BGP-based Clustering for Scalable and Reliable Gossip Broadcast *

M. Brahami[1], P. Th. Eugster[2], R. Guerraoui[1], S. B. Handurukande[1]

[1] Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)

[2] Sun Microsystems
Switzerland

Abstract. This paper presents a locality-based dissemination graph algorithm for scalable reliable broadcast. Our algorithm scales in terms of both network and memory usage. Processes only have "local knowledge" about each other. They organize themselves dynamically (right from the bootstrapping phase), according to join, leave or crash events, to form a locality-based dissemination graph. Broadcast messages can be disseminated using these graphs in large networks like the Internet, without relying on any special infrastructure or intermediate brokers. Roughly speaking, a dissemination graph consists of "non-crossing" (independent) trees that provide multiple paths between processes for improved broadcast efficiency and reliability. Each tree is constructed using BGP routing information about process "locality". We convey the feasibility of the algorithm using both simulation and experimental results and describe an application of our algorithm for broadcasting information streams.

Keywords: System design, Peer-to-peer communications, Content distribution, Multicast, Service overlay networks, Fault-tolerance, Broadcast streams.

1 Introduction

Traditional reliable broadcast algorithms [1] guarantee a very high level of reliability, despite message losses and process crashes, but scale poorly. Broadcast schemes like IP multicast [2] and MBone [3] are not widely available in the Internet and need specially configured routers (multicast routers and mrouted processes). Further more, these schemes do not cope well with message losses and process crashes.

Application level broadcast algorithms [4–10] have been recently proposed as a viable alternative. The general idea is to make use of some intermediate overlay network over the actual physical network, in order to cluster processes and achieve more efficient information dissemination. Such application level broadcast algorithms have good scalable and reliable properties. They can also be deployed easily in a large scale setting.

* *This work was sponsored by European Project PEPITO (IST 2001 33234).*

Many of these approaches exploit an abstract notion of "locality" in the clustering procedure in order to arrange processes according to network locality. Some approaches use centralized services [11, 12] to find "distance" between processes in the Internet. But, with a large number of processes, it becomes quickly infeasible to query a centralized service with all the existing process IDs to determine a "close" process for a given process. Round trip time (RTT) is also used to find "locality" information for clustering processes. But again this has the same draw back (the difficulty of finding RTT to many number of processes) as above with a large number of processes.

In this paper we present a locality-based broadcast algorithm. Our algorithm is scalable and provides a reasonable level of reliability. No specific infrastructure is needed (unlike for e.g., [3] and [13]). Processes have local knowledge about each other and self-organize (peer-to-peer based) in a dynamic content distribution scenario where processes join and leave in an ad-hoc fashion to form a dissemination graph. When a new process joins, it contacts one or more existing process/es to find a suitable place in the graph. In simple terms, this resembles a tree search to find a "locality" based suitable region. The notion of locality is based on BGP [14] routing information. Our scheme minimizes the usage of slow network links to reduce the delay incurred to deliver messages and reduce congestion in such links. It offers high level of reliability in a dynamic environment where processes leave the graph and crash. To achieve this, our graph building algorithm constructs "non-crossing" independent paths within the dissemination graph. We assume that there are few sources which broadcast with respect to the destinations. We show reliability properties using formal analysis and illustrate the feasibility of our approach through results obtained both with a prototype as well as with simulations.

In Section 2 we contrast our approach with related work. Our general broadcast architecture is presented in Section 3. The reliability of the scheme with independent trees is formally analysed in Section 4. A full algorithm which performs clustering and constructs dissemination graph with independent trees is described in Section 5. Section 6 presents simulation and prototype measurement results. Two possible applications of our general broadcast scheme is discussed in Section 7. Section 8 concludes this paper.

2 Related Work

BGP information is used in [15] to construct topological-aware Distributed Hash Tables (DHT). These DHTs are used to locate objects in an overlay network.

Application level broadcast have been widely described in the literature. We discuss here the efforts closest to ours.

In [7], several ways of arranging peers to form a hierarchy are mentioned, namely by selecting nodes either through 1) a random fashion 2) a round-robin fashion or 3) a smart-placement fashion (based on their network location). In a very large scale network, the network-oblivious schemes based on random and round-robin selections are obviously not adequate. A more efficient approach consists in selecting nodes using smart-placement. This is done in [7] through a centralized service. Though many such services are described (e.g., [11, 12]), to our knowledge, they are not available in the

Internet to be utilized as such. Even if such a service was available, it would not be clear how it could be used in a large scale setting.

Narada [4] is a multicast scheme with self-organizing capabilities. In this scheme, every member maintains a list of all other members, as well as a list of routing cost to every other member for paths between them. Then a per-source tree is constructed using an algorithm similar to DVMRP [16]. The scheme is very heavy in terms of memory and hence does not scale. Indeed, and according to the authors, the scheme is targeted towards medium size groups (with hundreds of members).

Scattercast [6] is based on an infrastructure (a set of servers known as agents) which needs to be deployed a priori. The agents construct an overlay network using a method similar to [4]. Individual clients are attached to close proximity scattercast clusters. The method used for automatic location of such a scattercast cluster is not presented (it is identified as a subject of future work by the author).

