Info. Sys. and e-Bus. Mgmt. manuscript No.
(will be inserted by the editor)

Interimistic Data Dissemination*

Sudarshan S. Chawathe,[1] **Abheek Anand**[2]

[1] Department of Computer Science
5752 Neville Hall, Room 224
University of Maine
Orono, Maine 04469-5752, USA
e-mail: `chaw@cs.umaine.edu`

[2] VMWare, Inc.
3145 Porter Drive
Palo Alto, California 94304, USA
e-mail: `spinlock@gmail.com`

Abstract *We present low-overhead protocols for disseminating streaming data in an interimistic environment in which the availability of computers, networks, and other resources is very unpredictable. An important example of such an environment is the state of computing and communication resources immediately following a natural disaster or a similar disruptive event. A key characteristic of such environments is the continual change in network membership and topology. Our protocols are based on using the intentional and extensional overlap among the data requirements of network nodes. We describe our testbed, CoDD, that implements these protocols and provide an experimental evaluation of the techniques.*

Key words interimistic data management; data dissemination; multicast.

1 Introduction

We address the problem of *interimistic* data dissemination in the chaotic environment in the minutes and hours following some catastrophic event, such as a natural disaster. A distinguishing feature of this work is that it

* A preliminary version of this paper appears in the Proceedings of the Third Workshop on E-Business, December 2004, Washington, D.C. This research is supported in part by the U.S. National Science Foundation with grants IIS-9984296, IIS-0081860, and CNS-0426683.

focuses on a very small window of time by data-management standards, ranging from hours to days, but not weeks, months, or years. We may think of the problem as that of temporary data management, emphasizing the premium on operating during the short time following a disaster and the relative irrelevance of long-term properties. This approach is in sharp contrast to the conventional bias of data-management systems, which favor long-term guarantees, such as consistency and durability, over short-term ones. We envision situations in which our methods are very useful for a few hours but are not necessary when the environment is back to normal.

As a motivating scenario that we use throughout this paper, consider the situation just after a devastating hurricane hitting a typical suburban county. Such an event is likely to cause widespread power failures, road blockages, floods, and communication failures, among other problems. Although much of the information infrastructure may be damaged or disabled, there likely are parts of it that function in a reasonable manner. For example, cellular and land-line phones may function in some locations and not others. Power and other utilities may have similar spotty availability. Now consider an ambulance responding to medical emergencies in this environment. It is likely that the crew is able to communicate with the central dispatch office using their reserved wireless frequencies. However, the office may not be able to provide the crew with the information it needs because of problems with its computers or those of other agencies on which it depends. For example, due to problems with traffic management systems, dispatch may not be able to route the ambulance around road closures and other hazards. Further, we must also consider the case in which the dispatch office suffers a catastrophic failure. Point-to-point communications between ambulance and other crews may still be possible over the air waves, but keeping track of a variety of information in an ad hoc manner over the radio quickly becomes unmanageable. As a simple example of a better solution, consider methods that maintain distributed soft state about recent positions of ambulances using on-board Global Positioning System units and wireless communications. Even a rudimentary ability to visualize trails of other vehicles is likely to provide useful information about road conditions.

In the above scenario, it is quite likely that public buildings such as schoolhouses will be converted into emergency shelters and medical centers. Such a center has a wealth of important information of interest to others, such as statistics on injuries, symptoms, and diagnoses that may help planners better judge the situation and quench any burgeoning outbreaks of disease. Conversely, such a center also has pressing information needs, such as the status of medical professionals and supplies in neighboring areas that could be called into service. Some of these needs may be met using telephones and other ad hoc communications. However, the quality of information would be greatly improved by the use of any computer and communication systems that are available at the schoolhouse, even though they are not provisioned for such service and may have variable availability. Although one may expect emergency workers to carry dedicated and well designed information systems, there is a need for information beyond these systems. For example, although not life-saving, the ability to inform people of the status of their homes and friends is valuable.

Methods for disseminating data in this environment face three major challenges: First, it is unlikely that centralized computing infrastructure is functional. Therefore, the data-dissemination methods cannot rely on the functioning of a single, or a few, critical hosts in the network. Rather, they must be capable of effectively disseminating data when few or none of the centralized facilities are functional. Second, the communication network is likely to be unreliable and functioning at only a small fraction of its capacity. Therefore, network bandwidth is a precious resource that data-dissemination methods must use judiciously. Third, the environment is very dynamic, with the availability of data sources, computing facilities, and communication network links changing by the minute. Similarly, the data needed at various sites also changes rapidly. Therefore, methods for data-dissemination in this environment must be capable of rapid adaptation to such changes in the available resources and required data.

With a few exceptions, work that addresses the problem of selective data dissemination is typically designed for a controlled, centralized environment. Siena [4] is an event-based publish-subscribe system that uses a set of special broker nodes in the network to maintain state and forward data to clients. It uses techniques such as query aggregation for efficient filtering, and is designed to work with multiple brokers to scale to a large set of subscribers. Continuous-query systems like NiagaraCQ [9] use grouped subscription queries, coupled with change-based and timer-based events, to notify interested clients of changes in an available data source. Work on filtering systems, such as XFilter [1] and YFilter [15], and querying systems, such as XSQ [22], focuses on fast algorithms for matching a data stream with filters and queries. Our work is designed to distribute data in a completely decentralized, autonomous environment.

The remainder of the paper is organized as follows. In Section 3, we develop the details of the problem of interimistic data dissemination noted above. We describe our solution, implemented in the CoDD system, in Section 3. Section 4 describes an experimental evaluation of our methods. We describe related work in Section 5 and conclude in Section 6.

2 Interimistic Data Dissemination

We now elaborate on the task of selective dissemination of data in the interimistic environment introduced in Section 1. Briefly, we may describe this problem as the interimistic version of the well-studied problem of selective dissemination of information. While the most obvious features that distinguish this version of the problem from earlier work arise from the environment (e.g., lack of provisioning, high failure rates), there are additional distinguishing features as well. There are multiple sources of data and there is no qualitative difference between the hosts that serve as data sources and those that subscribe to data. Further, the arrival rate of data is likely to be bursty and unpredictable.

Consider a stream of XML data emanating from a host. There is a set H of hosts that need to receive selected portions of this stream. Each host in H expresses the data of interest by sequentially subscribing to the data using *XPath* [11] *subscriptions* or standing queries. Nodes have a limit on the number of network connections (fan-out) they are able to support. We wish to determine a strategy for efficiently disseminating the documents according to the subscriptions while honoring the fan-out constraints. Further, the stream of documents is subject to frequent changes in characteristics, resulting in large changes in statistics. The main challenge is devising protocols that adapt to two kinds of changes at low overhead: (1) hosts joining and leaving the network and (2) changes in data characteristics.

Our task is to devise a strategy for data dissemination that satisfies the resource constraints and that provides each host in H with the requested data, over time. The actions required to follow the strategy at run time must be simple, efficient, and local. Operations that may trigger potentially inefficient wide-area changes, such as global query replanning, must be avoided. Specifically, a data dissemination strategy in this context consists of a collection of rules that determine the manner in which hosts transmit data amongst themselves at any point in time. In addition to specifying the initial setup, such a strategy indicates when and how the setup is modified, typically in response to changes in network membership, network topology, data characteristics, or query mix.

