of an internal domain are generated by aggregating the attributes of its child domains. Each domain is identified by its domain name. Unlike the DNS convention, Astrolabe uses Unix-style path names to identify domains. For example, "/usa/ny/ithaca/cornell" (rather than cornell.ithaca.ny.usa).

The implementation of Astrolabe is entirely peer-to-peer. Each host runs an agent process that communicates with other agents through an epidemic protocol or *gossip*. The data structures and protocols are designed such that the service scales well:

- The memory requirements on each host grow logarithmically with the membership size;
- The size of gossip messages grows logarithmically with the size of the membership;
- If configured well (more on this in Section 5), the gossip load on network links grows logarithmically with the size of the membership, and is independent of the update rate;
- The *latency* grows logarithmically with the size of the membership. Latency is defined as the time it takes to take a snapshot of the entire membership and aggregate all its attributes;
- Astrolabe is tolerant of severe message loss and host failures, and deals with network partitioning and recovery.

In practice, even if the gossip load is low (the agents in the demo version of Astrolabe gossip only once every five seconds), updates propagate very quickly, and is typically within a minute even for very large deployments [16].

In the description of the implementation of Astrolabe below, we omit all details except those that are necessary in order to understand the function of Astrolabe, and the issues that relate to peer-to-peer communication in the Internet. For an in-depth description of Astrolabe, please refer to [16].

As there is exactly one Astrolabe agent for each leaf domain, we name Astrolabe agents by their corresponding domain names. Each Astrolabe agent maintains, for each domain that it is a member of, a relational table called the *domain table*. For example, the agent "/a/b/c" has domain tables for "/", "/a", and "/a/b". A domain table of a domain contains a row for each of its child domains, and a column for each attribute name. One of the rows in the table is the agent's *own* row, which corresponds to that child domain that the agent is a member of as well. Using a SQL aggregation function, a domain table may be aggregated to form a single row. This row forms the *own* row in the parent's domain table of the agent.

Since multiple agents may be in the same domain, the corresponding domain table is replicated on all these agents. For example, agents "/a/b/c" and "/a/d/e" maintain both the "/" and "/a" tables.

A replicated table is kept consistent using an epidemic protocol. Each domain calculates, using an aggregation function, a small set of representative agents for its domain. Typically, Astrolabe is configured to use up to three representative agents for each domain. The representative agents of the child domains of a parent domain run the epidemic protocol for the parent's domain table. On a regular basis, say once a second, each agent X that is a representative for some child domain chooses another child domain at random, and then a representative agent Y within the chosen child domain, also at random. X sends the parent's table to Y. Y merges this table with its own table, and sends the result back to X, so that X and Y now agree on the contents of their tables.

The key here is how to merge the tables. The basic idea is as follows. Y adopts into the merged table rows from X for child domains that were not in Y's original table, as well as rows for child domains that are more current than in Y's original table. To determine currency, agents timestamp rows each time they are updated by writing (in case of leaf domains) or by generation (in case of internal domains). Unfortunately, this requires all clocks to be synchronized which, at least in today's Internet, is far from being the case.

To solve this problem, each row is tagged with the *generator*: the domain name of the agent that wrote or generated the row (in addition to the timestamp). Agents also maintain, for each row in each table, the set of generators from which they received updates for that row, along with the timestamp on the last received update. The merge operation is now executed as follows. When receiving a row in a gossip message, the agent adopts it, as is, if it is created by a previously unknown generator. If it is a known generator, the row is adopted if and only if the row's timestamp is more recent than the last received timestamp from that generator. This way, only timestamps from the same agent are compared with one another, and therefore no clock synchronization is necessary.

When no update has been received from a particular generator for some time period $T$, that generator is considered faulty, and forgotten. $T$ should be chosen so that the probability of any *old* gossips from this generator still going around is very low [16]. Since gossips disseminate in time $O(\log n)$, this typically is not very long, and can be determined by techniques of epidemic analysis or simulation. The aggregation function that determines the set of representatives for the corresponding domain will automatically assign a new representative. When there are no more representatives for the domain, and consequently the last generator of a row is removed, the row itself is deleted from its table.

