End-to-End Reliability for Best-Effort Content-Based Publish/Subscribe Networks

Amirhossein Malekpour
University of Lugano
Lugano, Switzerland
malekpoa@usi.ch

Antonio Carzaniga
University of Lugano
Lugano, Switzerland
antonio.carzaniga@usi.ch

Fernando Pedone
University of Lugano
Lugano, Switzerland
fernando.pedone@usi.ch

Giovanni Toffetti Carughi
University of Lugano
Lugano, Switzerland
toffettg@usi.ch

ABSTRACT

When it comes to reliability, there are two main categories of distributed publish/subscribe systems: reliable systems and best-effort systems. The former gives the highest priority to guaranteed and ordered delivery while the latter aims for high throughput and low delays. We propose a method to improve the delivery guarantees of the basic unreliable service offered by a best-effort publish/subscribe system. This method does not require any modification to the system's protocols or broker software, and instead simply uses the system's publish/subscribe API. The method is based on a technique, similar to reliable multicast, that enables subscribers to cooperatively recover lost messages. We experimentally demonstrate the effectiveness and performance of our recovery scheme in the presence of frequent message losses, and show that it enables subscribers to recover more than 70% of lost messages with minimum negative effects on the overall network performance.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms

Reliability,Performance

Keywords

Content-based networking, publish/subscribe

1. INTRODUCTION

Message oriented middleware, and in particular content-based publish/subscribe systems, have a wide range of applications in enterprise environments, in areas such as process control, work-flow, asset management, and system management. These applications usually have stringent requirements in terms of delivery guarantees, so to support such applications, system architects opt for publish/subscribe systems that provide guaranteed delivery. Various forms of reliability have been studied, along with various methods to achieve them [1, 2, 16, 8, 11]. However, in this paper we refer primarily to a form of reliable delivery also known as *message persistence* in such standards as the Java Messaging Service specification. According to this delivery mode, the service must guarantee not to loose messages due to failures of message brokers.

Persistent messages are clearly a desirable feature for applications, but they also have a cost in terms of lower throughput and greater end-to-end delay. In fact, in order to offer such higher delivery guarantees, the publish/subscribe system (distributed or not) must implement a store-and-forward mechanism whereby each broker must log each message onto a persistent storage before accepting the message for delivery from a client or from another broker, and such a commitment to deliver must also be confirmed to the sender.

On the contrary, systems in the "best-effort" class [10, 3, 6, 17] do not offer guaranteed delivery but instead try to maximize throughput and reduce end-to-end delay. So, typically, these systems do not log messages to a persistent storage, nor do they implement any mechanism to guarantee message ordering. Acknowledgment messages are not used and no explicit congestion control mechanism is in place. The advantage of these systems is that they allow for more streamlined message processing, with simpler protocols and with broker designs closer to those of network routers. Still, despite their better performance and simplicity, the unreliable nature of best-effort systems seems to limit their deployment significantly, especially in critical application domains.

Our goal in this work is to reduce the dichotomy between the reliable but more involved store-and-forward architecture, and the unreliable but streamlined best-effort architecture. In other words, our goal is to obtain a combination of the best features of both types of service. More specifically, we take a best-effort network as a basis for a content-based publish/subscribe service, and on top of that we design an end-to-end, probabilistic reliable service. The protocol is end-to-end in the sense that it requires no changes in the internal structure of the broker network or its protocols,

DEBS'11, July 11–15, 2011, New York, New York, USA.

and relies only on the participation of clients (though, the same technique we propose can be easily extended to take advantage of in-network caches). The resulting service is probabilistic in nature, in the sense that it can not guarantee delivery in an absolute sense, and in fact it probably would not achieve the same reliability as schemes that use stable storage within the network. Nevertheless, as we show experimentally, in practice the service can reduce message losses significantly, and with only minimal compromises in terms of throughput.

Our design is modeled after Scalable Reliable Multicast (SRM) [9]. Very briefly, SRM works as follows: a receiver that detects a message loss (using sequence numbers) tries to recover a copy of that message from other group members by multicasting a request to the group. SRM also uses an adaptive timer to reduce the number of duplicate requests and replies, and to cope with the effects of such duplicates.

However, SRM is not directly applicable to content-based publish/subscribe because messages are not explicitly addressed to a group, and therefore because it is not immediately clear how to address a request for a lost message. In fact, for the same reason, it is not even clear how to *detect* a message loss. This is because there is no clearly identifiable stream of messages other than what is published by a single source, and gaps in such a stream are very often due to the legitimate filtering of the content-based selection. In other words, simple sequence numbers do not allow a receiver to distinguish a lost message from a message that was legitimately filtered out by the receiver's subscriptions.

The protocol we propose addresses these two fundamental issues using a synthetic publication record that is attached to messages and that allows receivers to detect message losses and also to request the corresponding missing messages. We also extend SRM with a simple scheme for better cache management. We implemented and tested this protocol within a best-effort content-based publish/subscribe system. Our experiments show that the protocol is capable of recovering from a large number of messages losses in networks of different size and with different dynamics.

In Section 2 we first set the context of our work by discussing best-effort publish/subscribe networks and reliable IP multicast, and then present the problem we address and give an overview of the reliability protocol we propose as a solution. In Section 3 we detail the reliability protocol and its implementation. Section 4 presents the experimental evaluation and in Section 5 we review related research. Finally in Section 6 we offer some concluding remarks.

2. CONTEXT AND PRELIMINARIES

In order to put our end-to-end reliability protocol in the proper context, we briefly review best-effort content-based networks and the analogous reliability protocols for IP multicast. We also give a high-level summary of our solution.

2.1 Best-Effort Content-Based Networks

A best-effort content-based network is essentially a distributed publish/subscribe system architected as a datagram network, where routers act as event dispatchers. The addressing in this network is "content-based" in the sense that messages (publications) are addressed implicitly by their content and by the predicates (subscriptions) matching that content posed by receiver hosts. The communication service provided by the network is "best effort" in the sense that messages are treated as datagrams in an IP network, and therefore may be lost due to link instability and congestion in routers. A best-effort content-based network may be built as an overlay on top of an unreliable transport protocol like UDP, or on reliable TCP connections. There are also publish/subscribe systems that directly work atop IP multicast to disseminate events to large sets of subscribers [14]. However, regardless of the nature of the overlay or network underlay, congestion and errors in routers can still render the network unreliable. On the positive side, the goal of the best-effort design is to minimize the amount of state maintained within the network and reduce processing at each router, thereby promoting efficiency and scalability.

In this paper we assume a very common content-based publish/subscribe interface in which messages are at least in part structured as sets of *attributes*, and can be selected by subscriptions predicated upon the values of those attributes. In particular, the term *constraint* refers to a condition on the value of an attribute, the term *filter* denotes a logical conjunction of constraints (sometimes called a subscription) and the term *predicate* denotes a logical disjunction of filters (i.e., a set of subscriptions).

