Exploiting IP Multicast in Content-Based Publish-Subscribe Systems

Lukasz Opyrchal[1], Mark Astley[2], Joshua Auerbach[2], Guruduth Banavar[2], Robert Strom[2], and Daniel Sturman[2]

[1] Dept. of EECS, University of Michigan,
1301 Beal Avenue, Ann Arbor, MI 48109, USA
[2] IBM T.J. Watson Research Center,
30 Saw Mill River Rd., Hawthorne, NY 10532, USA

Abstract. Publish-subscribe systems are evolving toward using content-based subscription rather than subject-based subscription. A key problem in implementing such systems is that a straightforward mapping from matching sets to multicast groups produces a number of groups that rapidly grows beyond practical limits. This paper proposes a set of alternative algorithms for solving this problem, by: (1) using a smaller set of overbroad multicast groups, judiciously chosen to minimize imprecision; (2) issuing multiple multicasts to appropriately chosen clusters; or (3) sending an event over multiple hops each involving a multicast to a set of neighbors. We evaluate these algorithms on a simulated wide-area network. We find that (1) a simple flooding algorithm is viable over an extensive range of conditions; and (2) under conditions of high selectivity and high regionalism of subscriptions, the other approaches mentioned above perform significantly better; however, the specific algorithm to use depends upon the economics of deployment.

1 Introduction

Publish-subscribe systems provide a convenient approach for interconnecting applications on a distributed network. Publish-subscribe middleware is currently being deployed for application integration in many domains including financial, process automation, and transportation. In the publish-subscribe paradigm, information providers publish units of information called *events*, and information consumers subscribe to particular categories of events. The middleware ensures the timely delivery of published events to all interested subscribers.

The earliest publish-subscribe systems used *subject-based* subscription. In the past decade, systems supporting this paradigm have matured significantly, resulting in several academic and industrial strength solutions [3,16,18,19,21]. In subject-based subscription, each event is classified and labeled by the publisher as belonging to one of a fixed set of subjects (also known as groups, channels, or topics). Consumers subscribe to all the events within a particular subject or set of subjects. Except for the subject identifier, the information content of events is opaque to the middleware. A strength of this approach is the potential to easily leverage group-based multicast techniques to provide scalability and performance, by assigning each subject to a multicast group.

An emerging alternative to subject-based subscription is content-based subscription [1,2,20]. These systems support an *event schema* defining the type of information contained in each event. For example, applications interested in stock trades may use the event schema `[issue: string, price: dollar, volume: integer]`. A content-based subscription is a predicate against the event schema, such as `(issue="IBM" & price < 120 & volume > 1000)`. With content-based subscription, subscribers have the added flexibility of choosing filtering criteria along multiple dimensions, without requiring pre-definition of subjects. In our stock trading example, a subject-based subscriber is forced to select trades by issue name. In contrast, a content-based subscriber is free to use an orthogonal criterion, such as volume, or indeed a collection of criteria, such as issue, price and volume.

While content-based subscription is the more general and flexible paradigm, providing efficient and scalable implementations of such systems is still an open problem. In particular, existing group-based multicast techniques cannot readily be applied to this problem. Each subscriber may have a unique subscription, and therefore, each event may go to a widely varying group of subscribers. To naively map these subscribers into groups may require a number of groups exponential in the number of subscribers (i.e. 2^N).

In this paper, we explore a number of approaches for exploiting group-based multicast for event delivery in content-based publish-subscribe systems. In particular, we focus on being able to exploit widely available, best effort multicast such as IP Multicast [8], or reliable multicast techniques built on top of IP Multicast such as SRM [9].

We explore three approaches to reducing the number of groups needed: (1) reducing precision: i.e., sending to overly broad groups where brokers may receive events for which they have no client subscriptions, (2) multiple sends: i.e., sending an event on multiple multicast groups instead of making a single multicast, or (3) multi-hop routing: i.e. sending an event over a set of multiple hops each of which entails a multicast to a set of intermediate brokers.

We define and evaluate five algorithms – traditional flooding, plus four newly proposed algorithms – each of which exploits one or more of the above approaches. Each of the techniques we present in the paper is compared to an abstract algorithm which we call “ideal multicast.” Ideal multicast assumes that a perfect multicast group can be determined for each event. Ideal multicast provides a lower bound on network bandwidth utilization and latency.

We evaluate these algorithms on a simulated wide area network (WAN). This network consists of 100 multicast-enabled routers supporting 88 publish-subscribe servers (a.k.a. *brokers*), which include eight brokers with publishers and 80 brokers with a total of 10,000 subscribers.

The remainder of the paper is organized as follows. In Sect. 2, we describe all the evaluated algorithms. In Sect. 3, we provide details of the simulation setup that we use to evaluate the various algorithms, and we summarize our findings.

In Sect. 4, we review some of the previous work on event distribution systems using content-based subscription, and on applications of group multicast. Finally, Sect. 5 discusses conclusions of these experiments and suggests future directions for our work.

2 Group Multicast Algorithms

As mentioned earlier, the naive use of group-based multicast for implementing content-based publish-subscribe may require as many as 2^N groups where N is the number of communication end-points. Rather than treating each subscribing client as a communications end-point, we assume that the communication end-points are *brokers*, which are servers that manage client connection and event distribution. Brokers reduce the complexity of routing events by reducing the total number of endpoints known to the distribution system. Each end-point broker performs a local matching operation before forwarding the event to the subscribing clients. The local matching operation determines the set of interested clients connected to that broker. An implementation of matching for content-based subscription is described in [1] and shown to take time sub-linear in the number of subscriptions.

The current IPv4 specification for IP Multicast provides a maximum of 2^{24} locally scoped multicast addresses. The practical limit is smaller, since routing table space in backbone routers is a scarce resource [4]. Thus, it is important to reduce the number of groups needed. However, multicast technology is evolving rapidly, so it is difficult to know how few is "few enough." Rather than setting an arbitrary limit (other than the architected limit of $2^2 4$), we examine ways to reduce the number of groups needed for a given number of brokers, favoring approaches that use fewer groups over those that use greater numbers. We explore three general approaches to reducing the number of groups needed.

1. *Reduce group precision.* In this approach, events are sent to multicast groups that may contain brokers that do not have subscriptions for the event. In the extreme case, messages are sent to all brokers (Sect. 2.2). Another way to reduce precision is to combine groups to form larger groups until the number of groups is within an acceptable limit. This approach is explored in Sect. 2.6.
2. *Send multiple multicasts.* In this approach, the set of end-points is divided into mutually exclusive subsets, thereby reducing the total number of required groups. For example, if N endpoints are divided into two equal subsets, the number of groups required in each subset is $2^{N/2}$, and the total number of groups required is $2 \times 2^{N/2}$. However, each event must be sent to two groups in this case. This approach is explored by the algorithm in Sect. 2.3.
3. *Send over multiple hops.* In this approach, each publisher sends to a small subset of neighboring brokers, which in turn forward the event to their neighbors, and so on. This approach is explored in Sect. 2.5.

Hybrid approaches that combine more than one of the above approaches are also possible; one such algorithm, explored in Sect. 2.4, combines approaches 1 and 2 above.