The other side of domain membership, new domain discovery, is much more problematic in today's Internet. Originally Astrolabe depended on occasional IP multicasts, but as IP multicast is very poorly supported in the Internet, this was not an adequate solution. Section 7 is devoted to the issue of locating new peers.

Finally, new aggregation functions can be installed dynamically, and their dissemination piggybacks on the gossip protocol. This way an Astrolabe hierarchy can be customized for the applications that use it.

## 4 Addressing

Astrolabe supports communication using UDP/IP, using HTTP (on top of TCP/IP or SSL), or both. To support HTTP, Astrolabe agents act both as HTTP servers and clients. As we will see in Section 6, the use of an ALG is transparent to the sending agent, but requires some additional work by the receiving agent. Before we discuss the actual communication in more detail, we will first describe the concept of *realms*, and how addressing is done in Astrolabe.

A *realm* is a set of hosts and a communication protocol. For example, the tuple ("Cornell Computer Science Department", UDP) forms a realm, as does ("Core Internet", HTTP). ("Core Internet" is the set of those hosts on the main Internet that do not reside behind firewalls.) Each realm has a unique identifier of the form name:protocol, for example "cornellcs:udp" and "internet:http". The hosts in a realm form an equivalence class, in that they can all be accessed using the realm's protocol in the same way. A host can be in more than one realm, and can have more than one address in the same realm. No two hosts in the same realm can have the same address, but the same address may be used in different realms. (A similar addressing strategy was used in GTS [13].)

UDP addresses are of the form "IP-address:port" (e.g., "10.0.0.4:6422") or "DNS-name:port" (e.g., "rome.cs.cornell.edu:6422"). HTTP addresses are of the form "agent-name@TCP-address", where "agent-name" is the Astrolabe domain name of the agent, and "TCP-address," as in UDP addresses, consists of a port and either an IP address or a DNS name. For example, "/usa/ny/ithaca/cornell/cs/rome@10.0.0.4:2246".

We define an *extended address* to be the triple (realm identifier, address, preference). For example, ("cornellcs:udp", 10.0.0.4:6422, 5). A host has a set of these addresses, and can indicate its preference for certain addresses using the *preference* field. We call the set of extended addresses of a host the *contact* for that host. The contact is dynamic as addresses may appear and disappear over time as the administrator of the host connects to, or disconnects from, ISPs and VPNs.

Unlike agent's contacts, agent's names are constant. In order to simplify configuration, we observed that realms often coincide with Astrolabe domains, and thus named realms using their corresponding domain name. Thus, rather than "cornellcs:udp", we would use "/usa/ny/Ithaca/cornell/cs:udp". The core Internet coincides with the root domain, and is thus called "/:http".

Each domain in Astrolabe has an attribute called *contacts*, which contains the contacts of those agents in the domain that have been elected as representatives. This is done using an aggregation function that computes a union with a restricted output size. The *contacts* attribute of a leaf domain contains the singleton set with the contact of the agent of that leaf domain.

We will now briefly revisit Astrolabe's gossip protocol to show how this works in practice. When an agent wants to gossip the table of some domain, it has to come up with an address. First, the agent picks one of the table's rows at random and retrieves the *contacts* attribute from that row. The agent then picks one of the contacts at random. The resulting contact is a set of extended addresses. The agent removes the addresses of realms that it cannot reach (more on this below). If there is more than one remaining address, the agent has to make one more choice.

In order to make intelligent choices, each Astrolabe agent maintains statistics about addresses. This is simple to do, as each gossip message is followed by a response. Currently, an agent maintains for each extended address the following three values:

1. `outstanding`: the number of gossips sent since the last response was received;
2. `last_sent`: time of last gossip transmission;
3. `last_received`: time of last reception of a response.

If there is more than one extended address to choose from, the agent *scores* each address:

1. If there is no outstanding gossip, the score is the preference;
2. If it has been more than a minute since the last gossip was sent, the score is the preference;
3. If there is just one outstanding gossip, the score is the preference minus one;
4. In all other cases, the score is zero.

