A Peer-to-Peer Approach to Content-Based Publish/Subscribe

Wesley W. Terpstra Stefan Behnel Ludger Fiege
Andreas Zeidler Alejandro P. Buchmann
Department of Computer Science
Darmstadt University of Technology
D-64283 Darmstadt, Germany
terpstra@ito.tu-darmstadt.de, {behnel,fiege}@gkec.tu-darmstadt.de,
{az,buchmann}@informatik.tu-darmstadt.de

ABSTRACT

Publish/subscribe systems are successfully used to decouple distributed applications. However, their efficiency is closely tied to the topology of the underlying network, the design of which has been neglected. Peer-to-peer network topologies can offer inherently bounded delivery depth, load sharing, and self-organisation. In this paper, we present a content-based publish/subscribe system routed over a peer-to-peer topology graph. The implications of combining these approaches are explored and a particular implementation using elements from Rebeca and Chord is proven correct.

Keywords

Publish/subscribe, content-based routing, peer-to-peer networks, graph topology

1. INTRODUCTION

Publish/subscribe is a scalable and flexible communication paradigm which suits the needs of modern applications. A publish/subscribe service conveys published notifications from any producer to all interested consumers with a matching subscription set. In this manner clients do not use source/destination identifiers or addresses. This inherent *loose coupling* of producers and consumers is the primary advantage of these systems.

To achieve this loose coupling, consumers subscribe to specific kinds of event notifications. The most flexible selection criteria for notifications is realized by content-based selection. In this particular publish/subscribe model, notification messages are filtered according to their content. Event notifications propagate from a producer to interested consumers through a network of filters.

However, state of the art content-based publish/subscribe systems have static networks. While subscriptions change dynamically with the interests of the current clients, the routing network used to publish the notifications remains rather unchanged. Furthermore, the network is often chosen to be a tree in order to simplify the routing algorithms. This approach introduces single points of failure and bottlenecks. However, since the network topology significantly influences the performance of the overall system, it should be carefully selected to reduce network congestion, minimise routing depth, and preserve these properties in face of changing network nodes and failures. These properties are missing in current systems.

On the other hand, peer-to-peer (P2P) networks often address precisely these issues. They self-coordinate a very large network to achieve a common goal. The assumption made for the design of peer-to-peer networks is frequent node failure and changing participation. The networks typically have excellent routing depth guarantees. Also, as most peers are equal, traffic is often evenly distributed, reducing congestion. All peer-to-peer systems maintain their guarantees under the assumption of frequent failures.

Modern peer-to-peer networks have been used to great effect in implementing a form of multicast [17]. Topic-based publish/subscribe—a primitive addressing model—can be implemented on peer-to-peer multicast. The routing decisions of peer-to-peer networks are generally simplistic. Our publish/subscribe routing policy allows for decisions with selection criteria as flexible as content-based publish/subscribe.

To provide more reliable and scalable topologies for content-based filtering, more general graphs than a tree must be maintained. Our contribution in this paper is to take the graph topology and management of a peer-to-peer network and couple it with the highly flexible routing of a publish/subscribe system. Of particular interest, our network preserves the use of fully general filters, yet guarantees a logarithmic bound on delivery depth with evenly distributed congestion, even in the face of dynamic participation and failures.

The remainder of this paper is structured as follows. In section 2 we briefly outline Chord [18], a P2P overlay routing scheme, and REBECA [11], a content-based publish/subscribe system, both of which we borrow ideas from. After section 3

motivates the design of our system, we outline its formal properties. Section 4 proceeds to present and prove correct the publish and subscribe algorithms. We then discuss throughout section 5 how to preserve the required network and filter structure in the face of node and edge failures and joins. Finally, in section 6 and 7, we compare our system to existing work and outline the direction of our further research.

2. COMMUNICATION PARADIGMS

The communication architecture envisioned in this paper integrates content-based filtering strategies directly on top of a P2P network topology. To manage the filters, we reuse REBECA's algorithms for filter covering and merging [8, 10, 11]. The topology of our network is directly borrowed from the Chord P2P network [18]. However, the generalised routing algorithm used to implement the publish/subscribe interface is the focus of this paper. The maintenance of the graph is implemented using this generalised routing algorithm rather than Chord's native algorithm which we omit.

2.1 Rebeca

Processes in pub/sub systems (also known as *event-based systems* [8]) are clients of an underlying notification service and can act both as producers and consumers of messages, called event notifications or notifications for short. A *notification* is a message that describes an event. Notifications are injected into the event system via a `publish()` call rather than being published towards a specific receiver. They are conveyed by the underlying notification service to those consumers which have registered a matching subscription with `subscribe()`. Subscriptions describe the kind of notifications consumers are interested in.

