Dimacs Technical Report 2001-20
June 2001

Mobile Filters for Efficient Dissemination of Personalized Information Using Content-Based Multicast

by

Rahul Shah[*]
E-mail: sharahul@paul.rutgers.edu
Address: Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903

Ravi Jain, S. Rajagopalan, Farooq Anjum
E-mail: {rjain,sraj,farooq}@research.telcordia.com
Address: Telcordia Technologies
445 South Street, Morristown,
NJ 07960

DIMACS is a partnership of Rutgers University, Princeton University, At&T Labs-Research, Bell-labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

Abstract

There has been a surge of interest in the delivery of personalized information to users (e.g. personalized stocks or travel information), particularly as the amount of information readily available from sources like the World Wide Web increases, and mobile users with limited terminal device capabilities increasingly desire updated, targeted information in real time. When the number of information recipients is large and there is sufficient commonality in their interests, as is often the case, it is worthwhile to use multicast rather than unicast to deliver the information. However, traditional multicast services, e.g. at the IP level, do not consider the structure and semantics of the information in the multicast process. We consider the use of *Content-Based Multicast (CBM)* where extra content filtering is performed at the interior nodes of the multicast tree so as to reduce network bandwidth usage and delivery delay, as well as to reduce the computation required at the sources and sinks. Note that filtering could be performed at the IP level or, more likely, at higher software layers e.g. in applications such as publish-subscribe and event notification systems.

In this paper we evaluate the situations in which CBM is worthwhile. The benefits of CBM depend critically upon how well filters are placed at interior nodes of the multicast tree, and the costs depend upon those introduced by filters themselves. Further, we consider the benefits of allowing the filters to be mobile so as to respond to user mobility or changes in user interests, and the corresponding costs of filter mobility. We consider two criteria: minimizing total network bandwidth utilization and minimizing mean information delivery delay. For each criterion we also develop a heuristic that runs faster than the optimal algorithm. Finally, we evaluate all the algorithms by means of simulation experiments.

Our results indicate that filters can be effective in substantially reducing bandwidth and delay. We also find filter mobility is worthwhile if there is sufficient locality in the interests of users, or there is marked large-scale user mobility. We conclude with suggestions for further work.

1 Introduction

Future information systems will increasingly rely on multicast facilities for delivering the results of diverse applications to growing numbers of users. Multicast will increase in importance at all levels of information and networking systems. A case in point is the delivery of personalized information to users, particularly mobile users who desire real-time targeted information (e.g. stocks or traffic information.) Multicast is also used in the signaling and management protocols of networks, e.g. for congestion control [Kasera00]. It is also being used at the middleware level, e.g. in event subscription and notification mechanisms for Java and CORBA [OMG97]. And of course, multicast applications like Internet radio, chat, and conferencing are proliferating rapidly.

Along with this increase in the use and importance of multicast, we also observe a need to generalize the notion of multicast to bring its advantages to bear on more sophisticated and specialized uses. Traditional or *Basic Multicast (BM)* typically consists of a set (or *group*) of participants, of which (typically) one is a source of information and the other are sinks; any information generated by the source is delivered to the rest of the group, essentially by setting up a *multicast tree* from the source to the recipients. Typically the size of a group is more than a few hosts and much less than the total number of hosts in the system. Multicast is then highly desirable compared to broadcast (to all hosts in the system) as it consumes less network bandwidth and does not require any processing at hosts which are not intended recipients. Multicast is preferable to unicast (to each recipient in the group) as it requires less bandwidth as well as processing and delay at the source. Various enhancements to basic multicast seek to provide reliable delivery (e.g. every message is delivered exactly once) or causal delivery (message ordering is preserved), or both, and seek to do so efficiently in terms of reducing the overhead to set up and manage the group as recipients join and leave.

In this paper we do not focus on the process of setting up the basic multicast tree itself, and assume that an appropriate method has been used for this purpose. However, we observe that basic multicast does not concern itself with the content or structure of the information being delivered. Any structure in the information can only be accommodated either by setting up multiple multicast groups or by using the concept of multiple layers in the same multicast group as has been proposed for video applications. The latter though requires the data to be highly structured as the recipient cannot receive any arbitrary layers. As an example for the former, multicast groups corresponding to different sources (e.g. Internet radio channels or chat servers) can be set up. Once a recipient joins a group it receives all the information sent to the group; any further filtering or control can only be done by the end-user's application or the user herself. Thus there is a limit to the amount of filtering or personalization that can be achieved with basic multicast.

For example, consider disseminating real-time stock quotes for the New York Stock Exchange (NYSE). Using basic multicast the source has a limited set of options. One option is to transmit quotes for all NYSE stocks (currently about 5000) on a single multicast channel. Another is to divide the stocks into a limited number of groups (each with roughly equal number of recipients, say) where each recipient typically subscribes to multiple groups, filters out unnecessary information from each group, and collates the remaining quotes into a personalized result. Thus for a given set of information being disseminated, providing finer-grained filtering increases the number of multicast groups, which increases the burden on the recipient (the application or user) as well as the network to manage group membership. On the other hand, if fewer groups are used, the recipient has to filter out a large amount of unnecessary information and network bandwidth is utilized inefficiently.

The limitations of basic multicast become much more severe once the recipients desire more complex filtering and personalization, and especially if the information being delivered is unstructured or has limited meta-data. For instance, one recipient of an Internet radio channel may desire to block out all advertisements, except those for cars, which she is currently in the market for; another likes country music but cannot abide certain artists or songs; a third wants news but no more about the latest celebrity scandal; and so on. Even if the information is well-structured, e.g. the NYSE stock quotes example above, users can desire complex filtering e.g. "Inform me only if Telcordia stock changes by more than 10% in price or 5% in volume compared to yesterday".

We consider a form of *Content-Based Multicast (CBM)* that takes into account the structure and semantics of the information being disseminated and attempts to minimize both network utilization as well as the processing at the recipients by intelligently filtering the information being disseminated as it propagates towards each recipient. We consider a solution where this is done by means of *filters*, which can potentially be *mobile*, residing at (a subset of) the intermediate nodes in the network, either in IP or application-level routers or at attached CBM servers. A filter can then apply complex criteria and ensure that information propagates down the tree to a child only if it is desired by a user at a leaf in that child's subtree. While filters thus increase multicast efficiency they also introduce additional computation, complexity and delay in the multicast process, and hence the number of filters in the network should be limited. As users move or change their filtering criteria, or as the information being disseminated changes, filters may move from one interior node to another in response.

