

Adaptive Content-based Routing In General Overlay Topologies

*Guoli Li, Vinod Muthusamy and Hans-Arno Jacobsen
University of Toronto, Canada*

Abstract

This paper contributes publish/subscribe algorithms to support general overlay topologies, as opposed to traditional tree-based algorithms. Among other benefits, this enables message routes to adapt to dynamic conditions, such as congestion or failure, by choosing among alternate routing paths. Our design maintains the simple publish and subscribe protocol interface to clients and is easy to apply to existing systems. We exploit the flexibility of alternate routing paths to introduce a novel adaptive composite subscription routing algorithm that optimizes the cost of evaluating composite subscriptions by propagating composite subscriptions based on potential publication traffic and network status. We implement these algorithms in the PADRES publish/subscribe system and evaluate them in a controlled local environment and a wide-area PlanetLab deployment. Experiments show that the time to deliver notifications improves by 20% in a well connected network, and that the relative benefit of the adaptive algorithm over a fixed one increases with the connectivity of the network, the number of publishers, and path length.

1 Introduction

Publish/Subscribe systems provide a simple and effective method for disseminating data while maintaining a clean decoupling of data sources and sinks [10, 5, 8, 20, 18, 15, 12, 6]. This decoupling can enable the design of large, distributed, loosely coupled, and service-oriented systems that interoperate through simple publish and subscribe invocations. While there are many applications based on group communication [3] and topic-based publish/subscribe protocols [18, 6] such as information dissemination [20, 16], a variety of emerging applications benefit from the expressiveness and filtering capabilities of content-based publish/subscribe. These include network management and monitoring [2, 11], algorithmic

trading with complex event processing [14, 19], business process execution [21, 13], business activity monitoring [11] and workflow management [8].

Most existing publish/subscribe systems [12, 8, 17] 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, solutions for failure recovery and topology reconfiguration in an acyclic overlay can be complex since their repair actions must maintain the acyclic property of the overlay [9]. Maintaining the acyclic property is difficult since a broker often only knows about its direct neighbors and not the entire topology.

This paper focuses on enabling content-based routing in cyclic overlays, providing a foundation for simplifying the support of other features such as failure recovery and load balancing outlined above¹. Adequately handling cycles in the overlay and supporting general overlays also afford 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. Supporting general overlay topologies, however, requires changes to the content-based routing protocols in order to avoid routing messages in cycles. In a general overlay, multiple paths may exist between any two brokers, and a broker may receive duplicate messages over different paths from the same data source. In advertisement-based routing [5], a subscription may be routed in cycles due to multiple matching advertisements. Similarly, dupli-

¹Failure detection and recovery are not addressed in this paper

cate publications may arise due to cyclic routing paths. In subscription-based routing [17], where advertisements are not used, the situation is even worse since subscription flooding generates duplicate messages and publications may be routed in cycles, generating even more duplicate messages. Also, typically there are more subscriptions than advertisements (and more publications than subscriptions), which further exacerbates this problem. In both advertisement- and subscription-based routing, the broker must detect and discard wasteful duplicate messages. For instance, in a 500 broker topology with an average connectivity of 10 neighbors, a single advertisement induces 1485 duplicate messages that must be discarded.

Composite subscriptions support powerful event correlation capabilities, and are used in many sophisticated publish/subscribe applications [19, 15, 8, 11]. In this paper, we further evaluate our design by showing the potential for composite subscription routing in cyclic overlay topologies. The presence of redundant paths in general overlays offer additional possibilities for the routing of composite subscriptions, as there are more alternative points where a composite subscription can be split into its constituent *atomic subscriptions* and a composite event pattern can be detected. The composite event detection may happen at arbitrary brokers in the network based on some optimality criteria instead of being constrained by the broker topology. For instance, composite subscription evaluation may be dynamically placed close to high-volume publishers to filter out non-matching traffic before it is widely disseminated.

SIENA [5] also presents a technique to support content-based routing in general overlay topologies. However, the solution does not offer the potential to adaptively route based on network conditions. The routing paths setup for message dissemination are the shortest paths at the time that subscriptions are forwarded. Moreover, their solution discards duplicate messages, instead of avoiding them in the first place.

The potential for routing composite subscriptions in general topologies is not addressed in the literature. CEA [19] presents a composite event detection framework where mobile event detectors can migrate to other locations according to distribution policies. However, these policies are static rules specified at subscription time, and do not take dynamic network traffic and alternative paths into account. Furthermore, the migration of the event detector and the composite subscription routing are handled outside the publish/subscribe layer.

To address the above problems, this paper makes the following contributions. First, we develop extensions to the standard content-based routing protocol that enable message routing in general overlay topologies. Our design preserves the original simple publish and sub-

scribe interface to clients, and does not require changes to a broker's internal message matching algorithm allowing our approach to be easily integrated into existing publish/subscribe systems. Our solution applies to advertisement-based and to subscription-based routing, and exploits redundant routing paths so that publications can be routed to subscribers via the optimal path. An interesting byproduct of the algorithm is that message routing can be significantly improved by performing matching only once per message. Second, we develop a novel adaptive content-based 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 subscription propagation paths based on the publication traffic and the overlay network load distribution. Composite subscription *joint points*, where publications are filtered and correlated, are dynamically determined in a cyclic overlay based on network conditions. Third, the protocols are fully implemented and experimentally evaluated on topologies as large as 30 brokers in a controlled local environment and 50 brokers in a wide-area PlanetLab deployment.

2 Background and Related Work

Content-based Publish/Subscribe: A publish/subscribe system is comprised of *information producers*, who produce data in form of publications, *information consumers*, who express their interest in receiving publications with subscriptions, and a broker overlay network that connects data producers and consumers. In some systems, producers also send advertisements to define their publication space. In content-based routing (CBR), advertisements are broadcast over the overlay forming an advertisement tree, subscriptions are routed to potential publishers along their advertisement trees, and publications are delivered to subscribers along the paths built by the subscriptions.

Most existing publish/subscribe systems assume an acyclic overlay network. REBECA [12] explores advanced content-based routing algorithms based on an acyclic broker overlay network, and JEDI [8] uses a hierarchical overlay for event dispatching. SIENA [5], 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.

Group communication protocols [3] send messages to a predefined group of receivers with, possibly probabilistic, reliability guarantees. Incredibly successful [4], they

were designed for applications with very large, but relatively few, stable groups. In content-based routing applications, on the other hand, each message may be delivered to a different subset of receivers based on the message content. The Quicksilver [18] publish/subscribe protocol addresses, among other goals, scalability to a large number of groups, but still differs from this work in being a topic-based approach. To appreciate this point, 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 likely infeasible to manage such exponentially increasing numbers of groups, and content-based routing algorithms typically abandon the notion of groups and rely on some form of systematic filtering and routing based on message content. This work also differs in not explicitly addressing reliability but providing mechanisms—namely, relaxing acyclic topology requirements and providing alternate routing paths—to achieve reliability.