We note that we are required to provide a *distributed strategy over time*, not simply a centralized solution to a snapshot of the problem parameters at some instant in time. In particular, we cannot assume non-local knowledge of hosts, network conditions, and data repositories. Of course, we may convert a static centralized solution to a dynamic distributed using periodic re-evaluation and one of the several standard methods for maintaining distributed state. However, such an approach would be very inefficient for even a small number of hosts. A naive solution to this problem is to multicast the stream to each node, with query evaluation done at the individual nodes. However, this approach would entail a significant communication overhead, especially for participants with low-selectivity queries. We seek a method that organizes nodes into a network that takes advantage of the commonalities among subscriptions. However, our focus is on a low-overhead protocol that can respond rapidly to changes in membership, topology, subscriptions, and data characteristics.

3 Data Dissemination in CoDD

In our implementation of the CoDD dissemination network, nodes are organized as a tree rooted at the data source. Data from the source are selectively propagated down tree edges, with each node other than the root receiving data from exactly one other node, its parent in the tree. A tree-structured dissemination network simplifies the tasks of network maintenance as nodes join and leave the network, and as their subscriptions change over time. However, it also excludes certain dissemination methods that require non-tree networks. For example, it is possible, in general, for the data specified by a node's subscription to be routed separately to it via two or more paths in a network. Such strategies are permitted by non-tree dissemination networks, but not tree networks. On the other hand, the extra overhead and complexity associated with such networks may well negate the potential benefit of such strategies. In particular, it is important to note that we are addressing a very dynamic environment. Thus, the ability to rapidly and efficiently adapt the network to changes in the environment is more important than the ability to better optimize a static situation. In what follows, we will restrict our attention to tree-structured dissemination networks. (This restriction is on only the overlay network used for data dissemination and not on the underlying network; we do not make any assumptions about the structure of the latter.)

Each node maintains an aggregate filter query for each its children. This query denotes the set of documents needed by all descendants in its subtree. On receiving a document, a node propagates it to a child only if the document satisfies that child's aggregate filter query. The method used to aggregate queries depends on the query language in use, and has been addressed in prior work [7,4].

Adding a new node to the network potentially results in excess data flowing along the path from the root to the node. For each ancestor of the new node, this excess data consists of documents required by the new node that do not satisfy the subscription query of the ancestor. By choosing the parent of a new node judiciously, this overhead can be minimized. We may phrase the problem as one of minimizing the amount of data flowing on network links when each node receives at least the items satisfying its subscription. This definition assumes that the data emerging from the source in the future is known. In practice, we need an online method that places a node using only information known at the time a node joins the network. Below, we present an informal description of our method. The detailed descriptions of the join, leave, and reorganization protocols appear in the subsections that follow.

When a host h joins the network, or issues a new subscription, it first requests a direct connection to the source s of the data stream. If the request is granted, host h establishes the direct connection for the data it needs. However, it is likely that the source s is already connected to several hosts and cannot support any additional direct connections due to the fan-out constraint. In this case, s forwards the request to one of the hosts, c, that are directly connected to s and that can support additional connections. If no such host can be found, implying they are all operating at their fan-out capacity, the process is repeated, recursively. The process of adding

`n.parent`	`n`'s parent in overlay
`n.children`	list of `n`'s children
`n.ancestors`	list of `n`'s ancestors (including `n`)
`SubTree(n)`	subtree rooted at `n`
`n.query`	`n`'s subscription query
`n.query(D)`	items in `D` that satisfy `n.query`
`n.filter`	`n`'s aggregate filter query
`n.level`	`n`'s depth

Fig. 1 Terminology used by protocols

hosts in this manner results in a tree-structured dissemination network. During this process, connection requests are sent to hosts that are selected to maximize the overlap between the host's *effective subscription* and the new subscription. A host's effective subscription is the smallest subscription that contains (in a query-containment sense) the subscriptions of all hosts in the dissemination subtree rooted at that node.

The above *join protocol* is the static part of the data-dissemination strategy used by CoDD. The dynamic part of the strategy involves periodic reorganizations triggered by locally detected events such as a node's effective subscription being too large compared to its original subscription. The reorganization protocol is similar to the join protocol. The main difference is that instead of moving down the tree from s using *intensional overlap* of the subscriptions as a guide, we use *extensional overlap* of their recent results. The reason is two-fold. On the one hand, extensional overlaps, which are estimated using the intersection of the document identifiers of recent results for the two subscriptions, are determined much more efficiently than intensional overlaps, which require query-containment and common-subexpression computations [7]. On the other hand, they may be effective at determining overlaps that do not exist based on the known constraints on data, but that are exhibited by recent documents. Although intensional overlap certainly influences the extensional overlap, the relationship may be weak in practice. For example, queries that are not very close intensionally, such as `//computer[model = "iMac"]` and `//computer[color = "tangerine"]`), may have 100% overlap extensionally because of unknown dependencies. (Hypothetically, the only tangerine computers are iMacs.) Our protocols favor extensional techniques over intensional ones, although they use both.

3.1 Join Protocol

Figure 1 summarizes some of the terminology used in the following descriptions of our protocols. The System Join Protocol, outlined in Figures 2 and 3, defines a distributed procedure for a new node to attach itself to the CoDD network. In our descriptions, we use `send_sync_msg` to denote

```
// protocol for joining
proc join_codd() {
  seed_set s = send_sync_msg(system_agent,
                             TYPE_GETSEEDSET);
  list candidates = {};
  forall (node n in s) {
    send_async_msg(n, TYPE_ADD, this);
  }
  // wait for timeout, or all nodes to respond
  wait(candidates.length = s.size, TIMEOUT);
  if (candidates.length == 0) return FAIL;

  node p = min_cost_candidate(candidates);
  candidates.remove(p);

  while (send_sync_msg(p ,TYPE_ADDCHILD,
                     this) == FALSE) {
    if (candidates.isempty())
      return FAIL;
    p = min_cost_candidate(candidates);
    candidates.remove(p);
  }
  // wait to receive data
}
```

Fig. 2 CoDD Join Protocol

synchronous message passing, and `send_async_msg` to denote asynchronous communication. When the former is used, the return value is available immediately; the return value from a use of the latter is available only using a `message_handler` function that is asynchronously called when the action requested by the message is completed. The protocol uses a distributed greedy strategy to determine the insertion point for new nodes. An incoming node, `m`, obtains a seed set `S` of existing nodes using an out-of-band mechanism. Our implementation uses a special `system_agent` node for this purpose. Typically, the seed set `S` consists of the node, `ROOT`, that is the origin of the data stream. In general, the seed set is an arbitrary set designed to avoid a single point of congestion. For example, in a wireless network, the seed could be a randomly selected node in communication range of the new node. The `min_cost_candidate` function is used by each node to determine a point of connection to the network from among a set of potential candidates, by using a suitable cost-estimation function, such as network latency.