Essentially CBM reduces network bandwidth and recipient computation at the cost of increased computation in the network. CBM at the application level is increasingly important as the quantity and diversity of information being disseminated in information systems and networks like the Internet increases, and users suffering from information overload desire personalized information. A form of CBM is also useful at the middleware level [Carzaniga98, Aguilera98] and network signaling level [Kasera00]. In addition, CBM is increasingly important as wireless networks (ad-hoc networks, multi-hop packet radio networks, wireless LANs and wide-area cellular networks) proliferate, since wireless bandwidth is scarce and resources (in terms of computation, storage and battery life) at wireless devices are limited.

There has been some previous work in diverse areas that CBM can address [Carzaniga98, Aguilera98, Kasera00, Donahoo98]. Event-based middleware infrastructure is becoming increasingly common [Carzaniga98, Segall97, OMG97] and essentially consists of an event to which processes (*listeners* or *observers*) can subscribe; when the event occurs, only the listening processes are informed. This mechanism is very desirable as it provides loose coupling and yet good performance since listeners only receive notifications for precisely those events they are interested in. In [Carzaniga98] the authors define a useful formal syntax and semantics for the event notification service. It is clear that the service can be considered a special case of CBM. However, they do not consider in detail how the service would be implemented and do not consider how mobile filters could be used for the purpose.

[Kasera00] proposes to use active services to address the challenges presented by reliable multicast. Unlike the mechanism we propose they assume the active services are stationary. [Aguliera98] addresses the problem of providing an efficient matching algorithm suitable for a content based subscription system. [Banavar98] addresses the problem of matching the information being multicast with that being desired by leaves. However, their scheme assumes that the subscription pattern of users is static. Thus, unlike CBM, they do not use mobile filters and hence the proposed schemes can be considered as a special case of CBM. .

[Donahoo98] considers enhancements to the normal multicast along three directions. At the network layer they try to investigate the relationship between the application performance and the placement of the root of the multicast tree. They propose algorithms to influence the root placement as well as to migrate the root over a period of time thereby focusing on multicast routing. They also consider modifications to applications to take advantage of network layer multicast. Thus unlike our approach, this approach does require modifying the applications and they also do not consider the use of mobile filters. Finally they also study reliable multicast transport protocols. In [Joh00], the authors propose an application level multicasting system. In this work the authors focus on procedures to build efficient data distribution trees that adapt to changing network conditions.

[Opyr00] considers a problem similar to the problem that we consider but the solutions that they investigate are within the confines of normal IP multicast. Hence, they mainly consider ways of minimizing the number of multicast groups either by judicious clustering of the users or by minimizing the amount of "waste" information sent to users. It is though obvious that such an approach will not only be inefficient but will also become infeasible with a wide variability in the subscription requests of the different users in the system. [Zhou00] on the other hand propose a variation of IP multicast in which the source decides on the multicast receivers of a data stream based on the data contents.

The benefits of CBM depend critically upon how well filters are placed at interior nodes of the multicast tree, and the costs depend upon those introduced by filters themselves. Further, there is a tradeoff between the benefits of allowing the filters to be mobile so as to respond to user mobility or changes in user interests, and the corresponding costs of filter mobility. In this paper we thus address issues related to *mobile computing and networking* rather than wireless communication, i.e., mobility of software (filters) as well as users. In later work we will consider the effect of some of the links in the multicast tree being wireless.

The contributions of this paper can be summarized as follows. We consider two objectives for developing filter placements: minimizing total network bandwidth utilization and minimizing mean information delivery delay. For each objective we develop an algorithm for filter placement, show that it is optimal, and derive its time complexity. For each objective we also develop a heuristic that runs faster than the optimal algorithm. Finally, we evaluate all the algorithms by means of simulation experiments.

This paper is organized as follows. In sec. 2 we describe the system model, and the two placement objectives. In sec. 3 we develop an algorithm for filter placement so as to minimize bandwidth and show that it is optimal; in sec. 4 we develop an optimal algorithm for minimizing mean delay. In sec. 5 we develop heuristics corresponding to our optimal algorithms. In sec. 6 we describe results from simulation experiments to evaluate the optimal and heuristic algorithms. Finally in sec. 7 we end with a summary and some conclusions.

2 System model

We assume for simplicity that there is a single source of information connected by an IP network to a set of recipients. The source periodically receives updated information to be disseminated. We assume that a multicast tree has been set up using an appropriate protocol; the root is the source and the leaves are the recipients. For CBM we assume that there is a set of software modules, called *filters,* distributed at the interior nodes of this multicast tree. The filters reside at *filter platforms*, which can be IP routers or at servers attached to the local subnet of a router. We stress that our model does not require that all routers in the system be modified; an overlay of CBM servers can be laid out, similar to that proposed by [Carzaniga98]. In sec. 2.1 we describe the background on how the filters operate and how information is disseminated in the multicast tree, giving a numerical example in sec. 2.2. In sec. 2.3 we describe how the algorithm for determining where the filters should be placed in the multicast tree is carried out.

2.1 Subscription and matching

Filters can be written in any suitable language that allows execution across filter platforms. The filters may move from one filter platform to another. Thus, for example, filters could be mobile Java applets or programs, which are very widespread and successful on the Internet, or they could be filters as in Concordia [Kob99] or Aglets [Tai99] systems.

There need not be a filter at every interior node of the multicast tree. We define the *filter tree* as the subtree of the multicast tree such that each interior node of the filter tree has a filter, and the leaves of the filter tree are the leaves of the multicast tree. For convenience we assume there is always a filter at the root of the multicast tree, i.e., the root of the multicast tree is the root of the filter tree.

The information disseminated by the source is described using a hierarchical information schema of major keys, minor keys and subkeys where all are triples of the form *(type, name, value)*. As an example, an item of information for an IBM stock quote may be:

Major key: (string, exchange, NYSE)
Minor key: (string, symbol, IBM)
Subkeys: (float, prior_closing_price, 95), (float, current_price, 98),
(float, 52_week_high, 110)

Obviously various optimizations can be performed in this schema that we will ignore for simplicity.

Each filter collects requests for personalized information (*subscriptions*) from its children in the filter tree. A subscription consists of the description of an item along with a *filter criterion* specifying operators on its values. For example, in addition to the data item specified above (IBM stock quote) the subscription request for a particular user *u* could say:

Filter: (float, current_price, >, 100)

indicating the recipient is to be notified if the current price exceeds $100. Filter criteria can be combined with logical operators to make more complex filters.

The filter forwards subscriptions to its parent filter in accordance with a *subscription algorithm*. The filter maintains a subscription table of current subscriptions that it knows about and the child in the filter tree that each subscription originated from. If a new subscription request is subsumed by an existing subscription the filter updates the table only; otherwise it makes a new entry in the subscription table and forwards the subscription to its parent. Continuing the IBM stock quote example, if the filter received a new subscription request (from user *u* or any other user in its subtree) with criterion

Filter: (float, current_price, >, 110)

it would not forward the request to its parent.