This results in the following behavior. In the normal case, when gossips are followed by responses, the address of the highest preference is always used. If a single response got lost, the score becomes only slightly smaller. The intention is that if there is more than one address of the same preference, the ones that are only somewhat flaky become less preferential. If there are more losses, the score becomes such that the address is only used as a last resort. Once a minute, the score is, for a single send operation, reset to the original preference. This allows addresses to be occasionally re-tested.

## 5 Configuration

In order for Astrolabe to scale well, the domain hierarchy has to be set up with care. Each domain in Astrolabe runs an instance of the gossip protocol among the representatives of its child domains. The first concern to think about is the size of domains, that is, the number of child domains in a domain. If very large, the size of gossip messages, as well as the generated gossip load, will be large as well (they both grow linearly with the domain size). If chosen to be very small, the hierarchy becomes very deep, and latency will suffer. In practice, we find that a size of 25-100 child domains in a domain works well. Smaller sizes are possible too, at the cost of some additional latency, but larger sizes make the load unacceptably large.

The second concern is locality. The domains should be constructed ideally so that the number of network hops between its child domains' representatives is minimized, and so that independent domains (one domain is not an ancestor of the other) do not share any network links. If the Internet were a tree topology, the Astrolabe hierarchy should be preferably identical to this tree. In reality the edges of the Internet often resemble a tree topology, but the internal Internet is a complicated mesh of links that defies any resemblance to a tree.

In practice, this means that there is considerable freedom in designing the higher levels of the Astrolabe hierarchy, within the limits of the branching factor, but the lower levels should be mapped closely to the topology of the Internet edge. If we ignore the branching factor, this is not much different from the DNS hierarchy design. In DNS, too, the low levels often correspond closely to the network topology, while the high levels of the hierarchy have little correspondence to the Internet topology. Thus the main difference between DNS and Astrolabe configuration is the constrained branching factor of the Astrolabe hierarchy.

Astrolabe supports two forms of configuration: manual and automatic. The manual configuration supports various notions of security, including an integrated PKI infrastructure for the Astrolabe service. The automatic configuration is not secure. In order to foil all but the simplest forms of compromise, the communication is scrambled and signed using secret keys. The holy grail of peer-to-peer computing, a secure, self-configuring system, seems to be an unachievable dream. Below, we will focus on Astrolabe's insecure automatic configuration. More on Astrolabe security can be found in [16].

In an insecure version of Astrolabe, all an agent needs to know is

- Its domain name;
- The set of realms that it can send messages to;
- How to find peer agents to gossip with.

In the remainder of this section, we will look at the automatic configuration of domain names and realms. Peer location, which is another aspect of configuration, will be described in a Section 7.

Currently, we generate the Astrolabe domain name of an agent from the DNS domain name of the host, and the process identifier of the agent. We will first explain how this is done, and then provide the rational for this design. Say that the DNS domain name is $C_0.C_1.....C_k$, and the process id of the agent is $P$. We use a one-way hash function on $C_1...C_k$ (all but the first component of the domain name) to construct three 6-bit integers, $A_1$, $A_2$, and $A_3$. Finally, the Astrolabe domain name is constructed to be $/C_k/A_1/A_2/A_3/C_{k-1}/.../C_0/P$.

For example, say an agent runs as process 4365 on host "rome.cs.cornell.edu". By hashing "cs.cornell.edu" onto three 6-bit integers, we have effectively split the ".edu" domain up into $2^{18}$ pieces, as the ".edu" domain itself is much too large for a single Astrolabe domain. Using this construction, the Astrolabe "/edu" domain itself has at most 64 child domains. Say the three generated integers in our example are 25, 43, and 4 respectively. Then the Astrolabe domain name of the agent is "/edu/25/43/4/cornell/cs/rome/4365". (The three generated domains can be hidden from view if so desired.)

The hope is that each of the domains following "/edu/25/43/4" are of relatively limited size that can be supported by the Astrolabe protocol, and that these domains reflect the network topology to a close enough approximation. If in the future this turns out to be insufficient, we can update the downloadable executable to use more levels, or perhaps come up with an adaptive scheme. The addition of the process identifier makes it possible to run multiple agents on the same host.