2.1 The Ideal Algorithm

In an environment where we could have as many groups as we need, we could assign a multicast group to every required subset of the set of brokers. Every such group may be reached using a single multicast, and every event published is always sent to the group which contains exactly those brokers subscribing to the event. We call this the *Ideal* algorithm.

Of course, for any system with non-trivial size, the ideal algorithm requires an impractical number of multicast groups. This makes the ideal algorithm useless in practice. Nonetheless, the ideal algorithm provides a useful benchmark for evaluation – we expect the ideal algorithm to provide a lower bound on the performance of each of our multicast strategies.

2.2 Flooding

A simple solution to the problem of content-based routing is to send every published event to all brokers. In this approach, only one multicast group is needed consisting of all the brokers in the system.

A simple optimization to avoid sending events that do not match any subscribers is to first perform a matching operation at the publishing broker against all subscriptions. The additional overhead of this matching step (on the order of 100 microseconds for 10,000 subscriptions) is not significant relative to overall network latencies (on the order of a hundred milliseconds).

2.3 Clustered Group Multicast (CGM)

The CGM algorithm is based on the use of *clusters*: mutually exclusive subsets of brokers where each subset has its own set of multicast groups. We observe that if we divide N endpoints into 2 clusters, we reduce the number of groups in each cluster to $2^{N/2}$ groups, and the total number of groups to $2 \times 2^{N/2}$. The cost of this approach, however, is that it may be necessary to multicast an event twice: once to a group in each cluster. In general, if we divide N into C clusters, the total number of groups needed is given by $g = C * 2^{N/C}$. Figure 1 shows, for a given number of groups and number of clusters, the number of endpoints that can be supported. For example, if we have 2^{13} multicast groups available, we can support 80 broker end-points by dividing them into 8 clusters of 10 brokers each. Since the groups within a cluster enumerate all possible combinations of brokers, each broker must join half these groups (those that include the broker) at system configuration time.

Each broker contains an instance of the subscription matching engine with entries for all client subscriptions in the system. When an event is published,

Max Groups	# Clusters	# Endpoints
2^{24}	8	168
2^{24}	4	88
2^{17}	8	112
2^{17}	4	60
2^{13}	8	80
2^{13}	4	44

Fig. 1. Number of endpoints supported by CGM

the publisher's broker matches the event against all subscriptions, and sorts the resulting list of brokers by cluster. It then looks up the group in each cluster that contains exactly those brokers destined to receive the event. The publisher's broker then performs up to C multicasts, where C is the number of clusters. Some clusters may have no matching brokers and are therefore skipped.

The choice of cluster assignment has a significant impact on performance. For example, brokers that match a single subscription, but that are spread over multiple clusters will require multiple multicasts. One approach builds clusters by grouping brokers with similar subscription sets. Another uses geographic (or network) location data to group brokers into clusters. The algorithms described here use the latter approach.

2.4 Threshold Clustered Group Multicast (TCGM)

The CGM algorithm described above requires a number of groups that may be prohibitively large for many applications. The number of groups required may be reduced by reducing the precision of the algorithm. One approach to reducing the precision is to flood a cluster when more than a threshold number of brokers within that cluster need to receive an event. That is, the algorithm behaves like CGM unless the number of destinations in a cluster exceeds a threshold, at which point the event is multicast to the entire cluster. We call this algorithm Threshold CGM (or TCGM).

For each cluster, we pick a threshold $T < K$, where K is the size of the cluster. If an event matches more than T endpoints, the event is sent to all brokers in one cluster. Otherwise, the event is sent only to the brokers subscribed to the event (as in CGM). This algorithm requires multicast groups for all subsets of brokers in a cluster of size T or smaller, plus one additional multicast group for all brokers in the cluster. A closed form expression for the number of groups required is given in Fig. 4. Figure 2 compares group requirements for CGM and TCGM for three different values of threshold T, and for different numbers of brokers and clusters. The group requirement for TCGM is many orders of magnitude smaller than in the case of CGM.

Nodes	CGM	TCGM, T = 5	TCGM, T = 4	TCGM, T = 3
168, 8 clusters	16,777,216	223,168	60,376	12,496
112, 8 clusters	131,072	27,784	11,768	3,760
112, 4 clusters	1,073,741,824	489,756	96,632	14,732
88, 4 clusters	16,777,216	141,776	36,436	7,176

Fig. 2. Group requirements of CGM vs. TCGM

2.5 The Neighbor Matching Algorithm

The neighbor matching algorithm is derived from our earlier work [2]. In this approach, each broker designates a number of nearby brokers as "neighbors." Each broker performs just enough tests of the event content to determine which subset of its neighbors are on the next hop to a final destination broker.

There is one major difference between the earlier work and the use of neighbor matching in this paper: In the earlier work, we assumed a point-to-point link to each neighbor (which is why in that work, the algorithm was named "link matching"). In this paper, we are assuming that there is a multicast group for each possible combination of neighbor brokers. When an event arrives at a broker, the broker computes the set of brokers on the "next hop" and forwards the event to the corresponding group.

There are a number of potential advantages of this approach. First of all, it is more scalable as the number of brokers in the system grows. Each broker has to know about only its immediate neighbors, not about all the brokers. For k neighbors, a broker can have a maximum of 2^k groups. Furthermore, the knowledge of those group names does not need to be widely disseminated; only neighbors need to subscribe to a group.

The disadvantages of the approach are that there is extra processing required on brokers, extra bandwidth required on the links between brokers and the network, and potential extra delay from publisher to subscriber because of the extra hops required.

2.6 Group Approximation Algorithm

The group approximation algorithm is a single multicast approach which reduces the number of groups required by combining actual groups to approximate groups. This approach reduces precision because an approximate group often contains a superset of the brokers which match an event. That is, some brokers may receive *waste* events: events which do not match any subscription held by a broker. The volume of waste events can affect system performance. Thus, an important aspect of this technique is to construct approximate groups which minimize the volume of waste events received by each broker, given a fixed number of multicast groups.

One way to choose approximate groups is to make use of the information contained in the subscriptions stored at each broker. In particular, we may require a separate group for each disjoint *matching set* entailed by a collection of subscriptions. The matching set of a subscription is the set of events which satisfy the constraints of the subscription. Figure 3 gives example subscriptions and corresponding matching sets. Note that the ideal algorithm creates a multicast group for each disjoint matching set.

The intuition behind the group approximation algorithm is the observation that, in systems with large event schemas, many groups have a relatively low probability of receiving events. Therefore, combinations of such groups also have a relatively low probability of introducing wasted events.

