Self-stabilizing Routing in Publish-Subscribe Systems

Zhenhui Shen and Srikanta Tirthapura
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011
{zhshen,snt}@iastate.edu

Abstract

Publish-subscribe systems route events to interested subscribers through a distributed network of routing tables. We present a self-stabilizing algorithm that maintains these routing tables in a consistent distributed state, and recovers from faults in the network. Neighboring message routers periodically exchange their routing table state, and take corrective actions if (and only when) necessary. We formally prove that the resulting algorithm brings the system back to a legal global state if it starts out in a faulty state.

Further, we show how to reduce the size of the periodic message exchanges by exchanging "sketches" of the routing tables which are much smaller than the routing tables themselves. We present a message size/accuracy tradeoff of using these sketches, which are based on Bloom filters. We have simulated our algorithm, and present our results of studying the important special case of a transient edge failure in greater detail.

Due to space constraints, we will give an overview of our results in this paper, and we refer to our technical report for further details.

1 Introduction

Publish-subscribe systems provide a loosely coupled middleware to distributed applications. In such systems, messages from senders (or publishers) are routed to receivers (or subscribers) based on their content, rather than on a fixed destination address. Receivers in turn express their interest in receiving a certain class of messages by submitting *subscriptions*, which are predicates on the message content. If the publish-subscribe systems are to be scalable to large networks, then the event routing must be performed in a in a distributed fashion. Many systems such as [CRW01, ASS+99, Muh02] show how to construct a distributed network of routing tables to accomplish this task.

In this work, we are concerned with the fault tolerance of the routing tables. Faults are inevitable in such a distributed system built over a wide-area network. For example, a loss of a subscribe/unsubscribe message could lead the system to a state where the routing tables are inconsistent with each other. A more serious fault might be the corruption of a routing table, or even a router crash. Any fault detection/recovery mechanism must be *decentralized*, and should have a low overhead of detection/correction.

We propose a fault detection and recovery mechanism for distributed publish-subscribe networks based on *self-stabilization*. Self-stabilization is a general fault-tolerance technique, introduced by Dijkstra[Dij74]. Informally, a system is self-stabilizing if, starting from an arbitrary initial global state (perhaps faulty), it quickly reaches a "legal" global state. The advantage of self-stabilization is that it addresses all faults through an uniform mechanism. Rather than enumerate all possible faults that could occur in the network and take corrective actions for each of them, we present one set of rules and actions which handles all possible faults in the router state. When these actions are consistently followed, the system will be quickly restored to a "legal" state after router faults.

The correctness of the publish-subscribe system is a global system property. No single node in the system will be able to say whether or not the system is in a correct state. However, we show that the predicate specifying that the global system state is correct can be written as the conjunction of many local predicates, each of which can be checked in a decentralized way using local actions. This property will help us in designing a local algorithm for the fault-tolerance. We prove that our algorithm leads the system to a legal state, and the time taken is proportional to the diameter of the graph.

If implemented in a straightforward way, self-stabilization presents a large message overhead. At first glance, it seems necessary for the nodes to pass their

complete routing tables to their neighbors. The routing tables are typically large (a few thousands of subscriptions), and since this exchange has to take place periodically, this could lead to significant message traffic. To reduce this overhead of checking, we propose to communicate small space "sketches" of the routing tables, instead of the whole routing tables themselves. Our sketches are based on Bloom Filters[Blo70]. These can be used to detect inconsistencies between routing tables very accurately, and yield space improvements of two orders of magnitude.

We formally analyze the space/accuracy tradeoff of checking using our sketches. Our analysis of the false positive probability of using Bloom filters to check set equality is novel. All previous analyses of Bloom filters focused on the false positive probability for checking the set membership.