Scribe [6] is a channel-based publish/subscribe system built over the Pastry DHT. Scribe treats a channel name c as a DHT key and stores it at the channel root peer r . Subscriptions are sent towards r , with their reverse paths building a multicast tree. Publications are also sent to the channel root, and then follow the multicast tree to the subscribers. The content-based routing in this paper has more expressive filtering capabilities than Scribe’s channel-based approach. A different approach is taken in [7] where nodes are organized into communities based on their interests. A robust routing algorithm is developed, but does not guarantee that all interested subscribers are notified of a publication. An interesting containment-based partitioning scheme is presented for addressing this. RON [1] builds an overlay IP routing substrate that can detect and route around outages and performance failures. While the point-to-point routing in RON is different from our publish/subscribe interface, some of the path redundancy estimation and selection techniques complement this work.

In our approach, a set of dedicated publish/subscribe brokers rather than a peer-to-peer network is used. Moreover, we extended content-based routing to operate in general overlay networks minimizing redundant traffic caused by cycles in the network. Our focus lies on optimizing routing paths for publication message dissemination according to changing network conditions.

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 [19] 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.

PADRES [15] 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 set of atomic subscriptions connected by Boolean operators. It is represented by a tree in which the composite subscription is the root, atomic subscriptions are leaves, and operators are internal nodes. A composite subscription detects a composite event by correlating publications from multiple distributed data sources, with each data source satisfying an atomic subscription within the composite subscription. In addition, variables allow correlation of attribute values across publications. A composite subscription is routed as a unit towards its potential publishers until it reaches a *joint point broker*, which is the first broker at which data sources that may contribute to satisfying the composite subscription are located in different directions in the overlay network. *Topology-based composite subscription routing* [15], which requires an acyclic overlay and does not consider dynamic network conditions, splits the composite subscription at the joint point broker which then carries out the composite event detection. In this paper, we develop a novel adaptive composite subscription routing algorithm that seeks to minimize the network traffic and detection delay while correctly detecting composite events in a cyclic broker overlay. The dynamic composite event detection algorithm maintains the joint points during detection according to the changing network conditions. The approach benefits greatly from the flexibility made available by content-based routing in general overlays.

The PADRES System: PADRES (Publish/subscribe Applied to Distributed Resource Scheduling) [15] is a distributed, content-based publish/subscribe middleware platform. It uses a predicate-based subscription language, in which an atomic subscription is a conjunction of attribute-operator-value tuples and a composite subscription consists of atomic subscriptions connected by logical operators. The overlay network connecting the brokers forms the basis for message routing. Each message has a unique message identifier, which is a concatenation of a broker identifier and a broker-specific sequence number. Each broker records its overlay neighbors in an Overlay Routing Tables (ORT). A PADRES broker consists of a rule-based engine to perform scalable message routing and matching. Clients connect to the broker overlay and issue messages into the broker overlay. PADRES is based on advertisement-based

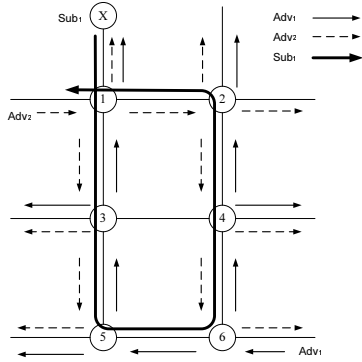


Figure 1: Subscription routing scenario 1

routing which is adopted from the SIENA [5] and REBECA [12] projects. In PADRES, with the ORT, advertisements are broadcast through the network forming an advertisement tree that spans all brokers. Advertisements are stored in the Subscription Routing Table (SRT) which is essentially a list of [advertisement, last hop] tuples. The subscriptions are propagated in reverse direction along the advertisement trees. Subscriptions are stored in the Publication Routing Table (PRT). Like the SRT, the PRT 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.

3 Routing in General Broker Overlays

Cycles in the overlay network introduce many problems in terms of misleading message routing paths and redundant network traffic. Consider the publish/subscribe overlay network from Fig. 1, there are two publishers issuing advertisements Adv_1 and Adv_2 . These advertisements are broadcast through the network to form an advertisement tree for each publisher, as indicated in Fig. 1. Note that brokers discard duplicate advertisements in order to remove cycles in advertisement propagation. Now, when Broker 1 receives a subscription Sub_1 matching both Adv_1 and Adv_2 from Broker X, Broker 1 forwards it to Broker 6 along the advertisement tree built by Adv_1 . Unfortunately, at Broker 6, Sub_1 matches Adv_2 and is routed back to Broker 1, forming a cycle.

Another scenario of a subscription cycle is shown in Fig. 2, 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 sub-

scription messages, they mislead subsequent publication dissemination, with redundant publications routed along the subscription cycle. Furthermore, just as subscriptions can form cycles switching repeatedly among multiple matching advertisement trees, publications may propagate in cycles as they switch among matching subscription routing paths.

In summary, due to cycles in the overlay, there may be duplicate advertisements, subscriptions or publications. However, since advertisements often make up the least number of messages, relative to subscriptions and publications, detecting and discarding advertisements is advantageous, provided subsequent cyclic routing of subscriptions and publications can be prevented. Indeed, our experiments show that in cyclic overlays, duplicate message may cause a three order of magnitude increase in message traffic. This problem is exacerbated in well-connected networks with many redundant paths, and special algorithms and data structures are required for detecting and discarding duplicates.

3.1 Extended CBR Protocol

Subscription and publication cycles are a result of subscriptions propagating back and forth over different intersecting advertisement trees. A subscription starting from a node of one advertisement tree is routed back to the original node via branches of other advertisement trees. Consequently, the problem amounts to distinguishing among advertisement trees so subscription propagation does not switch between trees, thus avoiding subscription cycles. We now describe our extensions to the standard content-based routing protocol for general overlay networks, using Fig. 3 as a running example. These extensions are contained within the routing protocol and do not modify the interface to the publish/subscribe 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