The major characteristic of a notification service is the data model of the transmitted notifications and the language used for subscribing. Subject- and type-based addressing exists [12, 2, 7], but content-based filtering [3] offers the most flexible scheme. Filters are boolean functions on the entire content of a notification.

The notification service relies on a network of brokers, which forward notifications according to filter-based routing tables. The topology of the system is often constrained to be an acyclic and connected graph (Fig. 1) for simplicity reasons. The edges are point-to-point connections, forming an overlay network, e.g., in an underlying TCP network. This model simplifies the implementation and reasoning about communication characteristics. The single network used for data dissemination is comparable to the single spanning tree approach of multicast algorithms [5]. The single tree, however, is a bottleneck of the system, as each node is a single point of failure and the central nodes are likely to carry the major part of the load.

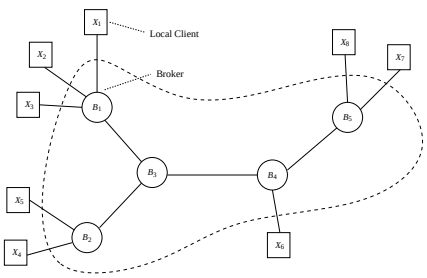

Figure 1: The router network of REBECA.

The major advantage of these systems is that the routing tables can direct the flow of notifications to only interested nodes. Each broker maintains a routing table which includes content-based filters. When routing, a notification only goes down a link if it is matched by a corresponding filter.

The simplest form of routing is *simple routing*: active filters are simply added to the routing tables with the link they originated from. Obviously, this is not optimal with respect to routing table sizes, which grow linearly with the number of subscriptions.

A first improvement is to check and combine filters that are equal. More generally, the *covering* routing strategy [4] tests whether a filter F_1 accepts a superset of notifications of a second filter F_2, and in this case replaces all occurrences of F_2 assigned to the same link in the routing table, significantly decreasing the table size.

In a second step, if no cover can be found in a given set of filters, *merging* can be used to create new filters that cover existing ones [10]. Only the resulting merged filter is forwarded to neighbour brokers, where it covers and replaces the base filters. Merging can be done either in a perfect or imperfect way. Perfectly merged filters only accept notifications that are accepted by at least one of its base filters, whereas imperfectly merged filters accept notifications beyond their base filters.

Imperfect routing table entries increase network traffic since notifications are accepted and forwarded to neighbour brokers only to be discarded by them. On the other hand, if a filter table already accepts a notification, then it does not need to be updated by a new, but covered subscription later.

2.2 Chord

Chord [18] is simple and sound peer-to-peer network. Its graph's topology provides several features which help us build an efficient content-filtered publish/subscribe filtering system with respect to delivery depth and load balancing.

Chord is a scalable system for node lookup in a dynamic peer-to-peer system with frequent node arrivals and departures. In this way it is similar to other Distributed Hash Tables (DHTs [14], [16]).

Chord itself supports just one operation: given a key as part of a large, circular key space $(0 \dots 2^{160} - 1)$, it maps the key onto a node. It achieves this in a scalable manner by limiting the routing information each node needs to only a few other nodes. Because the routing table is distributed, a Chord node communicates with its neighbours in order to perform a lookup.

In the steady state with an N-node system, each node maintains information about only $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. Chord maintains its routing information as nodes join and leave the system.

The Chord network topology is a more general form of figure 2. The basic principle is that each node keeps directed edges to nodes with distances $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \dots$ clockwise on the circle. Routing is achieved by successively refining the search via these edges. We borrow this concept of refinement from Chord and apply it to the more general concept of filter coverage.

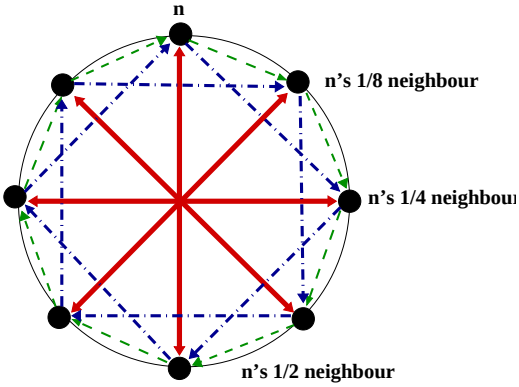

Figure 2: The Chord graph

3. DESIGN

In this section we outline what changes are required from the publish/subscribe system and what constraints we have on our network topology.

3.1 Extending the Publish/Subscribe

One of our goals in combining P2P with publish/subscribe was to remove the single bottleneck and point-of-failure of using exactly one tree for notifications and filter updates. To avoid introducing routing cycles within a more general, redundant graph, we must select for each notification a spanning subtree of the entire graph. During routing, we must provide a test to assure forwarding only along those edges which are in the subtree. To provide the routing algorithm with an understanding of how to select the edges for a subtree, we provide a topology component.