Next we have to determine the set of realms that the agent can reach. We assume that any agent can communicate to the "/:http" realm, that is, any agent can use HTTP to reach another agent on the core Internet. (Agents may use WPAD to determine the existence and location of an HTTP proxy automatically.) Furthermore, we assume that $/C_k/A_1/A_2/A_3/C_{k-1}/.../C_1$:udp ("/edu/25/43/4/cornell/cs:udp" in our example) is a realm, and that all agents within this realm can communicate with one another. In Section 7.4, we will see that this is not always the case and we have to fix this.

Currently, these are all the assumptions we make about realms. In practice this is sometimes conservative, and in that case agents that can communicate directly using UDP will use a ALG instead. In the next section, we describe how ALGs are configured and used.

## 6 Communication through an ALG

An Application Level Gateway (ALG) may be the only possibility for two agents to communicate (see Figure 1). Significant care should be taken in deploying and configuring ALGs. The number of ALGs is likely to be small compared to the number of agents using them, and thus they should be used judiciously in order not to overload them or the network links that connect them. Also, care should be taken that the peer-to-peer network remains tolerant of failures, and does not get partitioned when a single ALG server crashes or otherwise becomes unavailable, and that network security is not compromised. Finally, in order for the system to scale, it should be possible to add new ALG servers dynamically to the system as the number of Astrolabe agents grows. These new servers should be automatically discovered and used by the existing agents as well as the new ones.

Ideally, an ALG is located on the network path between a sender and a receiver, so that the number of hops that a message has to travel is not severely affected by the presence of an ALG. Since many senders may send messages to the same receiver, it follows that the ALG should be located as close to the receiver as possible. Thus, ideally, each firewall or NAT box has a companion ALG that serves receivers behind the firewall. In practice, we suspect that far fewer ALGs will be deployed, but it is still important for receivers to connect to the nearest-by ALG or ALGs.

Each receiver that wishes to receive messages through an ALG has to use HTTP requests to the ALG. In practice, this happens over a persistent TCP connection. In order to reduce the number of such connections to an ALG, not every host behind a firewall has to connect to the ALG. In Astrolabe, only representatives for the realm corresponding to the firewalled site gossip beyond the firewall boundaries, and only these agents (typically, two or three), need to receive through an ALG. The other agents learn indirectly of updates outside the realm through gossip with the representatives (see Figure 2).

In order for agents to locate ALGs, the ALGs themselves are situated in the Astrolabe hierarchy itself. Each ALG has a companion Astrolabe agent with a configured domain name. The *relays* attribute of the corresponding leaf domain is set to the singleton set containing the TCP/IP address of the ALG. This attribute is aggregated into internal domains in the same way as the *contacts* attribute, that is, through a union operator with a restricted output size.

An agent determines whether it is a representative for a firewalled site by monitoring the *contacts* attribute of the corresponding realm domain and noticing whether its contact is in there. When this becomes the case, the agent finds ALGs by traveling up the Astrolabe hierarchy starting in its realm and finding the *relays* attributes, stopping when it has located $k$ ALGs or when it reaches the root domain. To ensure fault tolerance, $k$ is typically chosen to be a small integer such as 2 (as in Figure 2). If no ALGs are found, agents resort to using a set of static built-in addresses of ALG servers that we deployed for this purpose.

For each ALG in the set, the agent generates a new extended address of the form (“domain-name@ALG”, “/:http”, preference), and adds this address to its contact set. The preference is chosen to be relatively low compared to its other addresses, so as to discourage its use. Finally, the agent sends an HTTP request to the ALG to receive the first message on this address.

When an agent determines it is no longer a representative, traffic to it using the ALG address will cease. When an agent is no longer a representative for a firewalled domain, *and* it has not seen any incoming traffic from the ALG server for some time (currently, two minutes), it terminates its TCP connection to the ALG and removes the ALG address from its contact set. In certain situations it is possible that traffic will remain arriving over the ALG connection indefinitely, however, so that the ALG connection never terminates (see Section 7.4).