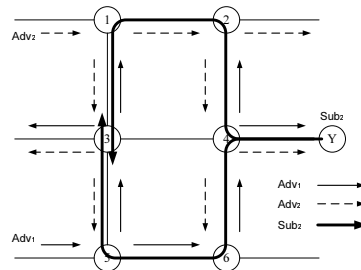


Figure 2: Subscription routing scenario 2

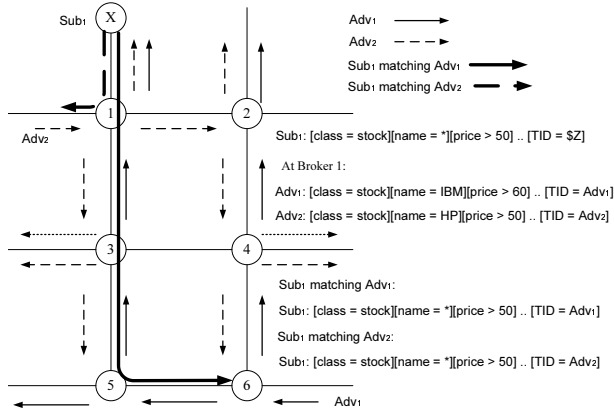


Figure 3: Subscription routing with TIDs: Scenario 1

the advertisement in its subscription routing table (SRT). In cyclic overlays, brokers may receive duplicate advertisement messages due to redundant overlay paths, so we extend this behavior such that brokers discard duplicate advertisements upon receipt. In this way, 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 TID predicate to the subscription, i.e., $[TID = _, \$Z]$, where $\$Z$ is a variable binding mechanism supported in our subscription language [15].² This mechanism extracts the value of a matching attribute; if the subscription matches an advertisement in the SRT, the advertisement’s TID is bound to the variable $\$Z$. For example, in Fig. 3, 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 forwarding is greatly sped up by the use of TIDs, by matching subscriptions against advertisements only once.

Subscriptions set up paths for routing publications.

²We refer to this predicate as an *existential predicate*, since it tests whether a given message contains an attribute-value pair with attribute equal to TID , but without regard to the value. The value is bound to the variable in the subscription. If the attribute-value pair does not exist the predicate is false.

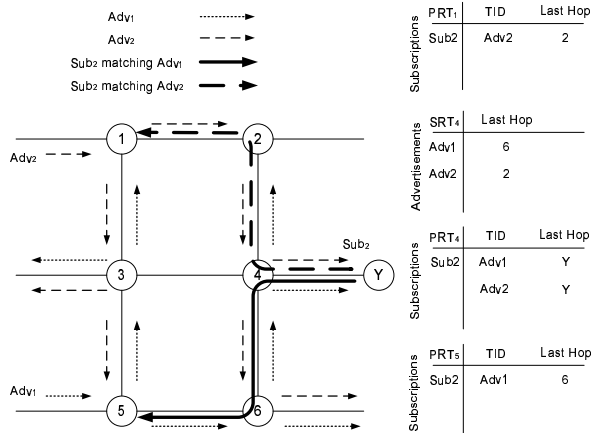


Figure 4: Subscription routing with TIDs: Scenario 2

We extend the publication routing table (PRT) to a list of $[\text{subscription}, \{\text{TID}, \text{last hop of subscription}\} \text{ pairs}]$, such as PRT_4 in Fig. 4. 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. If a new subscription is not in the PRT, it is inserted as a new record; otherwise the existing record is updated with the new $\{\text{TID}, \text{last hop of subscription}\}$ pair. When a subscription matches multiple advertisements, it may be bound to several TIDs, and will form alternate routing paths for subsequent publications if the subscription with different TIDs has different last hops. For instance in Fig. 5, Sub matches both Adv_1 and Adv_2 at Broker 1, and is assigned two TIDs in PRT_1 with different last hops. Note that copies of subscriptions with different TIDs propagate along their corresponding advertisement trees and these paths may diverge and re-converge 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.

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 business and peer-to-peer applications. Our approach takes advantage of this and uses multiple paths available at the subscription level.

Protocols such as simple and topology-based routing have been proposed for acyclic overlays [15]. In simple routing, a composite subscription is split into atomic parts at the first broker that receives it. Here, TID based routing can be applied to the atomic subscriptions in or-

der to support composite subscription routing in cyclic overlays. In topology-based routing, composite subscriptions are split based on the topology, and does not necessarily result in the most efficient use of network resources, especially when data sources have significantly different publication traffic. An adaptive composite subscription routing algorithm is proposed in Section 4.

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. As a result, publication propagation is simpler and faster in the TID approach. We call this algorithm *fixed publication routing*. This routing algorithm simplifies the cyclic overlay by generating advertisement trees for each publisher, and does not take advantage of alternate path information maintained at the subscription level, which is used in the *dynamic publication routing* algorithm discussed in Section 3.2.

Notice that with the TID extensions, subscriptions form a directed, cycle-free graph between a publisher and its 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 3.1 No broker receives duplicate publication messages.

Proof By way of contradiction, assume publication $p(c, t)$ with content c and TID t is received by some broker B_i twice.

Let broker B_0 be the broker that receives a publication $p(c, t)$ from the publisher. Without loss of generality, let B_i be the first broker to receive $p(c, t)$ twice. Notice that B_i must have received $p(c, t)$ from two different brokers B_m and B_n , since brokers never forward the same publication multiple times to a neighbor, denoted as $B_m \xrightarrow{p(c, t)} B_i$ and $B_n \xrightarrow{p(c, t)} B_i$.

Observe that if $p(c, t)$ is forwarded over link $B_m \rightarrow B_i$, then a subscription s with a TID bound to t was sent over link $B_i \rightarrow B_m$, denoted as $B_i \xrightarrow{s_t} B_m$. And if such a subscription was sent, then an advertisement adv with a TID t was sent over link $B_m \rightarrow B_i$, that is $B_m \xrightarrow{adv_t} B_i$. Similarly, we have $B_n \xrightarrow{adv_t} B_i$. The spanning advertisement tree rooted at B_0 . Then there must be two paths from root B_0 to B_i in the spanning tree.

This is a contradiction since trees only have one path between any two nodes. ■

Notice that it follows from Property 3.1 that no sub-

scriber receives duplicate publications, since brokers forward a publication at most once over a link, and no broker receives duplicate publications.

Unadvertisement and Unsubscription: Advertisements and subscriptions can be canceled with unadvertisement and unsubscription messages. The latter are propagated using similar logic as the former.

3.2 Dynamic Publication Routing

Subscriptions are routed to publishers along advertisement trees. If the advertisement trees of different data sources intersect without overlapping, multiple publication routing paths to a subscriber result. For example, in Fig. 5, *Sub* is forwarded to Broker 1 over two different routing paths (i.e., Path 1 via Brokers 5, 3, and 1, and Path 2 via Brokers 5, 3, 4, 2, and 1.) Publications of *Adv₁* take the path through Broker 3, while publications of *Adv₂* 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 *Adv₂* arriving at Broker 1, could be routed to Broker 3 by changing its TID to *Adv₁* instead of being routed to Broker 2.