In summary, our contributions are as follows:
(1)We present a local, self-stabilizing algorithm for adding fault-tolerance and recovery to publish-subscribe systems. We prove that irrespective of what global state the system begins in, it will reach a legal global state, and quantify the time (number of parallel steps) taken to do so.
(2)We present a way to reduce the overhead of fault-tolerance by communicating "sketches" of the routing tables as opposed to the routing tables themselves. We analyze the space-accuracy tradeoff provided by these sketches.
(3)We have simulated the self-stabilization algorithm, and these simulations reinforce our theoretical analysis. We found that though the algorithm is correct in all cases, it needs further tuning, and may not lead to the most efficient recovery in every case. We have analyzed an important special case, that of a transient edge failure, and this demonstrates how the choice of a timeout for initializing the fault recovery can be very important for the overall performance.

Related Work: Existing work study the fault tolerance problem from the perspective of topological changes, such as tree partitioning, grafting/pruning of branches. Siena[CRW01] briefly suggests using system primitives, like subscribe/unsubscribe, to allow subtrees to be merged or to be split. In [PCL03], the authors argue that subscribing and unsubscribing should be treated asymmetrically and propose an optimization over the Siena approach. In another paper [CFLP04], they optimize the special case of a link failure and a link formation occurring in parallel and push the reconfiguration overhead for this special case to a minimum.

On the other hand, some systems like Scribe[RKCD01] and Hermes [PB02] are built on top of a distributed hash table, which take advantage of the peer-to-peer routing substrate to achieve fault-tolerance.

All current work handles a limited range of faults such as message losses and link formation/failure; they do not provide a comprehensive mechanism like we do.

2 Model

We deal with a publish-subscribe network whose nodes are organized into a single spanning tree. We assume that all communication links are FIFO. Each node holds a routing table. Many data structures have been proposed for fast matching and forwarding of events[CW03, ASS$^+$99]; we will not be concerned with the exact form of this data structure. For our purposes, the routing table is a set of tuples of the form (sub, R) where sub is a subscription, and R is a set of neighboring nodes from which the subscription was received. If any event arrives that matches sub, then it is forwarded to all nodes in R, except for the node from which the event arrived.

Self-stabilizing algorithms can be built in a modular fashion[GM91]. Our algorithm stabilizes the state of the routing tables on a tree-based publish-subscribe system. This can be layered on top of another algorithm which stabilizes the spanning tree itself.

3 Self-stabilization Algorithm

The self-stabilization algorithm is concerned with the consistency between the routing tables held by the nodes. Neighboring nodes periodically exchange the states of their routing tables. A node takes corrective actions if its routing information is inconsistent with some neighbor. Each node makes local corrections independently and asynchronously. Through a sequence of local corrections, we restore the consistency among the distributed routing tables.

3.1 Local Legality Implies Global Legality

The definitions of legal local/global states lie at the core of our algorithm. Before defining them, we first introduce some notations and concepts.

An undirected edge connecting nodes a and b is denoted by $\{a, b\}$. It is composed of two directed edges, denoted by (a, b) and (b, a). For node v, let $N(v)$ denote the neighbor set of v. If we remove edge $\{a, b\}$ from the tree T, the whole tree is divided into two parts. The subtree rooted at a is denoted by T_a^b and the subtree rooted at b is denoted by T_b^a.

For directed edge (a, b), the *filter* $F_{a\to b}$ is the union of all subscriptions registered at node a, which sends matching events to node b; in other words, it is the set of all subscriptions that a has so far received from b. The

set S_b^a is the union of all subscriptions that are generated by nodes in the subtree T_b^a.

Definition 1 *A system is* quiescent *if there are no subscribe/unsubscribe messages in transit.*

Suppose the neighbor set of b, $N(b) = \{n_1, n_2, \ldots n_k, a\}$. Let $X_{b\to a}$ denote the subscriptions in transit from b to a and $Y_{b\to a}$ denote the unsubscriptions in transit from b to a. Let L_b denote the local subscriptions issued by node b. We first define what it means for an edge to be locally legal.

Definition 2 *The* directed edge (a, b) *is* locally legal, *iff* $F_{a\to b} \cup X_{b\to a} - Y_{b\to a} = \cup_{i=1}^{k} F_{b\to n_i} \cup L_b$.