Each filter sifts the information it receives from its parent filter in accordance with a *matching algorithm*. The filter compares the information item received with the subscriptions in its subscription table. It then composes a message to each child filter containing only the information (if any) matching the subscriptions originating from that child. For an example of a matching algorithm, see [Aguilera98].

Filters can be in one of four states: *active*, *migrating*, *spawning,* or *passive.* A filter performs the subscription and matching algorithms only in the active state. In the migrating state a filter physically relocates from one interior node to another. In the spawning state a filter either creates a new filter which will migrate to another node and become active, or will activate an existing filter at that node (by sending it an activation message). In the passive state the filter resides at a node and listens for activation messages from other filters but performs no subscription or matching algorithms.

The subscription and matching algorithms executed at the filters are fairly straightforward and we will not describe them further. Observe that they are distributed algorithms in the sense that each filter only operates using information in its local storage, and on-line algorithms in the sense that they are executed as each information item or subscription is received.

2.2 Example

We use the multicast tree of Figure 1 to illustrate the benefits of CBM. The subscription requests of each recipient (leaf), labeled A-I, for items (numbered 1 to 8 in this example) are shown along the bottom of the tree. These subscriptions propagate up the tree; interior nodes are labeled with the union of the subscriptions from the leaves in their subtree. The source sends the 8 (equal-sized) items of information requested by at least one recipient. With basic multicast, all 8 items are sent on all 15 links, resulting in 120 units of total traffic. Each recipient then must filter out all undesired items. If filters are placed at every interior node of the tree, the traffic can be reduced to 50 units, and the recipients need not perform any filtering.

However, recognizing that filters themselves represent a cost, suppose we restrict the number of filters (in this case, arbitrarily to 3), and place them at the shaded interior nodes. Then the total traffic is reduced to 63, and some recipients e.g. A-D, must perform filtering, while others e.g. E-I, need not. Now suppose some users leave, move or join the tree, or their subscriptions change; the total traffic will change. This raises the questions we address in the rest of this paper: (1) how should a limited number of filters be placed to be most effective, and (2) as conditions change, is it worthwhile to move filters in response?

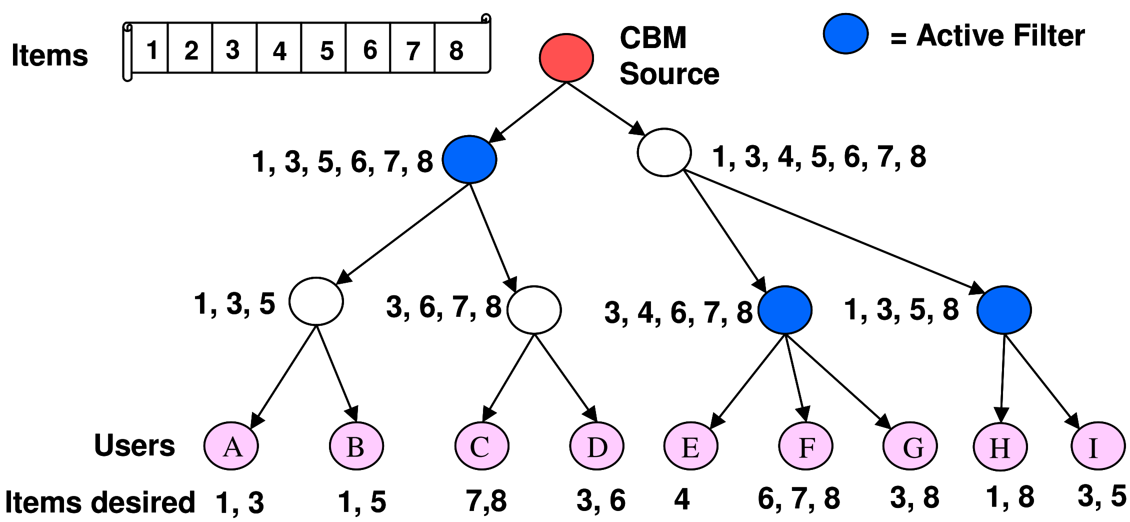

Figure 1: Example of decreased bandwidth utilization with filters (shaded vertices have filters present.)

2.3 Filter placement algorithm framework

A *filter placement* on a multicast tree $M = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ is a set $P \subseteq V$ where filters are placed at all vertices in P and on no other vertex in V. Let $|V| = n$. The root of M is denoted w and $Tree(v)$ denotes the subtree rooted at vertex $v \in V$. Thus $Tree(w) = M$.

All the filter placement algorithms we discuss operate in the CBM framework (see Figure 2). Note that this algorithm could in principle be executed at the source or at some other arbitrary server; for ease of exposition we assume it is executed at the source. Also note that the algorithms are centralized and operate upon the tree M that is in the source's memory. M represents a model of the actual multicast tree set up on the network, and not the multicast tree itself.

Users may make or modify their subscriptions at any time, and these subscriptions propagate up the tree to the source in accordance with the subscription algorithm described previously. The source collects the requests periodically into batches. At the end of each period the source runs the filter placement algorithm and calculates a new filter placement[1]. It then sends signaling messages to the filters in the tree to activate, passivate, migrate or spawn filters as necessary. The source then multicasts information which is the union of the subscriptions in the current batch. Although we show the pseudocode in Figure 2 as a single process, it is clear that some of the actions, e.g. collecting batch requests and calculating the new placement, could be done concurrently with the multicast of the information items.

For simplicity we make the following assumptions in the rest of the paper: (1) each information item is of unit size; (2) the size of an information item is much greater than the size of a signaling message; and (3) the delay to send the filters signaling messages, and for them to respond with the appropriate action, is much less than the time to multicast the information required in a batch.

The step of calculating the amount of information flow f required at each vertex based on the set of subscriptions B is as follows. For each leaf $v \in V$ the source calculates $f(v)$ as the size of the subscription request from that user. The source then repeats this process recursively up the tree. This information can be obtained by calcuating succesive unions. It can be calculated when the subscriptions are processed, with little extra effort. It is straightforward to see that this takes time linear in number of vertices.

We consider two objective functions to determine the effectiveness of a filter placement. The *total traffic* T is the sum of the bandwidth used in the multicast tree to satisfy a given set of subscriptions. The *mean delay* D is the average delay from the instant the source multicasts the information to the instant that a leaf receives it.

```
Content-Based Multicast framework (M) {
        /* M = (V, E) is the model of the
          multicast tree */
   B = {b(u): u is a leaf of M}, where b(u) is the set of
        items requested by user u;
   repeat every time period {
        Batch new subscription requests b' and
            cancellations c' into B' = {b'(u), c'(u)};
        B = {b(u) ∪ b'(u) \ c'(u)};
        Calculate the new incoming flow f(v)
            required at each vertex v ∈ V based on B;
                /* Find new filter placement */
        P = FilterPlacement(M, f, P);
        Issue instructions to enact the new
            placement P;
        Multicast the information requested in
            B over M;
   }
}
```

Figure 2: Pseudocode for CBM framework

[1] In the case of the heuristics to be described later, once filters are placed, we do not need all subscriptions to be sent up to the source. The filters can carry out the necessary calculations and only send summary information to the source. However, for simplicity we omit description of this improvement.