YOID [5] is an application level broadcast scheme which uses a general concept of "locality". However, no concrete hint is given on how locality information is determined when interconnecting processes.

Gossip-based broadcast algorithms [8, 17, 13, 18, 19] can also be viewed as application level broadcast schemes. While providing probabilistic guarantees on delivering messages, they have very good scalability properties and high resilience against failures and message losses. For instance, the algorithm presented in [13] is targeted toward small-scale WANs and relies on "gossip servers" which need to be setup and configured for each LAN manually. Hierarchical gossip-based broadcast algorithms presented in [18, 19] promote the idea of grouping processes (members) according to their locality, but no concrete way to exploit this locality is presented.

SplitStream [20], a content distribution scheme built on top of a DHT uses multiple paths between the content source and the destination. The topological placement of nodes is not done globally but limited to a few number of nodes (the "leaf set" of the DHT).

We describe in this paper a deterministic approach to message forwarding that limits the network usage. BGP-based clustering scheme can be applied both in a deterministic as well as in a randomized gossiping context.

3 Dissemination Architecture

In our dissemination scheme, the processes self organize in a peer-to-peer fashion to form a dissemination graph. The basic idea of our scheme is conveyed in Figure 1. Depending on their available resources (and possibly user-defined criteria), a subset of processes can be used to forward the messages they receive to other processes. The processes which are not capable of forwarding events to others (e.g., due to resource constraints) act as "pure clients" and receive events forwarded either by the source or by some other processes.

We construct a graph for message dissemination in a peer-to-peer basis. There is only one source for a given graph. For the sake of presentation simplicity, yet without loss of generality, we only discuss one such graph. The graph consists of independent trees connecting the source and all receiving processes. These trees are constructed

according to the network bandwidth between processes. Broadcast is achieved by forwarding messages along these trees.

Processes have parent-child relationships. A process P_1, which forwards the messages it receives to another process P_2, is called a *parent* of P_2 and P_2 is called a *child* of P_1. The number of possible children a parent can have is called the parent's *fanout*. The *fanout* of a process is chosen according to its capabilities in terms of computation and network resources. Our scheme aims at grouping processes which are "closer" in the Internet. This is achieved using BGP-based clustering as described below.

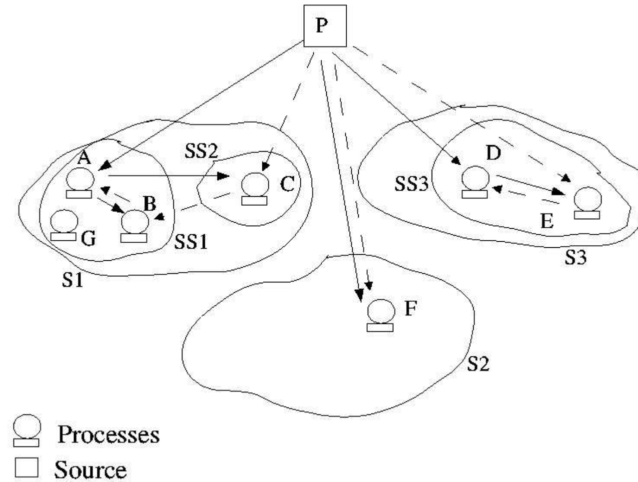

Fig. 1. Process-Assisted Broadcast.

3.1 BGP-based Clustering

Our clustering scheme relies on BGP [14] routing information. In particular, we use the notion of Autonomous System [14] (AS) to identify the segments of the Internet. Basically, an AS consists of a set of networks (a set of IP address blocks) which are managed by one administrative domain like an Internet Service Provider (ISP). Generally the networks inside a same AS have high bandwidth links connecting them. For example, the networks connecting universities in a country, large companies and organizations are typically in the same autonomous systems. In other terms, within such AS, individual networks and computers usually have high bandwidth links between them. Processes in a given autonomous system are arranged as neighbors in our dissemination graph.

It is possible for a process to find its AS using BGP information. This information is extracted using BGP routers or WHOIS servers [21, 22]. A service can be built on top of

BGP routers to provide this information directly from the lower level network in a real time fashion. On the other hand, there are WHOIS servers [22, 23] which provide BGP information in an off-line fashion. This service can be mirrored locally, for example at the source, to have a low response time. Also the network address of a process (which has an IP) can be obtained using such BGP routers and WHOIS servers. The network address represents more fine-grained segmentation of the network.

In this paper, we only consider the AS and do not use the network address. Each new process can know its corresponding AS when the process first contacts the service when joining the group. If a fine-grained division is required, the network address of a given process can also be used in the clustering. In such a case, processes are also grouped according to the local networks they belong to inside a given AS.

We also group ASs according to their countries. This helps a process to aggregate information about other processes: as a result, first, the size of the memory necessary to keep local information about neighbors is minimized. Second, there is a high probability to find a "closer" parent to a given process within the same country than from a different country.

These techniques can be used to divide the network into clusters and sub-clusters. A group of ASs in a country is called *cluster* where as individual ASs are *sub-clusters* within the cluster. This approach to find the "locality" can be efficiently applied in a large scale setting like in the Internet.

