Adaptive Content-based Routing In General Overlay Topologies

Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen

University of Toronto
gli@cs.toronto.edu, vinod@eecg.toronto.edu, jacobsen@eecg.toronto.edu
http://padres.msrg.toronto.edu

Abstract. This paper develops content-based publish/subscribe algorithms to support general overlay topologies, as opposed to traditional acyclic or tree-based topologies. Among other benefits, publication routes can adapt to dynamic conditions by choosing among alternate routing paths, and composite events can be detected at optimal points in the network. The algorithms are implemented in the PADRES publish/subscribe system and evaluated in a controlled local environment and a wide-area PlanetLab deployment. Atomic subscription notification delivery time improves by 20% in a well connected network, and composite subscriptions can be processed with 80% less network traffic and notifications delivered with about half the end to end delay.

1 Introduction

Publish/subscribe (pub/sub) is a powerful messaging paradigm that maintains a clean decoupling of data sources and sinks [3–5, 8, 14, 24, 31, 34]. Interoperating through simple publish and subscribe invocations is especially useful for the development of large, distributed, loosely coupled systems. While there are many applications based on group communication and topic-based pub/sub protocols such as information dissemination [17, 22], a large variety of emerging applications benefit from the expressiveness, filtering, distributed event correlation, and complex event processing capabilities of content-based pub/sub. These applications include RSS feed filtering [31], stock-market monitoring engines [33], system and network management and monitoring [7, 20], algorithmic trading with complex event processing [10, 29], business process execution [32], business activity monitoring [7] and workflow management [5]. Typically, content-based pub/sub systems are built as application-level overlays of content-based pub/sub brokers, with publishing data sources and subscribing data sinks connecting to the broker overlay as clients. In pub/sub systems, event *filtering* is supported by atomic subscriptions, whereas composite subscriptions are used by more sophisticated pub/sub applications to *correlate* events from distributed sources [5, 7, 14, 29]. To support the class of applications above, this paper considers both atomic and composite subscriptions.

Most existing pub/sub systems [5, 8, 19, 23] are based on an acyclic overlay broker network. With only one path between any pair of brokers or clients,

content-based routing is greatly simplified. Despite this success, an acyclic overlay offers limited flexibility to accommodate changing network conditions, is not robust with respect to broker failures, and introduces complexities for supporting other protocols, such as failure recovery. For example, since only one path exists between any pair of clients, an acyclic overlay cannot accommodate routing around congested, overloaded, or failed brokers. Furthermore, because acyclic networks are more vulnerable to partitions, failure recovery is more expensive and topology reconfiguration can be complex since their repair actions must maintain the acyclic property of the overlay [27]. Maintaining the acyclic property is difficult since a broker often only knows about its direct neighbors and not the entire topology.

However, supporting general overlay topologies requires changes to the standard content-based routing protocols in order to avoid routing messages in cycles. Consider a topology graph $G = (V, E)$ with vertices V and edges E. Broadcasting a single message in G will generate $\frac{|E|-|V|}{|E|}\%$ of redundant messages. For instance, in a 500 broker topology with an average connectivity of 10 neighbors, a single advertisement induces 2500 messages, 80% of which must be discarded.

This paper focuses on enabling content-based routing in cyclic overlays. Supporting such general overlays affords great flexibility for selecting an optimal routing path based on some optimality criteria or utility function, something not possible in acyclic overlays, where at most one path exists between any data source and sink pair. A novel dynamic routing algorithm takes advantage of this capability to route publications based on changing overlay-link statistics. The flexibility of a cyclic overlay is further exploited by a new composite event detection algorithm that efficiently correlates events from distributed data sources with the objective of reducing the dissemination of unnecessary messages and minimizing the notification delay. By allowing for general overlay topologies, the content-based pub/sub protocols in this paper also provide a foundation for potentially simplifying the support of other features such as failure recovery, load balancing, path reservation, or splitting message streams across multiple paths.

To address the above problems, this paper makes the following contributions. First, Section 3 develops significant extensions to standard content-based routing protocols to enable message routing in general overlay topologies. The design preserves the original simple publish and subscribe interface to clients, and does not require changes to a broker's internal message matching algorithm, allowing the approach to be easily integrated into existing systems. The solution applies to advertisement-based and to subscription-based routing, and exploits redundant routing paths so that publications can be routed to subscribers more optimally, for example, based on assessing load conditions on links. An interesting byproduct of the algorithm is that message routing can be significantly improved by performing matching only once per message, as opposed to existing approaches that match messages at each broker in the overlay. Second, Section 4 presents a novel dynamic routing algorithm for handling composite subscriptions in cyclic overlays. The algorithm is fully distributed, and seeks to minimize the message delay and network traffic by continually adjusting composite subscrip-

tion propagation paths based on the publication traffic and the overlay network load distribution. Third, Section 4 also develops a cost model that is used by the composite subscription routing algorithm to dynamically optimize the placement of composite subscription *join points*, where publications are filtered and correlated. Finally, in Section 5, implementations of the protocols are experimentally evaluated on various topologies in a controlled local network environment and in a wide-area PlanetLab deployment.

2 Related Work

Content-based Pub/Sub: Most pub/sub systems assume an acyclic overlay network. REBECA [8] explores advanced routing algorithms based on an acyclic broker overlay network, and JEDI [5] uses a hierarchical overlay for event dispatching. SIENA [4], however, proposes a routing protocol for general overlay networks, using reverse path forwarding to detect and discard duplicate messages. In this solution, any advertisement, subscription or publication message may be duplicated. As well, routing path adaptations to changing network conditions and the implications for composite event detection are not addressed.

There have been attempts to build content-based pub/sub systems over group multicast primitives such as IP multicast [6]. To appreciate the challenge in doing so, consider a scenario with N subscribers. In a content-based system, each message may be delivered to any subset of these subscribers, resulting in 2^N "groups". It is infeasible to manage such exponentially increasing numbers of groups, and the algorithms seek to construct a limited number of groups such that the number of groups any given message must be sent to is minimized and the precision of each group is maximized (i.e., minimize the number of group members that are not interested in events sent to that group). This is an NP-complete problem [2], but there have been attempts to develop heuristics to construct such groups [23, 30]. To avoid these complexities more recent content-based routing algorithms [4, 5], as well as the algorithms in this paper, have abandoned the notion of groups and rely on an overlay topology that performs filtering and routing based on message content.

The expressiveness and filtering capabilities of content-based routing comes at the price of larger routing table state. Techniques such as subscription covering [4], merging [13, 19] and summarization [34] reduce broker routing state, message traffic and matching overhead, and are orthogonal to the routing protocols in this paper.