3 Minimizing total traffic with *k* filters

We develop an algorithm for finding a filter placement so as to minimize the total bandwidth consumption for a given set of subscriptions assuming required flow values are provided at each vertex. As mentioned previously filters cause overheads in terms of processing and delay. For the problem of minimizing total traffic we model this constraint as a limitation that at most k filters may be placed within the multicast tree, i.e., the resulting filter placement $|P| \leq k$.

Here for simplicity, we describe the algorithm for binary multicast trees; an extension to arbitrary trees is shown in the Appendix. For a vertex $v \in V$ let $L(v)$ and $R(v)$ denote the left and right child of v respectively. We also define the *Lowest Tight Ancestor (LTA)* of v as the lowest ancestor of v whose parent has a filter, and denote it $A(v)$. If the parent of v has a filter, then $A(v) = v$.

Let $T(v, i, p)$ denote the minimum total traffic in $Tree(v)$ given that up to i filters can be placed in $Tree(v)$ and the LTA of v is $A(v) = p$. Thus the objective function is to minimize $T(w, k, w)$, where w is the root of M. (Recall that the source multicasts an item of information only if it is required by some user.) The total traffic can be expressed as the following recurrence relations, where for notational convenience we set $L(v) = l$, $R(v) = r$, $A(v) = p$.

If v is a leaf then

$$T(v, i, p) = 0 \quad \text{for all } p, i$$

Otherwise

$$T(v, i, p) = \underline{\min}\{$$
$$f(l) + f(r) + \underline{\min}\ [0 \leq j < i: T(l, j, l) + T(r, i - j - 1, r)], \quad \text{if } v \text{ has a filter}$$
$$2\, f(p) + \underline{\min}\ [0 \leq j \leq i: T(l, j, p) + T(r, i - j, p)], \quad \text{otherwise}$$
$$\}$$

Theorem 1. The minimum total traffic in the CBM tree rooted at w given that at most k filters can be placed in the tree is given by the recurrence relations for $T(w, k, w)$.

proof. The proof is by induction on the height h of the CBM tree. Clearly for $h = 0$, v must be a leaf so the total traffic is zero, which is minimum. For $h > 0$ consider an arbitrary vertex v (see Figure 3). The traffic entering its LTA is $f(p)$, since the filter at the parent of p will ensure that only the flow required in $Tree(p)$ is sent to p. Now $f(p)$ will also be the incoming flow to v since there is no filter between v and p to reduce or modify the flow in any way. If a filter is placed at v it will ensure that only the flows strictly required in $Tree(l)$ and $Tree(r)$ are sent to the left and right child respectively; otherwise, both children will receive flow $f(p)$. The decision to place a filter at v is made such as to minimize the total traffic assuming that j ($j = 0, 1, 2$ ..) filters are placed in the left subtree and the remaining filters are placed in the right subtree. Assuming minimality of values calculated at vertex l and vertex r by induction, we get the minimality of values calculated at v.

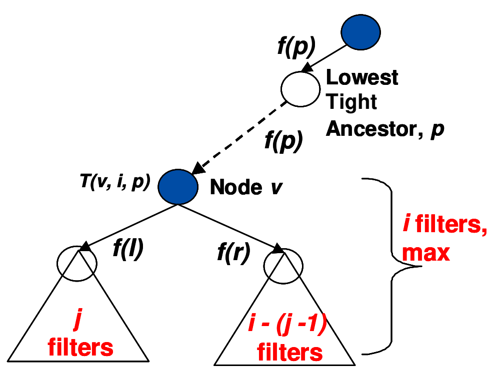

Figure 3: Filter placement to minimize total traffic

Theorem 2. The filter placement to achieve minimum total traffic in the CBM tree given that at most k filters can be placed in the tree can be computed in time $O(kn^2)$.

proof sketch. The optimal placement can be computed using dynamic programming. At each vertex a table of at most $O(k\ n)$ different values of T is maintained. To compute each value requires at most k comparisons. Since there are n vertices in M

for which these tables are to be computed the total time required is $O(k^2n^2)$. However, observe that relatively few vertices in the tree have both left as well as right subtrees of size comparable to k (say at least $k/2$). Hence typically for most vertices we require much less than k comparisons. A careful analysis of this similar to the one for the k-median problem on trees [Tamir96] shows that the time complexity is $O(kn^2)$.

4 Minimizing mean delay

We now consider the benefits of filters in reducing the mean delay of delivering information to individual users. Filtering reduces transmission delay by reducing the amount of information transmitted to nodes downstream. However, the filters themselves introduce a delay, leading to a trade-off as to whether it is worthwhile to place a filter at a node.

For simplicity we assume that (1) the delay on a link is proportional to the length of the message transmitted across the link, ignoring propagation delay and (2) the delay introduced by a filter is a (typically large) constant, F. Once again, for simplicity we describe the solution for binary trees. The extension to the general case is straightforward and is omitted.

Let $d(v, u, g)$ be the delay for sending traffic g from vertex v to leaf u in the tree M, i.e., it is the sum of the transmission delays for sending g over each link in the unique path from v to u. Then, assuming no intermediate filtering, the total delay for sending traffic g from vertex v to $Tree(v)$ is the sum of $d(v, u, g)$ over all leaves u in $Tree(v)$. In the following we will consider the problem of minimizing total delay, which is simpler in terms of notation; mean delay is minimized also as a result.

Now consider a CBM with a required flow f known for each vertex in the tree. For a vertex v with LTA $A(v) = p$, let $D(v, F, p)$ denote the minimum total delay in $Tree(v)$, given that filters introduce a delay F. Let $s(v)$ denote the number of leaves in $Tree(v)$, and c the link delay per bit. Then the minimum total delay can be expressed by the following recurrence relation, where $L(v) = l$, $R(v) = r$, $A(v) = p$.

If v is a leaf then

$D(v, F, p) = 0$ for all p

Otherwise

$$D(v, F, p) = \min\begin{cases} s(v)\,F + f(l)\,s(l)\,c + f(r)\,s(r)\,c + D(l, F, l) + D(r, F, r), & \text{if } v \text{ has a filter} \\ f(p)\,s(v)\,c + D(l, F, p) + D(r, F, p), & \text{otherwise} \end{cases}$$

Theorem 3. The minimum total delay in the CBM tree rooted at w given that filters introduce a delay F is given by the recurrence relations for $D(w, F, w)$.

proof. The proof is by induction on the height of the CBM tree, and is similar to the proof for Theorem 1. We omit details for brevity.