2.2 Reliable IP Multicast

A best-effort content-based network offers a service that is similar in nature to IP multicast. Since our goal is to improve the reliability of a best-effort content-based network, and specifically since we propose to do that with a pure end-to-end solution, we model our solution after the existing end-to-end reliability protocols developed for IP multicast. As we will discuss later, such protocols are not immediately applicable to a content-based addressing because of its greater expressiveness. Nevertheless, we review such protocols here and in particular we focus on Scalable Reliable Multicast (SRM) [9] as a basis for our reliability protocol.

In the context of IP networks, a number of reliability mechanisms have been proposed in the form of additions to the standard IP multicast. Among the most notable ones, Scalable Reliable Multicast (SRM) [9] and Reliable Multicast Transport Protocol (RMTP) [12] provide reliability without reliance on the routing infrastructure (i.e., "end-to-end") while Pragmatic General Multicast (PGM) [18] proposes additional functionalities to routers in order to provide reliability. In the terminology of these protocols, a *request* is a (broadcast or multicast) message whose function and meaning is similar to that of a negative acknowledgment (NACK), which is to request a missing message. The term *repair* refers to a reply (to a request) that carries the missing message. We use these terms with the same semantics in the rest of the paper.

We chose SRM as a basis for our reliability protocol for two reasons. First, SRM makes limited assumptions about the application logic, and second, it only relies upon a basic multicast service to recover lost messages without requiring any addition or modification to the underlying multicast service. Since in the paper we often refer to SRM semantics and internals, we now give a cursory description of how SRM achieves end-to-end reliability. For a detailed description we refer the reader to the original paper by Floyd et al. [9].

SRM is a general-purpose reliability protocol designed for large scale applications that use IP multicast. To be generic, SRM does not make any particular assumption about the formats and sizes of application-level messages. Thus, mes-

sage loss detection is not embedded in the protocol but is instead assumed to be part of the application logic. Once the application detects the loss of a message within a group, it multicasts a *request* to the same group. Other applications in the same group that received (and cached) the message respond by multicasting a *repair* to the group. In order to reduce duplicate requests and repairs for the same message, nodes hold their requests and repairs for an initially random delay. A node holding a request would then progressively back off by doubling the delay every time it receives a request for the same message. A node holding a repair would simply cancel the repair upon receiving the same repair.

The random delays are chosen uniformly in the interval $[C_1 d_{s,a}, (C_1+C_2)d_{s,a}]$ for request and $[D_1 d_{s,a}, (D_1+D_2)d_{s,a}]$ for repair messages where $d_{s,a}$ is the average message trip-time from the multicast source to the node and C_1, C_2, D_1 and D_2 are adjustable parameters. The protocol has an algorithm for automatic tuning of these parameters so that the likelihood of duplicates and the time to recover lost messages are lowered to a minimum.

2.3 Problem and Overview of the Solution

To extend the simple idea of cooperative message recovery to content-based networks, we must overcome two main technical problems. The first problem is to enable receivers to *detect* message losses. For some applications, specifically when messages are channeled into identifiable streams and subscribers receive all messages in a stream, this can be easily done by marking each message within its stream with a sequence number. In this case, a gap in the sequence numbers indicates a message loss. However, such streams do not exist in a content-based publish/subscribe network, where messages are delivered only if they match the interests of subscribers, and where such interests may partially overlap. In other words, with partially overlapping receivers' interests, it is impossible to assign sequence numbers to messages so as to obtain continuous sequences for all receivers. Therefore, in practice, a receiver can not distinguish a message that was lost from a message that was not delivered because it does not match the receiver's interests.

We address this problem by adding some information to each message that allows a receiver to determine, with some probability, if any of the latest publications of the sender that were not received was in fact of interest for the receiver. This information, which we call the *publication record*, consists of a set of Bloom filters, each encoding one of the sender's most recent publications. We discuss this encoding below.

The second problem is to *recover* a lost message that was determined to be of interest. As in SRM, we propose a cooperative recovery scheme whereby a lost message is recovered from some other application in the network that might have received and cached the message. In SRM a receiver would multicast a request for a lost message to the same multicast group to which the message was sent, which conveniently identifies all potential caches from which the message might be recovered. Unfortunately, in a content-based publish/subscribe network it is not as easy for a receiver to address other receivers of a given message. One way to reach potential caches is to send a request to *all* caches, which can be done by having all caching applications subscribe for a generic "request" message. However, that solution might incur an unacceptable overhead for caching applications and for the network in general, especially in situations where the network is already congested.

We address this second problem using the same publication record attached to messages. A receiver that receives a Bloom filter B_m from the publication record of a message m', and therefore determines that m was of interest, creates a request message that includes B_m. This request is intended for other applications interested in the same message m that are willing to serve as caches for lost messages, and therefore that would subscribe for request messages that match B_m.

The publication record, and in particular the encoding of messages into Bloom filters, is such that a subscription S can be evaluated against the Bloom filter B_m representing message m (without the original content of the message). This is what allows a receiver to verify that a message identified by a Bloom filter B in the publication record matches one of the receiver's subscriptions. So, our idea is to express this matching condition between a Bloom filter B and a receiver subscription S within another subscription. This is possible thanks to the structure of the Bloom filters that encode the message content, which we discuss below in Section 3.1.

Lastly, an application that has a cached copy of a lost message and that receives a request for a repair must somehow deliver that message to the requesting application. This can be done in a straightforward way through a direct (unicast) connection, or also through the publish/subscribe API by effectively republishing the message for the requesting receiver. Each of these solutions has advantages and disadvantages that we discuss below.

3. END-TO-END LOSS RECOVERY

3.1 Message Loss Detection

As stated in the previous section, in a content-based publish/subscribe network, where subscriptions can express partially overlapping interests, conventional sequence numbers are not sufficient to detect message losses. To remedy this problem we augment messages (publications) with an encoded summary of the latest publications of the same sender. This summary, which we call the sender's *publication record*, may allow a receiver to determine that a particular message that was not received was in fact lost.

A publication record is attached to a message m_k by its source s and consists of R entries representing the previous R messages $m_{k-1}, m_{k-2}, \ldots, m_{k-R}$ published by s. Each entry B_i is a Bloom filter obtained by encoding message m_{k-i} using the encoding scheme developed by Carzaniga et al. [4]. The encoding works as follows: first, a message m is mapped into a set of "categories" or "tags." For example, a message that contains the attributes (event=disk-failure, cause=overheating, priority=high) might be associated with tags "disk-failure," "overheating," and "high-priority." Then the set of tags is simply represented as a Bloom filter. In addition to defining sets of tags for messages, the encoding scheme also defines tags for subscriptions with the intended semantics that, if a message m matches a subscription S, then the tags associated with m are a *superset* of the tags associated with S. Carzaniga et al. describe one such encoding that is quite simple but that also incurs some false positives [4]. For the purpose of this paper we adopt the same encoding. In summary, a subscription S is encoded as a Bloom filter B_S (representing a set of tags) and a message

m is encoded as a Bloom filter B_m (representing a set of tags), and if m matches S then $B_m \supseteq B_S$ (where a Bloom filter B is interpreted as a set of bits).