Furthermore, we want the network to be robust when brokers change and fail. To keep the graph correct, we must maintain the system's routing tables to reflect the changes. Therefore, we add two components to our architecture that maintain the structure of the graph and the filters. The final change is semantic in nature: we have relaxed our assumption about broker failures—they can happen—so we no longer guarantee delivery of all published notifications which match a subscription.

Despite these changes, our implementation preserves the APIs used to implement/invoke filters and the API used to publish or subscribe for a notification. The two new maintenance components simply use the existing publish/subscribe API for communication and access the routing table. This design is illustrated in figure 3.

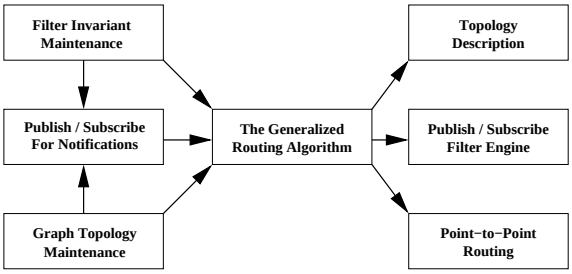

Figure 3: The components of our architecture

3.2 Topology of Many Trees

We know that we must generalise the graph. However, to keep the system functioning requires that the graph have structure. It is important that this structure be well understood so that we can implement it.

The nodes within our topology graph are special broker machines which publish/subscribe on behalf of themselves and possibly light-weight or poorly connected client machines. They maintain connections to several peer broker nodes in order to route notifications and subscriptions.

Each directed edge in our topology graph is a network link joining two brokers. The direction of the edge indicates in which direction published notifications flow. It is possible for two nodes to be connected by two edges, one in either direction. On each edge we associate exactly one content-based filter which determines if an event notification should be forwarded to the associated peer broker.

As previously noted, a more general graph than a tree is required. However, there is no purpose in sending a given notification or filter update to the same broker many times. In fact, we want to preserve the guarantee that a single published notification matches a given subscription at most once. Therefore, it makes sense to consider a spanning subtree of the graph for a given publish or subscribe operation. However, to balance the network congestion and reduce single points of failure, we use a different tree for every broker. That is, each broker is at the root of its own distinct tree to use for delivering a published notification.

The addition of multiple trees for notification presents some difficulties. Prior to our generalised routing, a subscription and notification followed the same tree. This will not lead to correct behaviour in our more general topology. If a notification is distributed through one tree, a subscription from any of the nodes in that tree must propagate up that tree; this remains unchanged. However, a subscribe must simultaneously propagate up *all* possible publish trees.

Let the path $u \rightsquigarrow v$ denote the route taken in the tree rooted at u to publish a notification to v. Then the subscription tree rooted at v must follow $u \rightsquigarrow v$ in reverse to reach u. That is, the same path must be taken. If this is not the case, then the path will not have filters which are supersets of each other, thus leading to a notification being dropped early. This will be discussed in more detail throughout the following section.

Finally, the distribution of notifications and filter updates in our network is essentially a broadcast algorithm. It is similar to an existing naive scheme [6] implemented on Chord. However, our broadcast notifications are heavily attenuated by the filters on our edges, while the filter updates are attenuated by filter covering.

4. IMPLEMENTATION

In order to make our proofs and explanation precise, we make use of some specific notation. All of the notation is collected here for quick reference and then explained in detail below.

4.1 Notation

We will refer to the topology graph as $G = (G_V, G_E)$ for the sets of vertices and edges respectively. K will denote the key space. E represents the set of all publishable event notifications. $N = |G_V|$ is the number of broker nodes in the graph.

- $(u, v) \in G_E$ denotes the directed edge from $u \in G_V$ to $v \in G_V$. In text, $(u \to v)$ is used to denote (u, v) for readability. $(u \rightsquigarrow v)$ denotes a path from u to v.
- $p(u \in G_V) = \{v \in G_V | (u, v) \in G_E\} \cup \{u\}$ denotes the set of peers a node u may forward a notification to. The loopback edge, $u \to u$, is used by u to express its (and its clients') subscriptions.
- $s(v \in G_V) = \{u \in G_V | (u, v) \in G_E\} \cup \{v\}$ denotes the set of peers a node v may receive a notification from. The loopback edge, $v \to v$, is used by v to publish notifications on its own (and its clients') behalf.
- $p_{\frac{1}{2^k}}(u \in G_V)$ denotes a peer $w \in p(u)$ such that w is at least $\frac{1}{2^k}$ clockwise around the circle from u.
- $key(u \in G_V) \mapsto k \in K$ is the function which maps a specific node u to its key value k.
- $r_{u \in G_V}(v \in G_V) \mapsto R \subseteq K$ returns the subset R of the key space K which node u holds v responsible for.
- $f((u \to v) \in G_E) \mapsto A \subseteq E$ is the function mapping an edge to the set A of event notifications which the corresponding filter accepts.