There have been a number of content-based pub/sub systems that exploit the properties of distributed hash tables (DHT) to achieve reliability. Hermes [28] builds an acyclic routing overlay over the underlying DHT topology but does not consider alternate publication routing paths as in this paper. Other approaches [3, 9, 21], construct distributed indexes to perform pub/sub matching. This paper, on the other hand, assumes a more traditional dedicated broker network model, one benefit of which is the lack of additional network and computation overhead associated with searching a distributed index to perform pub/

sub matching. The model in this paper can achieve lower delivery latencies when there are no failures, but still use alternate paths for publication routing in case of congestion or failure. Admittedly, the DHT protocols, designed for more hostile network, tend to be more fault-tolerant than the algorithms in this paper which assume a more reliable, dedicated broker network.

Pub/Sub systems have been developed for even more adverse environments such as mobile ad-hoc networks (MANET). These networks as inherently cyclic but the protocols [12, 26] require periodic refreshing of state among brokers due to the unreliability of nodes and links, an overhead that is not required by the work in this paper. As well, MANET brokers can exploit wireless broadcast channels to optimize message forwarding. For example, brokers in ODMRP [12] do not maintain forwarding tables, but only record if they lie on the path between sources and sinks in a given group. Brokers simply broadcast messages to their neighbours (discarding duplicates) until the message reaches the destinations. The protocols in this paper, on the other hand, cannot rely on broadcast transmission and also explicitly attempt to avoid duplicate message delivery. As well, ODMRP does not support the more complex content-based semantics.

Composite Subscriptions: A *composite subscription* correlates publications over time, and describes a complex event pattern. Supporting an expressive subscription language and determining the location of composite event detection in a distributed environment are difficult problems. CEA [29] proposes a Core Composite Event Language to express concurrent event patterns. The CEA language is compiled into an automata for distributed event detection supporting regular expression-type patterns. CEA employs polices to ensure that mobile event detectors are located at favorable locations, such as close to event sources. However, CEA's distribution polices do not consider the alternate paths and the dynamic load characteristics of the overlay network.

One of the key challenges in supporting composite subscriptions in a distributed pub/sub system is determining how the subscription should be decomposed and where in the network event collection and correlation should occur. While this problem is similar to query plan optimization in distributed DBMS [25] and distributed stream processing [11], data in a relation or a stream have a known schema which simplifies matching and routing. Moreover, a database query is evaluated once against existing data, while a subscription is evaluated against publications over time. This may result in different optimization strategies and cost models. In the IFLOW [11] distributed stream processing engine, a set of operators are installed in the network to process streams. IFLOW nodes are organized in a cluster hierarchy, with nodes higher in the hierarchy assigned more responsibility, whereas in our approach, brokers have equal responsibility.

3 Routing in General Overlays

The work in this paper builds on the PADRES distributed, content-based pub/sub middleware platform. PADRES uses advertisement-based routing adapted

from SIENA [4] and REBECA [8]. Each broker records its overlay neighbors in an Overlay Routing Tables (ORT), and advertisements are broadcast through the network forming an advertisement tree. Advertisements are stored in the Subscription Routing Table (SRT) which is logically a list of `[advertisement,last hop]` tuples. Subscriptions are propagated in reverse direction along the advertisement trees, and are stored in the Publication Routing Table (PRT), which is logically a list of `[subscription,last hop]` tuples, and is used to route publications. If a publication matches a subscription in the PRT, it is forwarded to the last hop broker of that subscription until it reaches the subscriber.

Each PADRES broker consists of an input queue, a rule-based matching engine, and a set of output queues. The matching engine takes messages from the input queue and routes it to proper output queues after matching the message content against the SRT and PRT. An output queue represents a destination which could be a broker, a client, a database or a JMS broker.

3.1 Challenges

Cycles in the network introduce misleading message routing paths and redundant traffic. In the network in Fig. 1(a), advertisements Adv_1 and Adv_2 are broadcast and form two advertisement trees. Since brokers discard duplicate advertisements, the trees shown in Fig. 1(a) may vary based on relative message delays. Now, a subscription Sub_1 matching both Adv_1 and Adv_2 arriving at Broker 1 from Broker X is forwarded both towards Broker 6 along the Adv_1 tree, and towards Adv_2. Unfortunately, at Broker 6, Sub_1 matches Adv_2 and is routed back to Broker 1, forming a cycle.

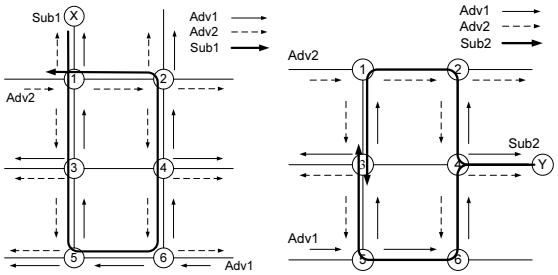
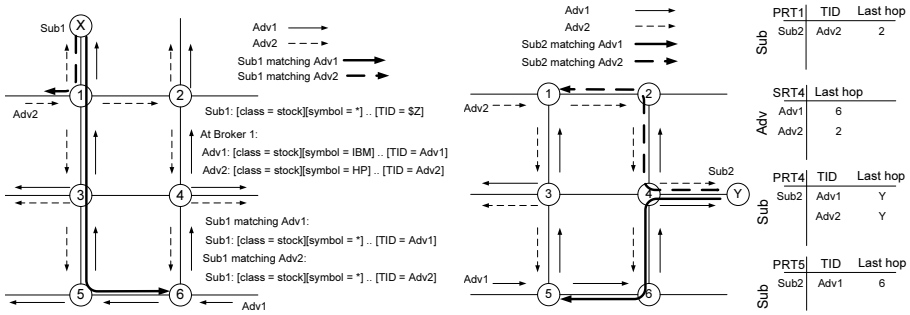


Fig. 1. Cyclic routing of subscriptions

Another scenario of a subscription cycle is shown in Fig. 1(b), where Broker 4, forwards Sub_2 to Brokers 6 and 2, following the paths built by Adv_1 and Adv_2, respectively. However instead of stopping at Brokers 5 and 1, the two copies of Sub_2 continue to be routed unnecessarily. The duplicate subscriptions are not detected until they arrive at the same broker, say Broker 3.

Subscription cycles not only cause redundant subscription messages, they can cause publications to be routed indefinitely in cycles as they keep switching among matching subscription routing paths.