When a subscriber receives a message m_k that carries a publication record $\langle B_1, \ldots B_R \rangle$ the subscriber checks whether it has received messages $m_{k-1}, \ldots, m_{k-R}$ from the same publisher. Then, for each message m_i that was not received, the subscriber checks whether the B_i entry in the publication record matches any of its subscriptions S. That is, the subscriber checks whether there is one of its subscriptions S such that $B_i \supseteq B_S$. (Of course, the subscriber does not have to recompute the encoding of its subscriptions for each message.) If one such subscription is found for B_i, then the subscriber concludes that message m_{k-i} was lost and may decide to try to recover the message.

3.2 Routing Requests

A subscriber may try to recover a lost message by requesting a copy of that message from other applications that received and cached the message. The problem is then to address such a request so as to reach all and only the receivers that might have cached a copy of the lost message.

We propose to transmit the request through the same publish/subscribe system, by constructing a special request message and by having receivers subscribe for those requests for messages that they might have received. So, an application with subscription S that is willing to cache messages for other applications must subscribe for requests for messages matching S. For example, a simplistic way to do that for a subscriber interested in, say, {news=sport, team=Yankees} would be to subscribe for a request message {reliability-message=request, news=sport, team=Yankees}. However, this simplistic approach does not work. For one thing, the second subscription is useless, since the first one would already match all corresponding request messages. More importantly, a subscription for a request that repeats the same constraints as a normal subscription would not work, because a receiver trying to recover a missing message may not be able to fill in the relevant attributes.

Consider in fact the following scenario. Application Alice subscribes for {team=Yankees} while application Bob subscribes for {news=sport}. Now, suppose Alice receives message m ={news=sport, team=Yankees} while the same message is lost on the way to Bob. Also suppose that Bob receives a following message m' ={news=sport, team=Mets} carrying a publication record that allows Bob to detect the loss of m. At this point, Bob can determine that the missing message m matches its own subscription {news=sport} and therefore might issue a request {reliability-message=request, news=sport}. However, that request would not reach Alice. In order for such a request to reach all potential caches, Bob would have to fill in all the attributes of m in the request. In other words, Bob would have to know m completely in order to issue an effective request to recover m.

We propose to overcome this problem by once again relying on the information contained in the publication record. In the scenario we just described, Bob does not know the content of the missing message m, but he does know its encoding B_m. Therefore, Bob could include B_m in a request message such that Alice could subscribe for it using an encoding of its own subscription. For example, suppose that Alice's subscription would be encoded in a Bloom filter B_A whose 1-bits are at positions 7 and 15 (all other bits are zero). Then, Alice could subscribe for {reliability-message=request, b7=true, b15=true} effectively representing B_A by means of attributes each representing one of the 1-bits in B_A. Notice that what matters is the *presence* of such attributes, not their value. Now, Bob could do the same by composing his request for m using the encoding B_m he finds in the publication record. And since the encoding is such that a message m matching a subscription S would be encoded by a Bloom filter B_m whose 1-bits are a *superset* of the subscription's Bloom filter B_S, Alice would have to receive that request.

With the ability to detect lost messages and to route requests to all potential caches for those messages, the recovery process proceeds in a similar fashion as in SRM. We illustrate this process through a an example. Consider a subscriber A that has an active subscription and suppose that A is willing to provide repairs for all the events matching that subscription. To do that, A encodes the subscription and issues a second subscription using the 1-bit attributes as described above. As subscribers intending to participate in the recovery mechanism, B and C also initially issue an additional subscription to receive matching repair requests. As we will detail below, this is done to suppress duplicate requests.

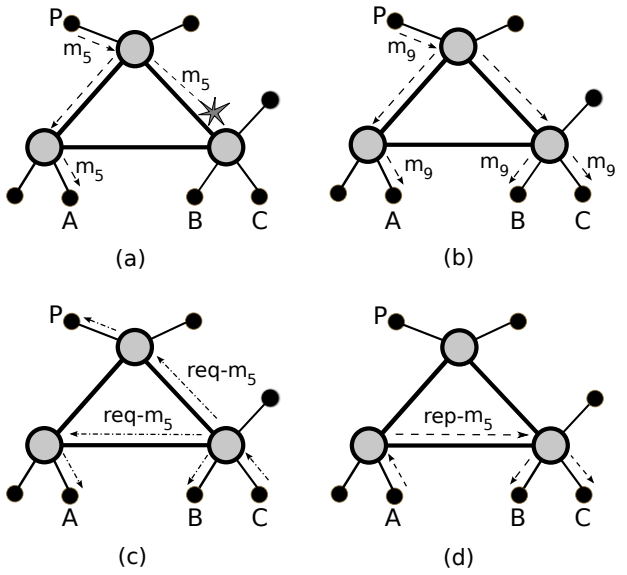


Figure 1: Message m_5 is lost before reaching B and C (a); having received m_9 (b), C publishes a request for m_5 (d); A replies with a repair (d).

Now, assume that later, a publisher P publishes an event m_5 matching A's subscription. The message is received by A but due to a temporary failure it is not received by B and C whose subscriptions also match m_5 (see Figure 1a). Later, P publishes another message m_9 that is received by B and C (because it matches their subscriptions). This message carries the publication record of P that includes the Bloom-filter encoding of m_5. Using that data, both B and C realize that they have missed m_5 (see Figure 1b).

The detection of a message loss triggers the recovery mechanism at B and C. Thus, B and C issue repair requests, but in doing that, they try to suppress duplicate requests. The two nodes start to count down from a randomly generated

timer (discussed in Section 2.2). Assume that C picks the earlier timeout between the two. Once C's timer expires, C publishes a request for m_5 by including the 1-bit attributes corresponding to the Bloom-filter encoding of m_5 as well as a unique message id for m_5 (source plus sequence number) that is also part of the publication record. This request is received by both B and A because it matches their request subscriptions (see Figure 1c). Upon receiving the request, B, which has a timer running on an identical request, reacts by delaying its own request by doubling its timer value.

A instead reacts by searching its message cache for a message with the given id, and when it finds it, publishes a repair message consisting of the original message m_5 with an additional attribute that indicates that it is a repair (and therefore a duplicate publication). The repair message reaches both B and C (see Figure 1d), at which point B cancels its pending request and both B and C deliver m_5.

Thanks to the expressiveness of content-based networking, this request routing technique allows for fine grained control over the request messages and their receivers. On the one hand, it allows clients to precisely describe request message they are willing to reply to, if any. On the other hand, it also allows for the implementation of special policies for the distribution of request messages, for example to confine request and repair messages within a single administrative domain. It is also easy to use this recovery protocol with designated cache nodes distributed in strategic points over the network that take on the responsibility of providing repairs, as suggested in RMTP [12] for reliable IP multicast.

3.3 Sending Repairs

As explained in the example of Figure 1, a caching subscriber can respond to a request by re-publishing a message and by flagging that publication as a repair. Alternatively, the cache may send the repair directly to the requesting node through a unicast connection. Re-publishing is advantageous when the same message is requested by several receivers—something that might happen with overlapping subscriptions and non-local faults. Conversely, a unicast reply may be a better option when no other receivers requested the same message, and when that message would reach many receivers that already received the original copy. There may also be security and authenticity issues with the repair messages that are provided by caching nodes. This can be seen as a general trust and security management problem in the context of publish/subscribe systems, which is out of the scope of this paper.