We define `compute_overhead(p, n)` to be the additional overhead incurred by node p if node n were to be added as a child of p. Overhead refers to the data that p must receive in order to propagate to n and that does not satisfy p's subscription. The function `max_overlap_node(q, n)` computes the node p in $n.children$ that is estimated to incur the least overhead upon

```
// message handler for join protocol
async proc message_handler(type t, args a) {
  switch (t) {
    case TYPE_ADD:
    // handle a request to add
      if (spare capacity available) {
        send_async_msg(a.sender,
                       TYPE_ADDRESULT, this);
        return;
      }
      node n = max_overlap_node(a.query, this);
      send_async_msg(n, TYPE_ADD, a);
    break;
    case TYPE_ADDRESULT:
    // received a response to add request
    // add the response to list of candidates
      candidates.add(a.sender);
    break;
    case TYPE_ADDCHILD:
    // add this node as a child
      this.children.add(a.sender, a.query);
      if (this.parent != null)
        send_async_msg(this.parent,
                       TYPE_ADD_QUERY, a.query);
        send_sync_response(a.sender, TRUE);
    break;
    case TYPE_ADD_QUERY:
    // update aggregate filter query
      this.query.add(a.query);
    // and update of parent as well
      if (this.parent != null)
        send_async_msg(this.parent,
                       TYPE_ADD_QUERY, a.query);
    break;
  }
}
```

Fig. 3 Message Handlers for Join Protocol

accepting a node with query q as a child. We refer to this node p as the *maximal-overlap child* of n for query q.

The join protocol for an incoming node `m` begin with `m` sending a `TYPE_ADD` message to each node in the seed-set S. Each node in S propagates this message recursively to its maximal-overlap children for *m.query*. (Ties in the overlap are resolved by the node `m` randomly.) When the message reaches a node whose current fanout is less than its maximum fanout, the new node is attached as a child of this node. By terminating the protocol when the first node that can accept new children is found, the protocol favors the construction of short trees, resulting in low latency. The early termination also results in low overheads: The number of messages exchanged is bounded by $h \times s$, where h is the maximum depth of the tree, and s is the size of the seed set. By making some simplifying assumptions about the data and query distribution, the network can be modeled as a random split tree [14]. With this model, the depth of the tree when constructed incrementally using the above protocol grows logarithmically in the size of the node set. Thus, we may expect the protocol to create balanced tree networks with low overheads. We note that the protocol is not guaranteed to position new nodes at locations that maximize overlap. However, such an optimal solution would require the protocol to examine the network more extensively, incurring additional overheads. We quantify the effects of this design decision in Section 4.

The maximal query overlap computation (function `max_overlap_node` in Figure 3) uses two methods: (1) an extensional method based on statistics on recently encountered data and (2) an intensional method based on structural overlaps among subscriptions. For the the first method, we maintain, at each node, statistics that describe the data encountered so far. These statistics are used to estimate the commonality of the requirements of a pair of nodes by comparing the document-sets received by them in the recent past. We use *bitmaps* and simple *summary structures* at each node to describe data that have passed through it. For each document arriving at a node, that node's bitmap has the bit for the document's ID set to 1. Nodes can efficiently determine overlap by exchanging and comparing these bitmaps.

In general, the summary structures depend on the data and query model. We describe below the scheme we use for XML documents and XPath twig queries. At each node, we maintain a *data-summary tree* that summarizes the set of XML documents encountered at that node. Each vertex of the tree has a document-ID bitmap in which the bit for a document-ID is set if the XML data tree (DOM tree) for the corresponding document had a similarly labeled vertex at the same position as in the summary tree. For example, consider document i with a root `n` that has two children: `left` and `right`. When this document arrives at a node, that node's data-summary tree is modified by setting i^{th} bit in the bitmaps of nodes corresponding to `n`, `left`, and `right`. If the subtree composed of these three nodes does not exist in the data-summary tree, then it is added. As new documents are encountered, this structure can be efficiently updated in a single pass on the XML tree of the new document. Lookups to check for the set of documents that match a given *twig* query are performed by tracing the twig pattern representing the given query on this structure, and checking to see which document ID bits are set in *each* vertex of the subtree traversed in this process. The query-overlap computation for a new node consists of comparing the subscription query of a new node with the set of documents received at prospective nodes as determined above. Our method uses a least-recently-used (LRU) policy to bound the size of the summary structure.

Although the above method works well once several documents have passed through the dissemination system, it is not meaningful immediately

```
proc leave_codd() {
  if (is_leaf(this)) {
    send_async_msg(this.parent, TYPE_LEAVE, this);
    return;
  }
  node n = max_overlap_node(this.query, this);
  send_async_msg(n, TYPE_NEWPARENT, this.parent);
  forall (node c in n.this.children) {
    send_async_msg(c, TYPE_NEWPARENT, n);
  }
}
```

Fig. 4 CoDD Leave Protocol

after the system is brought up because the statistics have not been seeded sufficiently. In this priming phase, CoDD uses a structural match between queries to estimate the overlap in their sets of matching documents. For XPath, our structural-match methods compares the XPath structures and literals in the queries in order to obtain an estimate of the intensional overlap. Any inaccuracies introduced by this method do not cause long-lasting inefficiencies because of reorganizations, as described in Section 3.3.

3.2 Leave Protocol

In CoDD, a node may leave the network gracefully after it has received the data it needs. The leave protocol for a node `n` reorganizes the subtree rooted at `n` locally. It replaces `n` by the child `n`* of `n` that has maximal overlap with `n`. By choosing a replacement with maximal overlap, the protocol attempts to restructure the subtree rooted at `n` in a manner that minimizes network traffic. The protocol then re-inserts each of the existing children of `n`, except `n`*, into the network using the join protocol described above, with seed set $\{$`n`$^*\}$. Finally, it uses a similar method to add each of `n`*'s children to the subtree rooted at `n`*. Thus, the only set of nodes affected by `n` leaving are the children of `n` and its replacement, `n`*. The protocol is summarized in Figure 4. When a node leaves the network ungracefully, its departure is detected by each of its children using keep-alive messages and timeouts. Each orphaned child re-inserts itself into the network by initiating the join protocol. However, it may use a modified set of seed nodes that reflects its knowledge of the network. For example, it may initiate the protocol at its earlier grandparent.

3.3 Reorganization Protocol

The reorganization protocol is the reactive component of CoDD. The protocol is initiated by a node when it observes a rise in the number of extraneous

```
// message handler for leave protocol
async proc message_handler(type t, args a) {
  switch (t) {
    case TYPE_LEAVE:
    // remove child
      this.children.remove(a.node);
      send_async_msg(this,
                     TYPE_REMOVE_QUERY, a);
    break;
    case TYPE_REMOVE_QUERY:
    // remove query from filter
      this.query.remove(a.query);
      if (this.parent != null)
        send_async_msg(this.parent,
                       TYPE_REMOVE_QUERY, a);
    break;
    case TYPE_NEWPARENT:
    // add node to new parent
      send_async_msg(a, TYPE_ADDCHILD, this);
    break;
}
```

Fig. 5 Message Handlers for Leave Protocol

documents it receives, and results in the subtree rooted at that node moving to another part of the network. The protocol associates a cost with the reorganization, and accepts a reorganization request only when the expected gain to the system outweighs the cost of the reorganization.