Due to the Chord graph in figure 2, we have $O(\log N)$ edges entering a node, and $O(\log N)$ edges leaving a node. We call those edges leaving a node u the publishing edges and denote the set of target nodes with $p(u)$. Those edges entering a node u, we call its subscription edges with $s(u)$ for the nodes respectively. This terminology is chosen to reflect the operations taken by u on its edges. Whenever edges are indicated such as $v \to w$, the edge is always interpreted with direction relative to publishing.

A node u's publishing edges are selected to have a doubling property taken from Chord. For $k = 1 \ldots 160$, u keeps an edge to the first node which is clockwise by a distance of $\frac{1}{2^k}$ or more along the circle. We denote these particular nodes as $p_{\frac{1}{2^k}}(u)$. Only one edge is kept to each peer node. There is a theorem related to Chord that the degree of u is $O(\log N)$ rather than 160 [18].

As in Chord, there is a key space K mapped onto the circle. Each node u has a location on the circle and therefore a key $key(u)$. During publishing, we must select a subtree. To achieve this, a node u takes all of the nodes in $p(u)$ and draws them in their appropriate locations on the circle as illustrated in figure 4. Now, u assigns responsibility $r_u(v)$ to each $v \in p(u)$ by taking the section of the circle clockwise of node v up to the next node. In this manner, the entire key space is partitioned.

Finally, each edge has a filter $f(u \to v)$. We say that the filter accepts an event notification $e \in E$ if $e \in f(u \to v)$. Otherwise, it is rejected. A filter f covers another filter g if $f \supseteq g$. Merging two filters is creating an h such that $h \supseteq f$ and $h \supseteq g$.

4.2 The Invariant

In order to make globally correct filter decisions locally at each node, we need guarantees about the overall structure of the filters. We present an invariant that ensures published notifications follow a path on which the filters are always subsets. In this manner, no early filter will reject a notification which would have been accepted later.

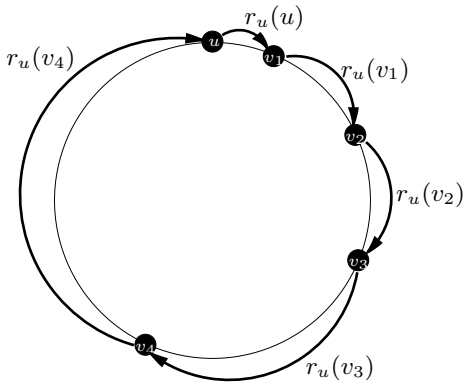

Figure 4: The partition of responsibility for u

Following our notation in section 4.1 the invariant can be expressed for $u \to v \to w$ as follows:

$$\forall u \in G_V \ \forall v \in p(u) \ \forall w \in p(v) : \\ r_u(v) \cap r_v(w) \neq \emptyset \Rightarrow f(u \to v) \supseteq f(v \to w)$$

The terms in this equation are illustrated for the Chord graph in figure 5.

4.3 Publishing Notifications

We make the guarantee that no node ever receives the same notification twice. Let u be a publisher of notification e and v a subscriber for event notifications F with $e \in F \subseteq E$. If no edges on the delivery path $u \rightsquigarrow v$ fail during delivery of e, we guarantee that v receives e.

Messages which carry an event notification include a range field, $r \in K$. This range field denotes that the receiving node u is responsible for delivering the notification to all interested parties in the range $[key(u), r)$ (where the key space is considered to wrap-around).

The publishing node w starts the algorithm by setting the initial range to the entire key space, and then imagining that the notification was received via the loopback edge $w \to w$. When a node u receives a notification, it intersects the range $[key(u), r)$ with the set $r_u(v) \subseteq K$ for each peer $v \in p(u)$. If this intersection is non-empty, the filter $f(u \to v)$ is tested. If the filter accepts the published notification, then it is forwarded down the edge $u \to v$ with a new range r' such that $[v, r') = r_u(v) \cap [u, r)$.

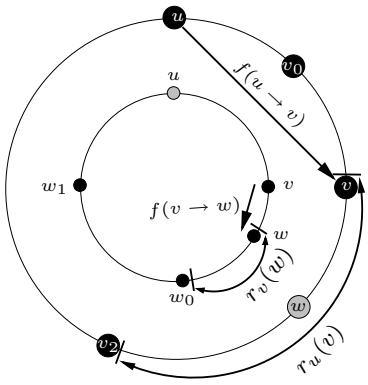

Figure 5: The terms of the invariant

The order in which edges are considered is in decreasing size of $|r_u(v)|$. In the case where the network contains $N = 2^k$ nodes, the published notifications follow the paths indicated in figure 6. Pulling this routing out of the circle, we see the binomial tree of figure 7. This is the reason underlying our guarantee of at most $O(\log N)$ deliver steps. We inherited this directly from Chord.