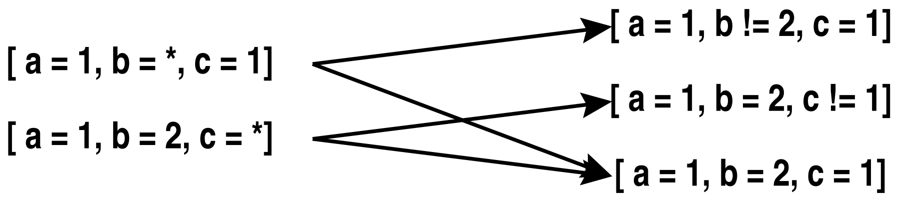

Fig. 3. Example subscriptions (on the left) and their corresponding disjoint matching sets (on the right) for the simple event schema `[a: integer, b: integer, c: integer]`. The events in the first set match the first subscription, events in the second set match the second subscription, and events in the third set match both

The group approximation algorithm operates as follows. Let g be the desired number of groups, then:

1. Determine the set of required multicast groups and their probability of receiving an event.
2. Combine pairs of groups until there are no more than g groups in the system.

The choice of groups to combine at each step has a significant impact on the waste generated by approximation. This waste may be characterized as follows. Given a multicast group G_i, define:

p_i	The probability that G_i will receive an event.
l_i	The loss factor of G_i. That is, the expected number of events wasted for each multicast to G_i.
b_i	The set of brokers with a subscription containing the matching set represented by G_i.

The expected waste induced by a group G_i is $p_i \times l_i$. The *net waste induced* (NWI) by combining two groups G_1 and G_2 is given by the expression $\text{NWI}(1,2) = (p_1 + p_2) \times l_{1,2} - p_1 \times l_1 - p_2 \times l_2$ where $l_{1,2}$ is given by:

$$l_{1,2} = \left(\frac{p_1}{p_1 + p_2}\right) \times (|b_2 - b_1| + l_1) + \left(\frac{p_2}{p_1 + p_2}\right) \times (|b_1 - b_2| + l_2)$$

and $|b_i - b_j|$ is the number of brokers in set b_i but not in b_j. Note that for the combined group $G_{1,2}$ we also have $p_{1,2} = p_1 + p_2$ and $b_{1,2} = b_1 \cup b_2$, where the former follows from the fact that G_1 and G_2 represent disjoint matching sets. Reducing the equations above, NWI$(1,2)$ may be expressed as $p_1 \times |b_2 - b_1| + p_2 \times |b_1 - b_2|$.

Typically, the set of disjoint matching sets is exponential in the number of subscriptions (several million for the simulations described in the next section). Moreover, the order in which we combine groups is significant. Therefore, an ideal group reduction involves a search over all possible orders of combining groups, and is therefore exponential in the size of the initial group set. Thus, heuristics are the only practical approach for deriving approximate groups using the expressions above. However, even in the case of polynomial heuristics, the exponential size of the initial group set is still a limiting factor[1].

In this paper, we use a hybrid approach where we approximate the set of initial groups, and then use a heuristic to reduce to a final group set. We approximate the initial group set by reducing the selectivity of subscriptions by eliminating rare attributes of the schema. We then combine groups to form an approximate group set by first sorting the initial groups from least to greatest according to probability of receiving an event. Groups with the same probability are further sorted from greatest to least according to the expression $|b_i| - l_i$. We then compute NWI(i,j) for each combination of the first 100 groups, and combine the pair with the minimal net waste induced. The combined group is reinserted and the algorithm is repeated until we have reduced to the desired number of groups.

The motivation for sorting the groups is that groups with small probability pay less of a penalty for non-optimal combinations. In the case of the second sorting term, the intuition is that groups with many members but little waste are more likely to overlap in a productive manner. As a further check, we find the best pair of groups to combine by considering the first 100 groups, rather than simply combining the first two groups in sorted order. This algorithm is $O(log\ n)$ for each combination step.

Note that some error is introduced by only considering a subset of subscription attributes. In particular, it is possible to discover an actual group at run-time which has no corresponding approximate group. In this case, we dynamically map the actual group to the smallest approximate group which is a superset of the actual group.

[1] One simple heuristic is to use a greedy algorithm which combines pairs of groups with minimal NWI. This algorithm is $\approx O(n^3)$ which is still prohibitively expensive for group sets with size in the millions.

2.7 Summary

Each of the algorithms described above makes specific tradeoffs in order to exploit multicast. While a common goal is to reduce the number of groups, there are several other criteria by which these algorithms may be categorized:

- *Precision:* Defined as the ratio $\frac{\#\ matched\ brokers}{\#\ receiving\ brokers}$. Precision is one measure of the amount of waste in the system.
- *Number of Multicasts:* The number of multicast sends required in order to distribute an event.
- *Total Number of Groups:* A bound on the total number of groups required in the system.
- *Groups Per Broker:* A bound on the number of groups each broker is required to join.
- *Configuration:* The stage at which multicast groups must be created. "Static" means that groups can be created before the subscription set is known.
- *Manageability:* An indication of the complexity and ease of management of a particular algorithm.

	Ideal	**Flooding**	**CGM**	**TCGM**	**Neighbor**	**Approx**
Precision	1	$\mathcal{P}_b$	1	$[\frac{T+1}{K}, 1]$	1	$1 - \frac{W}{N}$
# Mcasts	1	1	$1...C$	$1...C$	1 per hop	1
Groups	2^N	1	$C \times 2^B$	$C \times \sum_{i=0}^{T} \binom{B}{i}$	$N \times 2^k$	Configurable
Grps/Broker	2^{N-1}	1	2^{B-1}	$\sum_{i=0}^{T-1} \binom{B-1}{i}$	$k \times 2^{k-1}$	Variable
Config.	Static	Static	Static	Static	Static	Dynamic
Manag.	Hard++	Trivial	Moderate	Moderate	Moderate	Hard

Fig. 4. Summary of event distribution algorithms where N is the number of brokers, k is the average number of neighbors of each broker, $\mathcal{P}_b$ is the probability that an arbitrary broker will match an event, C is the number of clusters, B is the number of brokers in each cluster, T is the threshold value for TCGM, and W is the total waste induced by the group approximation algorithm

Figure 4 summarizes the characteristics of each of the algorithms under consideration. Note that the ideal algorithm is infeasible to implement in most systems and is only presented for comparison purposes.

3 Evaluation

We have implemented the multicast algorithms described in the previous section and tested them on a simulated network topology. The goals of our simulations were:

1. To measure the bandwidth utilization characteristics of the algorithms we developed as well as the simple flooding algorithm and the ideal algorithm.
2. To measure the latency characteristics for the same set of algorithms. We define latency as the delay from the time an event is published to the time it is delivered to a subscribing client.

It should be noted that if subscriptions are uniformly distributed over a geographic region, then for a high enough probability of match between a random event and a random subscription, and a small enough set of brokers, it follows from straightforward probability theory that most events will be required by all brokers, and thus the behavior of ideal multicast and the behavior of flooding will be the same. Therefore, we concentrate on evaluating other algorithms only where these conditions do not occur, or in other words, where the following conditions do occur:

1. *High selectivity.* The subscriptions are sufficiently selective that the average probability of a match is very low; or
2. *High regionalism.* The subscriptions are sufficiently non-uniform that certain kinds of events will have high interest in certain parts of the network and low interest in other parts of the network.