In summary, cycles in the overlay lead to duplicate advertisements, subscriptions or publications, a problem that is exacerbated in well-connected networks with many redundant paths. However, since there are relatively few advertisements, detecting and discarding advertisements is advantageous, provided subsequent cyclic routing of subscriptions and publications can be prevented.



(a) Solution for Scenario 1 (b) Solution for Scenario 2

Fig. 2. TID-based subscription routing

3.2 TID-based routing

We describe extensions to the standard content-based routing using Fig. 2(a) as a running example. These extensions are contained within the routing protocol and do not modify the interface to the pub/sub clients.

Advertisement: Each advertisement is assigned a unique *tree identifier* (TID) within the broker network. In our implementation we use message identifiers, which are unique in our system, as TIDs.

Normally, when a broker receives an advertisement, it broadcasts the advertisement to its neighbors and inserts the advertisement in its subscription routing table (SRT). For cyclic networks, we extend this behavior such that brokers discard duplicate advertisements upon receipt, so each advertisement forms a spanning advertisement tree distinguished by TIDs. As we will see, our approach only requires such duplicate detection for advertisements, which we expect to have fewer of than subscriptions and publications.

Subscription: When a broker receives a subscription from a subscriber, it adds an existential predicate [TID, =, $Z] that uses a variable binding mechanism.[1] If the subscription matches an advertisement in the SRT, the advertisement's TID is bound to the variable $Z. For example, in Fig. 2(a), Sub_1 is matched by both Adv_1 and Adv_2 at Broker 1. A copy of Sub_1 with TID bound to Adv_1 is forwarded to Broker 6 and another copy bound to Adv_2 is forwarded toward the publisher. In our implementation, the TID attribute may have a set of values associated with it so that only one copy of the subscription is forwarded if subscriptions with different TIDs have the same next hop.

A subscription with a bound TID value only propagates along the corresponding advertisement tree. Therefore, when a broker receives a subscription with a bound TID value, it can forward the subscription without matching the subscription against all the advertisements in the SRT. As a result, subscription

[1] The predicate tests whether a message contains a *TID* attribute, in which case the value is bound to the variable in the subscription. Otherwise, the predicate is false.

forwarding is greatly sped up by the use of TIDs, by matching subscriptions against advertisements only once.

Subscriptions set up paths for routing publications. We extend the publication routing table (PRT) to a list of `[subscription, {TID, lasthop of subscription}]`, such as PRT_4 in Fig. 2(b). Since a subscription may arrive at a broker via different paths labeled by TIDs, the PRT records the TID and the last hop broker of the incoming subscription. A subscription not in the PRT is inserted; otherwise the existing record is updated with the new {`TID, last hop of subscription`} pair.

A subscription matching multiple advertisements may be bound to several TIDs, and will form alternate routing paths for publications if the subscription with different TIDs has different last hops. For instance in Fig. 3, *Sub* matches both Adv_1 and Adv_2 at Broker 1, and is assigned two TIDs in PRT_1 with different last hops. Copies of subscriptions with different TIDs propagate along their corresponding advertisement trees and these paths may diverge and reconverge at a broker due to intersections among the advertisement trees. Thus, a broker may receive multiple copies of a subscription with different TIDs. These are not, however, duplicate messages as they correspond to different paths, and are stored in the PRT as potential routing path alternatives for publications.

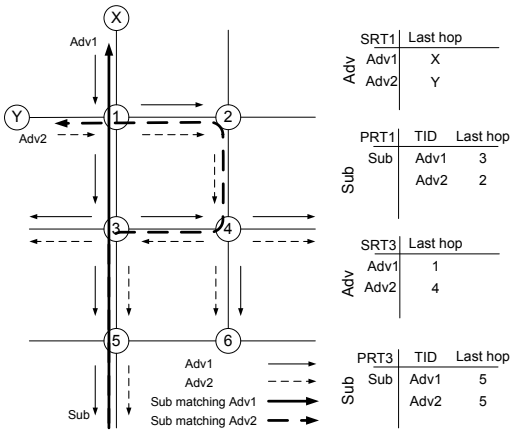


Fig. 3. Multiple publication routing paths

Alternative paths for publication routing are maintained in PRTs as subscription routing paths with different TIDs and destinations. More alternate paths are available if publishers' advertisement spaces overlap or subscribers are interested in similar publications, which is often the case for many applications with long-tailed workloads. Our approach takes advantage of this and uses multiple paths available at the subscription level.

Subscription covering, merging, and summarization optimizations [4, 13, 19, 34] eliminate redundant subscriptions and result in smaller routing tables. These optimizations can be applied among subscriptions with the same TID.

Publication: When a broker receives a publication from a publisher, the publication is assigned an identifier equal to the TID of its matching advertisement. From this point, the publication is propagated along the paths set up by matching subscriptions with the same TID without matching the content of the publication at each broker. This simple and fast *fixed publication routing* algorithm is enabled by the use of TIDs. Alternatively, the *dynamic publication routing* algorithm described in Section 3.3 exploits alternate paths.

Notice that with the TID extensions, subscriptions form a directed, cycle-free graph between publishers and their potentially interested subscribers, so publications are never forwarded in a cyclic manner. In the directed graph, there may be multiple paths between any pair of brokers depending on how subscriptions are routed along multiple advertisement trees. In fixed publication routing, brokers do not need to detect duplicate publications and, consequently, no bandwidth is wasted due to redundant publication traffic.

Property 1 *No broker receives duplicate publication messages.*[2]

It follows from Property 1 that no subscriber receives duplicate publications, since a subscriber connects to exactly one broker, brokers forward a publication at most once over a link, and no broker receives duplicate publications.

3.3 Dynamic Publication Routing

Subscriptions are routed to publishers along advertisement trees. If the advertisement trees of different data sources intersect, multiple publication routing paths to a subscriber result. In Fig. 3, *Sub* is forwarded to Broker 1 over two paths (Path 1 via Brokers 5, 3 and 1, and Path 2 via Brokers 5, 3, 4, 2 and 1). Publications of Adv_1 take the path through Broker 3, while publications of Adv_2 take the path through Broker 2 in the fixed routing approach. However, if the TID of a publication could be adapted in transit, a better path may be chosen. A publication of Adv_2 arriving at Broker 1, could be routed to Broker 3 by changing its TID to Adv_1 instead of being routed to Broker 2.