Definition 3 *The* undirected edge $\{a, b\}$ *is* locally legal, *iff both* (a, b) *and* (b, a) *are locally legal.*

We now define what it means for an edge to be globally legal.

Definition 4 *If the system is quiescent, then edge* $\{a, b\}$ *is globally legal, iff* $F_{a\to b} = S_b^a$ *and* $F_{b\to a} = S_a^b$.

We assume that every received subscription will be forwarded unless it is a duplicate. Definition 4 holds under this assumption. Regarding any other optimization in subscription forwarding, the computation of S and F can be adapted to keep Definition 4 valid.

Definition 5 *A publish-subscribe system is in a legal state if one of the two conditions holds:*
(1)it is quiescent and all edges are globally legal or
(2)it can be reached from a legal quiescent state by a finite sequence of transitions.

The global legality of edges is hard to check directly, since it is a predicate that involves the state of the whole system. However, the local legality of an edge can be (more) easily checked. We now state a theorem which shows that the predicate defining the global legality of the system can be written as the conjunction of many local predicates, one for each edge.

Theorem 1 *The publish-subscribe system is in a legal state iff every edge is locally legal.*

This proof and the following proofs are all omitted due to space constraints. We refer interested readers to our technical report[ST04].

3.2 The Edge Stabilization Algorithm

Given the above theorem, we only need to stabilize each directed edge into a legal state, and the system will reach a globally legal state. It is easy to set a (faulty) directed edge to a legal local state using appropriate subscriptions/unsubscriptions. However, stabilizing a faulty edge might "disturb" a neighboring edge, and cause it to move from a legal to an illegal state, so that the global state is still illegal. Informally speaking, such "disturbances" can flow only along a simple path in the tree, and have to eventually stop at a leaf. Thus, eventually the system will reach a globally legal state.

A timer is assigned to each directed edge in the network, and the (directed) edge stabilization procedure is initiated upon expiry of the timer. The period of the timer controls the frequency of stabilization, and hence the message overhead (more discussion of the timer appears in Section 5). The source node of a directed edge is responsible for the stabilization. A single round of the procedure consists of two phases: an observe phase followed by a correction phase. We describe the algorithm for directed edge (a, b). All the directed edges are being stabilized in parallel in this manner.

Variables:

(1) $N(a) = \{n_1, n_2, \ldots, n_k, b\}$
(2) $S(a) = \cup_{i=1}^{k} F_{a\to n_i} \cup L_a$
(3) $C_1 = S(a) - F_{b\to a}$
(4) $C_2 = F_{b\to a} - S(a)$

Actions at Node a

Event: timeout at t_1 (observe phase)

(1) compute $S(a)$ at time t_1
(2) send an "observer" to b
(3) reset the timer for the next round

Event: get the reply from b (correction phase)

(1) (comment: b's reply is $F_{b\to a}$)
(2) if ($S(a) = F_{b\to a}$)
(3) return;
(4) else
(5) compute C_1, C_2
(6) send C_1, C_2 to b

Actions at Node b

Event: receive an "observer" from a (observe phase)

(1) compute $F_{b\to a}$
(2) return $F_{b\to a}$ to a

Event: receive C_1 from a (correction phase)

(1) subscribe to each record contained in C_1

Event: receive C_2 from a (correction phase)

(1) unsubscribe to each record contained in C_2

It is important to note that the correction phase at node b is initiated only if edge (a, b), and hence the whole system was not in a legal state. Thus, if the system is in a legal state, then the self-stabilization will not add any additional subscriptions/unsubscriptions to the system.

Theorem 2 *Starting from any initial state (perhaps faulty), if (1)no further faults occur and (2)every directed edge in the tree executed the above stabilization process, then the system will reach a legal global state.*

4 Reducing the Message Overhead

An important component of the local stabilization algorithm is the checking of the equality between the two tables $S(a)$ and $F_{b\to a}$. One way to do this is to send the entire table $F_{b\to a}$ across from node b to node a, but this would result in a large message overhead for the following reasons:
(1)The objects being sent across and compared are large sets of subscriptions. These routing tables might contain thousands of subscriptions, and if each subscription takes a few tens of bytes, then these messages would be of the order of a few hundred kilobytes or more. In addition, comparing these large sets would be significant computational overhead. (2)Self-stabilization is a periodic system behavior, which further exacerbates the above problem.