Theorem 4. The filter placement to achieve minimum mean delay in the CBM tree given that filters introduce a delay F can be computed in time $O(n^2)$.

proof. The proof is similar to the proof for Theorem 2. We omit details for brevity.

5 Heuristics for filter placement

In the previous two sections we have described algorithms that determine an optimal filter placement for two objective functions: to minimize total traffic in the tree or minimize mean delay for users to receive information; these algorithms run in time $O(kn^2)$ and $O(n^2)$ respectively. It is desirable to have filter placements calculated quickly so that the source can respond quickly to changes in user subscriptions. Also, when user subscriptions change, the optimal algorithm may require many of the k filters to move; since filter mobility has a cost, it is desirable to restrict it in some way. In the following we develop a framework for running heuristics, and introduce heuristics that run in linear time for each of our objective functions, but are sub-optimal

Heuristic framework. The filter placement heuristics run in essentially the same framework of Figure 2 as the optimal algorithms, with the exception that the heuristics take into account not only that filters have costs (which the optimal algorithms model as constraints on the number of filters k, or as a filter delay F), but that moving a filter has a cost. Thus while the optimal algorithms calculate a new filter placement for each batch, where potentially the positions of all filters may change, in the heuristics we allow only one filter to be moved for each batch. (Clearly this constraint could be generalized to allow some $i \leq k$ filter moves, but as we shall see from simulations, the single-move heuristics work sufficiently well.)

Both the traffic-reducing and the delay-reducing heuristics calculate a metric called Importance, $I(v)$ for each v in M. Informally, the Importance of a vertex is an estimate of the value of placing or removing a filter at that vertex in terms of the objective function (reducing traffic or delay). The importance of a vertex depends on the current filter placements in the tree. The Importance is easier and quicker to calculate than the calculations in the optimal algorithm.

The Importance value is used, for each batch where a heuristic is run, to determine whether to move a filter, as follows. Consider the set of vertices that do not currently have a filter placed there; let vertex v of that set have the highest Importance $I(v)$, with ties broken arbitrarily. Similarly consider the set of vertices that do have a filter, and let vertex w of that set have the lowest Importance $I(w)$. Then the heuristic infers that it may be desirable to move the filter at w to v. The heuristic also considers vertices with filters whose parents do not have filters (or vice versa.)

The heuristic can be described as three steps.

1. *Importance flip.* Consider the most important vertex without a filter, v, and the least important vertex with a filter, w. If moving the filter from w to v will reduce the total traffic, the heuristic does so.
2. *Parent-child flip.* If the Importance flip would not reduce the traffic, the heuristic considers each vertex with a filter whose parent does not have a filter (or vice versa). If moving the filter to the vertex without the filter would reduce traffic, the heuristic does so.
3. If neither flip would reduce traffic, the heuristic does nothing.

5.1 Heuristic to reduce total traffic

We now describe the specific operation of the traffic-reducing heuristic given the framework described above.

The Importance metric for the minimum traffic case is calculated as follows. Let the *weight* of a subtree $Tree(v)$ of M be denoted $w(v)$. The weight represents the number of edges in $Tree(v)$ affected by the placement of a filter at v. Then, the Importance of a vertex v is defined by the following recurrence relations, where $L(v) = l, R(v) = r, A(v) = p$.

$$I(v) = (f(p) - f(l))\, w(l) + (f(p) - f(r))\, w(r)$$

where for a vertex x

$$w(x) = \begin{cases} 1, & \text{if } x \text{ has a filter} \\ 1 + w(L(x)) + w(R(x)), & \text{otherwise} \end{cases}$$

The intuition behind this Importance metric is sketched as follows. If a filter is placed at v then excess traffic will not be directed to l. (The same idea holds for r.) The amount of this excess traffic is $f(p) - f(l)$. If l also has a filter, then the savings will only apply to a single link, the link from v to l; otherwise, it will apply to all links from v until its next descendant that has a filter.

The intuition behind the Parent-child flip is sketched as follows. The Importance flip may not catch cases where moving a filter from a parent, say v, to a child, say r, is beneficial. Since there is no filter at r, $w(v)$, and hence $I(v)$, is greater than it would be otherwise. Further the filter at v means $f(r)$, and hence $I(r)$, is less than it would have been otherwise. These two effects together may incorrectly inhibit the movement of the filter from v to r.

Theorem 5. The execution time of the traffic-reducing heuristic is $O(n)$.

proof. The Importance for each vertex in M can be calculated using the recurrence relation above, and it is straightforward to show that no vertex is considered twice, so that the execution time for this calculation is $O(n)$. If an Importance flip has to be done, only the most important vertex without a filter and the least important vertex with a filter have to be found, which takes time $O(n)$., i.e., sorting all the vertices by Importance is not required. Once the two candidate vertices have been found by the Importance metric, determining whether flipping them reduces total traffic, given that all other filters remain unchanged, can also be done in time $O(n)$. The calculations for the Parent-child flip involve looking at each vertex and its parent only, which for each vertex is time $O(1)$, and so over the entire tree can be done in time $O(n)$.

5.2 Heuristic to reduce total delay

The delay-reducing heuristic operates in the same framework as the traffic-reducing heuristic. However, the Importance metric is calculated using the quantity $s(v)$, the number of leaves in $Tree(v)$, as follows. Once again $A(v) = p$.

$$I(v) = (f(p) - f(l))\, w(l)\, c + (f(p) - f(r))\, w(r)\, c - s(v)\, F$$

where for a vertex x

$$w(x) = \begin{cases} s(x), & \text{if } x \text{ has a filter} \\ s(x) + w(L(x)) + w(R(x)), & \text{otherwise} \end{cases}$$

The intuition behind this Importance metric is sketched as follows. The quantity $w(x)$ here is the sum of the lengths of parts of the paths from x to all leaves in $Tree(x)$. If a filter is placed at v then excess traffic not directed to l is $f(p) - f(l)$, resulting in a reduction in delay of $(f(p) - f(l))\, c$. (The same idea holds for r.) If l has a filter this reduced delay is enjoyed by all users in $Tree(l)$. If l has no filter, then the savings will also apply to all links from l until its next descendant that has a filter.

Theorem 6. The execution time of the delay-reducing heuristic is $O(n)$.

proof. Similar to Theorem 5, so omitted.