CoDD maintains a timeout parameter for each node, which defines a constraint on the amount of time that must pass between successive reorganization requests for any given node. A request for reorganization can be made periodically after a configurable time-limit has passed (the reorganization frequency), or on demand when the node detects an increase in the amount of spurious data it is receiving beyond a threshold (the reorganization overhead threshold). Thus, the protocol allows the system to react to increases in the data overhead, while the timeout restriction prevents nodes from unfairly initiating these requests very often.

We present a set of policies that can be used by CoDD to limit the overhead of reorganization without significant loss in performance. First, the choice of reorganization frequencies and overhead thresholds can be used to achieve a suitable tradeoff between performance and overhead. Second, *top-c* reorganization accepts the top c pending reorganization requests, where the nodes are ordered by decreasing overhead, and allows the system to move those nodes that are expected to give the highest potential benefit. Third, delta-thresholds chooses only those requests that have an expected benefit higher than a threshold value. This benefit is the difference in the overhead between the current and the new position for the node, and thus

```
proc reorganize() {
  best_parent_overhead = MAX_VALUE;
  queue prospective = {ROOT};
  best_parent = NULL;
  repeat {
    node c = prospective.dequeue();
    (curr_overhead, c.children) =
      send_sync_msg(c, TYPE_GETOVERHEAD, this);
    if (c has spare capacity) {
      if (curr_overhead < best_parent_overhead
          && c != this.parent) {
        best_parent = c;
        best_parent_overhead = curr_overhead;
      }
    }
    else if (2*curr_overhead <
             best_parent_overhead) {
      forall (node p in c.children)
        if (p != this)
          prospective.enqueue(p);
    }
  } until (prospective.length == 0);
  if (best_parent == NULL) {
    // unable to get better parent
    return FAIL;
  }
  send_async_msg(best_parent,
                 TYPE_ADDCHILD, this);
}
// message handler for reorganize protocol
sync proc message_handler(type t, args a) {
  switch (t) {
    case TYPE_GETOVERHEAD:
      // return overhead for reorganize
      overhead = compute_overhead(this, a.node);
      return (overhead, this.children);
   }
}
```

Fig. 6 CoDD Reorganization Protocol

allows only those nodes where the potential gain is known to outweigh the cost of the move.

A high-overhead node that is eligible for reorganization may send a request to a child node with which it has the least overlap (and thus contributes most to its overhead) to move to another part of the network. The protocol proceeds with the child node, `n`, sending a `TYPE_REORGANIZE` message to the `system_agent` node, which checks that the requesting node is eligible for reorganization. The node then proceeds as outlined in Figure 6.

It maintains a current best node while traversing the tree in a breadth-first manner. The protocol prunes search paths in the tree by estimating the overhead of examining the children of a current node, and using this estimate to stop the search along paths that are known to entail higher overheads than the current topology. The algorithm terminates by returning a new parent for the node that needs to be reorganized. The protocol maintains a queue of candidate nodes that may contain solutions. If the node currently examined has a free child slot, it is processed by comparing its overhead with that of the current best node. If the fanout constraint for the node is already satisfied, then the protocol tries to estimate if a child of this node could contain a better solution than the current one. If so, the protocol adds each child of this node to the *prospective* candidate queue. Otherwise, this subtree is pruned from the search space. The `compute_overhead(p, n)` function, as defined earlier, computes the overhead that is expected to be incurred upon adding n as a child of p. Upon determining the new point of attachment in the network, a reorganizing node uses the system join protocol to add itself to the new parent, and does so by sending an asynchronous `TYPE_ADDCHILD` message.

4 Experimental Evaluation

The CoDD protocols are simple and efficient, but are not guaranteed to produce an optimal solution to any given static configuration of the network. Given the highly dynamic nature of the interimistic environment that we have described earlier, the speed with which the protocols respond to network events, such as nodes joining and leaving, is more important than static optimality. Nevertheless, it is desirable that the data-dissemination network produced by the CoDD protocols be efficient in the sense that the amount of data transmitted in the network is small. We develop this idea further below and quantify the performance of CoDD using several metrics. We also compare CoDD with the Siena system [4].

The current implementation of CoDD consists of a set of Java classes that can be deployed in a distributed setting. We have also developed a simulator for nodes, allowing us to test a large number of configurations locally. We conducted our experiments using the Sun Java SDK version 1.4.1_01 on a PC-class machine with a 1.6 GHz Pentium IV processor and 256 MB of main memory, running the Redhat 9 distribution of GNU/Linux (kernel 2.4.22). We simulate network data transfers using standard interprocess communication mechanisms, such as local pipes for the simulator and sockets for the distributed deployment of CoDD. Each node runs an instance of a query engine corresponding to the data and query language in use. This engine evaluates each child's filter query on each data item, and enqueues the item to be sent to each of the children whose filter query it satisfies.

The primary metric we study is the network overhead. Consider the transfer of a data item over a network link. If the item does not satisfy the destination's subscription, the transfer is termed *extraneous.* (The item may be useful to one of the node's descendants in the multicast tree.) A node's overhead is computed by dividing the number of extraneous transfers sent to the node by the total number of transfers sent to it. Similarly, the system overhead is computed by dividing the number of extraneous transfers (occurring over all links in the network) by the total number of transfers. In some cases we are interested in measuring system average over the set of nodes that potentially receive extraneous items (i.e., the non-leaf nodes), and we refer to this as the interior node overhead.

Our test dataset is an abstraction of the mapping between data and subscriptions. We assign nodes to one or more of k category buckets. Intuitively, each bucket represents an interest class. We use $k = 20$ by default. Subscription queries are modeled by mapping each data item to every member of a fixed number of these buckets. The selectivity (s) of the query set for each experiment is defined as the fraction of the total number of documents a query is mapped to, averaged over the total number of queries. We model changes in data distribution by performing a random shuffle on the buckets, at a rate determined by the change-frequency parameter cf. Each shuffle operation consists of splitting c buckets into two sub-buckets and merging uniformly randomly chosen pairs of these sub-buckets to form new mappings. We use $c = k/5$ by default. The default synthetic dataset we used had a selectivity (s) of 0.2, with 400 nodes (n), and 10000 documents (d) being published in the system. The document distribution was made to change every 200 documents (cf), and the default fanout (f) was 6. The reorganization was initiated after every 200 documents were published in the system, by nodes that experienced an overhead greater than a threshold (*rthresh*) of 0.2.

Figure 7 summarizes the results of our experiments studying overhead with a changing document distribution for the default synthetic dataset described above, using a fanout of 6. The figure displays the overhead of the top 100 interior CoDD nodes with and without reorganization (CoDD-R and CoDD-NR, respectively), ranked according to their overhead. Overhead for leaf nodes is always zero, since they do not need to accept items for the purpose of transferring them to their descendants. Reorganization is seen to significantly lower overhead: the average overhead decreases by over 50% as a result of initiating reorganization, indicating that reorganization allows CoDD to adapt well to changing data distributions.