3.1 Simulated System

We simulate an eighty-eight broker publish-subscribe network deployed across a WAN. The WAN topology used in the simulations was generated using the Georgia Tech Internetwork Topology Models [5]. We used the transit-stub topology model [26] which approximates wide-area networks. The generated topology is shown in Fig. 5. It consists of three kinds of nodes: eighty broker nodes with only subscribing clients (rectangles), eight broker nodes with only publishing clients (double circles), and one hundred multicast-enabled router nodes (circles). Links between these nodes are of three types: backbone links (bold lines) that are OC-12 class (622Mbit), intermediate links (normal lines) that are OC-3 class (155Mbit), and fringe links (dotted lines) that are high-speed LAN class (100Mbit). Latencies are labeled on individual links.

The multicast routers in the network are state of the art *wire-speed* routers. That is, they are able to forward messages at the maximum bandwidth of their incoming links. However, for each outgoing link, there is an output queue for messages that are yet to be consumed by that link. Routers and links in the network are also loaded with traffic unrelated to publish-subscribe traffic. This ambient load is 25% of the link capacity on average, uniformly across the network[2].

Each of the eighty brokers with subscribing clients has twenty five clients connected to it, with an average of five subscriptions per client (giving 10000 total subscriptions). Subscriptions are generated randomly using an event schema

[2] We leave the study of a more realistic non-uniform ambient load as future work.

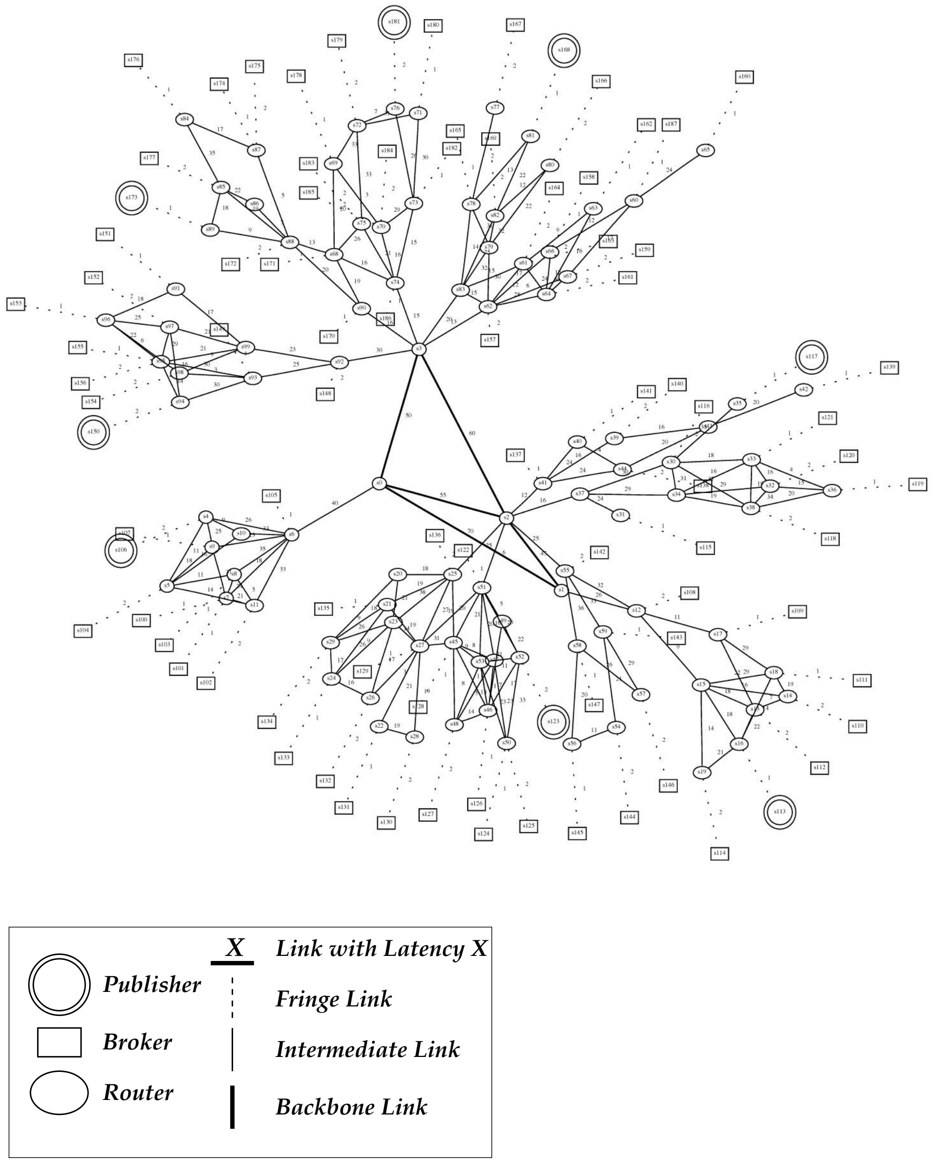

Fig. 5. Simulated network topology

with fifteen attributes, where each attribute has four possible values. For each attribute, a subscription gives either a concrete value chosen from a Zipf distribution, or a "don't care" value, which matches events with any value for the attribute. Subscriptions are generated randomly in such a way that the first attribute is a concrete value with probability of 0.98, and this probability decreases from the first to the last attribute. We vary the rate of this decrease to obtain results for different subscription match rates. For example, if the probability that an attribute is a concrete value decreases at the rate of 78%, each event matches about 2.24% of subscriptions. If the probability of a concrete value decreases at the rate of 88%, each event matches about 0.21% of subscriptions.

There are also 8 publishers in the network that publish events tracked by the simulator. Events are generated randomly, with attribute values in a Zipf distribution. Events arrive at the publishing brokers according to a Poisson distribution with mean arrival rate of 200 μs. The size of each event is 1KB.

When an event is published, it is matched at the publishing broker and then one of the previously described algorithms is used to forward it towards other brokers. Along the way, it incurs latency delays along different links as well as queuing delays at router output queues. Receiving brokers also perform a matching operation before forwarding to clients or to other brokers (in the case of neighbor matching). The brokers' CPU utilization for performing the matching operations is also modeled.

Simulations were run for all multicast algorithms described above with each run consisting of 5000 published events. In all cases, this number of events guarantees less than a 1% error rate (with 99% confidence) for the bandwidth measurements.

Additional Setup for Specific Algorithms

For the purposes of the neighbor matching algorithm of Sect. 2.5, the topology described above also specifies a "neighbor" relation between brokers, as shown in Fig. 6. Each circle in Fig. 6 corresponds to one of the broker nodes (rectangle or double circle) in Fig. 5. Latencies between neighbors represent latencies on the shortest path between corresponding brokers. In the particular experimental configuration tested here, we assign neighbor relationships based upon proximity in the network topology. We limit the number of neighbors so that the total number of groups used in the system is approximately 2^{13}. We chose 2^{13} to match the number of groups used in the simulation of CGM-8 and in the group approximation algorithm. Because no broker needs to know about any groups other than those used by its immediate neighbors, the number of groups known to any one broker is small — on the average a broker needs to know 100 groups to which it can send, and needs to join 325 groups from which it can receive an event.

For the purposes of the CGM algorithm of Section 2.3, we manually assigned each broker to one of the required number of clusters, based on its geographical location and its proximity to other brokers in the same cluster.