6 Simulation Results

In the following we describe simulation experiments to evaluate the effectiveness of the filter placement algorithms and investigate the regimes where they are worthwhile. There are a large number of possible scenarios that can be simulated. We have chosen several interesting scenarios that attempt to answer the following basic questions:

1. Can filters be effective in reducing CBM traffic? If so, how does the number of filters used affect the traffic reduction?
2. What is the impact of patterns in user subscription on the effectiveness of filters in reducing CBM traffic?
3. How effective is the traffic-reducing heuristic for some possible subscription patterns?
4. Under what circumstances does filter movement make sense for reducing CBM traffic, and how much impact can it have?

We also carried out simulation experiments to answer the same questions for minimizing mean delay. However, due to lack of space we do not include all of them here. The traffic results are shown in sec. 6.1, and the delay results in sec. 6.2. Our experiments show that in most cases the traffic and delay cases are qualitatively similar.

All the simulation experiments were carried out for a complete binary tree of seven levels, i.e., $n = 127$ vertices and 64 leaves (users). We assume that the source has $m = 64$ distinct information items that users can choose from. The probability that a user i, $0 < i < 65$, chooses an item j, $0 < j < 65$, is denoted $P(i, j)$.

6.1 Minimizing total traffic

6.1.1 Effectiveness of filters in reducing total traffic

In this experiment we allow users to have uniform subscription probabilities, i.e., $P(i, j) = P$. We vary k from 0 to 63, and measure the total traffic after the optimal algorithm has placed the k filters. The total traffic is measured in terms of messages, where one information item traversing one link counts as one message. The results are shown in Fig. 4.

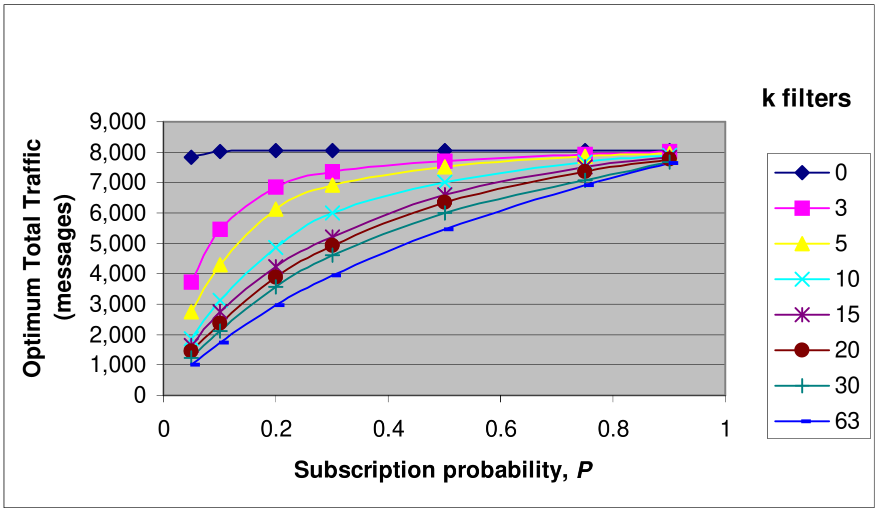

Figure 4: Total traffic with optimal placement of *k* filters, with uniform subscription probability and *k* varied

We see that when filters are introduced ($k > 0$) the total traffic can drop sharply. Filters have most effect when subscriptions are sparse (P is low), because as P increases they have less and less work to do; when $P = 1$ all users want all items and CBM is the same as basic multicast. (Note that when subscriptions are very sparse there is a small dip in traffic for $k = 0$ because at that low subscription rate not all m items are being requested by users.)

We also see that a small number of filters can have dramatic effects: with only 3 filters and subscription probabilities of less than 20%, traffic can be reduced by about 12-50%. There are diminishing returns as the number of filters is increased. We see that in this experiment with $k = 15$ filters we can obtain roughly 75% of the traffic reduction compared to $k = 63$ over a large range of subscription probabilities.

6.1.2 Impact of locality in subscriptions

We now consider the very likely case that user subscriptions are in fact not uniformly distributed, but that some kind of locality holds. We model locality in the following way: users that are close in the multicast tree tend to have similar interests, and a user's interests fall off rapidly in the surrounding items of interest. This locality model tends to capture a situation where locality in the multicast tree corresponds roughly to geographic locality (e.g. sibling leaves are physically close together) and users want location-based information (as is often the case with information such as weather, traffic, etc.) Clearly, other models of locality are also possible; we use this as one example.

Here, we have shown the case where average subscription probability is 0.2 and the probability $P(i, j) = P/N$ if $i = j$, and Pq^r/N otherwise, where N is a normalization factor which ensures average probability of 0.2, q is a *skew parameter* and r is the height of the lowest common ancestor of leaf i and leaf j. Thus r is a measure of the distance between i and j in the tree. The skew parameter q is 1 if subscriptions are uniform; the lower q, the more locality in the subscriptions.

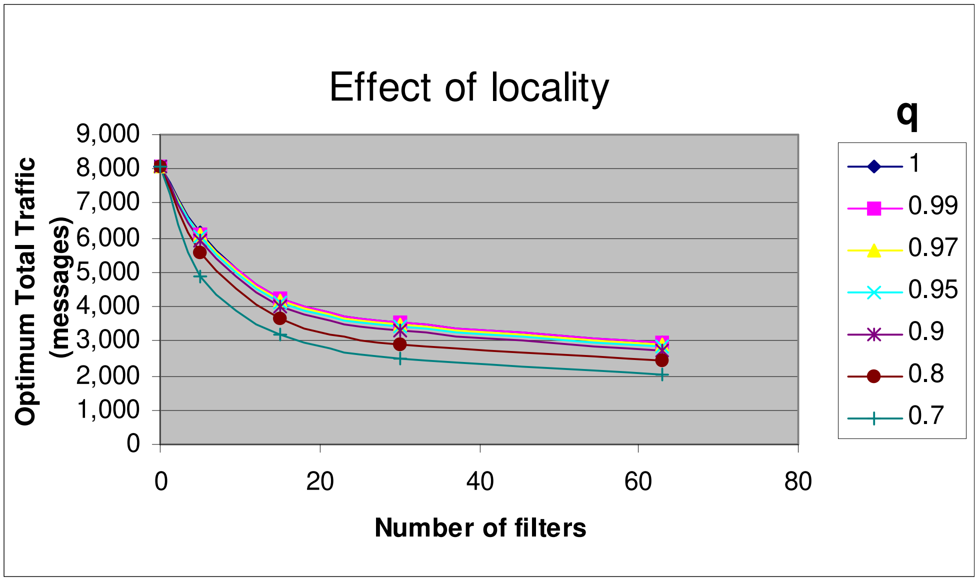

Figure 5: Effect of subscription locality on total traffic with optimal filter placement

We see that as before, in all cases, as the number of filters increases the total traffic drops, with diminishing benefits. We also see that with increasing locality filters are more beneficial, regardless of the number of filters. For $k = 3$ locality can decrease traffic further by about 15%, while at $k = 63$ almost 50% further reduction can be obtained.