We also compared our system with Siena [4], an event-based publish-subscribe system used to selectively disseminate data to a large set of subscribers. Siena uses a set of coordinated brokers, with each client being mapped to one broker. Brokers communicate with each other depending on the requirements of the clients they serve. This design is significantly different from that of CoDD, and is based on a centrally controlled setup of the server environment. In contrast, CoDD assumes no central coordination. Nevertheless, Siena is the implementation that is closest to CoDD. For purposes of comparison, we emulate CoDD's server-less environment

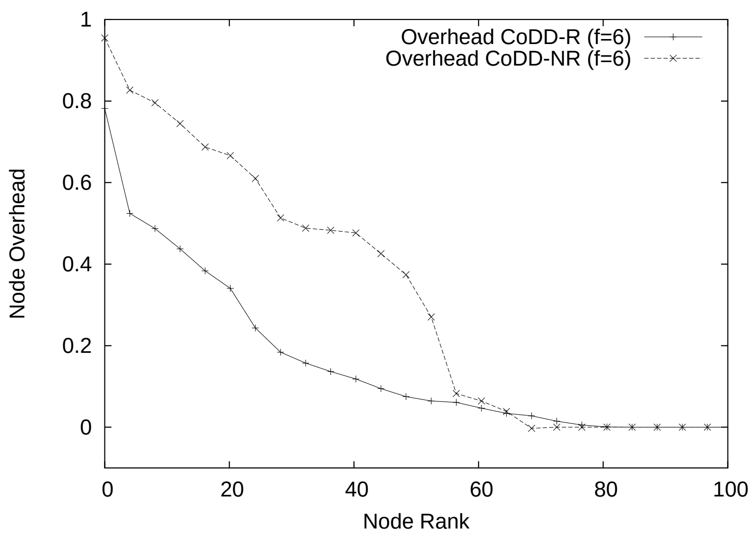

Fig. 7 Overhead versus Reorganization Frequency

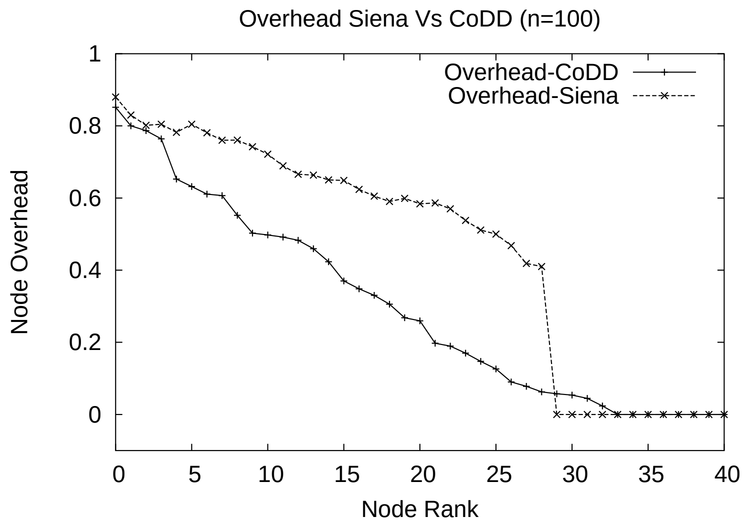

Fig. 8 Overhead of Siena versus CoDD

using Siena as follows. The network is built incrementally, with a fraction of randomly selected nodes designated as brokers. These nodes are connected in a balanced tree topology, and clients connect to a uniformly randomly selected server in this network. We use f clients per server, where f models the fanout constraint in the corresponding CoDD environment.

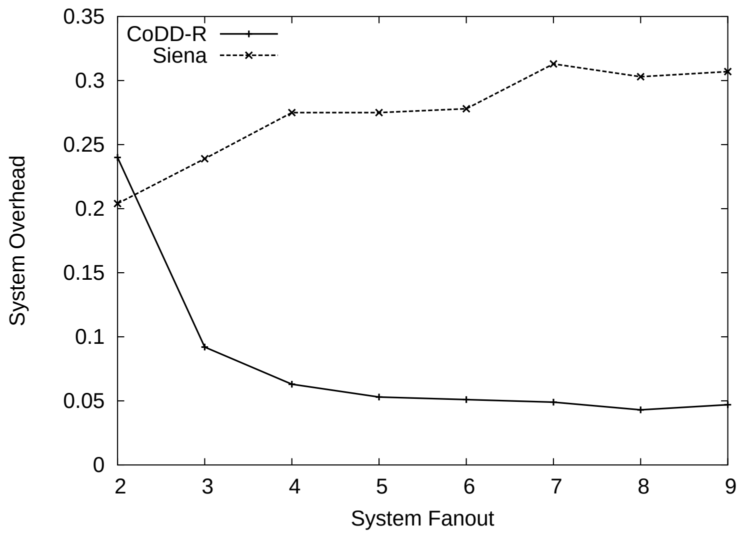

Fig. 9 Average Overhead versus Fanout

Figure 8 depicts the results of experiments comparing the overheads of Siena and CoDD using the default dataset. Our experimental setup favors Siena because we measure the overhead for only the designated broker nodes in Siena. We do not include the overhead incurred by Siena to to set up these brokers in the first place. (Recall that Siena, unlike CoDD, assumes some centralized control for this purpose.) Nevertheless, we note that CoDD perform well because it generates topologies that are data-aware. We note a sharp decrease in the overhead of Siena at $x = 29$. Recall that, in both CoDD and Siena protocols, only interior nodes in the data-dissemination tree incur overhead. However, Siena's centralized setup protocol creates a balanced tree whereas CoDD's completely distributed protocols result in trees with more variable structure. Therefore, the trees produced by CoDD have, in general, a larger fraction of leaf nodes than do the corresponding trees produced by Siena. The leaf nodes incur no overhead, and are responsible for the flat portion of the curve for Siena beyond $x = 29$ in the figure.

In Figure 9, we summarize our study of the effect of fanout on overhead. We observe that the overhead in CoDD drops rapidly as fanout is increased from 2 to 4, with further increases in fanout providing only minor improvements. This result suggests that it is not necessary to use high fanouts, which have the effect of increasing the processing and network load on each node. In contrast, Siena's overhead gradually increases as fanout is increased. This result may be explained by noting that increasing the fanout increases the number of children for each interior node in the tree. These

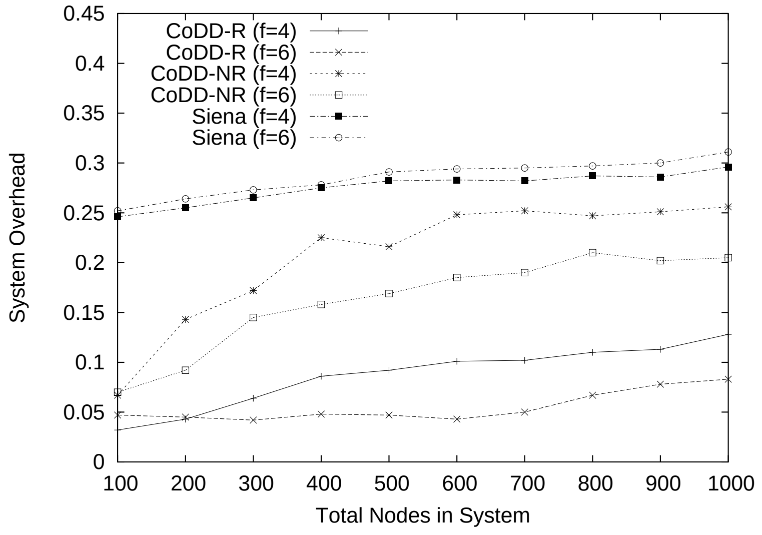

Fig. 10 Average Overhead in System

nodes are usually going to have little overlap with their new children, and thus their overhead is likely to increase significantly.