3.4 Message Cache

An important parameter that is not discussed in the original paper on SRM by Floyd et al. [9] is the amount of memory a caching node (generally, a subscriber) has to allocate to its message cache. The cache should be allocated and managed so as to obtain a high cache-hit ratio while also avoiding unnecessary caching and therefore saving memory resources. In this section we elaborate on this issue.

On the one hand, for performance reasons we would like to have nodes maintain the message cache in main memory, within the limits of their memory constraints. On the other hand, it is also desirable to limit memory usage to a minimum. Therefore, nodes need to know when to drop some of the messages from their cache. The best strategy depends on various parameters, such as end-to-end message delivery delay, sender publication rate, match probability, and timer parameters of the requesting nodes. A message cache of constant size is oblivious to such network dynamics and may sometime lead to memory waste and other times to cache misses. Instead, we seek an adaptive cache that would perform well under variable network conditions. In particular, we formulate an approximation for the optimum cache size focusing on the most common scenario.

Symbol	Meaning
λ	Publication rate
C_1, C_2	SRM request timers
P_m	Match probability
$d_{P,B}$	Trip time between requester and publisher
$d_{B,A}$	Trip time between requester and caching node
K	Cache size

Table 1: Parameters used in the calculation of the cache size.

We consider the message cache as a ring buffer with dynamic size, and we consider the problem of finding the optimum size of this ring buffer based on the parameters summarized in Table 3.4. (Indeed, our implementation uses a ring buffer for its message cache.) For this formulation we assume that message losses occur in bursts of S messages per second, where $1 \leq S < \lambda P_m$. We further assume that request messages are not lost.

Consider a publisher Z that publishes a message m_i. This message has two intended receivers: A that receives m_i and B that incurs a message loss and does not receive m_i. To find a lower bound for the size K of the message cache at A, we assume that A is closer to the publisher while B is farther away from the publisher in terms of end-to-end delay. Thus, on average, it takes $d_{Z,B} + \frac{1}{\lambda P_m} + \varepsilon$ time units for B to receive m_j, the next (relevant) message from Z, and detect the loss of m_i by investigating the publication record attached to m_j. We neglect ε, the small processing time of m_j. Therefore, the request message from B will be received by A after an expected delay of $(C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A}$ time units. Consequently, in order to be able to provide a repair for the lost message m_i, A has to store m_i in its ring buffer for at least $d_{Z,B} + \frac{1}{\lambda P_m} + (C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A}$ time units.

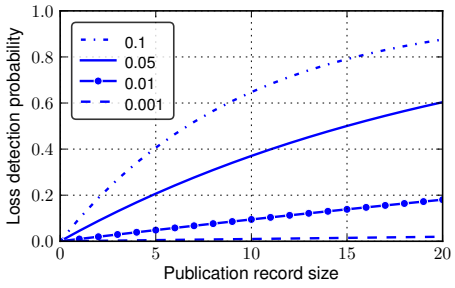
The minimal caching time we just computed determines, together with the arrival rate at A, determines a minimal cache size. Since A receives messages from the publisher with an expected inter-arrival time of $\frac{1}{\lambda P_m}$, and hence must overwrite m_i in its ring buffer approximately after $\frac{K}{\lambda P_m}$ time units. Thus, in order for A to be able to provide repair for m_i, the request must arrive at A before A overwrites m_i in its ring buffer. So we have:

$$d_{Z,B} + \frac{1}{\lambda P_m} + (C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A} \leq \frac{K}{\lambda P_m}$$

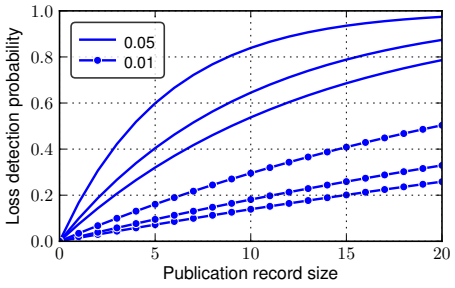
solving the inequality, we find a lower bound for the cache size K:

$$K > \lambda P_m[(C_1 + \frac{C_2}{2} + 1)d_{Z,B} + d_{B,A}]$$

The parameters of the above formula vary over time, and therefore must be continuously estimated. Each caching



(a)



(b)

Figure 2: Probability of loss detection for (a) different sizes of publication record and match probability, (b) different sizes of publication record, match probability and different number of nodes sharing a loss (2, 3, 5 from bottom to top in each line category).

node A, can directly measure λP_m which equals the reception rate from the publisher Z at A. The value of $(C_1 + \frac{C_2}{2})d_{Z,B})$ is not measurable by A and therefore is included in each request message sent by B. The auto-tuning algorithm in SRM works in a way that the requesting node is likely to be the closest node to the point of failure. Thus, the requesting node for a given publisher is likely to often be the same node. Thus, this value is averaged over received requests for message of the publisher Z with the assumption that requests are coming from the same node or nodes at the same distance from the publisher. Finally, the value of $d_{B,A}$ is easily found by the simple method described by Floyd et al. in [9].

Algorithm 1 Adjust cache size

```
if closets_repair_provider = True then
    a ← ((C1 + C2/2 + 1)dZ,B + dB,A)λPm + 1
    b ← 0
    if HitRatio < H then
        b ← 1.1K
    c ← max(a, b)
else
    c ← 0.9K
if Kmin ≤ c ≤ Kmax then
    K ← c
```

Another consideration is that based on SRM semantics, via auto-adjustable timer parameters, the node that provides repair messages is usually the closest node (in terms of end-to-end delay) to the point of message loss, which is in fact a method to minimize recovery time. Therefore, it is better to have this node handle caching more aggressively and other nodes gradually reduce their cache size, since they are not actively involved in providing repairs for messages coming from that specific publisher. Moreover, given that the above formulation gives a lower bound for K, caching nodes also need to monitor their average hit ratio, which is updated upon reception of each request, and if this ratio is below a preconfigured value, increase their cache size. Algorithm 1 shows this procedure, which a caching node executes periodically. H is a preconfigured hit ratio and K_{min} and K_{max} are minimum and maximum allowed cache size, respectively, and are configurable parameters. This algorithm is quick to increase the cache size as a reaction to sudden spikes in publication rate λ, and instead reacts with a gradual increase (10%) of the cache size when the network is stable but the hit ratio is lower than the configured level H. The algorithm is also rather conservative when it comes to reducing cache size, since we would like to avoid dropping cache entries too early as a result of abrupt changes in the network dynamics.

3.5 Discussion

The effectiveness of the proposed recovery protocol is primarily influenced by the effectiveness of the loss detection. Obviously, the publication record carried by each message must be limited in size due to practical limitations and also to limit traffic overhead. This is in fact, where the probabilistic nature of our recovery protocol stems from.