The clustering of processes promotes good communication between neighbors and minimizes the amount of "local" information about other processes without keeping global knowledge. It also helps a new process to find a suitable parent within an acceptable time duration without consuming too much computing and network resources. In other words, though there are many thousands of processes, the new process is provided with adequate information to select a suitable parent.

3.2 Parent Selection

Our scheme provides heuristic information for any new process help it to find a suitable set of parents. The source and a set of existing processes provide this information. To ensure scalability, the source and processes keep a minimum amount of information for providing this heuristic information. The source keeps the information about the clusters and the knowledge about a limited set of processes for each of these clusters. When a new process contacts the source, the source first checks the cluster to which the new process belongs. Then the source, using its local knowledge, checks if there is any process already in this particular cluster. If the source finds one or more processes which are already in this cluster, then the new process is provided with the IDs of those existing processes (if there are no existing processes in the cluster, the new process will join as a child to the source itself). The new process contacts these processes which are already in the cluster. Unlike the source, the processes inside a cluster have more precise information about other processes in their respective clusters. That is, inside a cluster each process knows to which sub-cluster it belongs. Also, processes know a subset of the processes in their sub-cluster as well as in other sub-clusters. As a result, if the new process p_n is in AS_n, then p_n will be redirected to any existing processes in

AS_n (details are in the Section 5). If there are no such existing processes in AS_n, p_n joins to some other existing processes.

The exact choice of a parent is made by the new process from the set of possible parents provided to the new process. In other terms, the new process measures the communication latency to each potential parent and also verifies the availability of an out-going link from each such parent. The parent is chosen such that it has an available out-going link with sufficiently low latency to the new process.

At this point, it should be noted that the trees can be re-arranged to make the scheme more efficient. For example, if a new process with a fanout greater than zero can not join because all the leaf processes of the tree are having a fanout of zero, then the tree will re-arrange by making the new process a new intermediate node in the tree.

Due to clustering, processes only keep a limited amount of information about other processes for the purpose of parent selection: this preserves scalability.

Example The parent selection can be elaborated using a simple diagram as shown in Figure 1. In this simple example, the processes belong to three clusters S1, S2 and S3. The source P knows about processes A, C, F, D and E from clusters S1, S2 and S3 respectively. (For the purpose of having a higher reliability, the source can know more than one process from each cluster). In cluster S1, there are two sub-cluster SS1 and SS2. Processes A and B are located inside SS1 while C is in SS2. Process C, which is also known by P, knows that process B is a child (of C) and belongs to sub-cluster SS1.

Suppose a new process G from SS1 wants to receive events from P. G first contacts the source P. P observes that G belongs to S1. As P already knows that A and C are in S1, P asks G either to join A and C, or obtain more precise information from them. Once G contacts C (or A), C observes that G is in SS1. G will be informed about B by C. Depending on the available resources, G joins A and B as a child. In a situation where none of the existing processes in a cluster can be assigned as a parent to a new process (for example, due to resource constraints), the tree will be re-arranged to make the new process an internal node in the tree (assuming the new process can forward messages to other processes) and previous leaf processes will remain leaf processes, in the newly arranged tree. If all the existing processes are not able to forward messages, and the new process also can not forward messages, the new process will join the source. A similar scenario applies to process F in S2 since F is the only process in that cluster.

4 Independent Trees

At this point it should be clear that the processes in the lower level of the dissemination tree rely on the proper functioning of the higher level processes which forward messages. If one such higher level process crashes, all the child processes of the crashed process will not receive events until the system reconfigures to construct the dissemination tree. The independent trees (i.e., non-crossing paths between the source and the receiving processes) are used to minimize the effect of such crashes and improve the efficiency as well as the reliability of the dissemination scheme. These multiple paths can be used in different ways depending on the nature of the application and the messages being broadcast. They are further discussed later in the paper.

Example Before continuing the elaboration of our scheme, let us consider a simple example. As shown in Figure 1, the source sends events along two separate paths (more than two paths are of course possible) for a given cluster, and hence any given process receives events along two different paths. For instance, process B in cluster S1 receives events via process A as well as C. In the case of failure (either A or C), B still receives a part or all events from one path (i.e., according to the configuration as discussed in Section 7).

When a new process joins, it must join multiple dissemination trees. For example, when G joins in Figure 1, it can select A as parent to receive events along one path and B as parent to receive events along the other path, provided that there are free out going links; if there are no outgoing links, the tree is re-arranged and the new process becomes an internal node (as described in the example of the previous section). Note that the new process G will receive messages from its own sub-cluster (from A and B) to minimize the transient traffic between sub-clusters. This reduces the message delay and congestion in links between such sub-clusters. Also, to *re-direct* G to B by process C, process C should keep some information about B and sub-cluster ID of B. These issues are described in Section 5. It is sub-optimal to have communication links between sub-clusters (like between SS1 and SS2) in these trees: but for a small number of independent trees and sufficiently large number of processes having non-zero out going links, such communication links between sub-clusters are limited.

When there is just a single process in a cluster as in S2, that process receives events on both paths from the source itself. As more processes join, the system reconfigures itself: for example, as in cluster S3. That is, in S3 process D and E exchange events that they do not receive directly from the source.