Figure 10 summarizes the results of experiments varying the number of nodes in the system, with two fanouts (4 and 6), using Siena and CoDD, the latter both with (CoDD-R) and without (CoDD-NR) reorganization. The figure suggests that the overhead is substantially lowered by reorganization. Further, the rate at which overhead increases with increasing network size is lower for the reorganization case.

4.1 Depth of nodes

The depth of a node in the overlay network built by CoDD affects properties such as the latency of data delivery. In general, the depth of a node varies over time due to reorganization. We define the *weighted average depth* of a node to be the average of the depths of that node in the network, where each depth is weighted by the number of documents received by the node at that depth. Smaller values of this metric are desirable, as they indicate a node receiving documents at a lower depth (and thus lower latency) overall. Figure 11 depicts the results of our experiments measuring node depths on the synthetic datasets, with and without reorganization. Reorganization is seen to reduce the weighted average depth significantly, indicating the benefits of adapting the topology to changing document distributions. The curves in Figure 11 have stepped shapes because depths occur in discrete values. The maximum node depth with reorganization is substantially lower than that without reorganization. As a result, there are fewer distinct values of

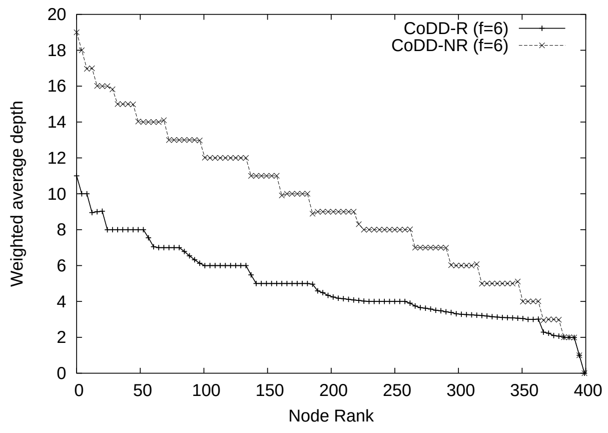

Fig. 11 Distribution of weighted average depths

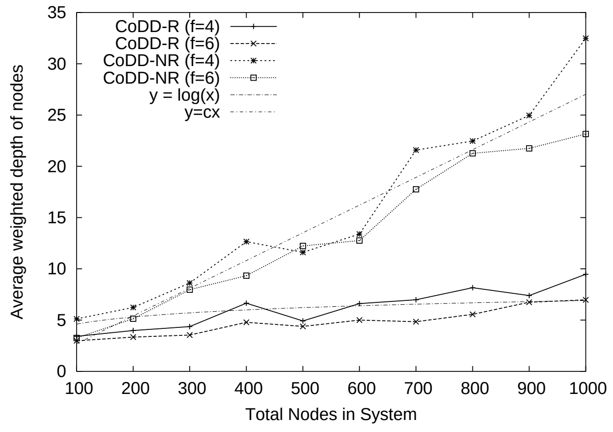

Fig. 12 Effect of network size on average depth

node depths, and thus fewer steps, in the curve for reorganization. Figure 12 depicts the rise in average depth with increasing network size. We note that the increase in depth is gradual, roughly following a $\log x$ curve, indicating that CoDDscales well with a large number of participants. However, without reorganization, depth is seen to increase linearly with network size.

4.2 XML datasets

For this set of experiments, we used XML datasets and XPath subscriptions. The datasets were generated using three real DTDs: (1) the News Industry Text Format (NITF) DTD, available at `http://www.nitf.org`, which is used by many major news agencies; (2) the Protein Sequence

	NITF	PSD	CIML
Max Depth	12	9	6
Avg Depth	5.83	4.37	3.32
Elements	132	66	78
Avg Raw Size (kb)	11.3	7.65	4.54
Recursive	N	N	N

Fig. 13 XML Document Data Used

Database (PSD) DTD, available at the Georgetown Protein Information Resource at `http://pir.georgetown.edu`; and (3) the Customer Identity/Name and Address Markup Language (CIML) DTD, available at `http://xml.coverpages.com/ciml.html`. We generated test data conforming to these DTDs using IBM's XML Generator tool (`http://www.alphaworks.ibm.com/tech/xmlgenerator`). Figure 13 summarizes the properties of the DTDs and datasets.

To generate a workload of XPath queries to be used as subscription queries for each of the nodes, we developed an XPath query generator. The query generator takes as input a DTD and a set of parameters describing query characteristics. These parameters include p_*, the probability of a location step using a wildcard ($*$); p_c, the probability of a node having more than one child; and p_λ, the probability of generating a closure axis.

Figure 14 summarizes the results of our experiments on the three datasets using queries generated with $(p_*, p_c, p_\lambda) = (0.2, 0.3, 0.2)$. The resulting XPath queries had measured selectivities of 0.28, 0.15 and 0.31 for the NITF, PSD and CIML datasets respectively, averaged over each query in the respective query workload. The y-axis of Figure 14 denotes the system overhead, as defined earlier. We measured the overhead in the system with (R) and without (NR) reorganization. Reorganization was triggered every 200 documents. As we may expect, the overhead for the NITF dataset is slightly lower than the overhead for the more complex PSD dataset. However, the variation in data characteristics do not result in significant problems for our protocols. In particular, we observe that CoDD, with reorganization, incurs overheads of less than 10%, even with low fanouts. Further, an important feature that is not immediately apparent is that the CoDD protocols do not include any specialization to the characteristics of any dataset. Thus, good performance is maintained even when, say, the dataset characteristics change from those of NITF to those of PSD.