We now discuss the impact of publication-record size on the probability of loss detection. Considering a subscriber S and a publisher P whose publication matches S's subscriptions with probability P_{match}, the probability that S detects the loss of a message sent by P depends on the message loss probability P_{loss} on the path from P to S, the matching probability P_{match}, and the size R of the publication record. Simply put, if each message carries a publication record of size R, then the loss of a message m will not be detected by its intended receiver if none of the R messages published after m are received by that receiver. So, the probability of loss detection is:

$$1 - (1 - P_{match} + P_{match}P_{loss})^R \quad (1)$$

Figure 2a shows the probability of loss detection for different sizes of publication record and different match probabilities for a loss probability of 0.01. As the figure suggests, when the match probability is 0.001 (i.e., out of every 1000 publications, only one message is expected to match the subscriber's interests) the loss detection mechanism is very inefficient. With 5% matching probability, a publication record of size 10 makes the loss-detection possible with a probability of 0.4. At a first glance, Figure 2a might suggest that this loss detection mechanism would limit the applicability of our recovery mechanism. While this is true for applications with very low match probability, note that Figure 2a demonstrates the worst case where there is only one intended receiver for the lost message. In other words, when a message loss is shared among multiple receivers, it is sufficient that only one of them detects the loss. This happens when subscribers have overlapping subscriptions.

As an example, let us consider a case of Figure 2a where

a node n has a match probability of 0.01. Let us also assume that there are k other nodes ($k = 1, 2, 4$ in Figure 2b) with the same matching probability as n and we also assume that all these nodes' subscriptions have a 50% overlap with n's subscriptions. That is, any message that matches n's subscription will match all of the other nodes' subscriptions with a probability of 0.5. Figure 2b (the three bottom lines) shows the growth of the loss-detection probability (the probability that at least on node detects a shared loss) with the growth of the publication-record size. The three top lines correspond to the case where nodes have a match probability of 0.05 and the rest of the scenario is similar. Notice that the growth of loss-detection probability with the size of publication record is faster for larger values of k.

Another consideration about this loss-detection mechanism is that if the publication rate (number of publications per time unit) is relatively low, detection of a message loss will be *late*, especially for subscribers with low match probability. This problem can be alleviated by periodic soft-state messages sent by the publisher. Such messages only serve the purpose of enhancing the loss detection on the receiver side. Receivers whose match probability is too low can subscribe for soft-state messages for faster and more successful loss detection.

Another way to mitigate the limitation of the proposed loss-detection mechanism is to maximize the number of messages that can be summarized into a publication record of a given size. Currently, we are investigating the *temporal locality* of events. That is, when two or more consecutive events sent out by a publisher have overlapping attribute/value pairs. We plan to exploit temporal locality to enhance the encoding scheme to allow for compression that is, merging the encoded format of a few similar events in a single Bloom filter. This might further increase the likelihood of a false positive however, we believe that the overall bandwidth usage will be improved with this compression mechanism. Entries of a publication record can be further compressed using compressed Bloom filters [13] to reduce space usage and bandwidth overhead.

4. EXPERIMENTAL EVALUATION

In this section we evaluate the performance of the recovery protocol with a focus on effectiveness and cost. More specifically, we are interested in measuring how many lost messages are recovered and how long it takes to recover them. Another practical question we explore is how much extra traffic is generated by the recovery protocol and what is the user-tangible impact on the ordinary traffic.

We note that the performance of the recovery protocol depends significantly on the choice of topology, workload, and message-loss probability. Our choice for these experimental settings does not aim to demonstrate the best-case performance of our protocol, but rather intends to examine its effectiveness under stress, in the presence of frequent message losses, and with conservatively chosen workloads that do not necessarily contribute to increase the effectiveness of the recovery protocol.

4.1 Experimental Setup and Workload

We implemented the recovery protocol as a pluggable module which integrates into any publish/subscribe application and protocol that provides a common publish/subscribe API. In particular, the publication record and other metadata required by the recovery protocol is attached to messages as an array of bytes treated by brokers as application payload not subject to the matching process. Most implementations of the Java Messaging Service (JMS) are capable of carrying an opaque payload and hence clients using such systems can take advantage of the recovery protocol when the system runs in best-effort mode. For the experiments presented in this section we used a recent implementation of the Siena publish/subscribe system that implements a best-effort content-based communication system [4].

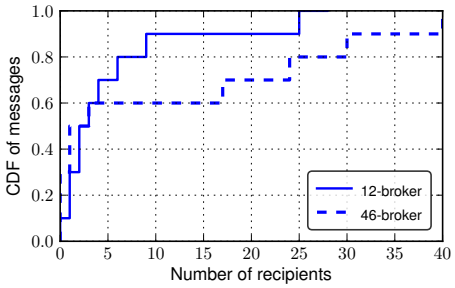
Our experiment testbed is a cluster of Dell PowerEdge with two dual-core 2GHz AMD Opteron processors and 4GB of main memory running Linux with a 2.6.32 kernel. Connectivity is through an isolated high-throughput Gigabit Ethernet switch. Broker software, client and recovery protocol are implemented in Java and run on the 64-bit open-JDK VM. Each physical machine hosts an broker that serves 5 instances of the client (as their home broker) running on the same machine. To simulate a wide area network, we used the Linux traffic control tools to apply delay and message loss on all inter-broker and links. Each link's delay d_i, is randomly chosen in the range of $[25, 75]$ milliseconds, which also continuously varies during the execution in the range of $[d_i - 5, d_i + 5]$ milliseconds. This variation of ± 5 milliseconds is typical of the Internet, based on different Internet measurements.[1]

We present the results for two sets of experiments with two different network topologies and workloads. The first topology is composed of 12 brokers with a diameter of 5 broker-hops in which out of the total of 60 clients, 6 nodes are publishers and 54 are subscribers. Then to probe scalability of the recovery protocol, the second experiment involves a larger topology of 46 brokers with a diameter of 11 broker-hops. Among the total of 230 clients, 10 are publishers and the rest are subscribers.

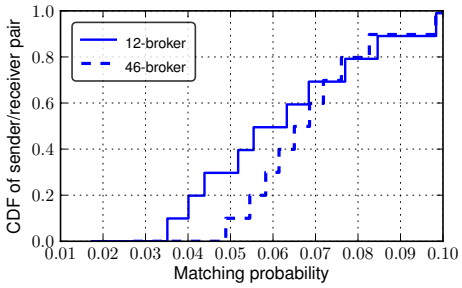
Using the Linux traffic control tools, we apply a link-level message loss probability of 0.01 to all inter-broker links (i.e., each link loses approximately one message out of each 100 messages). This loss probability is rather large because for a network of diameter 11 it sums to unrealistically large likelihoods of message loss for some endpoints. For example, for a sender and a receiver 11 broker-hops apart, the probability of message loss is as large as 0.1. This is a deliberate choice to stress-test the recovery protocol under very frequent message losses.