To guarantee the reliability of our scheme in the case of failures, two complementing paths should obviously not have common processes. The algorithm arranges processes according to the locality and rearranges them to make the scheme efficient as new processes join.

4.1 Reliability: An Analysis

The processes which take part in the message dissemination can be scattered all over the Internet and their behavior (in terms of join, leave and crash) is quite unpredictable. Either because the user terminates a process or due to failures, a process can be disconnected from the graph. Since many such processes act as parents, this might lead to form a disconnected tree causing inability to deliver messages to lower level processes in the tree. Of course, other trees in the graph could deliver messages and their operation is vital in such a scenario. We analyse the impact of such disconnections of trees on the reliability of the broadcast. For this, we use the following notions: 1) Mean Time Between Failures (MTBF) is the mean period of time a user may expect a given system to operate before a failure; 2) Mean Time To Recover (MTTR) is the mean period of time to recover the failed system (e.g., reconstruct a tree after disconnection). The availability of a system is defined as follows:

$$Availability\ (a) = \frac{MTBF}{MTBF + MTTR} \tag{1}$$

If the time is measured in minutes, the *down-time* per day, that is the mean time a given system is not available is :

$$Downtime = (1 - \ a) \times 24 \times 60 \tag{2}$$

Assuming the availability of a single tree is a, availability of k such trees, out of m trees in the graph, is A_k, where:

$$A_k = \binom{m}{k} a^k (1-a)^{m-k} \tag{3}$$

Hence the availability of at least k trees is α_k, where:

$$\alpha_k = \sum_{i=k}^{m} A_i = \sum_{i=k}^{m} \binom{m}{i} a^i (1-a)^{m-i} \tag{4}$$

For the proper functioning of the broadcast scheme, there should be at least one failure free tree at any given time. This tree can be used either to receive messages directly or to recover the messages (by retransmission from parent) once the message digests are received by a process (see Section 7 for more details). Availability of at least a single tree out of m trees is α_1, where:

$$\alpha_1 = \sum_{i=1}^{m} A_i = \sum_{i=1}^{m} \binom{m}{i} a^i (1-a)^{m-i} \tag{5}$$

Assume an extreme case where, for a given dissemination tree, MTBF is 15 minutes and MTTR is 4 minutes; that is a dissemination tree gets disconnected each 15 minutes on average due to a process leaving the graph and it takes 4 minutes to reconstruct the tree again (more on this is at the end of this section). Then using Equation 1 and 2 it can be seen that availability of a single tree (or any other scheme based just on a single path) is 0.7895 and the down-time is 303.15 minutes (around 5 hours) per day. Using Equation 5 and 2 it is possible to calculate the down-time of a system with m independent trees. For various values of m, the down-time per day is shown in Figure 2. For a dissemination scheme with 4 independent trees, the down-time is 2.82 minutes per day while with 6 trees it is 0.12 minutes per day.

This shows that, even in a very dynamic and unpredictable environment, where a trees fails at every 15 minutes (on average) due to processes leaving the graph, the reliability of the scheme can be improved considerably by having a few independent trees.

It should be noted that the reliability can also be improved by reducing MTTR. This can be done by localizing the reconstruction: that is, whenever a process crashes or leaves the tree, the neighbors of that process reconstruct the tree without involving all the processes in the tree. The construction of the entire tree takes more time than the localized reconstruction.

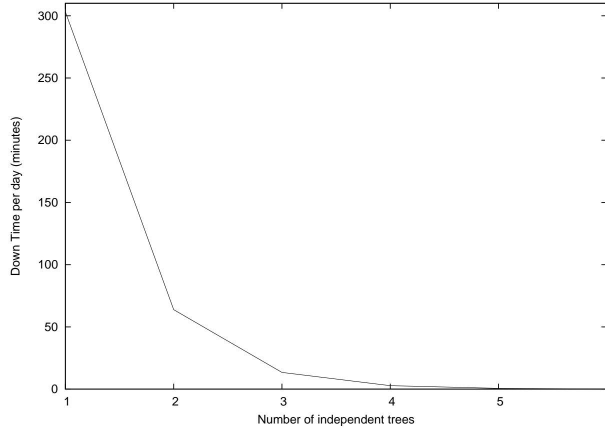

Fig. 2. Down-time of the broadcast scheme (with MTBF = 15min and MTTR = 4min)

5 Algorithm

In Figure 3 and 4, we describe the basic algorithm which constructs the dissemination graph. For presentation simplicity, we only show the major parts of the algorithm. When a process (potential receiver) joins a graph, the process sends a number of requests to the source (broadcaster) and then possibly to a set of other processes. It is important to note that the messages we refer to in this section are only *protocol* messages (e.g., *Join*, *SourceAccept*, *Accept*, etc.) that are used to construct the graph. They are not the actual application *broadcast* messages that is broadcast by the source. For all these *protocol* messages, the sender of a given message is denoted as P_s and the receiver is denoted as P_r.

The graph consists of m independent trees: each tree has a color to identify itself. A process receives messages from all the trees but only forwards messages from one tree, *T*. The color of a process is the color of that tree *T*. A child process that has the same color as parent *p*, is called a *direct* child of *p*. A parent has a list of all children denoted as *children_list* and list of all direct children denoted as *direct_c*. It is obvious that *direct_c* ⊂ *children_list* for its own sub-cluster.