A routing algorithm is required to determine the "best" path, based on metrics such as the fewest hops or the shortest end to end delay. While similar to the routing problem in IP networks, those solutions cannot be applied to pub/sub systems directly. In address-based routing, the shortest path can be calculated based on a global topology graph, such as with link state routing [18], whereas our brokers are only aware of their overlay neighbors, a property we wish to retain for scalability and manageability. More importantly, IP networks can rely on the clustering of addresses; all nodes in a part of the network may have the same network mask, and can be represented by a single routing table entry. In content-based pub/sub, however, nodes' addresses (i.e., their subscription), may not be clustered. This makes global optimizations infeasible, and instead we use a decentralized solution based on local link status information.

In the *dynamic publication routing* (DPR) algorithm, a broker forwards a message through the link with minimal cost, using the heuristic that this link is also on the minimum cost path to the destination. To dynamically select paths while network traffic loads or topologies change, each broker maintains a *Link Status Table* (LST). For example, to minimize the delay cost, the LST can store the link utilization ratio of each neighbor, and update the ratio whenever messages are sent or received over the link. The link utilization ratio is $U = \frac{r_{output}}{r_{sending}}$,

[2] The proof is presented in [15].

where r_{output} is the rate of messages entering the output queue corresponding to the link, and $r_{sending}$ is the rate of messages sent on that link. The link utilization ratio captures the queueing delay of a link. Other costs can also be modeled in the LST but are not considered in this paper.

When a broker receives a publication, for each matching subscription that may come from multiple links with different TIDs, it selects the link with minimal latency, and assigns the corresponding TID to the publication. The algorithm ensures that, for one subscription with different TIDs, each representing a path from a publisher to the subscriber, only one publication is forwarded to one of the potential neighbors. Also, only one copy of the publication is forwarded to a neighbor if several matching subscriptions come from the same last hop. For example, in Fig. 3 PRT_1 shows that publications matching Sub have two available paths, through Broker 2 or 3. Consulting the LST, the broker will forward the publication to the destination with minimal delay, say Broker 3.

When a broker fails or a link is overloaded, its neighbors will detect the failure or congestion as a result of messages queueing up in the corresponding output queue, which cause the link utilization ratio to increase. Consequently, publications will be routed around the failed or congested broker. While this approach tries to use as many available paths as possible to route messages around failures and congestions, it cannot guarantee delivery, such as in the case of a network partition, until the failures have been repaired by a separate module. Guaranteed delivery is out of the scope of this paper.

Modifying a publication p's TID in transit seems to change the set of subscribers notified of this publication, but this is not the case. Intuitively, the algorithm works because for any given subscription s_i from a subscriber S that matches p, p's TID is only changed to an advertisement's TID that also matches s_i. That means p will be delivered to S by "borrowing" branches of another advertisement tree. The DPR algorithm is formalized in Algorithm 1 and its correctness is proven in Property 2.

Property 2 *Changing a publication p's TID while in transit will not change the set of notified subscribers N.*[3]

Our solution exhibits useful properties. First, it retains the pub/sub client interface. No changes to the pub/sub matching algorithm are required, since TID attributes are matched just like any other attributes. Second, with the TID attribute, optimizations can be performed at each broker to speed up and simplify subscription and publication propagation. For example, subscriptions are matched only once while forwarded to publishers. Third, our approach generates duplicate messages only when broadcasting advertisements; subscription and publication forwarding do not create redundant traffic. Fourth, subscriptions may determine multiple routing paths for publications. The DPR algorithm can route publications around failed or congested brokers, making the system more robust to broker failures and dynamic network traffic. Moreover, the DPR algo-

[3] The proof is presented in [15].

Algorithm 1 Dynamic Publication Routing

Require: An incoming publication $p(c, TID_p)$
Ensure: forwardMsgs: A set of publications with destinations and updated TIDs
1: forwardMsgs = $\emptyset$
2: S = $\{s_i | p$ matches s_i in the PRT$\}$
3: **for** $s_i \in$ S **do**
4: $\wp = \{[TID_j, LastHop_j] | [s_i, TID_j, LastHop_j] \in$ PRT$\}$
5: Find m such that link utilization ratio of destination $LastHop_m$ is minimal in $\wp$
6: nextHop = $LastHop_m$
7: **if** p'.content$\neq$ content and p'.nextHop $\neq$ nextHop,$p' \in$ forwardMsgs **then**
8: p.TID = TID_m
9: p.nextHop = nextHop
10: forwardMsgs = forwardMsgs.add(p)
11: **end if**
12: **end for**
13: **return** forwardMsgs

rithm selects efficient routes based on network conditions to minimize notification delay. This is useful in applications with quality of service constraints.

4 Composite Subscription Routing

PADRES supports an expressive subscription language that can specify constraints on a publication's content and correlate publications in a distributed environment. A composite subscription consists of a Boolean function over atomic subscriptions. For example, the following subscription detects when Yahoo's stock opens at less than 22, and Microsoft's at greater than 31.

```
[class,eq,'STOCK'],[symbol,eq,'YHOO'],[open,<,22] &
[class,eq,'STOCK'],[symbol,eq,'MSFT'],[open,>,31]
```

The detection of event patterns is carried out by the broker network in a distributed manner. In *topology-based* composite subscription routing [14], a composite subscription is routed as a unit towards potential publishers until it reaches a broker B at which the potential data sources are located in different directions in the overlay network. The composite subscription is split at broker B, called the *join point broker*, and each individual part is routed to potential publishers separately. Later, matching publications are routed back to the join point broker, which carries out the composite event detection. Notice that topology-based routing requires an acyclic overlay and does not consider dynamic network conditions.

4.1 Challenges

In a general broker overlay, multiple paths exist between subscribers and publishers, and the topology-based composite subscription routing does not necessarily result in the most efficient use of network resources. For example, composite event detection would be less costly if the detection is close to publishers with a higher publishing rate, and in a cyclic overlay, more alternative locations for

composite event detection may be available. The overall savings are significant if the imbalance in detecting composite events at different locations is large.

In this paper, we develop a novel composite subscription routing algorithm that selects optimal join point brokers to minimize the network traffic and detection delay while correctly detecting composite events in a cyclic broker overlay. The approach benefits greatly from the flexibility made available by content-based routing in general overlays.

4.2 Dynamic Composite Subscription Routing

The *dynamic composite subscription routing* (DCSR) algorithm determines how a composite subscription should be split and routed. A composite subscription is represented as a tree where the internal nodes are operators, leaf nodes are atomic subscriptions, and the root node represents the composite subscription.