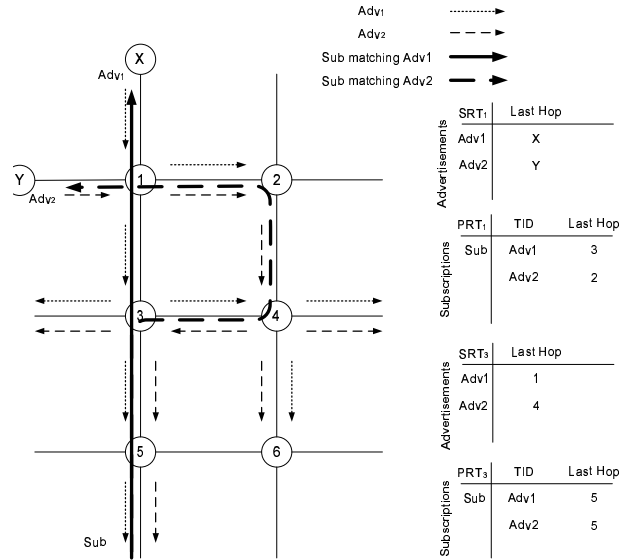


Figure 5: Multiple publication routing paths

A routing algorithm is required to determine the “best” path, that is, the one with “lowest cost”, such as the fewest number of hops or the shortest delay from source to destination. This problem coincides with the well known routing problem in computer networks, and solutions can be classified as global or decentralized. Global routing algorithms are based on complete, global knowledge of the broker network, such as Dijkstra’s algorithm. Since each broker only knows about its neighbors in our

broker network, we apply a decentralized solution based on local link status information. Algorithms that are widely used in practice, such as hot potato routing and reverse path routing, could be used here to select a “shortest” path for publication dissemination. The algorithms differ in their routing performance, overhead traffic, and implementation complexity.

We use an idea similar to the simple *hot potato routing*, which is not optimal but has been observed to perform well in practice. In this algorithm a broker forwards a message through the link with minimal delay, using the heuristic that this link is also on the minimum delay *path* to the destination. To dynamically select paths as the network traffic loads or topology change, each broker maintains a *Link Delay Table* (LDT) for its overlay neighbors, where link delays and link usages ratios are updated whenever it receives/delivers a message from/to this link. 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 next broker decides where to deliver the publication according to its LDT. The publication is routed hop by hop until it reaches its subscribers. It also 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. Only one copy of the publication is forwarded to a neighbor if several matching subscriptions come from the same last hop. When a broker fails, its neighbors will detect the failure which is indicated in their link delay table. As a result, incoming publications will be routed around the failed broker to their subscribers.

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 dynamic publication routing algorithm is formalized in Algorithm 1 and its correctness is proven in Property 3.2.

Property 3.2 Changing a publication p 's TID while in transit will not change the set of notified subscribers N .

Proof Consider a publication $p(c, Adv_1)$ with content c and TID Adv_1 , that is changed to $p'(c, Adv_2)$ where $Adv_1 \neq Adv_2$. We know that any publication in the set $P(c, t)$ where any $Adv_x \in t$ intersects s_i from subscriber S and c matches s_i will be delivered to S . The proof consists of two claims.

Claim 1: No subscriber $S \notin N$ receives p' . Note that $S \notin N$ implies that S is not interested in p , that is, it does not have a subscription s_i that matches p .

Algorithm 1 Dynamic Path Selection

Require: An incoming publication $p(c, TID_p)$

Ensure: forwardMsgs: A set of publication messages to forward, with destinations and updated TIDs

```

1: forwardMsgs =  $\emptyset$ 
2:  $S = \{s_i | p \text{ matches } s_i \text{ in the PRT}\}$ 
3: for  $s_i \in S$  do
4:    $\varphi = \{[TID_j, LastHop_j] | [s_i, TID_j, LastHop_j] \in \text{PRT}\}$ 
5:   Find  $m$  such that  $\text{LinkDelay}(LastHop_m)$  is minimal in paths  $\varphi$ 
6:    $\text{nextHop} = LastHop_m$ 
7:   if there is no  $p' \in \text{forwardMsgs}$  where  $p'.\text{content} == p.\text{content}$ 
     and  $p'.\text{nextHop} == \text{nextHop}$  then
8:      $p.\text{TID} = TID_m$ 
9:      $p.\text{nextHop} = \text{nextHop}$ 
10:     $\text{forwardMsgs} = \text{forwardMsgs.add}(p)$ 
11:   end if
12: end for
13: return forwardMsgs

```

By way of contradiction, assume an $S \notin N$ receives p' . Consider the Broker X that sent p' to S . According to Algorithm 1, this means that Broker X found an s_i in its PRT whose last hop is S and which matches p' . Therefore, subscriber S sent a subscription s_i that matched publication p' . Since p and p' have the same publication content, s_i also matches p , meaning that S is interested in p and $S \in N$, which is a contradiction.

Claim 2: All subscribers $S \in N$ receive p' . Again, note that $S \in N$ implies that S is interested in p , and has a subscription s_i that matches p .

By way of contradiction, assume an $S \in N$ does not receive p' . Therefore, it must be that $p'(c, Adv_2) \notin P(c, t)$, in particular, $Adv_2 \notin t$. But we know that $p(c, Adv_1) \in P(c, t)$, and according to Algorithm 1, $p(c, Adv_1)$ is changed to $p'(c, Adv_2)$ only if both Adv_1 and Adv_2 intersect s_i , implying that $\{Adv_1, Adv_2\} \in t$. Then it must be that $p'(c, Adv_2) \in P(c, t)$. This is a contradiction.

Together, the above two claims prove that the set of notified subscriber N does not change when a publication's TID is changed in transit. ■

Our solution exhibits six useful properties. First, it retains the publish/subscribe client interface, and only makes a few changes to the routing protocol: one predicate and one attribute-value pair are added to each subscription and publication, respectively. No changes to the publish/subscribe matching algorithm are required, since TID attributes are matched just like any other attribute. Adding an attribute to resolve cycles in the overlay is easy to realize in existing publish/subscribe systems. 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; subscrip-

tion and publication forwarding do not create redundant traffic. Fourth, subscriptions may determine multiple routing paths for publications. If a broker fails, its neighbors may detect the failure and update their link delay tables. As a result, the dynamic publication routing algorithm can route publications around the failed broker, making the system more robust to broker failures. Fifth, the dynamic publication routing algorithm selects efficient routes based on network conditions to minimize notification delay. This is useful in applications with quality of service constraints. Last, while we described the algorithm in terms of the standard advertisement-based routing, it works in a similar manner for approaches not employing advertisements.

4 Composite Subscription Routing

As described in Section 2, composite subscriptions can be used to specify complex event patterns. Many sophisticated applications benefit from the powerful publication correlation capabilities provided by composite subscriptions [19, 15] including network management tools [2, 11] where sophisticated intrusion detection patterns must be monitored, business process execution engines [21] that need to observe the status of multiple dependent jobs, and complex event processing engines [14] that need to react to complex event patterns over time and space to support applications such as algorithmic trading.

In a general broker overlay, multiple paths exist between subscribers and publishers, and composite subscription routing becomes more complicated. In this section, we extend the content-based routing protocol for composite subscriptions in cyclic overlays.