## 7 Locating Peers

In an auto-configuring peer-to-peer system, peers have to find each other automatically. The same mechanisms should also unify partitioned components of a peer-to-peer system. That is, peer location is simply a specific case of partition repair. In Astrolabe, peers can be found using the following four mechanisms:

1. Direct gossip
2. Multicast and Broadcast
3. Configuration
4. Indirect gossip

We already described the first mechanism: when one agent X gossips to another agent Y, Y will learn about all the agents that X knows. Multicast and broadcast provide simple ways to discover peers automatically, but care need be taken not to create broadcast storms or response implosions. Using configuration and indirect gossip, connections that go beyond multicast and broadcast can be initiated. Below we will describe the latter three mechanisms in more detail.

### 7.1 Multicast and Broadcast

By IP broadcasting or IP multicasting a gossip message, a single sender can update the tables of any agent that receives the message. In order to avoid a large load of broadcasts and multicasts, the following policies are observed. For broadcast, each agent maintains a timer. When the timer expires, the agent broadcasts a gossip on its local LAN. The timer is reset each time the agent receives such a broadcast. The setting of the timer is slightly randomized in order to prevent synchronization effects that result in the agents

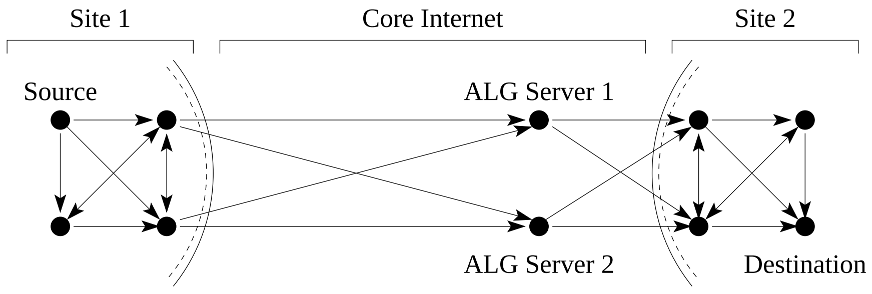

**Figure 2. The many ways gossip can travel from a source host in Site 1 to a destination host in Site 2. Each site has four hosts, two of which are representatives, behind a firewall. The representatives of Site 2 connect to two different ALG servers to receive messages from outside their firewall.**

broadcasting all at the same time. In the demonstration version, it is set so that a broadcast is generated once every 20 seconds on average.

For multicast, each agent makes use of the fact that any agent knows, approximately, how many agents there are in each domain. This is done by aggregating the *nmembers* attribute (initialized to one in the leaf nodes) by summation. In particular, the *nmembers* attribute of the root domain contains the approximate total number of agents in the Astrolabe hierarchy. Each agent multicasts with a rate that is inversely proportional to this number, so that the overall rate of multicasts is independent of the total number of agents. The demonstration version also uses a total rate of once every 20 seconds for multicasts.

Receivers of multicast and broadcast messages should not respond as they do with other gossip messages, as this could result in a large implosion of messages back to the sender, and it is unnecessary to do so. Receivers of such messages learn about the sender and its direct peers, and will from then on occasionally gossip back resulting in a complete merge. For example, say that A and B know each other, as do C and D. Also say that at some point D broadcasts a message, which only B receives. B will now gossip this information on to A, and gossip about itself and A back to D. Finally, D gossips this information on to C, and the merge is complete.

## 7.2 Configuration

In order to find peers that cannot be reached by multicast, agents can be configured to gossip with agents at well-known addresses. The addresses of ALG agents make an obvious choice for this purpose. The demonstration version of the Astrolabe agent comes with a set of built-in addresses of ALG agents that are located on the core Internet. When an agent starts up, it will first attempt to find other agents using built-in multicast and broadcast addresses. If an agent turns out to be a representative for the root domain, the agent will occasionally gossip to the well-known ALG agents in order to integrate with the global Astrolabe network.

## 7.3 Indirect Gossip

In practice, the multicast, broadcast, and configuration techniques described above did turn out to be insufficient. The problem is that our automatic address and realm generation makes mistakes. For example, “sulfur.cs.cornell.edu” may be a laptop, which is sometimes connected directly to the “cs.cornell.edu” network, but may also be connected from other places using VPN technology like PPTP [10], or even be connected to the Internet but without a PPTP connection set up. The automatic address and realm generation techniques, however, led an Astrolabe agent on the laptop believe it was in the firewalled domain of “cs.cornell.edu” (see Figure 3).