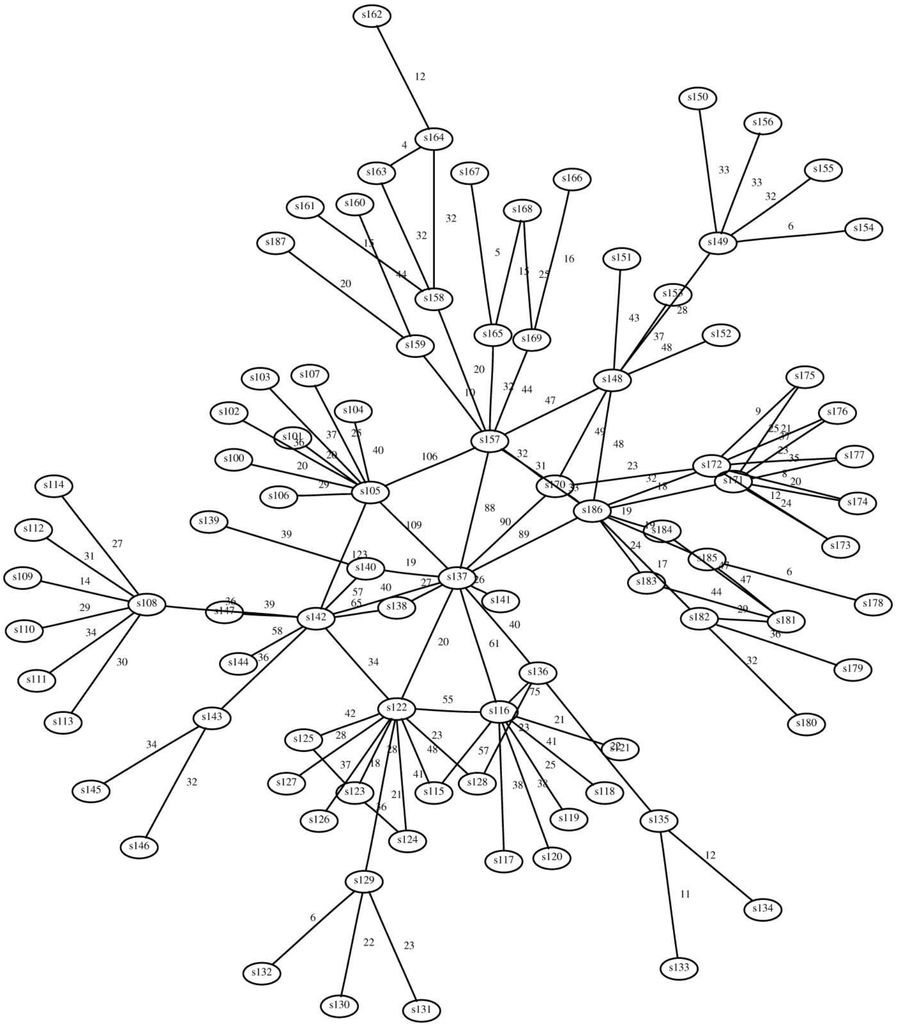

Fig. 6. Broker "neighbor" relations

Subscription Distributions

We ran simulations for two kinds of subscription distributions: non-regional, and regional. In the non-regional distribution, the subscriptions are assigned randomly with a uniform distribution, without regard to the location of the subscribing client. In the regional distribution, topologically nearby clients are more likely to be interested in the same events. This is achieved by assigning one attribute of the event schema as the "regionalism" attribute. The value of the regionalism attribute is a number between one and four, corresponding to its "cluster" as determined by the CGM clustering algorithm (with 4 regions) described above. With probability p, a subscription for a client in region i specifies an interest in an event with value i for the regionalism attribute; otherwise it specifies a don't-care for this attribute. This probability p is a simulation parameter we call the *degree of regionalism.* At $p = 0$, the distribution is equivalent to the non-regional distribution. We refer to the distribution at $p = 1$ as "total regionalism".

The regionalism simulations were further refined according to whether or not publisher events are assigned a regionalism attribute based upon the location of the publisher. In one scenario, called "publisher regionalism", all events are assigned a regionalism attribute value equal to the publisher's region number; in the other scenario, the regionalism attribute is assigned randomly. As it turned out, the results of the simulations were not sensitive to publisher regionalism. We therefore present only the results with non-regional subscriptions and regional subscriptions.

3.2 Bandwidth Utilization Results

To study the bandwidth utilization of the multicast algorithms described earlier, we divide the links into three classes: backbone links, intermediate (router to router) links, and fringe (router to broker) links, corresponding to link types in Sect. 3.1. This classification is based not only on the bandwidth capacity, but also on economic and administrative considerations. For example, the cost of using a backbone link may be different from that of a fringe link. Similarly, economic decisions regarding fringe links may affect the way in which subscriptions on a broker are managed. For these reasons, we believe that these three classes of links must be studied separately.

Highly Selective Non-Regional Subscriptions

For non-regional subscriptions, the various approaches are only distinguishable when match rates are low (e.g., less than 3%). Figure 7 charts the mean bandwidth utilization per published event at various subscription match rates, for different classes of links. On backbone links, the graph shows that cluster-based algorithms use a factor of two or three more bandwidth on the backbone than the other algorithms. This is because these approaches send multiple messages for each published event. The other algorithms perform similar to each other on the backbone, and are close to ideal for almost all match rates. One interesting observation here is that the neighbor matching algorithm is slightly more efficient than the ideal algorithm even though it uses multiple sends, one per hop. This is because hops from neighbor to neighbor may use an optimal number of backbone links although a sub-optimal number of links overall.

On intermediate links, all the algorithms outperform flooding if the subscription set is highly selective. In particular, the neighbor matching and CGM-4 algorithms perform better than others (excluding ideal) for subscription match rates below 1.5%. Thus, if an application has a stable match rate in this region, one of these algorithms may prove to be suitable. However, at higher match rates, these algorithms perform worse than a simple flooding approach. Similarly, CGM-8 and TCGM perform worse than flooding for anything but the most highly selective subscriptions. The group approximation algorithm does no worse than flooding asymptotically, but offers a slight benefit for match rates below 0.5%.

On fringe links, bandwidth utilization is closely related to the precision of algorithms. Single-hop precise algorithms, such as the cluster-based algorithms perform similar to the ideal algorithm, the difference being the extra usage of fringe links from publisher brokers. Single-hop imprecise algorithms, such as TCGM and Approx utilize more bandwidth on the fringes, and quickly approach flooding. Neighbor matching, although precise, utilizes worse amounts of bandwidth on the fringes since it is based on multiple hops between brokers (which are always on the fringes).

As expected, all algorithms (even ideal) eventually converge to the same (or worse) bandwidth usage as the flooding approach. With 125 subscriptions per broker and 2% subscription match rate, the fact that subscriptions are uniformly distributed (as opposed to regionally distributed) gives a 92% probability that an arbitrary broker will have a subscription matching a particular event. This means that over 90% of the brokers receive each published event.