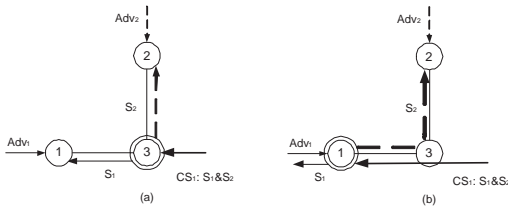


Figure 6: Adaptive composite subscription routing

In a general overlay where multiple paths are available to route publications, the topology-based composite subscription routing does not necessarily result in the most efficient use of network resources. An example is given in Fig. 6, where CS_1 arrives at Broker 3 and it is matched when both S_1 and S_2 are matched. The publishers potentially satisfying the composite subscription are determined by advertisements at the broker. In the example according to topology-based composite subscription routing, Broker 3 is the joint point broker for CS_1 , where

the composite subscription is split into atomic subscriptions, S_1 and S_2 . Each atomic subscription is forwarded in the direction of a publisher emitting publication traffic potentially matching the subscription. Publications are collected, filtered and correlated at the joint point broker, where the actual composite event detection happens based on the standard composite subscription routing mechanism in PADRES [15].

In Fig. 6, if the number of publications matching S_1 outweighs the number matching S_2 , then composite event detection would be less costly if the detection is moved from Broker 3 to Broker 1. While performing the detection at Broker 1 adds the cost (e.g., the number of hops, bandwidth, delay etc.) of moving publications for S_2 from Broker 3 to Broker 1, it eliminates the cost of moving publications for S_1 from Broker 1 to Broker 3. Furthermore, 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.

To determine the best composite event detection location, it is necessary to estimate data set sizes for subscriptions at a particular broker and the cost of evaluating composite subscriptions at a broker and its neighbors. We address both points below.

4.1 Data Set Size

To estimate the potential data set sizes attracted by subscriptions, we maintain statistics of publications at each broker, such as the publication rate and the distribution of an attribute. The overlap function $f(S \cap A)$ between a subscription S and a matching advertisement A indicates what publications may be forwarded to a broker. The ratio of actual publications over all potential publications determined by an advertisement is the *reduction factor* of the advertisement relative to the subscription. Then, the data set size of an atomic subscription is:

Definition 4.1: Data Set Size of Atomic Subscription Suppose an atomic subscription, S , matches a set of advertisements, $\mathbb{A} = \{A_i \mid P(S) \cap P(A_i) \neq \emptyset, i = 1..n\}$, at a broker. Then:

$$|P(S)| = \sum_{A_i \in \mathbb{A}} R_i * \prod_{a_j \in Attr(S)} \frac{\int d_{a_j} f(S_{p_j} \cap A_{p_j})}{\int d_{a_j} f(A_{p_j})}$$

In Definition 4.1, $P(x)$ is the set of publications matching x , R_i is the publication rate of A_i representing a publisher, and d_{a_j} is the distribution function for attribute a_j . Multiplying R_i by the reduction factor gives the expected publication size from A_i . The publication set size of S is the sum of publications from its matching advertisements.

For composite subscriptions, the estimation of the data set size depends on the composite subscription operators.

Definition 4.2:Data Set Size of Composite Subscription
Consider a composite subscription $CS = S_l \text{ op } S_r$, where S_l and S_r may be composite subscriptions. Then:

$$|P(CS)| = \begin{cases} |P(S_l)| + |P(S_r)| & \text{if } op = ||; \\ \min(|P(S_l)|, |P(S_r)|) * \prod_{a_i \in Var} \frac{\int d_{a_i} f(S_{p_i} \cap A_{p_i})}{\int d_{a_i} f(A_{p_i})} & \text{if } op = \&. \end{cases}$$

Based on Definition 4.2, we can estimate the data set size of an arbitrary composite subscription with a recursive algorithm based on the data set sizes of the individual atomic subscriptions and the operators in the composite subscription, which is represented by a tree. If the operator is OR ($||$), $P(CS)$ is the sum of its children; if the operator is AND ($\&$), $P(CS)$ is the minimum data set size of its children times the reduction factor. Attributes contributing to the reduction factor are variable bindings Var ³ between each child node.

4.2 Composite Event Detection Cost

We discuss the overhead incurred by moving composite event detection away from the joint point broker for two different models, using the example in Fig. 6.

Traffic-based model: In this model, we assume that each link has b bit/sec bandwidth, and d ms message delay. The parameter affecting the detection location is the publication traffic for a given composite subscription. The overhead traffic cost of evaluating composite subscription at a broker where one of the child subscriptions comes from, say Broker 1 in Fig. 6, is: $TrafficCost(S_1) = |P(S_2)| - |P(S_1)| + |P(CS)|$.

The overhead traffic cost includes the cost of sending publications for S_2 from Broker 3 to Broker 1 and the cost of sending CS 's notification set from Broker 1 to Broker 3, but we save in sending publications for S_1 from Broker 1 to Broker 3. Similarly, we can estimate the cost of evaluating CS at Broker 2. Since all links have the same bandwidth and delay, the delay cost is: $DelayCost(S_1) = \frac{TrafficCost(S_1)}{b} + d$.

QoS-based model: In this model, we estimate the overhead cost based on the publication traffic and the QoS of the overlay network. If network links have different bandwidth and delay properties, the delay cost function is: $DelayCost(S_1) = \frac{TrafficCost(S_1)}{b_{13}} + d_{13}$. The link properties are updated dynamically based on actual network traffic.

4.3 Adaptive Routing Algorithm

Based on the dynamic overhead estimation, we propose the adaptive composite subscription routing algorithm

³A variable binding is similar to an equi-join in SQL. A variable value should be the same in each atomic subscription.

Algorithm 2 Adaptive Composite Subscription Routing

Require: An incoming composite subscription cs
Ensure: Destination tree of a composite subscription

- 1: Initialized desTree according to cs
- 2: **if** cs is a leaf node **then**
- 3: desTree.root.dest = get cs 's destination set from SRT
- 4: **else**
- 5: desTree.left = adaptive-cs-routing(cs.left)
- 6: desTree.right = adaptive-cs-routing(cs.right)
- 7: **if** desTree.left.dest == desTree.right.dest **then**
- 8: desTree.root.dest = desTree.left.dest
- 9: **else**
- 10: **if** desTree.root.op == OR ($||$) **then**
- 11: desTree.root.dest = current broker
- 12: **end if**
- 13: **if** desTree.root.op == AND ($\&$) **then**
- 14: Estimate $|P(cs.left)|$ and $|P(cs.right)|$
- 15: Estimate overhead cost of detecting cs
- 16: desTree.root.dest = the dest with min overhead cost
- 17: **end if**
- 18: **end if**
- 19: **end if**
- 20: **return** desTree

shown in Algorithm 2. This recursive algorithm computes a composite subscription's destination tree, which dictates how the subscription is 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. An example is given in Fig. 7. The algorithm traverses the 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 processes the left and right children's destination trees 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 and the connecting operator is OR, the root's destination is the current broker, which means the composite subscription is split at the current broker. If the children have different destinations and the operator connecting the two children is AND, the algorithm estimates the data set sizes of the children and the overhead costs at the children's destinations, and the destination with minimal overhead is assigned to the root node. The recursive algorithm assigns destinations to the tree nodes bottom up.