A process is considered as *full* if it can not have -more- outgoing links. A process which has a parent from another sub-cluster than its own, is called a *sub-cluster head*.

5.1 Basic Sequence of Messages

We describe here, the basic sequence of messages shown in the algorithm (Figure 3 and 4). When first joining, the new process, P_n, finds its country and AS (autonomous system). Then P_n sends a *Join* message to the source: the source replies either with a *SourceAccept* or *Redirect* message. If an existing process P_e (other than the source) can accept P_n as a child, then P_e sends an *Accept* message to P_n. If a process detects

1: For connecting to the broadcast group:
2: **get country c and AS as**
3: send *join(c, as)* message to S

4: upon receiving a **Join(c,as)** from P_s by P_r
5: **if** P_s and P_r are in same AS **then**
6:   **if** P_r $\neg$ $full$ **then**
7:     P_r sends *accept(|children_list|,|direct_c|)* to P_s
8:   **else**
9:     P_r sends *redirect(direct children of P_r in same AS)*
10: **else**
11:   **if** P_r is a sub-cluster-head **then**
12:     **if** $\exists$ P_e $\in$ routing_table **such that** P_e.AS=as **then**
13:       P_r sends *redirect(P_e)* message to P_s
14:       **else** *Do as 15 to 18*
15:   **else if** P_r $\neg$ full **then**
16:     P_r sends *accept(|children_list|,|direct_c|)* to P_s
17:   **else**
18:     P_r sends *redirects(direct children of P_r)* to P_s

19: upon receiving a **accept(|children_list|,|direct_c|)**
20: set P_r.parent =P_s
21: **if** all m accept messages are received **then**
22:   **if** parent P_i of P_r is from another sub-cluster **then**
23:     select color of P_i
24:     P_r sends *Route(P_r_id,AS of P_r)*
25:   **else**
26:     select a parent who needs more direct children {*using parameter of the message*}
27:   P_r sends *SetColor(P_r_id,color of P_r)* to P_s

28: upon receiving a **Redirect(list)**
29: let P = set of potential parents
30: **if** P_s is the source **then**
31:   **for all** $p \in$ *list* **do**
32:     send *join(c,as)* messages
33: **else**
34:   $P \leftarrow P \cup$ list
35:   $t\leftarrow$ select one from P
36:   $P \leftarrow P \setminus t$
37:   send *join(c,as)* message to t

38: upon receiving a **SourceAccept(list, clr)**
39: set color to clr
40: **for all** $p \in$ *list* **do**
41:   send *join* messages

42: upon receiving a **Route(process_id, p_AS)**
43: **if** P_r is a sub-cluster-head **then**
44:   routing_table$\leftarrow$ (process_id, p_AS)
45:   forward *Route(P_r_id, p_AS)* to parent of P_r
46: **else**
47:   forward *Route(process_id, p_AS)* to parent of P_r

48: upon receiving a **SetColor(process_id, clr)**
49: $children_list \leftarrow children_list \cup$ (process_id, clr)

Fig. 3. Graph Building Algorithm: At every receiver process

```
1: upon receiving a Join(c,as) from P_s
2: let C=immediate children of S in country c
3: if |C| < m then
4:    C← C ∪ P_s
5:    for all p ∈ C  do
6:       clr← select color for p
7:       send SourceAccept(C,clr) message
8: else
9:    send redirect(C) to P_s
```

Fig. 4. Graph Building Algorithm: At the source S.

itself as a sub-cluster head, then it sends a *Route* message to its parent: all the processes forward this message to their parents. While sub-cluster heads alter this message before forwarding, others forward the message as it is. Once it is received by the source, the message is discarded. After receiving all m *Accept* messages, the new process P_n decides on its color: then P_n sends a *SetColor* message to its parents. The parents update their *children_list* and *direct_c* lists accordingly.

These messages (both requests and responses) and the tasks associated with them are shown in Figure 3 and 4. These request and response messages (simply stated as "messages" in this section) and associated tasks are described briefly next.

5.2 Messages

Join This message is sent by P_s to P_r when a process P_s needs to join a group. First the message is sent to the source. Then, depending on the response this message will be sent to other processes. The source might respond with a *SourceAccept* or *Redirect* message. Other processes respond with an *Accept* message or *Redirect* message. If the processes (not source) P_s and P_r are not in the same sub-cluster and P_r is a sub-cluster head, then P_r performs a routing table lookup in *routing_table* to find another process that is in the same sub-cluster as of P_s.

SourceAccept This message is sent by the source (as a response to a *join* message) when the source decides to accept a receiver process as its own child. That is, when there are less than m processes in a cluster C (country) at the bootstrapping (initial) phase, these messages are sent to construct the initial set of children and the graph. The information about the process ids (of source's children in the country C) and their corresponding color is also sent along with this message. Then each process sets its color according to the request of the source and sends a *join* message to all other processes as indicated by *process_ids*.