The correctness argument is structured as an inductive proof on the size of $|R_u|$. The inductive hypothesis is:

A node u receives a notification e for range $R_u = [key(u), r)$. Let $S_u \subseteq R_u$ be those nodes which have a subscription for e. u guarantees to deliver e to all $x \in S_u$, at most once to all $y \in R_u$, and never to $z \notin R_u$.

Assuming the algorithm works for $|R_u| < P$, we will prove that it works for of $|R_u| = P$. Then we will argue that the algorithm terminates, completing the proof.

For all $v \in p(u)$, let $R'_v = r_u(v) \cap R_u$, and $S'_v = r_u(v) \cap S_u$. Recall that $K = \bigcup_{v \in p(u)} r_u(v)$ is a partition. Therefore, $R_u = \bigcup_{v \in p(u)} R'_v$ and $S_u = \bigcup_{v \in p(u)} S'_v$ are partitions of R_u and S_u respectively.

Because u has a peer $p_{\frac{1}{2^{160}}}(u)$, u is the only node such that $key(u) \in r_u(u)$. Therefore, u simply delivers e to each of the subscribed clients exactly once and no one else.

Now, u decides to forward e to $v \neq u$ with range R'_v iff $R'_v = r_u(v) \cap R_u \neq \emptyset$ and $e \in f(u \to v)$. v would only need to receive e if there is a $w \in p(v)$ such that $r_v(w) \cap R'_v \neq \emptyset$ and $e \in f(v \to w)$. Suppose that u does not forward e, but v needed e. This means $e \notin f(u \to v)$. Pick $k \in r_v(w) \cap R'_v$. Then, $k \in r_v(w)$ and $k \in R'_v \subseteq r_u(v)$. Hence $r_u(v) \cap r_v(w) \neq \emptyset$. By our invariant, then $f(u \to v) \supseteq f(v \to w)$. Since $e \in f(v \to w)$ then $e \in f(u \to v)$. This is a contradiction.

Hence, if v needs e for any nodes S'_v, v receives e. Furthermore, since $|R'_v| < |R_u|$, the inductive hypothesis holds.

As each element R'_v of the partition of R_u met the guarantee of no delivery to $z \notin R'_v$ and at most once to $y \in R'_v$, R_u meets these guarantees by unioning the disjoint R'_v sets. Furthermore, as each R'_v delivered to each $x \in S'_v \subseteq R'_v$, then u delivered to each S_v as it is the union. Therefore, u met all of the guarantees for a problem of size $|R_u|$.

Since we know that every node u sees e at most once and u reduces the number of nodes remaining by one, the algorithm must terminate. Therefore, our algorithm makes the claimed guarantees. Less formally, the delivery depth is related to the binomial tree and is thus $O(\log N)$.

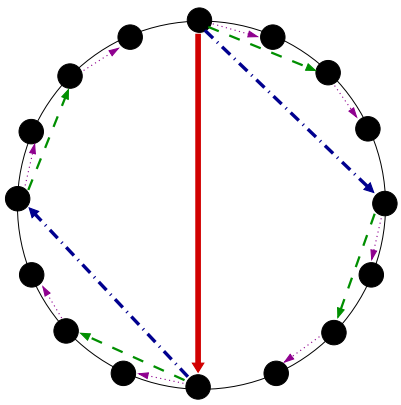

Figure 6: Notification propagation

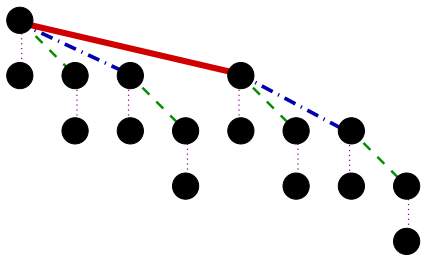

Figure 7: Binomial tree for publish/subscribe

4.4 Subscribing/Unsubscribing

A subscription simply updates the loopback filter on the originating node and then reestablishes the invariant.

Where the publish operation was delivered clockwise on the Chord ring, the subscribe operation is delivered counterclockwise. Where the publish followed decreasing responsibility order, the subscribe follows increasing responsibility order. Figure 8 illustrates this for the case $N = 2^k$.

Messages which update the invariant include only the state of the new filter to apply on that edge. The goal of the subscribe algorithm is to reach every node u along the same path that a publish would follow to reach the subscriber from u.

We require that every $v \in G_V$ must also know what $r_u(v)$ is for every $u \in s(v)$. That is, it must know what keys each of the publishers pointing at it hold it accountable for.