Our approach to reducing this overhead is as follows. Instead of sending the entire routing tables across, we send only a sketch of the table to the neighboring node. This sketch takes much smaller space than the table itself. These sketches are compared at the neighboring node, and if they are found to be inconsistent, then there must be a fault, and now a full comparison of the routing tables is initiated to recover from the fault. For the common case when the routing tables are consistent with each other, the whole routing table will not have to be sent across, leading to a very efficient checking process.

However, these sketches are inherently lossy. There is some probability that the routing tables are actually inconsistent, but the sketches do not reveal it. We are able to quantify this probability of a false positive, and we show that a sketch which uses only a few bits per subscription is able to provide a false positive probability of less than 10^{-3}.

We note that this is the first work, to our knowledge which suggests compression of messages in this way in a self-stabilizing algorithm. We summarize the desirable properties of a sketch:

(1) The size of the sketch should be small compared to the original routing table size
(2) It should be able to detect inconsistencies with high probability, and with low computational overhead
(3) The cost of maintenance should be low, i.e. every time a subscription or unsubscription is received, we should be able to update the sketch quickly.
(4) Since we need to compute the union of routing tables while checking, we need to be able to (quickly) compute the sketch of the union of sets given the sketches of the sets.

We considered various techniques for maintaining these sketches, including hashing and checksums. Our final solution, which satisfies all the above properties, is based on *Bloom filters*. Below we analyze the various tradeoffs associated with using a Bloom filter for the purpose of testing equality between sets.

4.1 Bloom Filter for Testing Set Equality

A Bloom filter is a compact representation of a set to support membership queries. It was invented by Burton Bloom in 1970 [Blo70]. In [FCAB00], it is shown how to derive the probability of a false positive for a membership query in using the Bloom filter.

In self-stabilization, we do not use a Bloom filter to test for set membership, but to compare if two sets are equal. More precisely, we want to check if the union of a few sets ($S(a) = \cup_{i=1}^{k} F_{a\to n_i} \cup L_a$) equals another set ($F_{b\to a}$). This calls for a new analysis of the tradeoffs between the false positives and the parameters of the Bloom filter. We now sketch our analysis of the false positive probability for the context of set equality, and graph the resulting tradeoffs obtained.

Let B_S denote the Bloom filter of set S. Let m denote the size (in bits) of the Bloom filter, and k the number of hash functions. We want to compute the probability that B_A and B_B are equal, though A and B are unequal. Let $\alpha = e^{-\frac{k|B|}{m}}$ and $\beta = e^{-\frac{k|A|}{m}}$. Let p denote the false positive probability, i.e. the probability that $B_A = B_B$ though $A \neq B$.

Theorem 3

$$p \leq \min\{p_A, p_B\}$$

where

$$p_A < \alpha \cdot \prod_{i=2}^{k\cdot|A-B|} \left(\frac{i-1}{m} + \left(1 - \frac{i-1}{m}\right)\cdot\alpha\right) \quad (1)$$

and

$$p_B < \beta \cdot \prod_{j=2}^{k\cdot|B-A|} \left(\frac{j-1}{m} + \left(1 - \frac{j-1}{m}\right)\cdot\beta\right) \quad (2)$$

We now sketch the false positive probabilities for various values of $k, m/n$ and $|A-B|$ in Figure 1. Clearly, the probability decreases (leading to a more accurate test for equality) when the difference between the sets is getting large. As k increases, the computation overhead for maintaining the Bloom filter increases. The parameter m/n is the number of bits used per element. As it increases, the space overhead also increases, but the false positive probability decreases. Thus, by using 4 bits per element and 2 hash functions, the false positive rate for sets differing by 3 elements is only about 0.001, and this can be further decreased by increasing k or m/n.