Regional Subscriptions

For regional subscriptions, all the algorithms have the same relative performance (with the exception of approx) but show a marked improvement over flooding as the degree of regionalism (as given in Sect. 2) is increased. Figure 8 illustrates the effect of regionalism on the various approaches at a fixed match rate of 3%. At the top of the figure, intermediate link utilization is plotted as a function of the degree of regionalism. It is interesting to note that none of the algorithms perform significantly better than flooding until regional correlation reaches 0.75. At this point, ideal, neighbor matching and CGM-4 begin to show successively better improvement as the degree of regionalism increases. CGM-8 and TCGM-4(3) also show improvement but do not compare favorably with flooding until regional correlation is close to 1. The one exception to this trend is the group approximation algorithm which does not improve because the set of required groups is approximated and regionalism is not accounted for. In

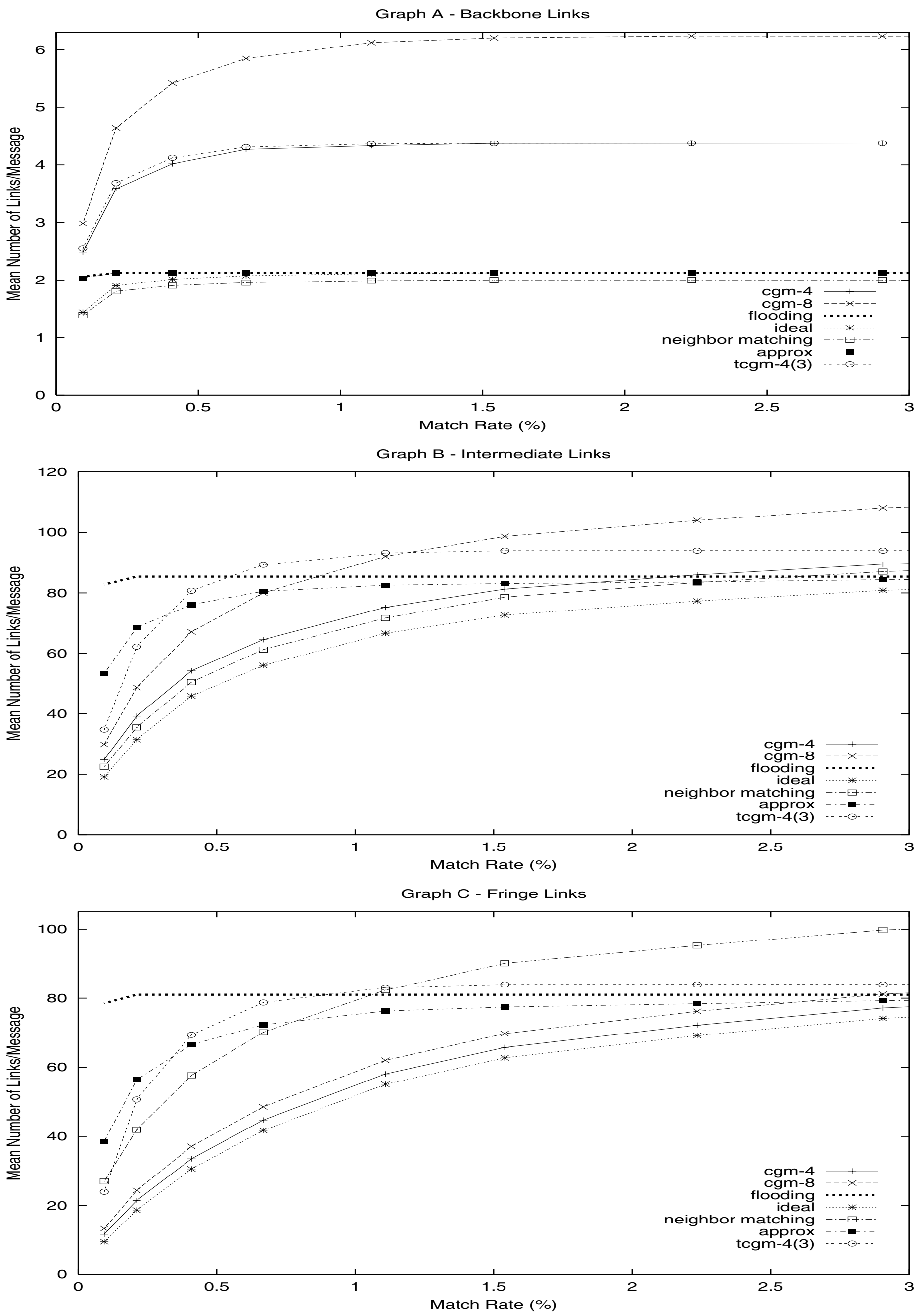

Fig. 7. Link utilization results for non-regional subscriptions

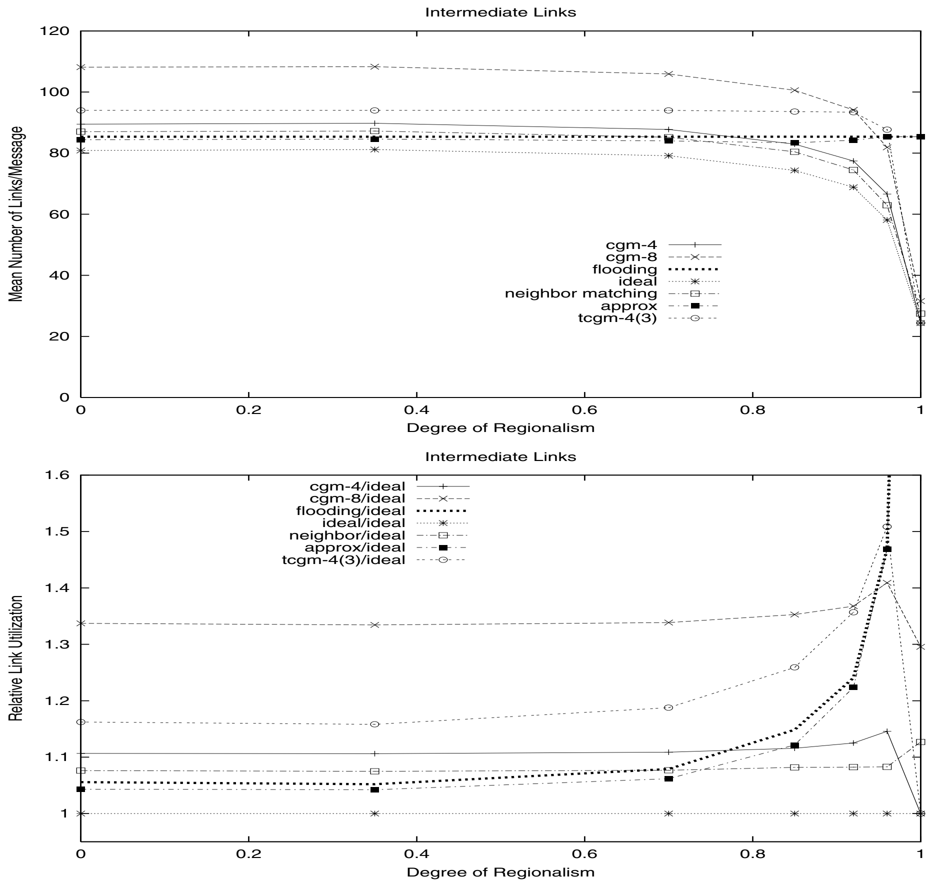

Fig. 8. Effect of regionalism on intermediate link utilization

particular, it is possible for groups from separate regions to be combined into a single group during the group combination phase[3].