When a node v receives a filter update from a peer $w \in p(v)$ with filter g, it checks if the filter $f(v \to w)$ already covers g. If f does cover g, v may choose not to update its filter. If f does not cover g, v must set $f(v \to w) \supseteq g$. If the filter did not change, v stops processing.

Next, v considers all of its subscription edges to peers $u \in s(v)$. If $r_u(v) \cap r_v(w) \neq \emptyset$, then v recalculates an appropriate filter h defined below which it propagates to u.

$$h = \bigcup_{\substack{x \in p(v): \\ r_u(v) \cap r_v(x) \neq \emptyset}} f(v \to x)$$

For unsubscribe, the exact same procedure is followed if the loopback filter was changed.

This algorithm directly maintains the invariant; we only need to prove that it terminates. We show that the number

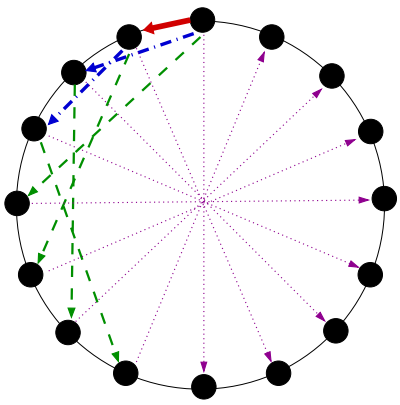

Figure 8: Filter update propagation

of the edge, k, decreases and therefore the corresponding distance spanned in the key space $\frac{|K|}{2^k}$ must increase. What this tells us is that eventually the algorithm must terminate with $k = 1$, the edge which crosses half the circle. In fact, the number of steps is again, not 160, but $O(\log N)$ due to the binomial tree. The following sketch of the proof uses a node x to separate $r_u(v)$ and $r_v(w)$ as figure 9 illustrates.

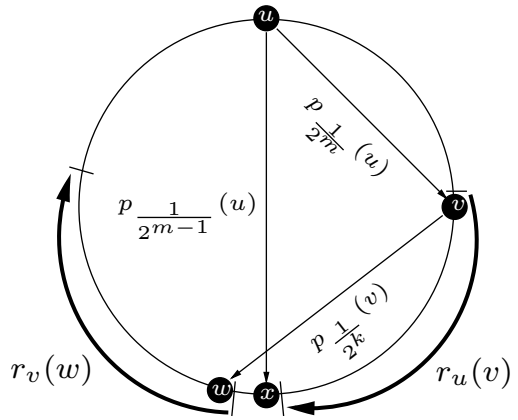

Figure 9: Visual termination proof

Recall that v received a filter update from $w = p_{\frac{1}{2^k}}(v)$. Consider the case where v needs to update $u \in s(v)$ such that $v = p_{\frac{1}{2^m}}(u)$. For a contradiction, set $m \geq k$. u chose v to be the first node clockwise of it by a distance of at least $\frac{|K|}{2^m}$. u also chose a $x = p_{\frac{1}{2^{m-1}}}(u)$ which has a distance of at least $\frac{|K|}{2^{m-1}}$. w is similarily at least $\frac{|K|}{2^k}$ clockwise from v. Therefore, w has clockwise distance from u of at least

$$\frac{|K|}{2^m} + \frac{|K|}{2^k} \geq \frac{|K|}{2^m} + \frac{|K|}{2^m} \geq \frac{|K|}{2^{m-1}}$$

So, w must be at least as far clockwise from u as x. Recall the range $r_u(v)$ is counter-clockwise of $key(x)$ due to the construction in section 4.1. Since w is clockwise of x, the range $r_v(w)$ must be clockwise of x. Hence $r_u(v) \cap r_v(w) = \emptyset$. This contradicts the assumption that u needed to be updated by v.

5. DEALING WITH FAILURES

Fault tolerance and recovery are very important issues in a distributed environment such as a peer-to-peer system. Node and edge failures are commonplace. Since our system relies on correct structure, we have to provide a maintenance algorithm that allows us to recover from edge and node failures. The high redundancy in our system allows us to handle these in an efficient way.

In the face of edge failures on the delivery path, we guarantee only best-effort delivery semantics. However, the reader may have noticed that our redundant topology could use retransmission to guarantee *at least once* (but not exactly once) delivery to all subscribed nodes in case of any simultaneous $\log N - 1$ edge failures. A significant amount of history and an increased message overhead is still required to give a delivery guarantee in the face of a network split. Therefore, we chose to omit this form of failure recovery to allow applications to make the trade-off themselves.

5.1 Keeping the Graph Chord Shaped

There are two events which can temporarily break the Chord graph. A new node may join and its publish and subscribe edges will not be up-to date. Worse, edges may fail. This case is more critical since if a neighbour goes missing, the entire circle may be cut.