4.3 Effects of reorganization

In Figure 15, we summarize an experiment that studied the temporal variance of overhead, and the effects of reorganization. We used the the default synthetic dataset for this experiment. Time is measured in increments of events denoting a document being published or or a reorganization occur-

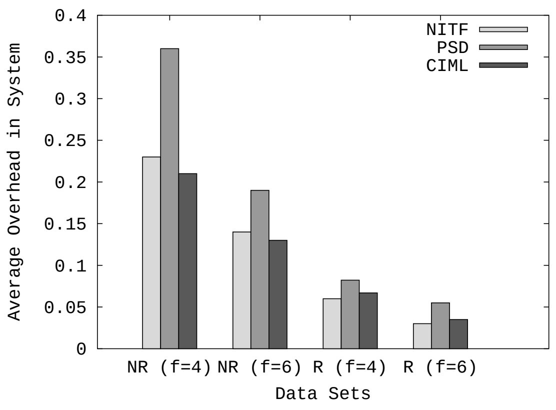

Fig. 14 Performance for different datasets

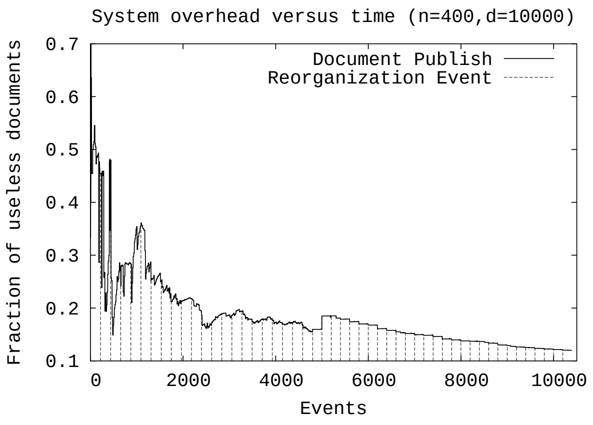

Fig. 15 Overhead versus Time

ring. We measure the average overhead of all the nodes in the system that have positive overhead. That is, we ignore leaf nodes, which have zero overhead. The impulses in the figure indicate reorganization events. The average overhead is seen to drop appreciably near the end of the curve, suggesting the system converging to a low overhead configuration. The results compare favorably with the overhead at interior nodes in a Siena-like configuration, indicating the benefits of the data-aware topologies used by CoDD.

We also analyzed the data from the previous experiment to measure the effects of reorganization events on the overhead of the node that initiates the reorganization. In Figure 16, we quantify the effect of the reorganization by depicting on the y-axis the ratio of the overhead after the reorganization (and before the next one) with the overhead before the organization. The x-axis measures time in units of reorganization events, and unlike the

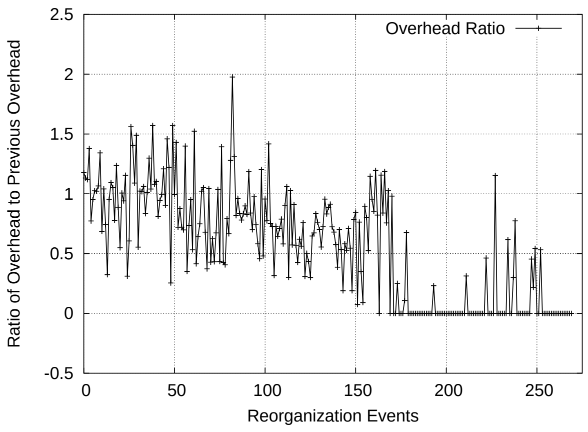

Fig. 16 Reorganization Effects

previous figure, does not include intermediary events like node joins and leaves. The graph indicates that initially the overhead ratio fluctuates significantly. Early reorganizations often do not benefit the requesting node. However, the later reorganizations are much more successful. This result may be explained by noting that the statistics maintained by the system improve over time, allowing it to make more informed decisions.

5 Related Work

Siena [4] is a closely related event-based publish-subscribe system. Clients in Siena subscribe to messages by registering subscription queries. Data messages are routed using cooperating brokers that share subscription information. The primary data model used in Siena represents events as a set of typed attributes. Queries express constraints over a subset of these attributes. Subscriptions are aggregated using internal data structures that describe a partial order on the covering relationships of these queries, and that can be efficiently traversed to match incoming notifications. The brokers comprise a centralized infrastructure that the protocol maintains, and scalability is achieved by adding brokers to the system. SIFT [25] is a keyword-based text filtering and dissemination system for Internet News articles. It supports two profile models, one that does exact boolean predicate matches, and one that does a fuzzy match using a similarity value assigned to every (document, profile) pair. The Gryphon [18] system implements data transformation and dissemination using a similar architecture. An information flow graph is used to describe the flow of events in the system, and each node in the graph implements operators that merge, transform, filter and interpret data items. This graph is then optimized and deployed on a network of brokers, which are used for data delivery. CoDD differs from these systems by using protocols that are decentralized, and that work well with loosely-coupled subscriber nodes without centrally controlled servers. CoDD also enables resource-poor nodes to participate because nodes can manage their level of cooperation using capacity constraints and reorganization. Another difference is the use of extensional overlaps in CoDD. While this feature precludes methods that take advantage of specific features of the query language, it permits easy application of our protocols to a variety of data models and query languages.

Xlyeme [21] is a system used in conjunction with publish-subscribe systems that tackles the problem of finding the events associated with monitoring queries that are satisfied by incoming documents. It uses a hierarchy of hash tables to index sets of atomic events that compose more complex events. Le Subscribe [16] proposes a predicate model to specify subscriptions. To match incoming events (i.e., a set of attribute value pairs) with predicates efficiently, all predicates are indexed, and all subscriptions are clustered by their common conjunctive predicates. A cost model and algorithms are developed to find good clustering structure and to dynamically optimize it. A common feature of these systems, in contrast with CoDD, is the use of restricted profile languages and data structures tailored to the complexity of the languages in order to achieve high throughput.

Network layer multicast [13] is an efficient mechanism for packet delivery in one-to-many data transfer applications. It uses protocols operating in the interior of the network that create a connected tree, with the clients at the leaves of the trees, such that the length of the path from the root to each client along the tree is close to the underlying network path. End system multicast [10, 20] pushes the functionality needed to create this tree to edge nodes on the network, and the data is transmitted over an overlay architecture. The tradeoff for using this end-to-end approach is a sub-optimal performance in terms of network paths and number of duplicate packets on each link. The hierarchical approach used by CoDD is similar to the approach taken in NICE [3], an end system multicast protocol that organizes nodes in a layered tree hierarchy, and is designed to have low control overheads and scale well with the size of the participant set. The fanout constraint is captured in NICE using a parameter that determines the number of nodes at each layers, and bounds the number of outgoing connections for each node. CoDD works in a more general environment, where the data is disseminated selectively to participating clients, and the topology is constructed according to implicit constraints imposed by each clients requirements and the data distribution.

Content-based multicast (CBM) in ad-hoc networks [27] describes a similar problem in wireless sensor networks, where the data needs to be multicast to a subset of the group based on its content. Sensors push data to neighboring nodes, and clients pull data from these nodes. Geographical regions are divided into blocks, and an election mechanism creates leader nodes for each block that are responsible for coordinating the local dissemination. In a CBM system, mobile node express interest in data that is a d distance away, or that will be available after time t (assuming a relative-velocity of the node towards the source). In contrast, CoDD uses general-purpose query languages that allow nodes to express more precise subscription interests. Further, CBM concentrates on wireless node mobility, and uses leader nodes to simulate a centralized infrastructure, while CoDD assumes a decentralized, connected environment that allows it to make more general-purpose optimizations (like reorganization).