The bottom of Fig. 8 illustrates the performance of each algorithm relative to the ideal algorithm. The peaks in the graph indicate regions where the ideal algorithm improves at a faster rate than the other algorithms. The CGM-4 and TCGM algorithms converge with ideal at total regionalism because the four regions used in the experiment correspond exactly with the four clusters used in these algorithms[4].

[3] The group approximation algorithm can be refined to take regions into account while combining groups in order to eliminate this effect.

[4] Also, under total regionalism, all matching subscriptions will be in the same region. As a result, a match rate of 3% gives a high probability that every broker in a cluster will require an event. Thus the flooding aspect of TCGM has no detrimental effect.

Summary

These results illustrate that in scenarios with high selectivity (match rates in the 1% range) or high regionalism (degree of regionalism greater than 0.8), the algorithm of choice will depend on the economics of deployment. If the cost of fringe links is the highest, a cluster-based algorithm may be feasible, provided that the number of groups required can be supported. If intermediate links are most expensive, that may suggest the neighbor matching approach. If backbone is expensive, anything but the cluster-based algorithms are acceptable. A weighted sum of the bandwidth utilizations, where the weights are based on the cost of using each class of links, will suggest the optimal algorithm.

3.3 Latency Results

The latency metric compares the average time taken by an event to travel from a publisher to all subscribers. It turns out that the latency of all algorithms except neighbor matching were virtually identical for all match rates. All these algorithms do not differ since the event publish rate used in our simulations was not high enough to induce queueing delays at the various routers. Even under regionalism, event rates were not sufficiently high to show any latency variation. In all cases, however, neighbor matching was about 25% slower because of the delays introduced by performing partial matching at broker nodes on intermediate hops.

4 Related Work

The background of this study, and work related to it, will be reviewed in two phases. First, we examine the event distribution algorithms of those systems that support non-trivial subscription languages, with respect to how (if at all) these systems exploit group multicast at the network level. Second, we examine algorithms that employ multiple IP multicast groups, with respect to how closely their semantics resemble those of content-based subscription systems.

4.1 Event Distribution Systems

Relatively few event distribution systems [25] allow subscriptions to be expressed as predicates over the entire message content. A few noteworthy examples of this emerging category are SIENA [6], READY [10], Elvin [20], JEDI [7], Yeast [12], GEM [15], and Gryphon [2]. All of these systems support rich subscription predicates, and thus face problems of scalability in their event distribution algorithms.

However, pure content-based systems are only one endpoint on a scale of subscription “richness,” and an increasing number of publish-subscribe systems may be expected to experience aspects of the problem explored here. The Java Message Service (JMS) [22] enables the use of *message selectors*, which are predicates over a set of message *properties*. Message designers are free to store information in properties rather than the message body, making the resulting system behave more like a content-based system. The OMG Notification Service [17] describes structured events with a "filterable body" portion. Many vendors are implementing JMS, or the OMG Notification service, or both, which has the result of making this form of subscription more popular. The TIB/Rendezvous system [23] available from the TIBCO corporation has a hierarchy of subjects and permits subscription patterns over the resulting segmented subject field, also approximating some of the richness available with content-based subscription.

The actual event distribution algorithms employed by the "richer" systems vary. Some systems, such as Yeast [12] and Elvin [20], are centralized, with a single server to which all events are first sent. The server evaluates subscription expressions and sends the results to individual subscribers. Multicast is not used. The Elvin server supports a "quench" function, wherein publishers are able to find out if an event has any subscribers at all: such events are not sent to the server.

TIB/Rendezvous uses LAN broadcast to deliver all events, and performs event filtering in daemon processes at client machines. An extension to use IP multicast [8] instead of LAN broadcast has been accomplished and it is reportedly in use by some customers.[5] This extends the reach of the Rendezvous solution to a somewhat wider network, but the solution still employs a single group and is optimized for the LAN case, where the cost of multicast and unicast are similar.

Both SIENA [6], and our previous work in the Gryphon project [2] explored algorithms that delivered events over a logical network of brokers. These algorithms delivered events only to interested subscribers, employed only links that were along a path to an interested subscriber, and sent each message at most once over each link. Both papers characterized their algorithms as forms of multicast, but neither system actually exploited multicast services at the network level: their implementation assumed only point-to-point links.

READY [10] is a new, distributed version of Yeast. It offers two ways for publishers and subscribers to connect to event brokers, via TCP connections, or via a "reliable multicast" provider. However, what they mean by reliable multicast is itself an event-based middleware layer such as TIB/Rendezvous or IONA's OrbixTalk [11]. Whether or not network-layer multicast is exploited depends on how the underlying product achieves its reliable multicast semantics. READY employs a peer group of equivalent servers rather than a graph of servers as in SIENA or Gryphon.

Both READY and TIB/Rendezvous provide specialized routers between administrative domains (called "boundary routers" in READY and "routing daemons" in Rendezvous). The assumption is that the publishers and subscribers within an administrative domain have high levels of traffic, while messages cross domain boundaries less frequently. Elvin lists a similar function as future work [20].

As far as we can determine, all previous solutions either do not use group multicast at the network level, or employ a single group with filtering at the

[5] See **http://www.rv.tibco.com/faq.html**.

clients, or modify the second technique only at boundaries between administrative domains. We wish, in contrast, to use network-level multicast as a flexible building block in developing a specialized content-based multicast solution.

4.2 Other Algorithms That Exploit IP Multicast

Publish-subscribe systems are not the only domain in which information is periodically delivered to a set of clients whose membership may vary from delivery to delivery. IP multicast was, of course, designed for the case where the set of interested clients was the same for a large set of related deliveries. So, the need to use multiple, possibly overlapping, IP multicast groups may be expected to arise in numerous domains.

One domain where the use of multiple IP groups is becoming popular is web caching. The Adaptive Web Caching proposal [27] proposes a dynamically maintained mesh of overlapping multicast groups, over which trees are implicitly formed with web servers at their root and caches as nodes. A mixture of multicast and unicast transmissions are used in constructing the protocol. Caching is based on requests from clients, rather than pro-active "pushes" from servers, so the relevance of this proposal to publish-subscribe systems is limited.

Other web caching proposals, however, have used a model in which servers push content to proxy caches based on predictions concerning likely interest in particular pages. This is much more like a publish-subscribe system. MMO [14] and LPC [24] are two recent examples of multicast "push" caching proposals that assign caches to multiple IP multicast groups based on clusters of web pages that are expected to have "similar" hit patterns.

As far as we can determine, proposals in which web caches belong to multiple IP multicast groups have assumed that the number of groups will be modest, and that the limit of IP multicast addressing is not a factor in the scalability of the proposals. In contrast, the present study contemplates algorithms in which the number of groups can become a factor in scalability, and considers tradeoffs to minimize the number of groups.