The DCSR algorithm traverses the subscription tree as follows: if the root of the tree is a leaf node, that is, an atomic subscription, the atomic subscription's next hop destination in the SRT is assigned to the root, and TID-based routing is applied to each atomic subscription. Otherwise, the algorithm recursively processes the left and right children separately. If the two children have the same destination, the root node is assigned this destination, and the composite subscription is routed to the next hop as a whole. If the children have different destinations, the algorithm has three choices: assigning the current broker or one of the children's destinations to the root node.

The composite subscription detection cost is estimated based on the cost model in Section 4.3. A destination broker with minimum cost will be assigned to the root node. If the root's destination is the current broker, the composite subscription is split. Otherwise, it is routed to one of the neighbors where further decisions are made. The recursive algorithm assigns destinations to the tree nodes bottom up.

4.3 Cost Model

A broker routing a composite subscription makes local optimal decisions based on the knowledge available at itself and its neighbors. The cost function can capture the use of resources such as memory, CPU, and communication. The *total routing cost* of a composite subscription CS at a broker is

$TRC(CS) = RC(CS) + \sum_{i=1}^{n} RC_{N_i}(CS_{N_i})$,

and includes the *routing cost* of CS at the broker and neighbor brokers (N_i, i = 1..n) where publications contributing to CS may come from. CS_{N_i} denotes the part of CS routed to broker N_i, and may be an atomic or composite subscription.

Routing cost at each broker: The cost of a composite subscription CS at a broker includes not only the time needed to match publications (from n neighbors) against CS, but also the time these publications spend in the input queue of the broker, and the time that matching results (to m neighbors) spend in the output queues. This cost is modeled as

Let $D = \int_{Min(a_i)}^{Max(a_i)} d_{a_i}$
$sf_A(a_i = val) = d_{a_i}(val)/D$
$sf_A(a_i > val) = \int_{val}^{Max(a_i)} d_{a_i}/D$
$sf_A(a_i < val) = \int_{Min(a_i)}^{val} d_{a_i}/D$

Table 1. Selection Factor

$$|P(CS)| = \begin{cases} |P(S_l)| + |P(S_r)| & \text{if } op = \|; \\ min(|P(S_l)|, |P(S_r)|) & \text{if } op = \&. \end{cases}$$

Table 2. Subscription Cardinality

$RC(CS) = \sum_{i=1}^{n} T_{in}|P(CS_{N_i})| + \sum_{i=1}^{n} T_m|P(CS_{N_i})| + \sum_{i=1}^{m} T_{out_i}|P(CS)|$, where T_m is the average matching time at a broker, T_{in} and T_{out_i} are the average time messages spend in the input and output queues to the i^{th} neighbor, respectively. $|P(S)|$, defined below, is the cardinality of (atomic or composite) subscription S. To compute the cost at a neighbor, brokers periodically exchange information such as T_{in} and T_m. This information is incorporated into an M/M/1 queueing model to estimate queueing times at neighbor brokers as a result of the additional traffic attracted by splitting a composite subscription there.

Subscription cardinality: The cardinality a subscription S, denoted $|P(S)|$, is the number of matches of S per unit time. First, we define the *selection factor* of subscription S. If S is atomic, its selection factor with respect to advertisement A is defined as $sf_A(S) = \prod_{i=1}^{k} sf_A(p_i)$, where p_i is the predicate of attribute a_i in S and $sf_A(p_i)$ is the selection factor of predicate p_i. The selection factors of individual predicates are computed based on the predicate operator, and the distribution of attribute values d_{a_i} across publications, as shown in Table 1.

An advertisement's attribute distributions are disseminated as a histogram as part of the advertisement. We note two sources of inaccuracy in the selection factor estimation that arise from having to tradeoff accuracy with cost. First, the above equations do not consider the joint distribution among attributes, which would improve the estimation but require more information to be disseminated. Also, the accuracy of the attribute distributions themselves depends on the histogram bucket size and frequency of updates. We leave the exploration of these tradeoffs for future work.

We can now calculate the cardinality of an atomic subscription S that intersects advertisements $\{A_1, A_2, \ldots, A_q\}$, as $|P(S)| = \sum_{i=1}^{q} r_i * sf_{A_i}(S)$, where r_i is the publication rate associated with advertisement A_i. The cardinality of a composite subscription $CS = S_l \ \ op \ \ S_r$ is shown in Table 2.

4.4 Example

We now give an example of how the DCSR algorithm can route composite subscriptions based on the publication traffic and the status of the overlay. Fig. 4 shows a possible routing solution for composite subscription $CS = S_1 \& S_2$, where the S_i are atomic. At Broker 6, CS is routed as a whole towards Broker 4, where the destinations of both S_1 and S_2 are different, and as a result, Broker 4 is the join point broker of CS in the topology-based routing algorithm. If the amount of data from S_1 is significantly larger than that from S_2, it may be more efficient to evaluate CS at Broker 3 rather than Broker 4. In the DCSR algorithm

this dynamic evaluation occurs at each broker until a broker decides to split the composite subscription, say at Broker 1, in which case Broker 1 becomes the join point broker for CS.

Since network conditions, such as delay and bandwidth, may change, the join point chosen by the DCSR algorithm may not remain optimal, and should be computed dynamically. If the join point broker finds a broker that is able to detect the composite event with a cost lower than itself, it initiates a join point movement [16].

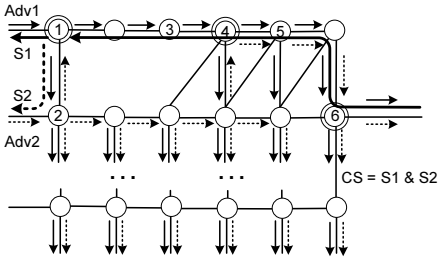


Fig. 4. Dynamic CS routing

The DCSR algorithm determines the location that minimizes the network traffic and message delay costs of evaluating composite subscriptions. The DPR algorithm from Section 3.3 further reduces the cost of composite event detection.