5 Simulations

The self-stabilization procedure is a periodic system behavior, whose period is controlled by a

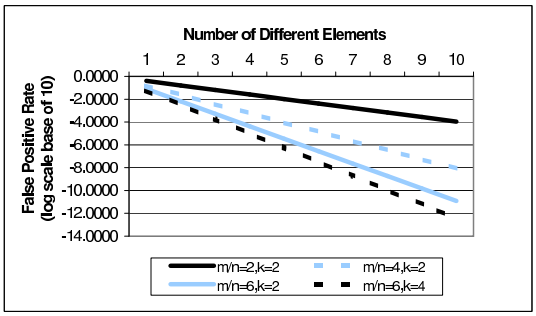

Figure 1. The Probability of a False Positive Under Various m/n and k Combinations.

The y-axis is the error probability, in a log-scale. The x-axis is the size of the difference, $|A - B|$.

timer(Section 3.2). Since this period is fixed, the procedure is executed at regular intervals. However, this regular behavior does not always yield the best performance, though it is always correct. One special case is that of a transient edge failure, which we consider now.

In face of an edge failure, some subscriptions become obsolete, since they were generated by nodes in a partition which is now unreachable. These subscriptions have to be (usually) removed, else the endpoints of the broken edge will continue to receive useless events. But, if the broken edge comes back up quickly, then these obsolete subscriptions become useful again. In such a case, it is better to delay the unsubscribing after the link failure.

Since the setting of the timer is independent from the event of link failure, there is no synchronization between the time of the link failure and the activation time of the next round of stabilization. As a result, the next round of stabilization might kick in soon after the link failure, which may not be ideal for the performance.

A better approach is to reset the timer based on the network condition. This leads to our *adaptive strategy*, which delays the stabilization timeout until the node receives a sufficient number of unwanted events, rather than depending on a fixed timeout.

We conclude by presenting some experimental results which compare various timeout strategies. Our performance metric is the *reconfiguration overhead*, which is defined to be the total number of hops traversed by both unsubscribe/re-subscribe messages and unwanted events under a single link failure. We study the following three strategies:

(1)The "strawman algorithm" (the name borrowed from [PCL03]). Upon a link failure, it resets the timer and initiates the stabilization without delay.

(2)The "static algorithm" with no change to the preset timer.

(3)Our "adaptive algorithm". Upon a link failure, the timer is reset, but the stabilization is activated once a sufficient number of unwanted events are received, or when the link is re-formed.

5.1 Simulation Setting

In our simulations, both an event and a subscription are chosen to be 3-character random strings. An event matches a subscription if the two strings are identical. The topology is a single spanning tree, consisting of 100 nodes.Each simulation scenario is uniquely identified by a combination of the following parameters:

Publish Rate: The publish rate regulates the system load. We simulate two scenarios: a *light system load* using a publish interval of 5.0 seconds and a *heavy system load* using a publish interval of 0.1 seconds.

Subscribe Rate: The subscribe rate controls the density of the subscriptions. We set the subscribe interval to be 2.0 seconds. In addition, each router can subscribe to at most 20 event patterns.

Fixed Delay: This is the timeout used by the "static" algorithm. We choose two timeout values: a longer one of 10 seconds and a shorter one to be 3 seconds.