We now describe the dynamic algorithm which routes a composite subscription based on the potential publication traffic and the status of the current overlay network. Fig. 7 shows a possible routing solution for composite subscription $CS = \{S_1 \& S_2\} || S_3$. At Broker 9, CS is routed as a whole to Broker 8, where the destinations of both S_1 and S_2 are Broker 4, and as a result the destina-

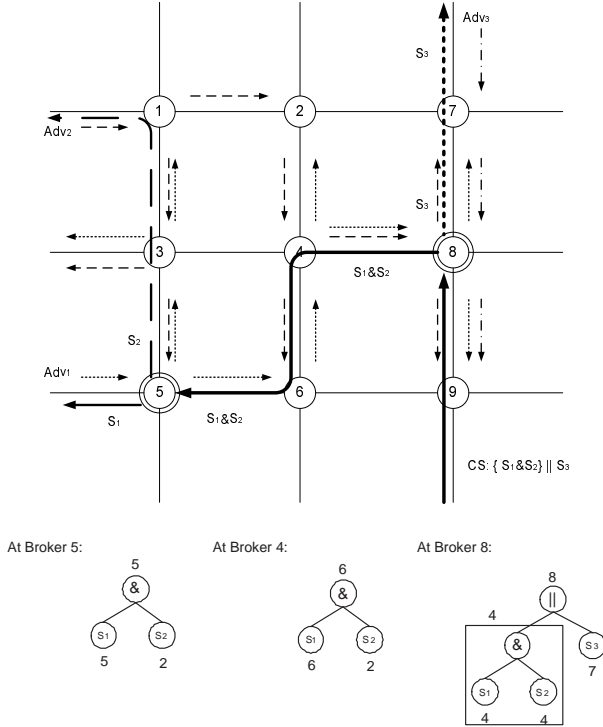


Figure 7: Adaptive composite subscription routing

tion of their parent node is Broker 4 as well. S_3 should be forwarded to Broker 7, and since the operator of the root node is OR, CS is split at Broker 8. When $S_1 \& S_2$ arrives at Broker 4, according to the SRT, Broker 4 is the joint point broker of $S_1 \& S_2$ in the topology-based routing algorithm. If the data set size of S_1 is significantly larger than that of S_2 , it may be more efficient to evaluate $S_1 \& S_2$ at Broker 6 rather than Broker 4. This dynamic evaluation happens at each broker until a broker decides to split the composite subscription, say at Broker 5. According to the adaptive routing algorithm, Brokers 8 and 5 are the joint point brokers for CS , instead of Brokers 8 and 4. After the routing, joint point brokers are ready for composite event detection.

The advantage of performing adaptive composite subscription routing is that it determines the initial composite event detection location that minimizes the network traffic and message delay costs of evaluating composite subscriptions. Moreover, dynamic publication routing for general overlays further reduces the cost of composite event detection.

4.4 Dynamic Joint Point Movement

Since network conditions, such as delay and bandwidth, may change, the joint point broker chosen during composite subscription routing may no longer be optimal,

and joint points should be computed dynamically.

The broker maintaining a joint point evaluates the cost model periodically. If the joint point broker finds a broker that is able to detect the composite event with a cost lower than itself, it initiates a joint point movement. State transfers between the original joint point and the new one include the routing path information and partial matching state. The routing path is concerned with subscriptions. Each part of the composite subscription should be routed to the proper destinations in order to keep the routing information consistent. The partial matching state concerns publications. Publications that partially match composite subscriptions are stored at the joint point broker, and this partial matching state should be delivered to the new joint point for subsequent composite event detection. Strategies to stabilize the joint point movement, such as movement frequency and cost difference thresholds are future work.

5 Evaluation

In this section, we experimentally evaluate the routing protocol in general overlay topologies. A workload generator synthesizes publish/subscribe messages used in the experiments by selecting attributes and values following a Zipf distribution from a list of twenty attribute names and given value ranges, respectively. The Zipf distribution models locality among the advertisements, subscriptions and publications.

We explore the properties of the routing protocols by deploying a broker overlay in a controlled 15-node local network consisting of nodes with 4GB of memory and 3.0GHz CPUs. We evaluate the protocols in two overlays, each consisting of 30 brokers scattered across the network. One overlay is well connected with an average connection degree (D) of 4, while the other is less connected with $D = 2$. Each node in the topologies is a complete content-based publish/subscribe broker that implements the protocols described in this paper. More details on the broker design and implementation are available in [15]. In our experiments we compare the end to end notification delay of fixed (Section 3.1) and dynamic (Section 3.2) publication routing algorithms and measure the CPU and memory usage per broker. For adaptive composite subscription routing, we observe the placement of the joint point broker by measuring network traffic. To demonstrate robustness and scalability, we also deploy a broker network on PlanetLab.

End to end notification delay When a publication matches a subscription, a notification is delivered to the subscriber. The notification delay is computed as the time between publishing at the publisher to the corresponding notification at the subscriber. This delay is a function of the number of hops from the publisher to sub-

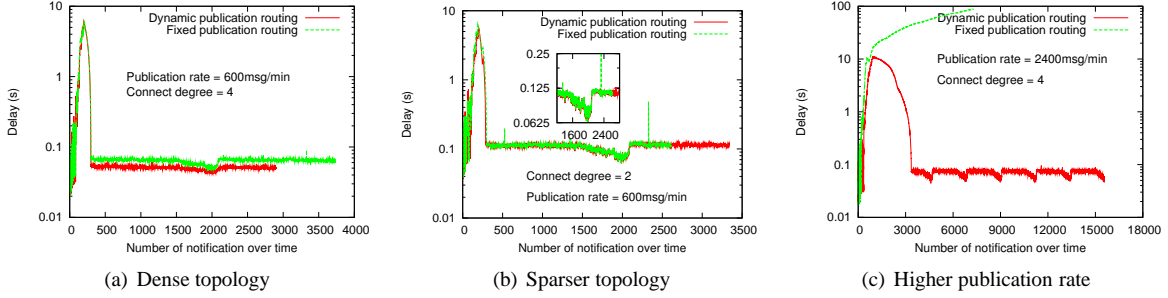


Figure 8: End to end notification delay

scriber, 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 queuing delay, processing and matching time, and transmission delay at each hop. We measure the end to end delay in three scenarios.