In both experiments, we use synthetic workloads with varying publication rates that simulate sudden short-term spikes in publication rate of publishers, to simulate bursty traffic that causes queuing delays. In these experiments, all the nodes participate in the recovery process. That is, they all volunteer to provide repair for messages that they receive. As shown in Section 3.1, two crucial factors in the loss detection and hence in the effectiveness of the recovery protocol are the matching probability and the number of receivers for a message. To be conservative in our evaluation, we choose workloads that exhibit low subscription/publication match probability (the probability that a message matches a node's subscription) and a low number of receiver per message. Figure 3 characterizes the two workloads for the 12-broker and 46-broker experiments. As Figure 3a shows, in both work-

[1]For example see measurements by RIPE Network Coordination Centre (RIPE) available at http://www.ripe.net/data-tools/stats/ttm/ttm-data



(a)



(b)

Figure 3: (a) Number of receivers for cumulative distribution of messages. (b) Match probability for cumulative distribution of subscriber/publisher pairs.

loads 50% of the messages have at most two receivers. Figure 3b plots the subscription/publication match probability for each pair of subscriber and publisher. In both workloads, 80% of the pairs have a match probability of less than 0.08 while the maximum match probability is not higher than 0.1.

Figure 4, plots the aggregate publication and delivery rates without the recovery protocol during the course of the experiments, which runs for a total of 5 minutes. The rapid changes in delivery rate is due to spikes in publication rates.

4.2 Recovery Effectiveness

First we look at the number of false negatives (that is, the number of messages that were not delivered to their intended receivers) with and without the recovery protocol. One determining factor in the effectiveness of the protocol is the size of the publication record. Figure 5 illustrates the effectiveness of the recovery protocol with different publication-record sizes. The y-axis shows the total number of false negatives (message losses) and zero on the x-axis represents the case where no recovery protocol is in place. The decrease in the number of false negatives is more pronounced in the 46-broker network while it is slower in the 12-broker network. Indeed, in the 12-broker experiment, growing the publication record size from 1 to 10 only halves the number of false negatives while in the 46-broker networks the false negatives are reduces by a factor of 3, approximately.

Our calculation of the probability of message loss detection by Equation 1 as well as Figure 2a explain this result, which is mostly due to the characteristics of the workload, i.e., small matching probability and small subscriber/message, which hinder a more effective loss detection in the 12-broker network. Furthermore, our experiments with smaller message loss probabilities showed that the exponential effect of publication record size on the recovery effectiveness is more substantial when message loss probability is smaller, which is also explained by Equation 1. For instance, in the 46-broker network when loss probability is 0.001, the recovery protocol with a publication record size of 10 reduces the number of false negatives by more than 8 times.

We now examine the network dynamics and the corresponding protocol behavior over time. To do that, we focus on the experiments with the best effectiveness results, that is, with a publication record of size 10. We choose this case because a larger size for the publication record generates larger amounts of network traffic and hence by studying this case we gain a better understanding of its impact on the network.

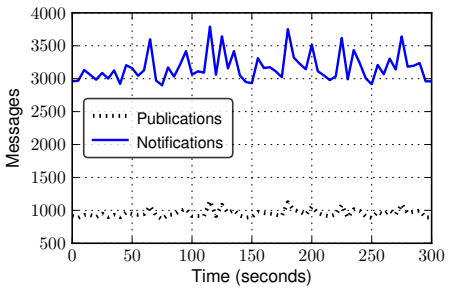
We begin our probe by looking at the aggregate rate of false negatives with and without the recovery protocol during runtime, shown in Figure 6. The reduction in the false negatives that is almost a factor of two for the 12-broker and a factor of three for the 46-broker network is persistent during the whole course of the experiment. So, at a high level, the recovery protocol does not show any pathological behavior during the experiment.

We now proceed to examine another aspect of the recovery effectiveness, namely the time it takes to recover a lost message. Figure 7 shows the end-to-end delay of the original (non-repair) messages as well as the repair messages. Note that the delay of repair messages is in fact the time difference between their publication by the original publisher and their delivery to the indented receiver as a repair. Figure 7 also plots the request/repair delay, which is the time difference between multicast of the first request for a certain message and the first repair for that message. An interesting observation is that the request/repair delay in both networks is relatively small: for 80% of the messages in 12-broker and 46-broker networks, this delay is below 200 and 350 milliseconds, respectively, while the total time to recover missing messages is considerably larger. This means that a large part of the recovery delay is due to "late" loss detection, which is a result of low matching probability and/or low publication rate. This is not surprising, since the publication rate of each publisher varies between 20 and 500 messages per second in the 12-broker, and between 20 and 250 messages per second in the 46-broker networks, and hence, with a match probability of 0.05, it might take up to 1 second to detect a message loss.

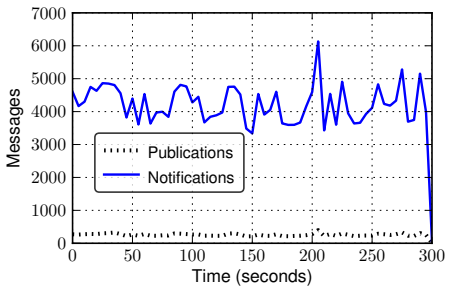
Also, note that our choice of large message loss probability causes many of the requests to be lost and so, some requests must be reissued for a second or a third time, each time after a timeout. In fact, in other experiments with 46-broker network when we applied smaller link loss probabilities, we observed that the request/repair delays were almost 50% smaller, which in turn resulted in smaller recovery times.

4.3 Performance and Network Overhead

We now turn our attention to the operating costs of the recovery protocol. In particular, We consider two measures: network usage and the overall impact on the receivers in terms of the extra delivery delay that the original messages incur. The extra network load is caused by the publication record attached to each message as well as the traffic of request and repair messages. In our workload all messages

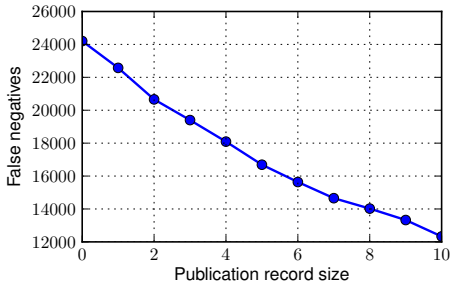


(a)

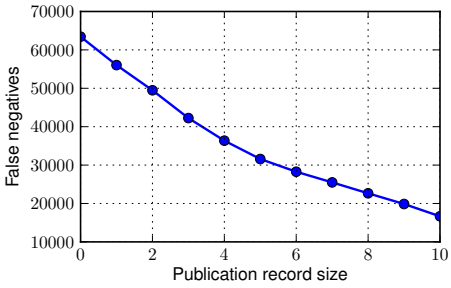


(b)

Figure 4: Aggregate publication and notification rates in (a) 12-broker and (b) 46-broker networks.



(a)



(b)

Figure 5: Impact of publication-record size on the effectiveness of the recovery protocol in (a) 12-broker and (b) 46-broker networks.

have 10 attributes and to produce each entry of a publication record we encoded a message in a Bloom filter of size 256 bits. Thus, a message with a publication record of size 10 carries 320 bytes of extra information. We deliberately used this large number of attributes and large-size Bloom filters to examine the negative effects of the recovery scheme in a rather extreme case. In reality though, where messages usually have smaller number of attributes, Bloom filters of size 128 or 64 would suffice and cause less network overhead.