6.1.3 Effectiveness of traffic-reducing heuristic

We next consider the effectiveness of the heuristic and compare it with the optimal algorithm, as follows. The optimal algorithm is run at the start of the experiment, and generates a placement, called *init*. For 49 subsequent time units (i.e., 50 batches total), user subscriptions are allowed to vary, and for each batch the heuristic and the optimal algorithm are compared. The heuristic is constrained to move only one filter at each time unit, while the optimal algorithm may move any number of k filters.

The experiment is carried out with users having uniform subscriptions with $P = 0.3$, and $k = 15$ filters. At each time unit, 20% of the users (probabilistically) change their subscriptions. The total traffic with the initial placement (*init*), the optimal algorithm (*opt*) and the heuristic (*heu*) are shown in Figure 6.

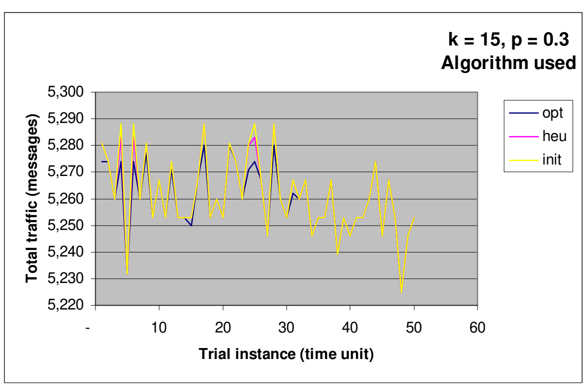

Figure 6: Total traffic with initial placement, optimal algorithm, and heuristic, with uniform subscriptions

We see that for uniform subscriptions in this case the initial placement performs very well, and in fact neither the optimal nor the heuristic algorithm do much better. The reason for this is straightforward: with uniform subscription probabilities, once filters have been placed what really matters is the quantity of information desired by each user; this is especially the case with relatively large subscription probability and relatively large number of filters.

6.1.4 Effect of filter movement

We now consider a situation where filter movement makes an impact on total traffic. The situation we consider is where users themselves move, for two mobility models.

Commuter user mobility model. The situation we consider is where users themselves move, in a manner similar to commuters.We assume that users in the left and right subtrees have a subscription skew of q. (See Figure 7.) Thus at time unit $t = 0$, users in the left subtree have subscriptions with uniform probability $P = 0.3 + q$, while those in the right have $0.3 - q$. For $t = i > 0$, however, the probability of user i in the left subtree is swapped with that of user i in the right. In this manner, probabilistically speaking, we model the movement of users from the left to the right (e.g. from work to home) as a change in subscription probabilities.

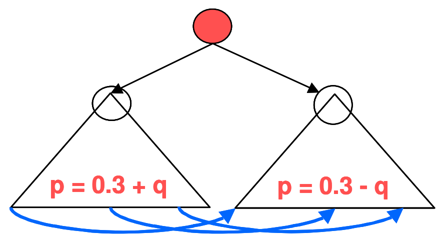

Figure 7: Commuter user mobility model

With this model, we see that in fact filter mobility becomes very important. In figure 8 we show the extra reduction in total traffic when filters are mobile as against static, as a percentage, for different values of skew q. When q=0 there is no skew and initial position of filters is good enough. By allowing mobility we don't get much extra reduction. Then regrdless of number of filters used, there is little change in the total traffic when mobility of filters is allowed, since subscriptions are uniform and stay so. However, as skew is increased, indicating more user mobility, the mobility of filters becomes more effective. For a given skew we observe that, for around k= 16 filters, mobility of filters is less effective. This is because when number of filters is 16, they tend to place themselves at the middle level of the complete binary tree. And this structure being very stable, there is not much mobility of filter can achieve there.

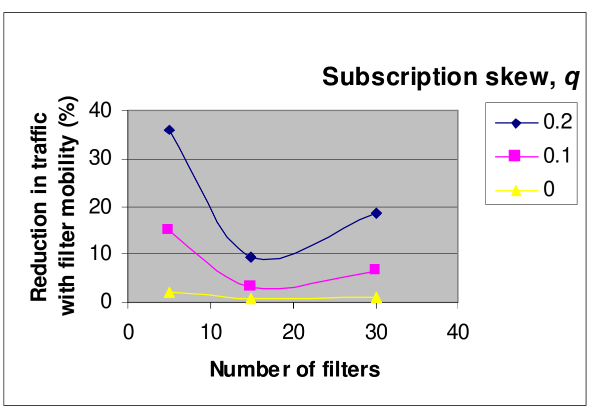

Figure 8: Effect of user mobility and filter mobility on total traffic

Extreme user mobility model. We now consider an extreme example of user mobility. At time $t = 0$, every user subscribes to every message with probability $P = 0.3$ and there are $k = 15$ filters. Then at $t = i$, user i (numbering from left to right) leaves the system and cancels its subscription. We see that with time the initial placement becomes more and more ineffective, while the optimal and heuristic algorithms perform very well. We also notice that the heuristic performs almost as well as the algorithm in this case.

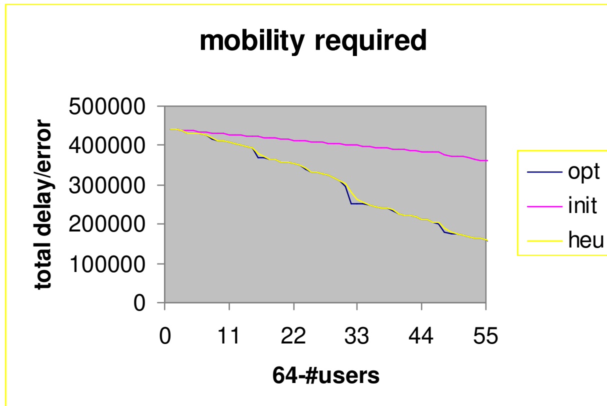

Figure 9: user mobility scenario where the initial placement performs poorly

6.2 Minimizing mean delay

6.2.1 Effectiveness of filters

We carry out the same experiment as in sec. 6.1, but for minimizing mean delay. Thus users have uniform subscription probabilities. To model the cost of filters, we vary the delay F introduced by an filter from 100 to 2000 time units. The results are qualitatively similar to those for minimizing total traffic, and indicate that filters can be very effective.