5 Evaluation

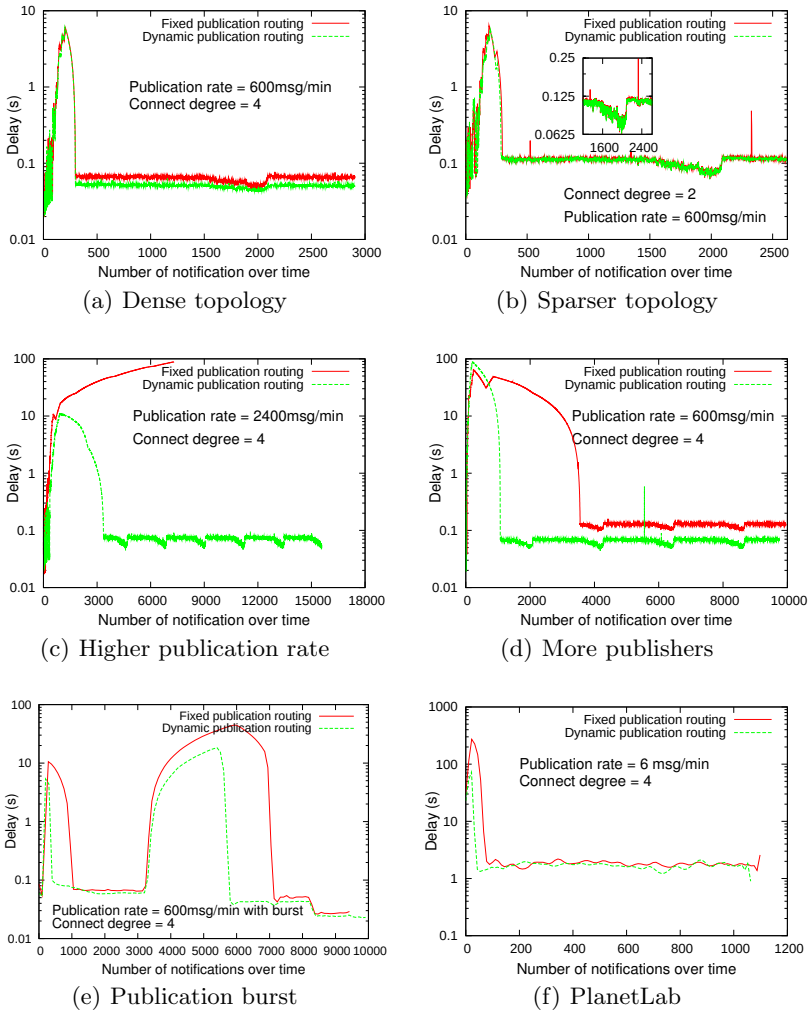
In this section, we experimentally evaluate our routing protocols in general overlay topologies. For publications, we use stock quote traces obtained from Yahoo! Finance. Lacking real subscription traces, we generate subscriptions for the stock quote publication with predicates following a Zipf distribution in order to model locality among subscribers.[4] We explore the properties of the routing protocols by deploying a broker overlay in a controlled 20 machine local network consisting of machines with 4GB of memory and 1.86GHz CPUs. Unless otherwise stated, we evaluate the protocols in a 30 broker with an average connection degree (D) of 4. Each node is a complete content-based pub/sub broker that implements the protocols described in this paper. As well, 20 publishers and 30 subscribers join the system within the first 30 seconds of the experiment.

We compare the end to end notification delay of the fixed (Section 3.2) and dynamic (Section 3.3) algorithms and measure the CPU and memory usage per broker. For dynamic composite subscription routing, we observe the placement of the join point broker by measuring network traffic. To demonstrate robustness and scalability, we also evaluate broker networks deployed on PlanetLab.

End to End Notification Delay The notification delay is computed as the average time between publishing at the publisher to the corresponding notification at the subscriber. This delay varies with the number of hops from the publisher to subscriber and the workload in the broker network, the latter of which depends on factors such as the overall publication rate and the number of subscriptions per broker. The delay metric includes the queueing delay, processing and matching time, and transmission delay at each hop.

When the publishers and subscribers join the system, advertisements are broadcast and subscriptions are matched with advertisements and forwarded to-

[4] The datasets used in the experiments are available at [1].



(a) Dense topology (b) Sparser topology

(c) Higher publication rate (d) More publishers

(e) Publication burst (f) PlanetLab

Fig. 5. End to end notification delay

wards potential publishers. This initial traffic contributes to network congestion and large notification delays, and is visible in Fig. 5(a) where the end to end delay is plotted (in log scale) for each notification. After initialization, the delays of both dynamic and fixed routing algorithms stabilize, with the dynamic algorithm producing a 20.3% shorter average delay than fixed routing. Our results show that brokers in the overlay have different traffic loads. In the fixed routing case, the output queueing delay of an overloaded broker constitutes 25% of the total end to end routing delay. The dynamic algorithm is able to detect such congestion and balance publication forwarding across the overlay.

In a less connected network, the benefit of dynamic routing is diminished due to a lack of alternate paths. In Fig. 5(b), with $D = 2$, the dynamic routing delay is only about 4.7% better than that of fixed routing. At some points in the experiment, the dynamic approach even performs worse than the fixed one because of the overhead of the path selection algorithm. While Fig. 5(a) and Fig. 5(b) demonstrate that the dynamic approach benefits from a well connected overlay, the benefit is not proportional to the connection degree. When the experiment is repeated with a fully connected topology, the delay with the dynamic approach is 4.1% worse than with the fixed approach. In this case, the overhead of maintaining link information does not offset gains in alternate path selection since all publishers and subscribers are already one hop from one another.

We observe that an increase in the publication rate causes the fixed routing approach to suffer worse notification delays. For instance, in Fig. 5(c) when the publication rate is increased to 2400 msg/min, the fixed algorithm becomes overloaded with messages queueing up at brokers along the routing path, whereas the dynamic routing algorithm continues to operate by offloading the high workload across alternate paths. The results suggest that dynamic routing is more stable and can handle heavier workloads, especially in a well connected network. Incidentally, the small but periodic decreases in delay are an artifact of the periodic publication workload. The publications near the end of the workload match fewer subscribers and are filtered by our routing algorithms before they propagate far into the network. This results in less congestion in the network and faster routing of the remaining publications. This phenomenon is apparent in all of the experiments that use this workload.

In the second scenario, we increase the number of publishers to 40, and their advertisements overlap each other. As described in Section 3.2, overlapping advertisements increase the number of potential alternate paths. Fig. 5(d) shows that the delay with the dynamic algorithm stabilizes 5 minutes sooner than with the fixed algorithm. Furthermore, the end to end delay with dynamic routing is 46.5% less than that with fixed routing. (We note again the log scale of the delay axis in Fig. 5). This compares with a relative benefit of 20.3% in Fig. 5(a) where there were fewer publishers and hence fewer alternate paths.