In the first scenario, we construct a 30 broker network, and connect 20 publishers and 30 subscribers to the system within the first 30 seconds of the experiment. One of the publishers publishes, and four of the subscribers issue subscriptions that match some publications. The subscribers are placed such that the longest path in the cyclic network from the publisher to them is 12, 10, 8, and 6 hops. We vary the publication rate of the publisher and measure the notification delay at the subscribers' side. Each data point in the following figures is the end to end delay of one notification received by the subscriber that is 12 hops from the publisher in the overlay.

When the publishers and subscribers join the system, advertisements are broadcast and subscriptions are matched with advertisements and forwarded to potential publishers. This initial traffic contributes to network congestion and large notification delays, and is visible in Fig. 8(a) where the end to end delay is plotted (in log scale) for each notification. After the initialization phase, the delays of both dynamic and fixed routing algorithms stabilize, with the dynamic algorithm producing a 20.3% shorter average delay than fixed routing. This is because the dynamic routing algorithm can explore alternate paths at each hop, and balance publication forwarding across multiple neighbors.

In a less connected network, the benefit of dynamic routing is diminished due to a lack of alternate paths. In Fig. 8(b), with $D = 2$, the dynamic routing delay is almost identical to that of fixed routing. We enlarge part of the plot in Fig. 8(b) to illustrate the slight average improvement of 4.65%. 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 Figs. 8(a) and 8(b) demonstrate that the dynamic approach benefits from a well connected overlay,

the benefit is not proportional to the connection degree. When the same experiment is repeated with a fully connected topology, the delay with the dynamic approach is 4.07% worse than with fixed approach. In this case, the overhead of maintaining link information in the dynamic algorithm can not be offset by 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 a worse notification delay. For instance, in Fig. 8(c) when the publication rate is increased to 2400 msg/min, the fixed algorithm becomes overloaded with messages queuing 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 Sec 3.1, overlapping advertisements increase the number of potential alternate paths explored. In Fig. 9(a), we observe 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. 9.) This compares with a relative benefit of 20.3% in Fig. 8(a) where there were fewer publishers and hence fewer alternate paths explored.

In the third scenario, we evaluate a dynamic workload. A new publisher now joins the system 5 minutes after the other clients and publishes a burst of 1400 msg/min for 1

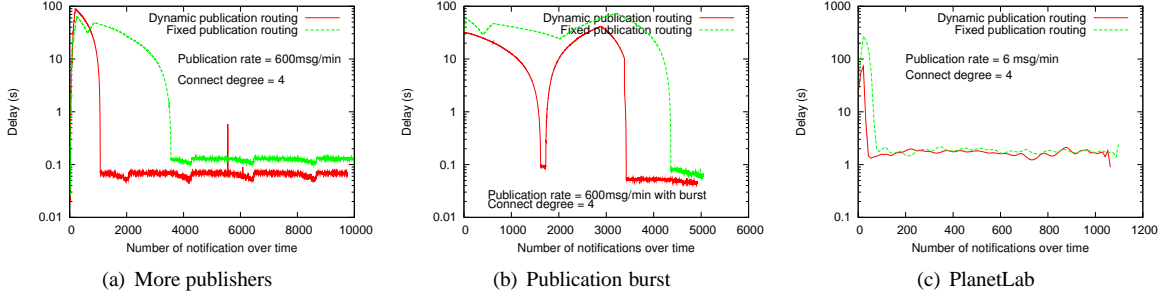


Figure 9: End to end notification delay

minute. As illustrated in Fig. 9(b), 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 1, 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 1, with a 6 hop path length, there is even a 0.78% performance degradation resulting from the overhead of the dynamic algorithm. When the distance increases to 12 hops, the improvement improves to 18.57%. We also observe, reading down the columns in Table 1, that the delay between the 6 and 12 hop subscribers with the fixed approach is 57.65%, while the corresponding difference with the dynamic algorithm is only 27.39%. 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	47.202	47.568	-0.78%
10 Hops	64.477	52.895	17.96%
12 Hops	74.416	60.598	18.57%
Max Diff	57.65%	27.39%	

Table 1: Effect of subscriber distance on delay

From the above results, we make the important observation that our publish/subscribe routing algorithms not only support cyclic overlays, but *benefit* from them by reducing the notification delay and increasing resilience to network loads. To demonstrate the robustness and the scalability of our approach, we repeat our experiments on PlanetLab with a 50 broker overlay network. While the heterogeneous and shared nature of the PlanetLab nodes and network make it difficult to derive repeatable and reliable results, our evaluations on PlanetLab support the

conclusions made from the controlled environment experiments. For example, Fig. 9(c) 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. Once the TID attribute is bound (see Section 3.1), 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. While we have not yet implemented this optimization, a quick calculation demonstrates the potential. For the dynamic algorithm in the experiment associated with Fig. 8(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.

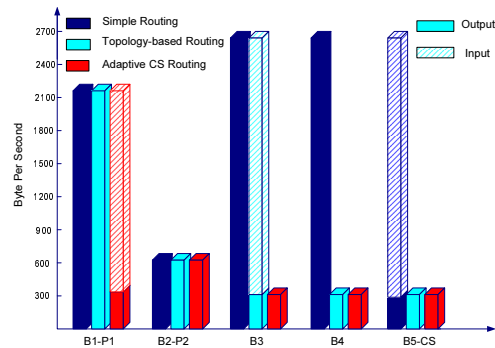


Figure 11: Composite subscription routing

Overhead of Dynamic Publication Routing In this experiment, we measure the CPU and memory usage per broker at the servers. Fig. 10 shows the average CPU and memory usage of all 30 brokers over time with a publication rate of 600 msg/min. In the dynamic approach, we need to periodically (10 ms in this experiment) maintain

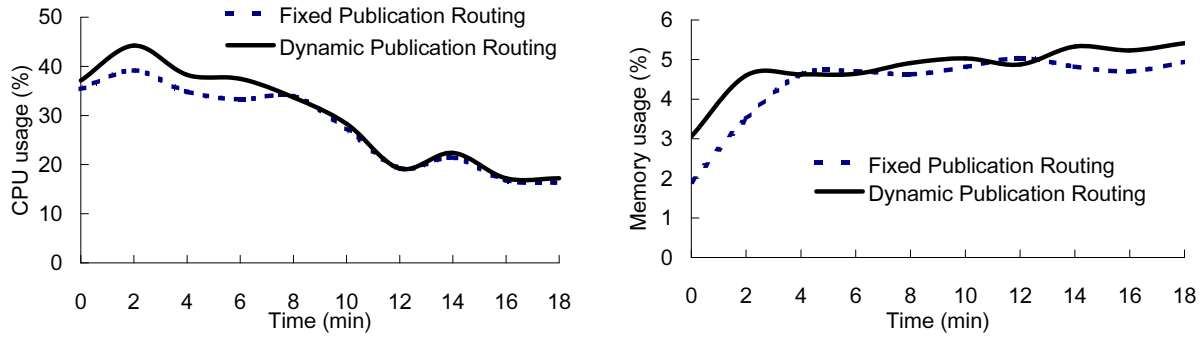


Figure 10: CPU and memory usage

the *link delay table* at each broker. When a publication arrives, the broker selects a “better” path based on the updated *link delay table*. When the broker overlay is setup, system control messages, including broker initialization and overlay routing information, are routed through the system. After the system has stabilized, publishers and subscribers then issue their messages. During the initialization phase, the busiest broker, which has the most neighbors, consumes up to 83.7% of the CPU processing capacity; while the other brokers use only 16.6% of the CPU. During the publishing phase, the broker that the publisher connects to consumes 44.3% of the CPU and 10.2% of the memory. The CPU and memory usage may increase when more subscribers and publishers are connected to the overlay. The results show that brokers consume similar CPU and memory usage in both approaches. That is, applying the dynamic approach, we can benefit from the reduction of notification delay and the resilience to publication workloads without consuming much more system resources.