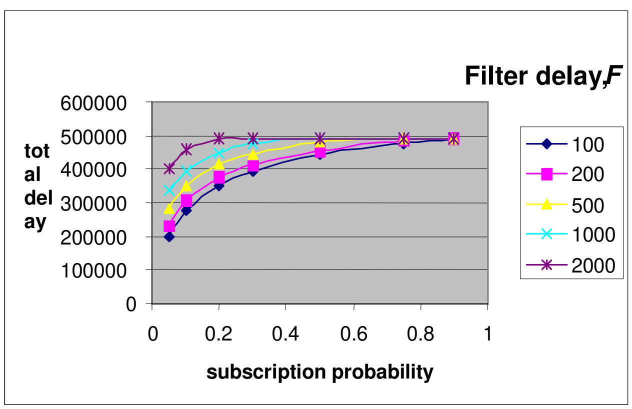

Figure 10: Effect of filters on total (hence mean) delay, for uniform subscriptions and varying filter delays

6.2.2 Impact of locality

This section is very similar to section 6.12. We model the locality in the same way. We observe that filters are more effective when subscriptions have more locality bias.

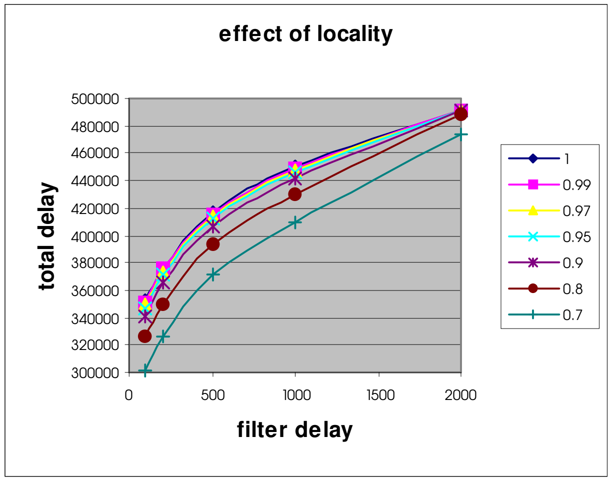

Figure 11: Effect of subscription locality on delay

7 Conclusions

In this work we have considered the effect of using mobile filters for achieving personalized delivery using the paradigm of content-based multicast. Personalized information delivery has become an important and lucrative application-level service. CBM is also applicable to other important facilities, e.g. event subscription and notification. However, the effectiveness of CBM depends upon the placement of the filters, and their movement in response to changes in user subscriptions and motion.

We have developed optimal algorithms for placement of filters to minimize total traffic as well as to minimize the mean delay experienced by users in receiving information. Both algorithms run in time proportional to the square of the number of vertices in the multicast tree. We have then developed simple heuristics that run in linear time but which are sub-optimal.

Through simulation experiments we have demonstrated that, for the situations studied, filters can be very effective in reducing total traffic as well as mean delay, and that a relative small number of filters can be very effective. We also show that the effectiveness of filters increases with locality in users' subscription patterns. The simple heuristics work well, especially when user subscriptions are distributed uniformly and are relatively stable, but in extreme cases can cause poor performance. Finally, filter mobility may not be beneficial in general; it is effective if there are large-scale changes in user subscriptions or in user locations.

Proposed work for the future includes further investigations into other aspects of this problem such as the costs associated with the instantiation and migration of filters, the delay caused due to processing by filters as well consideration of more complicated topologies. An actual implementation using Aglets technology is also in progress.

References

[Aguilera98] Aguilera et al, "Matching Events in a Content-based Subscription System", *http://www.research.ibm.com/gryphon*

[Banavar98] Banavar et al., "An efficient multicast protocol for content-based publish-subscribe systems", *Technical report, IBM* 1998.

[Carzaniga98] Carzaniga et al, "Design of Scalable Event Notification Service: Interface and Architecture", *Tech Report CU-CS-863-98, University of Colorado, Dept of Computer Science,* 1998.

[Donahoo98] M. Donahoo, "Application-based Enhancement to Network-Layer Multicast." *Ph.D thesis, Georgia Inst of Technology,* 1998.

[Joh00] Janotti et al, "Overcast: Reliable Multicasting with an Overlay network", *Proc. OSDI 2000, San Diego,* October 2000.

[Kasera00] Kasera et al, “Scalable Fair Reliable Multicast Using Active Services”, *IEEE Network Magazine,* Jan/Feb 2000

[Kob99] Koblick R., “Concordia”, *Comm ACM, 42(3): 96-97,* Mar 1999.

[OMG97] Object Management Group, “CORBAServices: Common Object Services Specification”, *www.omg.org*, December 1997.

[Opyr00] Opyrchal et al. “Exploiting IP Multicast in Content-Based Publish-Subscribe Systems”, *Middleware 2000, LNCS 1795*, *pp. 185-207,* 2000.

[Segall97] Segall B. and D.Arnold, “Elvin has left the building: A publish subscribe notification service with quenching”, *Proc. AUUG97*, September 1997.

[Tai99] Tai H. and K. Kosaka, “The Aglets project”, *Comm ACM, 42(3): 100-101*, Mar 1999.

[Tamir96] A.Tamir. “An O(pn^2) algorithm for p-median and related problems on tree graphs”, *Operations Research Letters, 19:59-94,* 1996.

[Zhou00] Zhou H. and Suresh Singh, “Content based multicast (CBM) in ad hoc networks”, *Proc. First Annual Workshop on Mobile and Adhoc Networking and Computing* 2000, pp 51-60

Appendix: Optimal Algorithm for placing k agents in an arbitrary (non-binary) tree

We assume the same terminology as in section 3. Let $v_1, v_2, \ldots, v_j$ be children of vertex v. The algorithm is given by following dynamic programming recurrences.

If v is a leaf then

$T(v,i,p)=0$ for all p,i

Otherwise

$T(v,i,p)=\underline{\min}\{$

$\underline{\text{sum}}[1 \leq l \leq j: f(v_l)] + \underline{\min}[a_1+a_2+..+a_j=i-1: \underline{\text{sum}}[0\leq l\leq j: T(v_l,a_l,v_l)]]$, if v has a filter

$j.f(p)+\underline{\min}\ [a_1+a_2+...+a_j=i: \underline{\text{sum}}[0\leq l\leq j: T(v_l,a_l,p)]]$ …otherwise

$\}$

To calculate the values of $a_1, a_2, \ldots, a_j$ which lead to the minimum in the above equations, we define

$$S(v_m,i,p)=\underline{\min}\ [a_m+a_{m+1}+..+a_j=i: \underline{\text{sum}}[m\leq l\leq j: T(v_l,a_l,p)]]$$

The value of S can be calculated by following recursions

$$S(v_j,i,p) = T(v_j,i,p)$$
$$S(v_m,i,p)= \underline{\min}[0\leq l\leq i: T(v_m,l,p) + S(v_{m+1},i-l,p)]$$