Accept This message is sent by a process P_s to another process P_r as a response to a *join* message when P_s decides to accept P_r as a child. If one parent is from another

sub-cluster, P_r selects the color of this parent. As a result, subsequent processes from the same cluster (as of P_r) can find a parent from within their own cluster. This reduces the number of links between the sub-clusters. Since such links are associated with greater delays (than links in the same cluster), by reducing such links, the efficiency is increased. The two parameters (number of elements in *children_list* and in *direct_c*) allow a child to decide its color in an efficient manner. Using this parameter it is possible to estimate whether one parent is not having adequate direct children (i.e., enough out-going links) of its own color. Then P_r can select the color of the process which does not have adequate direct children. In other terms, this selection criterion helps to have enough out-going links of each color.

Redirect This message is sent by P_s to P_r in response to a *join* message sent by P_r (who is looking for a parent). The parameter, *list*, is a set of possible parents to P_r. If P_s is the source, then ($|\text{list}| = m$) P_r will send *m* number of *join* messages to construct *m* trees. If P_s is not the source, then P_r explores each process in "list" to find a suitable parent by sending *join* messages to them. This join-redirect set of messages resembles a search in a tree to find a possible parent.

Route This message is used to update the routing information in sub-cluster heads, (a sub-cluster head is a process whose parent comes from a different sub-cluster than his own). These processes keep a small amount of routing information. As a result, given a process from a particular sub-cluster AS_i, the sub-cluster head knows whether there are any other processes in this AS_i, in the sub-tree below the sub-cluster head. If such a process exists, a new process which tries to join can be redirected appropriately. In short, this message helps to group processes of the same sub-cluster together. This message has a parameter $<process_id, p_AS>$, where p_AS is the sub-cluster which could be reached via $process_id$. Sub-cluster heads alter the $process_id$ and forwards the new *route* message to their parents.

SetColor This message is sent to P_r by P_s in response to an *accept* message indicating that P_r is selected as the parent. The parameter "color" indicates the color of P_s. P_r keeps information about ID of P_s and color of P_s.

For the presentation simplicity, the Figure 3 and 4 show only the major parts of the algorithm. The procedure of broadcasting (application message forwarding) is not shown in the algorithm; but this is simply done by forwarding messages along the edges of the trees by each process starting from source. The information a process stores (e.g., *children_list*) has the nature "soft state"; that is, these informations need to be refreshed periodically (e.g., in this case by children). In other terms, in order to be in the *children_list*, children need to inform parents about their presence periodically. This nature of "soft state" information enables to handle crashes and leaving of processes without any notification.

When a process (in particular an intermediate node in a tree) crashes (or leaves without any notification), one of the path (of a given color) in the dissemination graph is broken. Under such circumstances (once the children observe the crash of parent), after a timeout, the children of crashed process should initiate a procedure to reconstruct the broken path. The naive approach would be that these children contact the source

again to construct a path with the color of the broken one (other paths are unaffected). This naive solution is not optimal in terms of communication steps, but consumes less memory as it does not require additional information for the reconstruction procedure.

6 Performance

In this section we present results obtained from 1) a prototype implementation as well as from 2) simulations. We use the prototype to test the feasibility of BGP-based clustering in a real world scenario as well as to evaluate the performance of the dissemination scheme. The feasibility of clustering method in a larger scale is also tested using a simulation. We also use simulation to check the performance of the system with different values of fanout.

6.1 Prototype Implementation

In this real experiment, we used a source which broadcasts messages with the size of 1kb each to a total of 210 processes.

Setting A source publishes over a modem connection (56kbps) from country A and in autonomous system X. We used 150 processes in one country (A), and in autonomous system Y, while 60 other processes were in another country (B), and in the same autonomous system. A random value between 1 to 4 was chosen as the "fanout"(i.e., max. number of outgoing links) for each process. This simulates a real-life setting where the fanout of an individual process depends on its network bandwidth and user preferences. As described in Section 3, the country is specified for each process and the autonomous system is obtained using the WHOIS [21] service.

Results Table 1 summarizes the results obtained in this experiment. The details of these measurements are discussed next.

Country	A	B
Number of processes	150	60
Maximum depth	8	8
Average depth	5.3	5.7
Maximum join latency (ms)	1758	1302
Average join latency (ms)	952	1150
Minimum join latency (ms)	686	757
Maximum delay (ms)	213	267
Average delay (ms)	195	258
Minimum delay (ms)	177	182

Table 1. Communication between two Clusters.

Maximum and average depth: The depth of a process reflects the number of hops taken by a message before being delivered to that process. The maximum time taken to disseminate a message depends on the maximum depth of the dissemination tree used for its dissemination. The average depth in contrast is the average number of hops taken by messages before being received by processes.

Maximum and minimum join latency: In the dissemination scheme, a process joins a suitable parent in the join phase. As processes are redirected progressively starting from the source (see Section 3.2) to other processes, there is a certain latency when joining the dissemination tree. The maximum and minimum latencies are depicted by these two values.

Maximum, average, and minimum propagation delay: As each message is forwarded a given number of times by the processes, there is a delay before a message is received by each process. The maximum propagation delay is the largest delay incurred when receiving a message in the system. This delay occurs for the bottom most process in the dissemination tree. Similarly, minimum and average propagation delay represents the minimum and average delay incurred in the dissemination process.