Upon a new node v joining, we must update $s(v)$ so that v receives published notifications. We define a special node-join notification which is published when v connects and includes $k = key(v)$. All brokers u subscribe for a node-join notification in $[key(u) + \frac{|K|}{2^k}, key(p_{\frac{1}{2^k}}(u))$. In this manner, when a v joins inside this range, u can update its $p_{\frac{1}{2^k}}(u)$ peer to be v since it is closer along the circle.

To assist in updating the new node's publish links, all nodes subscribe for a node-location notification within the key range up to the first node counter-clockwise. When a node connects, it publishes a node-location event notification to find the best node to use for its $\frac{1}{2}, \frac{1}{4}, \ldots$ publish edges.

Sometimes nodes fail, and take with them their associated edges. Although most edges can be reestablished, the edges comprising the perimeter of the circle are irreplacable. We need to prevent the network from falling apart. Research has indicated [9] that a network needs $O(\log N)$ edges to expect to keep one edge alive during a network partition. Therefore, every node keeps $\log N$ edges to those nodes which are immediately clockwise from it. These are discovered with a published node-location notification. Only the first edge is actually used. The other edges simply sit dormant waiting to be used in case the perimeter is cut.

If a non-perimeter edge $u \rightarrow v$ fails, then node u seeks a new neighbour via the discovery procedure above. During discovery, the responsibility $r_u(v)$ for the failed node v must be delegated to u's first neighbour counter-clockwise of v.

It should be noted that the published notifications in this section can be filtered with a range inclusion test. As we know from Chord, notifications so filtered can be routed in only $O(\log N)$ messages.

5.2 Keeping the Invariant True

In addition to keeping the graph appropriately shaped, the nodes must preserve our invariant for filters. Again, there are two cases where the invariant becomes violated: new edges, and edge failures.

When a node joins, publishers will connect to it via the graph preserving strategy of section 5.1. Further, new edges may be created as the graph tries to preserve the appropriate edge link structure.

When a new edge is created, the receiving node should be informed of what range it is responsible for by a published responsibility-change event notification. These notifications are filtered so as to never propagate beyond one hop.

Upon receiving a responsibility-change notification (which the local invariant maintenance component is subscribed for), a node sends an update message containing the computed filter union of section 4.4. The propagation of this filter update proceeds normally, establishing the invariant.

When an edge disappears, section 5.1 dictated a responsibility change. In this case, the filter must be corrected. To achieve this, a notification for a responsibility-change event is published as in the case of a new edge. This then reestablishes the invariant.

5.3 Missing Maintenance Notifications

In the correction routines above, we cannot assume reliable event notification delivery since edges are failing. However, this assumption is not required.

For the responsibility-change event, the notification only travels down exactly one edge. If that edge fails, the node simply sends a new responsibility-change event notification to the newly responsible neighbour.

For the node-discovery event, if no appropriate publishing peers connect in a timely manner, the node retransmits the notification. Duplicate messages are not an issue here since no node connects to the same peer twice.

For the node-join notification, the node can also retransmit. When a link is established, keys are exchanged. Thus, a node can deduce whether it is missing some subscription peer edges.

Therefore, the above two resolution algorithms will be robust in the face of network flux and efficient since they use very easily filtered notifications.

6. RELATED WORK

Most P2P systems focus on efficient, distributed search, often limiting the search to hash key lookup. These systems favour the anonymous request/reply scheme and use either structured hash tables, simple broadcast networks, redundant centralised indices, or a combination.

Recent work on the scalable design of P2P overlay networks has introduced a new class of structured networks called Distributed Hash Tables (DHT's). Well known representatives include CAN [14], Chord [18], and Pastry [16]. All of these systems were built to allow efficient key lookup.

Most structured networks allow for a multicast extension to the DHT lookup scheme. Generally, these schemes are very efficient, requiring only $O(\log N)$ messages for subscribe and only wasting a few messages in the interior tree nodes during publish.

The scheme proposed for DHT's such as Chord and Pastry in [17] is to map each multicast group number to a specific node and then have it act as a rendezvous node for that group. Joining a group means to lookup the rendezvous node and have the nodes on the lookup path record the route back to the new members. A variant scheme based on CAN [15] was proposed. The idea is to have the rendezvous node act as an entry point to a distinct overlay network composed only of the group members. Thus no messages are wasted on publish since every node is interested.

So far, all publish/subscribe systems based on structured networks implement only topic based publish/subscribe. A system which uses the described overlay multicast, called Scribe [17], is implemented on top of Pastry. The mapping of topics onto multicast groups is done by simply hashing the topic name. Hermes [13] uses a similar approach, also based on Pastry. Additionally, the system tries to get around the limitations of topic based publish/subscribe by implementing a so-called "type and attribute based" publish/subscribe model. It extends the expressiveness of subscriptions and aims to allow multiple inheritance in event types.