To illustrate what will happen, consider three agents /a/b/c, /a/b/d, and /a/e/f. Say the first two agents are each in their own firewalled realms, while the third is on the core Internet. As a result, the first two agents can communicate with /a/e/f, but not with each other. Thus /a/b/c will gossip to /a/e/f, claiming it is the only agent in /a/b. /a/b/d will do the same. The gossip protocol will cause /a/e/f to believe that /a/b has only one agent in it, when in reality there

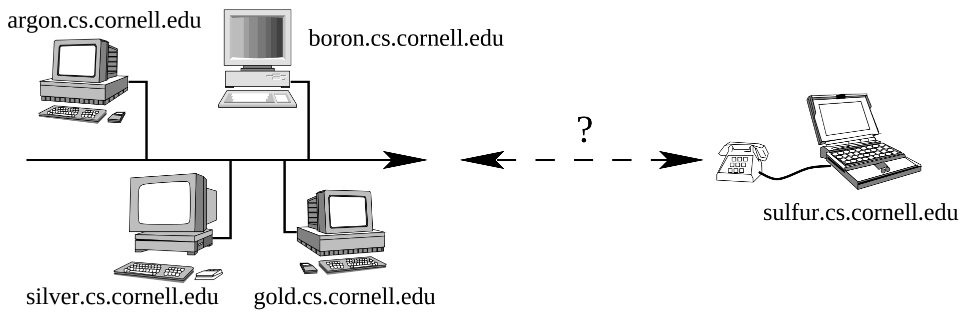

**Figure 3. Auto-configuration based on DNS may cause the laptop to think it is in the same realm as the other computers. However, multicast may be disabled so that discovery is not possible this way. Also, in the absence of a VPN, the only form of point-to-point communication may be through an ALG.**

are two. In general, the information gathered by /a/b/c and /a/b/d will never be aggregated together, resulting in two different versions of attributes for /a/b. /a/e/f will switch back and forth between these two versions as gossips from /a/b/c and /a/b/d arrive.

We considered fixing the automatic realm generation problem, but for two reasons we elected a different approach. The first reason is that we could not come up with a realm generation strategy that always works. It would have required extensive network topology discovery (*e.g.*, [4, 12]), which would always have been prone to mistakes. The other reason is that the approach described below increases the robustness of the Astrolabe gossip protocols significantly, and may solve further unanticipated problems.

The idea is that /a/e/f will gossip information about /a/b/d back to /a/b/c, and vice versa. Remember that /a/e/f already maintains for each domain a list of representatives. In particular, for domain /a/b, it maintains both /a/b/c and /a/b/d as representatives, in order to implement the replacement strategy of the merge algorithm described of Section 3. Originally agents would maintain only the timestamps associated with updates from representatives. We extended this by also maintaining the contacts of the representatives.

Now, when /a/e/f gossips to /a/b/c, it includes for domain /a/b the contact of /a/b/d. In general, when an agent X gossips to agent Y, X includes in the gossip message the contacts for all the domains that Y is in and that X knows about. When /a/b/c receives this gossip message, it finds out about /a/b/d, recognizes that it does not know about this representative, and gossips to it. Now that /a/b/c and /a/b/d know about each other and are communicating, they will merge their tables and aggregate the information for /a/b correctly.

## 7.4 A Problem Remains

There is still a remaining issue, which fortunately does not turn out to be a show stopper. Say there are many agents in /a/b, but that they are split up in two sites S and T that are separated by firewalls, so that an agent in S can only communicate with an agent in T by using an ALG. They should have been configured into two separate realms, but they are not. Indirect gossip through some other agent will allow them to locate one another, and form a single domain /a/b containing both the agents in S and the agents in T. Only the representatives of /a/b are required to receive messages through the ALG. There are two cases of interest: either all the representatives are in one of the sites S or T, or some are in S and some are in T.