The third scenario evaluates a dynamic workload. A new publisher now joins the system five minutes after the other clients, whose publication rates range from 400∼800msg/min, and publishes a burst of 1500 msg/min for two minutes. We further remove a publisher after the burst. As illustrated in Fig. 5(e), the dynamic approach stabilizes first as in previous experiments. While both algorithms suffer from increasing delays due to the burst, the dynamic algorithm maintains a smaller delay, is able to recover faster after the burst, and has a smaller steady state delay. Overall, the dynamic approach is more resilient to dynamic workloads.

In Table 3, we list the average end to end delay experienced by three subscribers that are 6, 10, and 12 hops away from the publisher. When the publisher and subscriber are close to one another, the dynamic algorithm has fewer opportunities to find suitable alternate paths. In Table 3, with a 6 hop path length,

there is even a 0.8% performance degradation resulting from the overhead of the dynamic algorithm. When the distance increases to 12 hops, the improvement is up to 18.6%. We also observe, reading down the columns in Table 3, that the delay between the 6 and 12 hop subscribers with the fixed approach is 57.7%, while the corresponding difference with the dynamic algorithm is only 27.4%. This suggests that the dynamic approach is less sensitive not only to publication traffic but also to the path lengths between subscribers and publishers.

Distance	**Fixed (ms)**	**Dynamic (ms)**	*Improvement*
6 hops	47.2	47.6	-0.8%
10 hops	64.5	52.9	18.0%
12 hops	74.4	60.6	18.6%
Max diff	57.7%	27.4%	

Table 3. Effect of subscriber distance on delay

An important observation from the above results is that our pub/sub routing algorithms actually *benefit* from a cyclic overlay by reducing the notification delay and increasing the resilience to loads. To demonstrate the robustness and scalability of our approach, we repeat our experiments on PlanetLab with a 50 broker overlay network. The heterogeneous and shared PlanetLab nodes and network make it difficult to derive repeatable and reliable results, but our evaluations on PlanetLab support the conclusions made from the controlled environment experiments. Fig. 5(f) confirms that the dynamic algorithm stabilizes faster than the fixed one, and has a smaller notification delay.

Faster Matching Both the fixed and dynamic routing algorithms have the potential to dramatically improve routing and matching performance. It takes our matching engine about 4.5 ms to match a publication against over 200,000 subscriptions per hop [14]. Once the TID attribute is bound (see Section 3.2), subsequent brokers only need to match the TID attribute instead of the full publication content. This can provide significant savings, especially with complex subscriptions, large workloads, or long path lengths. For the dynamic algorithm in the experiment associated with Fig. 5(a), 1926 publications issued by publishers resulted in 16997 publication messages, requiring 16997 matching operations by brokers. With TID routing, where only the first broker performs full matching, only $1926/16997 = 11.3\%$ of the matching operations are required.

Overhead of Dynamic Publication Routing We measure the CPU and memory usage for all 30 brokers in the network over time for the experiment shown in Fig. 5(a). In the dynamic approach, we need to periodically (10 *ms* in this experiment) update a *link status table* at each broker. When a publication arrives, the broker selects a better path based on the link status table. The average CPU and memory usage per broker in the dynamic routing approach is 6.3% and 8.9% higher than those of the fixed routing approach, respectively. The busiest broker in the dynamic approach has the most neighbors and consumes up to 83.7% of the CPU processing capacity, while the other brokers only consume 16.6% of the CPU. The results show that fixed and dynamic publication routing consume similar CPU and memory usage. Therefore, the dynamic algorithm

reduces the notification delay and the resilience to publication workloads without consuming much more system resources.

Dynamic Routing with Failures In this experiment, we connect 20 publishers and 30 subscribers to a fully connected 30 broker network. We simulate broker failures by randomly killing some of the brokers, and measure the end to end delay of the dynamic routing approach with failures. The number of failures the system can tolerate depends on the connectivity of the network, and the position of the broker in the overlay. For example, a failure that partitions the network will render it impossible to deliver publications across the partitions. In Fig. 6, immediately after the first broker failure, which occurs after about the 1000^{th} notification, the notification delay increases by up to 89.1%. Routing around the failed broker temporarily introduces congestion at some other broker, but the dynamic routing algorithm detects the congestion and automatically balances the traffic among the remaining alternate paths. When the second broker failure occurs at around the 7000^{th} notification, fewer alternate paths are available, and the notification delay now increases up to 217.5%. This time, however, the notification delay stabilizes at about 23.1% higher than before the failure, because the alternate paths the algorithm finds are longer. This experiment shows that the dynamic routing approach can route messages around failures and balance traffic among alternative paths.

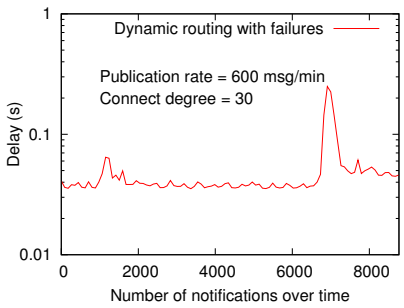


Fig. 6. Failures

Dynamic CS Routing We evaluate the DCSR algorithm on PlanetLab using a topology similar to that in Fig. 4. Twenty publishers publish at rates ranging from 100 to 600 per minute, and 30 subscribers issue subscriptions, with one of them being a composite subscription. In Fig. 7 we measure the bandwidth of certain brokers located on the composite subscription routing path. The solid bars represent the number of outgoing messages at a broker, and the hatched bars are the number of incoming messages that are not forwarded. Note that the sum of the solid and hatched bars represents the total number of incoming messages as a broker. We also measure notification delays in Fig. 8, as measured from when the last publication contributing to the composite subscription is published to when the subscriber receives the notification.

The composite subscription is issued at Broker 6 in Fig. 4, and is routed to its potential publishers using simple routing, topology-based routing or DCSR. In simple routing, a composite subscription is split into atomic subscriptions at the broker that first receives the composite subscription from a subscriber. In this case, the split occurs at Broker 6, and all publications must be routed to this broker where the composite subscription is evaluated and unmatched publications finally filtered out. This is illustrated in Fig. 7 where we see that with simple routing, only Broker 6 filters any messages, and therefore all preceding

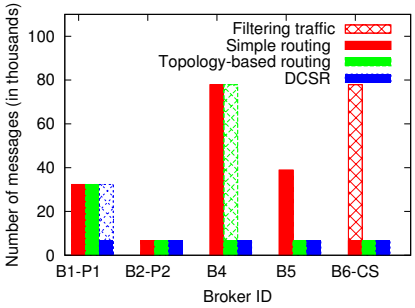


Fig. 7. CS traffic

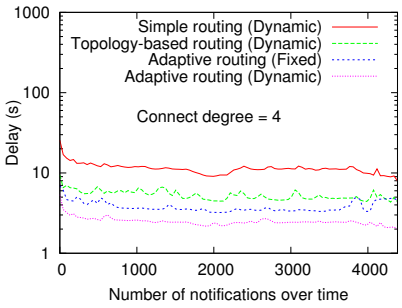


Fig. 8. CS delay

brokers incur higher than necessary message load. In topology-based routing, however, Broker 4 is the join point broker, and we observe the filtering that occurs here in Fig. 7, as well as the reduced message loads at Brokers 5 and 6. The DCSR algorithm determines the composite subscription detection location based on the potential publication traffic. In our topology, the publisher at Broker 1 has a higher publication rate, and hence this broker is an efficient point to detect the composite subscription. Fig. 7 shows that filtering indeed occurs at Broker 1 and that all subsequent brokers enjoy a reduced message load.