The above networks efficiently implement a simple form of publish/subscribe. However, the topic based approach suffers from limited expressiveness and selectivity by only allowing predefined topics. Therefore, they leave a large part of the filtering to the leaf nodes of the event dissemination graph. Content-based filtering, on the other hand, aims to deliver only useful matches.

7. FUTURE WORK

While we do not assume any specific filter model to be implemented on top of our infrastructure, there are a number of general constraints on content-based filter models as described in [11]. In addition to these, our system favours filter models that allow to limit the size of update messages between the nodes. It remains to be seen what filter models can actually be implemented efficiently on top of our architecture.

Then, measuring the performance of this network is a difficult task. The performance is closely tied to the content filter selected, the probability distribution of subscribes, the distribution of publishes, and the ratio of publishes to subscribes. The most important open question here is where to put the trade-off between filter update and event notification messages.

Our proofs of correctness rely primarily on partitioning responsibility among the peers. The maintenance of the graph also depends on the peer-to-peer network, but the filter maintenance does not. We are interested in examining other peer-to-peer networks as substrates for our algorithm, particularly those with guarantees about the path length of the underlying network.

It is an interesting challenge to investigate the impact of physical locality on the behaviour of publish or subscribe. We expect that there are trade-offs to be made in choosing which edges connect physically proximate nodes.

8. CONCLUSION

In this paper, we presented a content-based publish/subscribe system built on top of a dynamic peer-to-peer overlay network. It distributes load equally by maintaining independent delivery trees for each node. This allows us to use a generalization of the publish/subscribe routing strategy. Separate components ensure that the network self-organises to maintain the optimal topology and can survive simultaneous failure of up to half of its nodes. We argue that because our system delivers via binomial trees, message delivery paths are logarithmically bounded. Details of the algorithms are provided and proven correct.

The main advantage of our system is the unique combination of the high expressiveness of content-based filters and the scalability and fault tolerance of a peer-to-peer system.

9. ACKNOWLEDGEMENTS

This work was partially founded by the German National Science Foundation (DFG) as part of the Graduate Colleges "Enabling Technologies for E-Commerce" and "System Integration for Ubiquitous Computing".

10. REFERENCES

[1] *The 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Claremont Hotel, Berkeley, CA, USA, Feb. 2003.

[2] J. Bates, J. Bacon, K. Moody, and M. Spiteri. Using events for the scalable federation of heterogeneous components. In P. Guedes and J. Bacon, editors, *Proceedings of the 8th ACM SIGOPS European*

Workshop: Support for Composing Distributed Applications, pages 58–65, Sintra, Portugal, Sept. 1998.

[3] A. Carzaniga, D. R. Rosenblum, and A. L. Wolf. Challenges for distributed event services: Scalability vs. expressiveness. In W. Emmerich and V. Gruhn, editors, *ICSE '99 Workshop on Engineering Distributed Objects (EDO '99)*, May 1999.

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

[5] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[6] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)* [1].

[7] P. T. Eugster, R. Guerraoui, and C. H. Damm. On objects and events. In L. Northrop and J. Vlissides, editors, *Proceedings of the OOPSLA '01 Conference on Object Oriented Programming Systems Languages and Applications*, pages 254–269, Tampa Bay, FL, USA, 2001. ACM Press.

[8] L. Fiege, G. Mühl, and F. C. Gärtner. Modular event-based systems. *The Knowledge Engineering Review*, 17(4):55–85, 2003. to appear.

[9] M. F. Kaashœk and D. R. Karger. Koorde: A simple degree-optimal hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)* [1].

[10] G. Mühl. Generic constraints for content-based publish/subscribe systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, volume 2172 of *LNCS*, pages 211–225, Trento, Italy, 2001. Springer-Verlag.

[11] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[12] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus—an architecture for extensible distributed systems. In B. Liskov, editor, *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 58–68, Asheville, NC, USA, Dec. 1993. ACM Press.

[13] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In J. Bacon, L. Fiege, R. Guerraoui, A. Jacobsen, and G. Mühl, editors, *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable Content-Addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, San Diego, California, United States, 2001. ACM Press.

[15] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using Content-Addressable networks. In J. Crowcroft and M. Hofmann, editors, *Proceedings of the Third International COST264 Workshop (NGC 2001)*, volume 2233 of *LNCS*, pages 14–29. Springer-Verlag, nov 2001.

[16] A. Rowstron and P. Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, 2001. Springer-Verlag.

[17] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In J. Crowcroft and M. Hofmann, editors, *Third International Conference on Networked Group Communication (NGC 2001)*, volume 2233 of *LNCS*, pages 30–43, London, UK, 2001. Springer-Verlag.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, California, USA, 2001.