6.2 Simulations

We performed a set of simulations to analyze the performance of our clustering scheme beyond the above (admittedly limited) setting involving only 2 countries and 3 autonomous systems. We were interested in finding 1) the maximum delay (in terms of hops) incurred due to successive forwarding of messages between processes, and 2) the impact of the fanout on the maximum delay.

Setting We simulated a set of IP clients which are globally distributed. To achieve this, we used a set of IP addresses of hosts which had recently accessed our laboratory web site. To group IP addresses into clusters and sub-clusters within each country, we used the WHOIS [21] service from [22].

We associated, -with each IP address-, a random integer f such that f is bounded by $1 \leqslant f \leqslant k$. The parameter f depicts the fanout or the number of out-going links from a process to other processes. We varied k such that k= 2,3,4,5 and did the simulations for each k. As a result, we represent processes which are capable of forwarding messages to up to k other processes as well as ones that can forward messages to only 1 other process.

Results For each value of k we constructed the dissemination tree as shown in Figure 1 (Y-axis = depth, X-axis = fanout) and found the maximum and average depth of the tree to estimate the maximum delay incurred due to hops when disseminating the messages. Since the delay incurred for each message depends on the number of hops the message has, it is critical to limit the number of hops. The maximum number of hops a message will have is equal to the maximum depth of the tree in our dissemination scheme. Since it is more general and appropriate to express the delay in terms of hops in end processes based (peer-based) systems, we used hops as the measure of the delay in our results. Figure 5 shows the delay for each value of k. Here, -for example- when k=2, the fanout of processes can have a value of 1 or 2.

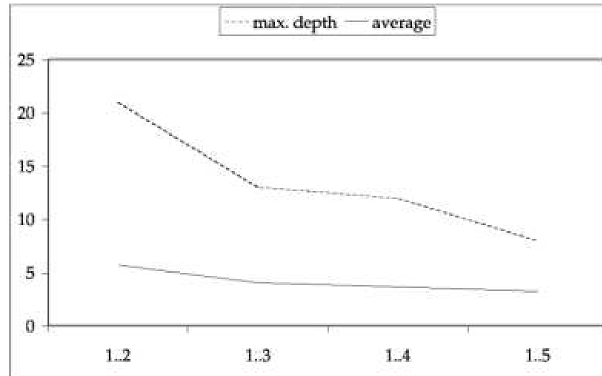

Fig. 5. Maximum Delay for Different Values of k.

7 Applications

In this section we describe two specific applications that can be built on top of our broadcast. First, we present a scheme suitable for streaming media. Second, we describe a general gossip-based broadcast that will reduce the network usage (by using message digests) while preserving the necessary redundancy to handle message losses and failures.

7.1 Broadcasting Stream Data

Stream data such as audio and video consists of samples which are broadcast in a fragmented fashion. These fragments, which form a "signal" (e.g., a video frame, an audio clip), are re-assembled at the receiver. Fragmentation can be done such that even if a set of fragments are lost, this does not necessarily invalidate an entire message (e.g., a video frame, an audio clip). There are number of coding schemes (e.g., [24, 25]) which can deliver adequate quality stream data in spite of high level of message loss. Our independent dissemination trees can use such coding schemes very efficiently to deliver stream data. These successive (or interleaving) fragments (packets) can be routed in a round-robin style over multiple trees.

In such a scenario, the crash of a path does not cause a complete loss of the broadcast. The crash of a path only degrades the quality of the signal until that path is reconstructed. This technique is hence applicable whenever the composition of single events/messages generates higher quality aggregated data.

In the context of streams, our scheme applies particularly well since the redundant trees are efficiently used: that is, the redundant trees do not disseminate duplicate messages but messages that can augment other messages.

7.2 Deterministic Gossip

The independent trees of the dissemination graph can be used to implement gossip-based dissemination scheme in a deterministic fashion. In other terms, as the simplest case, processes receive the same set of messages via independent and redundant trees (e.g., m different trees). The source can set the parameter m according to the level of redundancy required by the application to circumvent message losses and process failures.

Another approach is to send messages in k (where $k < m$; e.g., $k = 1$) trees and use other $m - k$ trees to send digests (i.e., message IDs) of those messages. As a result, when the system operates without process failures and message losses, a process receives actual messages k times and message digests from $m - k$ trees. In the case of k trees that send actual messages fail, the process still receives message digests from other independent trees. Under such circumstances, a process is aware that it is not receiving all messages that are being broadcast. Then (after a time-out) the process can ask for messages from its parent in the correctly functioning trees from which it receives the digest of the actual message. This recovery phase of messages is efficient since it is done using neighbors of a given process (localized recovery) instead of using the source itself. This method of deterministic gossip could help to reduce the amount of network usage while still maintaining good reliability properties by having redundancy.

8 Conclusion

This paper presents a scalable gossip broadcast algorithm with good reliability properties. Broadcast is achieved using a graph, consisting of processes grouped according to their locality. Processes (including the broadcaster) forward messages to a limited number of other neighbors. This number is defined according to their capabilities in terms of resources. The processes only know about limited number of other processes.