Bandwidth of Adaptive CS Routing This experiment consists of four publishers with different publication rates: two at a rate of 100 msg/s, and the other two at a much slower 2 msg/s rate. A subscriber also connects to the network and issues a composite subscription. We focus on the bandwidth of five particular brokers (Brokers B1 to B5) located on the composite subscription routing path in the network.

The composite subscription is issued at Broker B5, and is routed to its potential publishers according to three different strategies: simple routing, topology-based routing and adaptive composite subscription routing. 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, composite event detection occurs at Broker B5. In topology-based routing, a composite subscription is forwarded through the broker network as a whole until it reaches a *joint point broker* where its atomic subscriptions are forwarded in different directions in the topology. Broker B3 is the joint point broker in this scenario. The

adaptive routing algorithm determines the detection location based on the potential publication traffic. In our topology, the faster publisher connects to Broker B1, and hence is an efficient point to detect the composite subscription. The slower publisher connects to Broker B2.

The input and output bandwidth consumptions of the brokers with the three different strategies are presented in Fig. 11. With simple routing, B5 evaluates the composite subscription and filters out unwanted publication traffic. Therefore, there is a great reduction from the input traffic to the output at B5, as shown in Fig. 11. Since the filtering happens at the last broker along the paths from publishers to the subscriber, the overall network traffic is high. The topology-based algorithm improves the broker traffic compared to the simple routing by moving the joint point into the network, while the adaptive composite subscription routing algorithm dramatically reduces the bandwidth by moving the joint point close to the publishers according to our cost model, with Brokers B1 and B3 experiencing 85.7% less traffic than in the simple routing case. Furthermore, all brokers after the joint point broker along the routing path experience traffic reduction. We expect these benefits to be even more pronounced in larger networks since longer composite subscription paths in such topologies offer the potential for more savings.

6 Conclusions

Current publish/subscribe systems [12, 8, 17] 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. Our approach retains the original publish/subscribe interface and matching algorithms so it may be easily integrated in existing publish/subscribe systems. Our approach minimizes redundant traffic induced by cycles in the overlay and reduces message routing delay. Content-based routing in a gen-

eral overlay improves the scalability and robustness of publish/subscribe systems by offering routing path alternatives. We exploit this flexibility to devise an adaptive composite event detection algorithm that reduces network traffic by up to 85% in one experiment.

Experiments in a local 30 broker topology indicate notifications can be delivered about 20% faster in a well connected network. Furthermore, we observe that the relative benefit of the dynamic algorithm over the fixed one actually increases when the system is stressed in various ways including increases in the connectivity of the network, overall publication rate, number of publishers, or path length between publishers and subscribers. However, the overhead of the dynamic algorithm may not be offset by gains in extremes such as short paths or very high connectivity. The use of TIDs to discriminate among duplicate advertisements affords a very useful mechanism to significantly improve content-based routing by requiring messages to only be matched once when the TID attribute is bound to value. In one scenario we estimate that almost 90% of matching operations can be eliminated with this technique. Also, preliminary results in a wide area PlanetLab environment with up to 50 nodes confirm many of the conclusions derived from the local experiments, and suggest that the unreliable nature of PlanetLab network benefits greatly from robust algorithms that can route around congested or failed nodes.

The work is motivated by applications such as network monitoring and management [2, 11], complex event processing [14] to support applications such as algorithmic trading, and distributed workflow management and business process execution [13, 15]. These applications require a flexible, decoupled and robust messaging substrate. Our routing protocols to handle cyclic overlays and novel adaptive composite subscription routing protocol make the publish/subscribe system more flexible to overlay topology changes and more robust to broker failures and recovery. Moreover, applications may have quality of service constraints on activities, such as execution time for a service or for an entire business process, which may affect message routing at the publish/subscribe layer. With our contributions, publications may select “optimal” paths to matching subscribers and composite subscriptions are routed to the “best” event detection locations in order to satisfy potential quality of service constraints at the application level. Future work will investigate the feasibility of supporting some of the above applications with quality of service guarantees using the content-based publish/subscribe paradigm.

References

[1] D. G. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *ACM SOSP*, October 2001.

[2] K. L. B. Mukherjee, L. T. Heberlein. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[3] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[4] K. P. Briman. A review of experiences with reliable multicast. *Softw. Pract. Exper.*, 29(9):741–774, 1999.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM ToCS*, 19(3):332–383, 2001.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct 2002.

[7] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par*, volume 3648, pages 1194–1204. Springer, 2005.

[8] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE ToSE*, 27(9), 2001.

[9] G. Cugola, G. Picco, and A. Murphy. Towards dynamic reconfiguration of distributed publish-subscribe middleware. In *Intl. SEM Workshop*, 2002.

[10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[11] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *ACM SIGKDD*, pages 53–62, San Diego, CA, 1999.

[12] L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. Engineering event-based systems with scopes. In *ECOOP*, pages 309–333, London, UK, 2002. Springer-Verlag.

[13] IBM and Microsoft. Business process execution language for web services version 1.0.

[14] I. Koenig. Event processing as a core capability of your content distribution fabric. In *Gartner Event Processing Summit, Orlando, Florida*, 2007.

[15] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *ACM/IFIP/USENIX Middleware*, Grenoble, France, 2005.

[16] A. Nayate, M. Dahlin, and A. Iyengar. Transparent information dissemination. In *ACM/IFIP/USENIX Middleware*, Toronto, Canada, October 2004.

[17] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting IP multicast in content-based publish-subscribe systems. In *ACM International Conference on Distributed systems platforms*, 2000.

[18] K. Ostrowski and K. Birman. Extensible web services architecture for notification in large-scale systems. In *IEEE ICWS*, pages 383–392, Washington, DC, USA, 2006. IEEE Computer Society.

[19] P. R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks*, January/February 2004.

[20] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *NSDI*, April 2007.

[21] C. Schuler, H. Schuldt, and H.-J. Schek. Supporting reliable transactional business processes by publish/subscribe techniques. In *TES*, pages 118–131, 2001.