Figure 8 illustrates the aggregate rate of request and repair messages during the experiment as well as the aggregate rate of publications to be used as a reference measure. Ideally, for each lost message there must be only one request and one repair message. However, in many cases request and repair messages are also lost, which is indeed the reason why in Figure 8 the number of requests is more than the number of repairs. Interestingly, we observed that in both networks the multicast suppression mechanism built in SRM works favorably well. More specifically, in the 12-broker network more than 90% of the request and 80% of the repair messages were not duplicated while in the 46-broker network these values were 80% and 70%, respectively. This larger duplicate number in the 46-broker network is due to the network's greater diameter, which is twice the diameter of the 12-broker network. A higher diameter has a slight affect on SRM's multicast suppression mechanism, but more importantly leads to more frequent losses of request/repair messages.

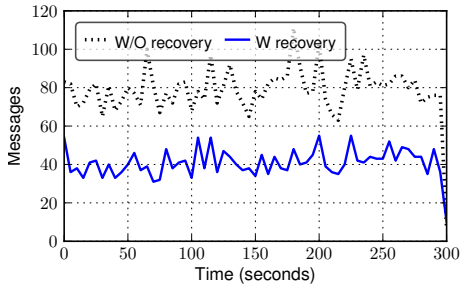
In effect, the overall and user-visible overhead is the change in the delivery delay of the messages when the recovery protocol is active and causes extra network traffic. Figure 9 shows the end-to-end delivery delay for cumulative distribution of messages with and without the recovery scheme. As the diagram suggests, the recovery protocol shifts the plot of end-to-end delay without recovery to the right, which implies a constant increase in the end-to-end delivery delay of all messages. Nevertheless, given the minimum and maximum values of end-to-end delay without recovery, an increase of 25 milliseconds in delay is not prohibitively large, since the dominant network traffic is the ordinary publication traffic and so, request/repair messages do not cause tangible impact on the overall network performance.

4.4 Adaptive Cache

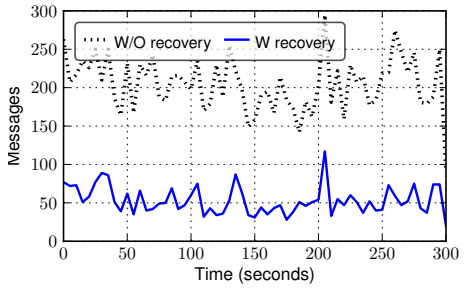
We now examine the performance of the adaptive message cache by measuring the hit rate and the size of message cache in the network. A cache hit occurs when a request for a message is sent out and at least one of the nodes that received the request, is able to provide a repair. Thus, considering the whole network as a single cache, we define hit ratio as the ratio between total number of cache hits to the total number of requests.

In our experiment we assigned values of 5 and 300 to K_{min} and K_{max}, respectively, with an initial cache size of K_{min}. The the target hit ratio H (see Algorithm 1) was set to 1. Figure 10 plots the changes in hit ratio during the experiment. In both networks, the hit ratio grows rapidly in the first few seconds of the experiment and then does not exhibit large changes during the rest of the experiment despite the continuous oscillation of the publication rates.

The effectiveness of the adaptive cache is further demonstrated by Figure 11, which shows the minimum, mean, and maximum cache size among all the network nodes during the experiment. Despite the high hit ratio, the amount of mem-

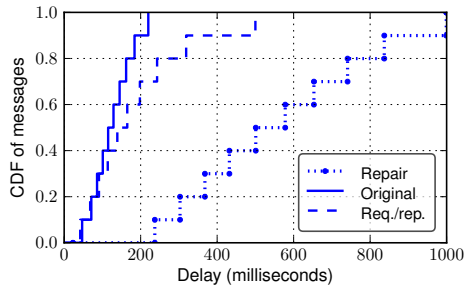


(a)

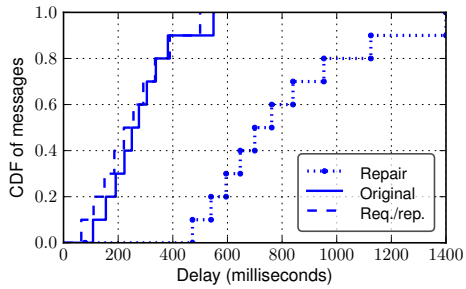


(b)

Figure 6: (a) Changes in the aggregate rate of false negatives (message loss) with and without the recovery protocol, for (a) 12-broker and (b) 46-broker networks.



(a)



(b)

Figure 7: Cumulative distribution of the end-to-end delay for original and repair messages and request/repair delay for (a) 12-broker and (b) 46-broker networks.

ory used for caching was a small. This is because most of the nodes keep their cache size to a minimum (as evidenced by small value of the mean cache size). In turn, this is because the adaptive cache mechanism causes the nodes that do not actively participate in providing repair messages to reduce their cache size. On the other hand, nodes that are actively providing repairs adjust their cache size to accommodate changes in publication rate and improve hit ratio.

5. RELATED WORK

As we mentioned in Section 1, prominent implementations of publish/subscribe systems have taken two different approaches concerning reliable delivery. Systems in the first category strive to ensure that all subscribers receive all published messages that match their subscriptions [1, 2, 16, 8, 11]. Guaranteed delivery and service availability are provided by replication of brokers and logging of messages onto durable storage. Moreover, reception of messages by the intermediate brokers and the final recipients are acknowledged at each hop. These are known as *store and forward* systems. The main problem of the store and forward design is that it induces high delivery delays. Additionally, when the publication rate is high, logging messages onto disk might contribute to congestion.

In the second category there are *best effort* systems [10, 3, 6, 17]. These systems treat messages as datagrams in a network, and therefore do not store messages onto disk, nor they require any acknowledgment when transmitting messages from broker to broker or from a broker to a client. The advantage of the best-effort design is its simplicity and therefore its ability to scale in terms of throughput.

The idea of reliable message delivery in a publish/subscribe system without support from the broker network has been previously proposed in the work by Ostrowski and Birman[15]. The authors briefly mention the possibility of combining reliable multicast protocols like SRM and RMTP with an unreliable publish/subscribe protocol. However, the proposed general approach is limited to topic-based publish/subscribe systems. Even so, this work lacks a precise and concrete plan on how it can be implemented.

Costa et al. [5] and Esposito et al. [7] propose using epidemic and peer-to-peer techniques to provide reliability in publish/subscribe networks. Similar to our protocol, the work by Costa et al. [5] provides reliability in probabilistic terms. We believe that the work presented here is more generic and easier to implement and deploy.