In the first case, without loss of generality let us assume all representatives of /a/b are in S. All agents in T can send gossip messages to the representatives of S through the ALG, which are then gossiped on to the other agents in S. But it appears as if there is no way to send gossip messages to agents in T. Luckily, agents do not terminate their connection with the ALG unless they no longer receive messages through the ALG (see Section 6). As the only way for agents in S to send messages to agents in T is through the ALG, the agents in T that had ALG addresses will maintain their ALG addresses. Through *transitivity of gossip*, all updates will spread among all agents in /a/b. The second case is similar, and somewhat less problematic, as the representatives in S can gossip without restriction with the representatives in T.

Although this works in practice, the solution is not ideal. We are investigating if it is possible to analyze the address statistics that are kept at each agent to determine if such a configuration mistake has been made, in which we may be able to adjust the assigned addresses and realms dynamically.

## 8 Conclusion

In this paper, we presented how the Astrolabe service achieves efficient and scalable peer-to-peer communication, auto-configuration, and peer discovery. Firewall and NAT technology, and the lack of IP multicast support, make this complicated. The popular solution, the deployment of Application Level Gateways (ALGs), is inefficient and may lead to reduced robustness. In the solution implemented by Astrolabe, a novel addressing scheme allows peers to avoid using ALGs when possible.

Astrolabe's auto-configuration is based on mimicking the DNS domain hierarchy, although some adjustments had to be made in order to enforce Astrolabe's constraints on branching factors used in the domain tree. Astrolabe's architecture limits the number of ALGs that need be used, as only the representatives of firewalled domains need to interact through ALGs.

Peer discovery is accomplished through broadcast, multicast, and configuration, but primarily through Astrolabe's gossip protocols. An extension to these protocols, called *indirect gossip*, allows mistakes in the auto-configuration process to be masked and significantly increases the robustness of the gossip protocols.

Once up an running, the Astrolabe service itself may be of great use to other collaborative applications. For example, we have developed an application-level multicast and publish/subscribe service that works in the face of firewalls and NAT using Astrolabe. The service takes care both of efficient message routing and listener discovery, depending heavily on Astrolabe's aggregation functionality. It is described in [16].

## Acknowledgements

We would like to thank the following people for various contributions to this paper: Ken Birman, Tim Clark, Werner Vogels, Lidong Zhou, and the anonymous reviewers.

## References

[1] P. Almquist. Towards requirements for IP routers, Nov. 1994. RFC 1716, edited by F. Kastenholz.

[2] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of the Eighteenth ACM Symp. on Operating Systems Principles*, pages 131–145, Banff, Canada, Oct. 2001.

[3] M. Borella and G. Montenegro. RSIP: Address sharing with end-to-end security. In *Proc. of the USENIX Special Workshop on Intelligence at the Network Edge*, San Francisco, CA, Mar. 2000.

[4] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz. Topology discovery in heterogeneous IP networks. In *Proc. of IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000.

[5] B. Carpenter. Internet transparency, Feb. 2000. RFC 2775.

[6] R. Droms. Dynamic Host Configuration Protocol, Mar. 1997. RFC 2131.

[7] K. Egevang and P. Francis. The IP Network Address Translator (NAT), May 1994. RFC 1631.

[8] N. Freed. Behavior of and requirements for Internet firewalls, Oct. 2000. RFC 2979.

[9] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

[10] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol (PPTP), July 1999. RFC 2637.

[11] M. Holdrege and P. Srisuresh. Protocol complications with the IP network address translator, Jan. 2001. RFC 3027.

[12] B. Lowekamp, D. O'Hallaron, and T. Gross. Topology discovery for large Ethernet networks. In *Proc. of the '01 Symp. on Communications Architectures & Protocols*, San Diego, CA, Aug. 2001. ACM SIGCOMM.

[13] S. Maffeis, W. Bischofberger, and K.-U. Matzel. A generic multicast transport service to support disconnected operation. *Wireless Networks*, 2:87–96, 1996.

[14] A. Oram, editor. *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[15] S. Savage, N. Caldwell, and T. Anderson. The case for informed transport protocols. In *Proc. of the Seventh Workshop on Hot Topics in Operating Systems*, Rio Rico, AZ, Mar. 1999.

[16] R. van Renesse, K. Birman, D. Dumitriu, and W. Vogels. Scalable management and data mining using Astrolabe. In *Proc. of the First International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002.