5 Conclusions and Future Work

One important result of this study is that the flooding algorithm is viable over an extensive range of conditions. As pointed out earlier, when subscription patterns do not vary by location in the network, even a fairly low match rate guarantees that all or nearly all brokers will have some subscription matching each event. For instance, under our simulation parameters (10,000 subscriptions distributed among 80 brokers), with a match rate of about 3%, each event goes to over 91% of the 80 brokers. That is, there is less than 9% wasted work on the fringe links and in the destination brokers if that event were broadcast to all brokers (there is an even smaller percentage of wasted work on the other links). Even with a match rate as low as 1.5%, each event goes to over 77% of all brokers. Therefore, it is only useful to examine the non-flooding algorithms for cases with high selectivity (i.e., match rates are low or highly variable), or high regionalism (i.e., where the probability of match is biased according to the location of the broker).

The algorithms being studied here do not begin to perform significantly better than flooding until the match rate drops below 1%. CGM performs well in this region, but still requires a very large number of groups and does not scale well. Neighbor matching is the best of the candidates if intermediate link bandwidth is the most important, but suffers in terms of latency. The group approximation algorithm is potentially scalable but performs worse than neighbor matching in the low match-rate region.

The case in which subscriptions display what regionalism is an important one. In this case, flooding is less likely to perform well because subscriptions are localized and wide-scale dissemination of events will unnecessarily congest the network. Thus, it is not surprising that many of the approaches described in this paper begin to perform better than flooding when more than 75% of subscriptions have a regional correlation. In particular, CGM and neighbor matching may provide significant bandwidth savings in these highly regional scenarios. These results suggest a hybrid approach where our multicast techniques are only utilized during high regionalism conditions. In particular, an important future direction is to discover such conditions dynamically, and to exploit them in creating small numbers of groups tailored to the most likely patterns of event deliveries.

In evaluating multicast techniques, we have emphasized performance based on a static set of subscriptions, based on the assumption that events are published far more frequently than subscription changes. However, many systems are likely to experience a flux of subscriptions. Thus, multicast groups may need to be periodically reconstructed as subscription sets change. Of the approaches considered, flooding, CGM, and neighbor matching are the most resilient to subscription set changes, since these approaches organize brokers into multicast groups which are fixed at system configuration time. For group approximation, subscription changes may alter the waste incurred by existing groups. In the worst case, the entire set of approximate groups must be reconstructed from scratch. Some overhead may be reduced in each of these approaches by performing group reconstruction at idle times and using flooding for new subscriptions in the interim. On the other hand, if subscription regionalism is also a dynamic feature then both flooding and CGM may suffer in performance. Flooding, for example, does not account for regionalism. Similarly, CGM may suffer from an unfortunate choice of regions at configuration time. In contrast, the neighbor matching algorithm is more adaptable to dynamically forming regions.

Any practical solution is likely to incorporate a hybrid of technologies. It may be cost effective to incorporate a certain degree of higher-level function in routers. For instance, neighbor matching or group approximation may be combined with a form of network multicast that permits sending to a subset of a group, as in AIM [13]. Moreover, as broker networks are consolidated and grow into the hundreds of brokers, even clustering will not significantly reduce the number of required groups. Thus, it may be necessary to consider structuring the larger network hierarchically, using different multicast algorithms internally within the subnetworks and across subnetworks.

6 Acknowledgements

The authors wish to thank Arthur Goldberg for his comments and suggestions regarding the cluster multicast algorithm, and Dilip Kandlur for his help in understanding IP router characteristics. The authors also thank Sumeer Bhola and the reviewers for their comments.

References

1. Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching Events in a Content-Based Subscription System. In *Proceedings of Principles of Distributed Computing (PODC '99)*, Atlanta, GA, May 1999. 186, 187
2. Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In *International Conference on Distributed Computing Systems (ICDCS '99)*, June 1999. 186, 190, 202, 203
3. Ken P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993. 185
4. S. Bradner and A. Mankin. *The Recommendation for the IP Next Generation Protocol*. IETF. RFC 1752. 187
5. Ken Calvert, Matt Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997. 194
6. Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, December 1998. Available from http://www.cs.colorado.edu/~carzanig/papers/. 202, 203
7. G. Cugola, E. DiNitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. Submitted to Transactions on Software Engineering. 202
8. S. Deering. *Host Extensions for IP Multicasting*. IETF. RFC 1112. 186, 203
9. S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997. 186
10. R. Gruber, B Krishnamurthy, and E. Panagos. An Architecture of the READY Event Notification System. In *Proceedings of the Middleware Workshop at the International Conference on Distributed Computing Systems 1999, Austin, TX*, June 1999. 202, 203
11. IONA Corporation. *OrbixTalk Fact Sheet*. http://www.iona.com/products/messaging/talk/index.html. 203
12. B. Krishnamurthy and D. Rosenblum. Yeast: A general purpose event-action system. *IEEE Transactions on Software Engineering*, 21(10), October 1995. 202, 203

13. B. N. Levine and J.J. Garcia-Luna-Aceves. Improving internet multicast with routing labels. In *Proc. IEEE International Conference on Network Protocols*, pages 241–50, October 1997. 205
14. Dan Li and David R. Cheriton. Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast. In *Proceedings of the USENIX Symposium on Internet Technology and Systems*, Boulder, Colorado, 1999. 204
15. M. Mansouri-Samani and M. Sloman. A Generalized Event Monitoring Language for Distributed Systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(2), June 1997. 202
16. Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. Consul: A Communication Substrate for Fault-Tolerant Distributed Programs. Technical Report TR 91-32, Dept. of Computer Science, The University of Arizona, November 1991. 185
17. Object Management Group. *Notification Service*. http://www.omg.org/cgi-bin/doc?telecom/98-06-15. 203
18. Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus - An Architecture for Extensible Distributed Systems. *Operating Systems Review*, 27(5), December 1993. 185
19. David Powell. Group Communication. *Communications of the ACM*, 39(4):50–97, April 1996. (Guest Editor). 185
20. Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Australia, September 1997. 186, 202, 203
21. Dale Skeen. Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview. Technical report, Vitria Technology Inc., 1996. http://www.vitria.com. 185
22. Sun Microsystems. *Java Message Service*. http://java.sun.com/products/jms. 202
23. TIBCO. *TIB/Rendezvous White Paper*. http://www.rv.tibco.com/whitepaper.html. 203
24. J. Touch and A. S. Hughes. The LSAM Proxy Cache - a Multicast Distributed Virtual Cache. *Computer Networks and ISDN Systems*, 30(22–23), November 1998. 204
25. Workshop on Internet Scale Event Notification. See http://www.ics.uci.edu/IRUS/wisen/wisen98 for details. 202
26. Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom '99*, San Francisco, CA, April 1996. 194
27. L. Zhang, S.Floyd, and V. Jacobson. Adaptive Web Caching. In *Proceedings of the 2nd NLANR Web Cache Workshop*, Boulder, Colorado, 1997. http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.ps. 204