To group processes according to their locality, a clustering scheme based on BGP information is used. This scheme arranges processes in the Internet according to their "locality". Consequently, message delays between processes and transient broadcast traffic between large networks (autonomous systems) are reduced by localizing the majority of broadcast traffic within clusters and sub-clusters. The clustering scheme, together with the local knowledge of processes, help new processes find a suitable "place" within the graph by using few communication steps.

The clustering approach we use to arrange processes can be applied to various applications (e.g., peer-to-peer applications) and other broadcast algorithms such as [26, 27, 18]. Our dissemination graph with multiple independent paths is particularly suitable for broadcasting streaming data such as audio and video media.

The processes self-organize to construct the graph which consists of "non-crossing" (independent) trees. These trees, which evolve in a dynamic environment, are used to forward messages. As shown in the paper, even in extreme cases where processes leave the dissemination graph often, it is possible to have good reliability properties by limiting the down-time to a required level. We also convey the feasibility of our approach both using simulations and experimental results.

References

1. Hadzilacos, V., Toueg, S.: 5: Fault-Tolerant Broadcasts and Related Problems. In: Distributed Systems. 2nd edn. Addison-Wesley (1993) 97–145
2. Deering, S.: Host extensions for IP multicasting; RFC 1112. Internet Requests for Comments (1989)
3. Eriksson, H.: Mbone: The multicast backbone. Communications of the ACM **37** (1994)
4. hua Chu, Y., Rao, S.G., Seshan, S., Zhang, H.: A case for end system multicast. In: IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast. (2002)
5. Francis, P.: Yoid: Extending the internet multicast architecture. http://www.isi.edu/div7/yoid/docs/index.html (2000)
6. Chawathe, Y.: Scattercast: An adaptable broadcast distribution framework. In: Special issue of the ACM Multimedia Systems Journal on Multimedia Distribution. (2002)
7. Deshpande, H., Bawa, M., Garcia-Molina, H.: Streaming live media over a peer-to-peer network. http://dbpubs.stanford.edu/pub/2002-21 (2002)
8. Birman, K., Hayden, M., O.Ozkasap, Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems **17** (1999) 41–88
9. Li, Z., Mohapatra, P.: Hostcast: A new overlay multicasting protocol. In: Proceedings of the IEEE International Communications Conference (ICC). (2003)
10. Castro, M., Jones, M., Kermarrec, A.M., Rowstron, A., Theimer, M., Wang, H., Wolman, A.: An evaluation of scalable application-level multicast built using peer-to-peer overlay networks. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM). (2003)
11. Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., Zhang, L.: Idmaps: A global internet host distance estimation service. In: IEEE/ACM Trans. on Networking, Oct. 2001. (2001)
12. Theilmann, W., Rothermel, K.: Dynamic distance maps of the internet. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM). (2000)
13. Lin, M.J., Marzullo, K.: Directional gossip: Gossip in a wide area network. In: Proceedings of European Dependable Computing Conference (EDCC). (1999) 364–379
14. Rekhter, Y., Li, T.: A border gateway protocol 4 (bgp-4). RFC-1771, http://www.ietf.org/rfc/rfc1771.txt (1995)
15. L.Garces-Erice, Ross, K.W., Biersack, E.W., Felber, P.A., Urvoy-Keller, G.: Topology-centric look-up service. In: Proceedings of COST264/ACM Fifth International Workshop on Networked Group Communications (NGC). (2003)
16. Deering, S.: Multicast routing in internetworks and extended lans. In: Proceedings of ACM SIGCOMM. (1988)
17. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kermarrec, A.M., Kouznetsov, P.: Lightweight probabilistic broadcast. In: Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN 2001). (2001)
18. Gupta, I., Kermarrec, A.M., Ganesh, A.: Adaptive and efficient epidemic-style protocols for reliable and scalable multicast. In: Proceedings of 20th Symposium on Reliable and Distributed Systems (SRDS 2002). (2002)
19. Xiao, Z., Birman, K.: Randomized error recovery algorithm for reliable multicast. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM). (2001)
20. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in a cooperative environment. In: Proceedings of The ACM Symposium on Operating Systems Principles (SOSP). (2003)

21. Harrenstien, K., Stahl, M., Feinler, E.: Rfc 954: Nicname/whois. http://www.rfc-editor.org/rfc/rfc954.txt (1985)
22. Web, M.: The routing arbiter project. http://www.ra.net/ (2002)
23. Bourcier, P.: Cyberabuse. whois.cyberabuse.org (2001)
24. Cai, J., Chen, C.W.: Fec-based video streaming over packet loss networks with pre-interleaving. In: Proceedings of IEEE International Conference on Information Technology: Coding and Computing (ITCC '01). (2001)
25. Leslie, B., Sandler, M.: Packet Loss Resilient, Scalable Audio Compression and Streaming for Wired and Wireless IP Networks (White Paper). (2002)
26. van Renesse, R., Birman, K., Vogels, W.: Astrolabe: A robust and scalable technology for distributed systems monitoring, management, and data mining. ACM Transactions on Computer Systems **21** (2003)
27. Eugster, P.T., Guerraoui, R.: Probabilistic multicast. In: IEEE International Conference on Dependable Systems and Networks (DSN 2002). (2002)