To summarize, the topology-based composite subscription routing algorithm imposes less traffic than simple routing by moving the join point into the network, and the DCSR algorithm further reduces traffic by moving the join point closer towards congested publishers as indicated by the cost model. In the scenario in Fig. 7, compared to simple routing, the DCSR algorithm reduces the traffic at Brokers 1 by 79.5%, a reduction that is also enjoyed by all brokers downstream of the join point.

In Fig. 8 we see that with the DPR algorithm, the simple composite subscription routing performs the worst, and the DCSR the best. Even compared to the topology-based approach, the DCSR algorithm manages to reduce the notification delay by 55%, by filtering out messages early in the network and hence reducing queueing delays. In this scenario, we also evaluate fixed and dynamic publication routing with the DCSR algorithm. Fig. 8 shows that the dynamic approach improves the delay by 40.1% compared to fixed publication routing. We expect the benefits to be even more pronounced in larger networks since longer composite subscription paths in such topologies offer the potential for more savings in terms of traffic and delay.

6 Conclusions

Current pub/sub systems [5, 8, 19, 23] assume an acyclic broker overlay network and do not provide special mechanisms to cope with cycles in the overlay. In this paper, we introduce a content-based routing protocol for general overlays supporting both atomic and composite subscription routing. Content-based routing

in a general overlay improves the scalability and robustness of pub/sub systems by offering routing path alternatives. Our approach retains the original pub/sub interface and matching algorithms so it may be easily integrated in existing pub/sub systems. It also minimizes redundant traffic induced by cycles in the overlay and reduces message routing delay. Our protocols allow publications to select optimal paths to matching subscribers and composite subscriptions can be routed to the best event detection locations in order to satisfy potential quality of service constraints at the application level.

Experiments in both wide-area PlanetLab and controlled local environments confirm the benefits of the dynamic publication and composite event routing algorithms. Publication end to end routing is about 20% faster, stabilizes sooner after a burst, and is able to route around certain failures in the network. As well, dynamically determining the optimal composite subscription processing location saves about 80% of the network traffic and reduces end to end delay by more than half.

A worthwhile future research direction for this work is to investigate the feasibility of supporting some of the motivating applications from Section 1 with quality of service guarantees. As well, it would be useful to quantitatively compare content-based pub/sub algorithms designed for acyclic topologies to those in this paper.

Acknowledgments: The research was funded in part by CA, CFI, IBM, NSERC, OCE, OIT, and Sun. We would like to thank Serge Mankovski from CA and our colleagues including Bala Maniymaran for providing valuable feedback on earlier drafts of this paper. The completion of this research was also made possible thanks to Bell Canada's support through its Bell University Laboratories R&D program.

References

1. Experiment datasets. http://research.msrg.utoronto.ca/Padres/DataSets.
2. M. Adler, Z. Ge, J. Kurose, D. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. In *IEEE ICNP*, 2001.
3. I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe DHT network. In *IEEE ICDCS*, 2006.
4. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM ToCS*, 2001.
5. G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE TSE*, 2001.
6. S. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM ToCS*, 1990.
7. T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *ACM SIGKDD*, 1999.
8. L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. Engineering event-based systems with scopes. In *ECOOP*, 2002.
9. A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In *ACM Middleware*, 2004.

10. I. Koenig. Event processing as a core capability of your content distribution fabric. In *Gartner Event Processing Summit, Orlando, Florida*, 2007.
11. V. Kumar, Z. Cai, et al. Implementing diverse messaging models with self-managing properties using IFLOW. In *IEEE ICAC*, 2006.
12. S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A performance comparison study of ad hoc wireless multicast protocols. In *INFOCOM*, 2000.
13. G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *IEEE ICDCS*, 2005.
14. G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *ACM Middleware*, 2005.
15. G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. *TR CSRG-584*, University of Toronto, July 2008.
16. G. Li, V. Muthusamy, and H.-A. Jacobsen. Subscribing to the past in content-based publish/subscribe. *TR CSRG-585*, University of Toronto, Jan 2008.
17. H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for Web micronews. In *IMC*, 2005.
18. D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Academic Press, 2007.
19. G. Mühl. Generic constraints for content-based publish/subscribe systems. In *CoopIS*, 2001.
20. B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *IEEE Network*, 1994.
21. V. Muthusamy and H.-A. Jacobsen. Small-scale peer-to-peer publish/subscribe. In *MobiQuitous P2PKM*, 2005.
22. A. Nayate, M. Dahlin, and A. Iyengar. Transparent information dissemination. In *ACM Middleware*, 2004.
23. L. Opyrchal, M. Astley, et al. Exploiting IP multicast in content-based publish-subscribe systems. In *ACM Middleware*, 2000.
24. K. Ostrowski and K. Birman. Extensible Web services architecture for notification in large-scale systems. In *IEEE ICWS*, 2006.
25. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
26. M. Petrovic, V. Muthusamy, and H.-A. Jacobsen. Content-based routing in mobile ad hoc networks. In *MOBIQUITOUS*, 2005.
27. G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in the presence of topological reconfiguration. In *IEEE ICDCS*, 2003.
28. P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *IEEE ICDCS*, 2002.
29. P. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *IEEE Network*, 2004.
30. A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. In *IEEE ICDCS*, 2002.
31. I. Rose, R. Murty, et al. Cobra: Content-based filtering and aggregation of blogs and RSS feeds. In *NSDI*, 2007.
32. C. Schuler, H. Schuldt, and H.-J. Schek. Supporting reliable transactional business processes by publish/subscribe techniques. In *TES*, 2001.
33. Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *IASTED PDCS*, 2005.
34. P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *IEEE ICDCS*, 2004.