Filtering systems are designed to efficiently match data items with a set of predicates expressed in a query language specific to the data model. XFilter [1] uses an XML data model and encodes XPath queries as FSMs by mapping location steps in the expression to machine states. Arriving XML documents are parsed with an event-based (SAX) parser, and the events raised during parsing are used to drive the FSMs through their various transitions. A query is determined to match a document if during parsing an accepting state is reached in the corresponding FSM. YFilter [15] extends this idea for multiple queries, by merging common prefixes into a single NFA, which is processed once for multiple queries. This approach makes it possible to share processing costs between multiple queries, and reduces the number of machine states that need to be maintained. There has also been related work on maintaining index data structures for efficiently filtering data against a set of queries. In contrast with XFilter, XTrie [8] indexes on input substrings that contain parent-child operators rather than element names. This allows it to share the processing of common substrings among queries using this index, and allows it to decrease the number of index probes necessary and avoid redundant matchings. In our description of CoDD, we present policies to create topology structures that are independent of the data model being used. The filtering component in the system can be replaced by any of these techniques to enable fast filtering of data streams without any change to the topology management protocols.

Work on *Query Merging* seeks to reduce the cost of answering a set of subscription queries by grouping similar queries, such that the aggregated queries match a superset of the documents matched by each input query. Given a set Q of queries and a cost model for answering these queries, the task is that of determining a collection M of subsets of Q such that evaluating queries as suggested by M minimizes cost. Previous work has shown QM to be NP-Hard in the general case ($|Q| > 2$), by reducing it to the set cover problem [12]. This work also quantifies the effect of merging for various cost models, and presents a set of heuristics for approximating the QM problem. While CoDD uses similar techniques to group queries, it does so without assuming a-priori knowledge of all nodes (and their respective queries) that will participate in the dissemination. Instead of relying on heuristics to try and optimize the query merging, it uses adaptive reorganization combined with data-independent statistics on query matches to maintain low overhead topologies.

The decentralized services provided by CoDD have several characteristics that are similar in nature to the resource discovery problem in pervasive computing environments. For example, in the VIA system [5], domains organize into clusters based on their resources. Resources are described using a list of attribute-value pairs. Queries are specified as a set of constraints on a subset of these attributes. Clusters are built using upper-bound queries that are created by replacing some constraints by wildcard constraints. These upper-bound queries are then used to discover commonalities in metadata and to build a topology corresponding to these commonalities in a bottom-up fashion. VIA* [6] extends these techniques to allow queries to be generalized based on an impedance parameter. The query impedance is the average number of times a data attribute does not match a query attribute, and describes the relevant importance of that attribute in contributing to query matches. The extensional method for grouping nodes used by CoDD may be viewed as a generalization of this idea. The emphasis in CoDD is low-overhead protocols for dynamic environments that emphasize node autonomy. In VIA, groups of nodes are managed by an administrative domain controller. It would be interesting to explore combinations of these methods.

Web-service interfaces provide an attractive mechanism for CoDD peers to interact in a manner specified by our protocol. It should be interesting to define such interfaces using established and emerging methods for the design of Web-service interfaces [17]. The data-dissemination requirements in the scenario described in Section 1 are a special case of the general problem of providing group-oriented services in a mobile environment [23]. The CoDD system, as described in this paper, uses a syntactic method for matching data to subscriptions. However, since the CoDD protocols do not make strong assumptions about the details of this matching method, it is not difficult to extend them to use other methods, such as those based on ontologies [2]. In generalizing our work beyond data dissemination to more interactive communications between peers in the network, it may be useful to build on work on matchmaking engines such as IPSI-PF [24]. In addition to the best-effort method used by CoDD to provide efficient dissemination, it is interesting to consider methods based on quality-of-service guarantees [26,19].

6 Conclusion

We motivated the need for data management in interimistic environments in which computing and communication resources are continually changing. We discussed the characteristics of such environments and the design guidelines resulting from them. A key feature distinguishing methods that are viable in this environment from those found in prior work is that currency and short-term properties are much more important than long-term properties. We presented our work on the problem of disseminating data in such environments using a server-less network of computers that is in a state of continual change. Our methods are based on simple, low-overhead network-formation and reorganization protocols. We presented our experimental evaluation of these methods. A notable feature of our methods is the emphasis on extensional overlap of queries, determined on recently encountered data, over intensional overlap, defined using the logical analysis of queries. Our results suggest that these protocols, though simple to describe and implement, are well suited to coping with the changes to network membership, network topology, data characteristics, and query mix.

References

1. M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 53–64, 2000.
2. I. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko. Ontology-driven Web services composition platform. *Information Systems and e-Business Management*, 3(2):175–199, 2005.
3. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, Aug. 2002.
4. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, Aug. 2001.
5. P. Castro, B. Greenstein, R. Muntz, P. Kermani, C. Bisdikian, and M. Papadopouli. Locating application data across service discovery domains. In *Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM)*, pages 28–42, 2001.
6. P. Castro and R. Muntz. An adaptive approach to indexing pervasive data. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 2001.
7. C. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Aug. 2002.
8. C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 235–244, Feb. 2002.
9. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 379–390, May 2000.
10. Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, Santa Clara, California, June 2000.
11. J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C Recommendation. `http://www.w3.org/`, Nov. 1999.
12. A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Efficient query subscription processing in a multicast environment. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 83–83, Mar. 2000.
13. S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 8(2):85–110, 1990.

14. L. Devroye. Universal limit laws for depths in random trees. *SIAM Journal on Computing*, 28(2):409–432, 1999.
15. Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Transactions on Database Systems (TODS)*, 28(4):467–516, 2003.
16. F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2001.
17. G. Feuerlicht. Design of service interfaces for e-business applications using data normalization techniques. *Information Systems and e-Business Management*, 3(4):363–376, Dec. 2005.
18. The Gryphon messaging system. `http://www.research.ibm.com/gryphon/`.
19. Y. Huang, X. Geng, and A. B. Whinston. Network mapping services for dynamic selection of web services: promises and challenges. *Information Systems and e-Business Management*, 3(3):281–297, Oct. 2005.
20. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr.. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Symposium on Operating System Design and Implementation*, pages 197–212, Oct. 2000.
21. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the web. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 437–448. ACM Press, 2001.
22. F. Peng and S. S. Chawathe. XSQ: A streaming XPath engine. *ACM Transactions on Database Systems (TODS)*, 30(2):577–623, June 2005.
23. U. Varshney. Group-oriented mobile services: requirements and solutions. *Information Systems and e-Business Management*, 2(4):325–335, Sept. 2004.
24. A. Wombacher, B. Mahleko, and E. Neuhold. IPSI-PF a business process matchmaking engine based on annotated finite state automata. *Information Systems and e-Business Management*, 3(2):127–150, July 2005.
25. T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems (TODS)*, 25(4):529–565, 1999.
26. T. Yu and K.-J. Lin. Service selection algorithms for Web services with end-to-end QoS constraints. *Information Systems and e-Business Management*, 3(2):103–126, July 2005.
27. H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proceedings of the 1st ACM International Symposium on Mobile ad hoc Networking and Computing*, pages 51–60. IEEE Press, Aug. 2000.