6. FINAL REMARKS

The goal of this paper is to enhance the reliability of best-effort publish/subscribe systems with a minimum reduction in the high throughput and end-to-end delivery delay that such systems offer. Towards this end, we develop an end-to-end method that effectively considers the broker network as a black box. In essence, our solution consists of two components. The first is a message-loss detection mechanism; the second is a routing scheme to deliver request messages to nodes that volunteer to provide repairs. Both mechanisms are built on top of a standard publish/subscribe API, and therefore neither requires any modification to the broker network. Also, both mechanisms are based on a synthetic publication record attached to messages and in turn based on a space-efficient encoding of messages and subscriptions into

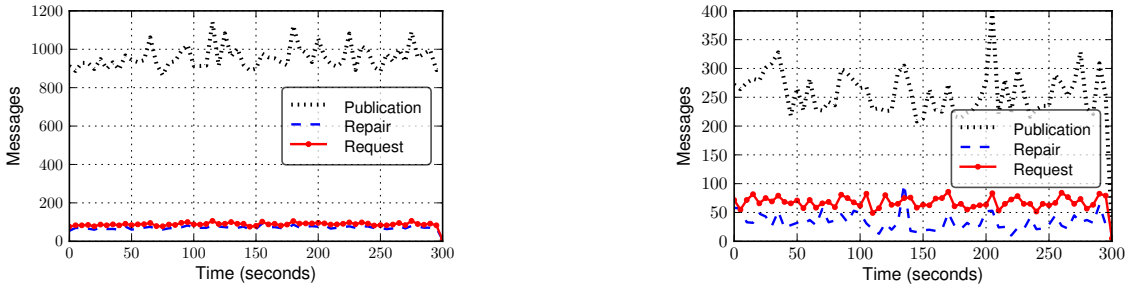


Figure 8: (a) Aggregate publication rate, request, and repair messages during the experiment for (a) 12-broker and (b) 46-broker networks.

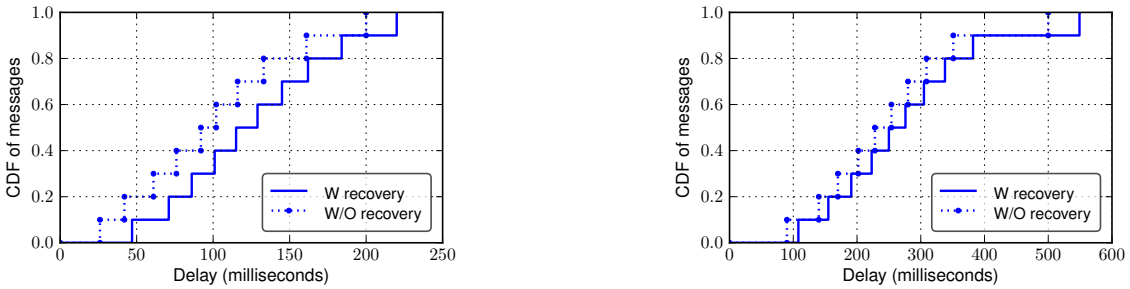


Figure 9: Delivery delay with and without the recovery protocol in (a) 12-broker and (b) 46-broker networks.

Boom filters. Through experimental evaluation, we show that this end-to-end reliability method is effective in recovering lost messages even in the presence of highly unstable network conditions and low link reliability.

We believe that best-effort content-based systems, thanks to their good performance and relatively simple architecture, have great potentials to emerge as a general-purpose communication paradigm integrated into the network fabric. Similar to IP networks, we believe that higher level guarantees, like message ordering and reliable delivery, ought to be built atop the basic service offered by the broker network, by involving the clients and potentially other dedicated network components in the process. Indeed, this work is a part of a project whose aim is to devise a *transport protocol* for best-effort content-based networking.

Acknowledgments

This work was supported in part by the Swiss National Science Foundation under grant number 200020-120188/1.

7. REFERENCES

[1] S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 7–16, Washington, DC, USA, 2002. IEEE Computer Society.

[2] S. Bhola, Y. Zhao, and J. Auerbach. Scalably supporting durable subscriptions in a publish/subscribe system. In *In proceeding of the international conference on dependable systems and networks (DSN 2003)*, pages 57–66, 2003.

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

[4] A. Carzaniga, G. Toffetti Carughi, C. Hall, and A. L. Wolf. Practical high-throughput content-based routing using unicast state and probabilistic encodings. Technical Report 2009/06, Faculty of Informatics, University of Lugano, Aug. 2009.

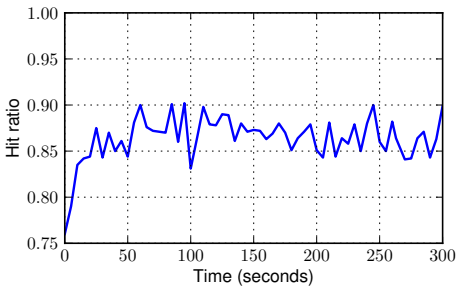
[5] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[6] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

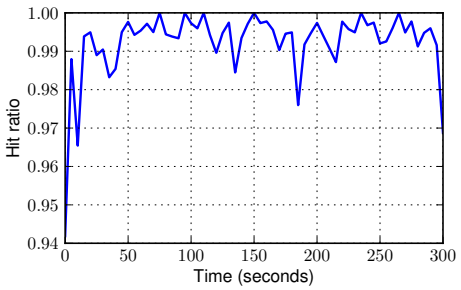
[7] C. Esposito, D. Cotroneo, and A. Gokhale. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 16:1–16:12, New York, NY, USA, 2009. ACM.

[8] E. Fidler, H. A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. In *In 8th International Conference on Feature Interactions in Telecommunications and Software Systems*, pages 12–30, 2005.

[9] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for

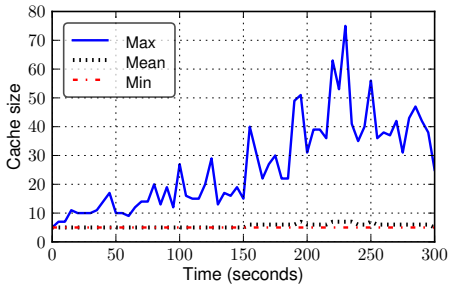


(a)

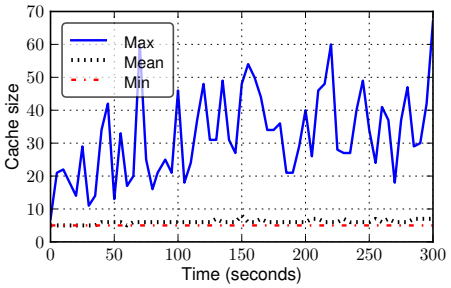


(b)

Figure 10: Changes in the cache hit ratio in (a) 12-broker and (b) 46-broker networks.



(a)



(b)

Figure 11: Changes in the minimum, mean, and maximum cache size of all nodes in (a) 12-broker and (b) 46-broker networks

light-weight sessions and application level framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803, 1997.

[10] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line Speed Publish/Subscribe Inter-networking. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 195–206, New York, NY, USA, 2009. ACM.

[11] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *SRDS '09: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 41–50, Washington, DC, USA, 2009. IEEE Computer Society.

[12] J. C. Lin and J. L. Sanjoy. Rmtp: A reliable multicast transport protocol. In *IEEE Journal on Selected Areas in Communications*, pages 1414–1424, 1996.

[13] M. Mitzenmacher. Compressed bloom filters. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, PODC '01, pages 144–150, New York, NY, USA, 2001. ACM.

[14] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *IFIP/ACM International Conference on Distributed systems platforms*, Middleware '00, pages 185–207, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[15] K. Ostrowski and K. Birman. Extensible web services architecture for notification in large-scale systems. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 383–392, Washington, DC, USA, 2006. IEEE Computer Society.

[16] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society.

[17] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using xml. *SIGOPS Oper. Syst. Rev.*, 35(5):160–173, 2001.

[18] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. Pgm reliable transport protocol specification, 2001.