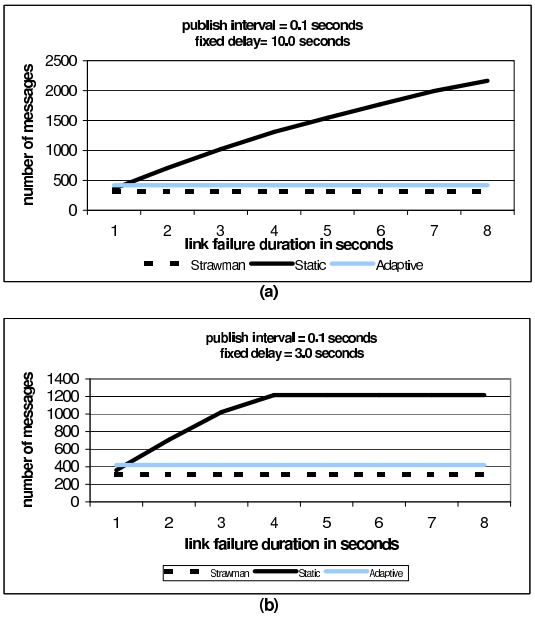

Figure 2. Reconfiguration Overhead Under Heavy System Load

Under a heavy load (Figure 2), the endpoints expect to receive more unwanted events. It's ideal to unsubscribe early to limit the increasing cost of unwanted events. In Figure 2, the strawman curve has the lowest

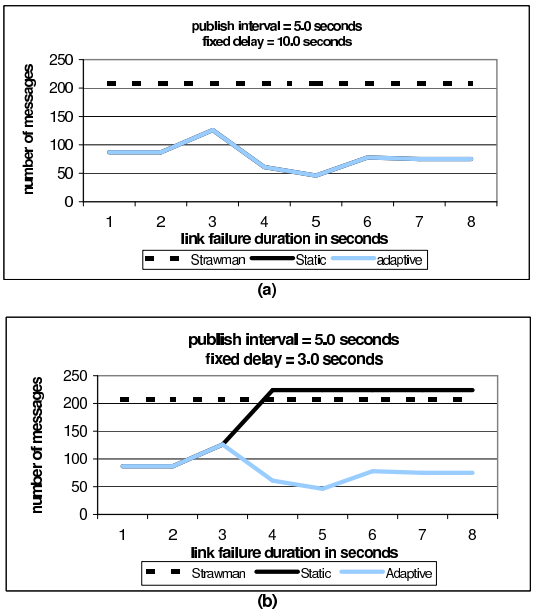

Figure 3. Reconfiguration Overhead Under Light System Load

overhead, as it unsubscribes without delay. The adaptive curve has a slight increase, for it delays unsubscribing a little bit. The static algorithm has a poor performance, but a shorter timeout brings down the overhead by 50%.

Under a light load (Figure 3), the cost of unwanted events is negligible due to the rare occurences of unwanted events. It is better to delay unsubscribing. Therefore any static algorithm with large timeout performs well under this condition. Meanwhile the adaptive algorithm also yields the same amount of cost. This time both the strawman and the static strategy with short timeout value generate huge overhead. As a comparison, our adaptive algorithm saves two thirds of the cost of the strawman approach.

In summary, neither the *static* nor the *strawman* do well for both the lightly loaded and heavily loaded cases. However, the *adaptive* algorithm for triggering the reconfiguration shows a good (though not optimal) performance in both cases.

References

[ASS+99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 1999.

[Blo70] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7), Jul 1970.

[CFLP04] Gianpaolo Cugola, David Frey, Amy L.Murphy, and Gian Pietro Picco. Minimizing the reconfiguration overhead in content based publish subscribe. In *Proceedings of the 19th ACM Symposium on Applied Computing (SAC04)*, Mar 2004.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[CW03] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, 2003.

[Dij74] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[FCAB00] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[GM91] M. G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40:448–458, 1991.

[Muh02] G. Muhl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[PB02] Peter Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

[PCL03] Gian Pietro Picco, Gianpaolo Cugola, and Amy L.Murphy. Efficient content-based event dispatching in the presence of topological reconfiguration. In *Proceedings of the 23rd International Conference on Distributed Computing Systems(ICDCS'03)*, pages 234–244, 2003.

[RKCD01] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of the Third International COST264 Workshop (NGC 2001)*, nov 2001.

[ST04] Zhenhui Shen and Srikanta Tirthapura. Self-stabilizing routing in publish-subscribe systems. Technical Report TR-2004-04-4, Iowa State University, Electrical and Computer Engineering, 2004. http://archives.ece.iastate.edu/archive/00000033/.