Towards Byzantine Fault Tolerant Publish/Subscribe: A State Machine Approach

Leander Jehl and Hein Meling
Department of Electrical Engineering and Computer Science
University of Stavanger, Norway
leander.jehl@uis.no

ABSTRACT

More than a decade of research has gone into techniques aimed at tolerating arbitrary failures in client/server interaction, using consensus based replication. These works made Byzantine fault tolerance possible [5], competitive [18], robust [7], and feasible to apply [6]. In this paper we establish a connection between the pub/sub interaction model and consensus based replication protocols, that makes the above results applicable to the design of large scale event-based middleware. We propose a Byzantine fault tolerant pub/sub system, on a tree-based overlay, tolerating a configurable number of failures in any part of the system, with minimal divergence from traditional pub/sub specifications and forwarding schemes.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Fault Tolerance*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Reliability

Keywords

Publish subscribe, Byzantine fault tolerance, State machine replication

1. INTRODUCTION

There is a growing trend towards event-based interaction between system components [1, 4]. A primary reason for this trend is that more and more workflows are naturally event-driven, e.g. a user tweeting to his followers or bidding in an auction. Event-based interaction allows a *subscriber* to indicate its interests by sending a subscription to the *service*. Then, upon receiving a publication from a *publisher*, the service can *match and forward* the publication to interested subscribers. This is referred to as the pub/sub model.

HotDep November 03 - 06 2013, Farmington, PA, USA

We consider distributed content-based pub/sub, where subscribers connect to an overlay network of *brokers* representing the service, and matching is based on one or more attributes of a publication. We refer to publishers, subscribers, and brokers as *agents*. The salient features of the pub/sub model are that publishers and subscribers are decoupled [11], interacting only through the service, and that subscription based filtering is used to drop unwanted publications.

In this paper we propose a Byzantine Fault Tolerant (BFT) pub/sub architecture built on a tree-based overlay network. Designing such a system is motivated by future application needs, as we envision that BFT pub/sub can play a role in preventing cheaters in multiplayer online games [2] or fraud in online trading/auctions, or to increase resilience to attacks for an intrusion detection system [12].

By implementing BFT, a pub/sub system can enforce that all publications originate from authenticated publishers, that can be held accountable for their publications, and that publications are fairly distributed to all subscribers, preventing faulty publishers from sending exclusive information to certain subscribers.

Previous works [14, 16] on pub/sub have applied ad hoc techniques for handling failures in a relaxed failure model, such as assuming that publishers are non-Byzantine [16]. We argue that this assumption is unreasonable, since publishers might behave Byzantine for the same reasons as other agents in pub/sub.

Contributions: (1) Formalizing BFT pub/sub in a state machine model akin to the Paxos agents: proposers, acceptors, and learners [17], making it easier to reason about its properties. (2) Our solution is safe against Byzantine publishers. (3) It can update subscriptions concurrently with publication forwarding, which is not covered by [16]. (4) Our approach uses symmetric encryption which outperforms classical signatures used in [16] by several orders or magnitude.

Prerequisite: Our objective is to derive a BFT pub/sub system, starting from non-fault-tolerant pub/sub. To tolerate failures of the pub/sub agents, they are replicated using established protocols for state machine replication (SMR), that lend themselves to formal reasoning. We assume an initial pub/sub system in which the agents are organized into a tree overlay [4]. This is a common and practical assumption for many systems; we leave it for future work to consider more general overlays. Moreover, there exist several extensions [24, 23, 9] for pub/sub on a tree overlay. With this overlay as a basis, we hope that these extensions can also be adapted to our system.

In our solution, we wish to retain the original tree overlay as much as possible, adding nodes and links only when necessary. In particular, the normal operation forwarding paths should be similar to the original overlay.

2. SPECIFICATIONS AND DESIGN PRINCIPLES

We first give a pub/sub specification, and introduce fault tolerance. We then reduce the problem to a single publisher, and make the connection to consensus.

What distinguishes pub/sub forwarding from multicast is its ability to filter publications based on subscriptions. That is, we can selectively forward only those publications that have matching subscribers. In content-based pub/sub, this matching may operate on multiple attributes of a publication. We can formalize the safety and liveness conditions of a pub/sub system with the following properties [20]:

Relevance If a subscriber S receives a publication p, then S issued a subscription s, matching p.

Authentication If a subscriber S receives a publication p, then p was published by some publisher P.

Uniqueness A subscriber S receives no publication twice.

Liveness If a subscriber S issues a subscription s and never changes it, then S receives all publications matching s, except a finite number[1].

2.1 Fault Tolerant Specification

For a fault tolerant pub/sub system, the above specification is insufficient. Specifically, we must make certain assumptions about the publishers and subscribers. That is, we can think of them as clients of a fault tolerant service. Therefore, if subscribers are (crash) faulty, the system cannot guarantee that they deliver a publication. Further, a Byzantine subscriber cannot even be prevented from delivering incorrect publications. Thus, we require the above safety properties to hold for *correct subscribers*, if no more than f failures occur. We say that a publisher (subscriber) is correct if it never fails during an execution. We require Liveness to hold for correct subscribers and *correct publications*. A publication is *correct*, if it was issued by a correct publisher, or delivered by at least one correct subscriber.

To ensure Uniqueness, we assume that all publications are different. In the presence of *Byzantine publishers*, this means that two publications with the same content, but different authenticators are considered different. Finally, we note that Liveness is not restricted to correct publishers. Thus our Liveness property guarantees fairness:

Fairness: If two correct subscribers issue the same subscription s and never change it, the stream of publications they receive from a faulty publisher only differs by a finite number of publications.

2.2 Problem Reduction: Single Publisher

We note that the safety and liveness conditions hold, *iff* they hold separately for every publisher. Thus, for simplicity we restrict the above specification to a single publisher; we call this *single source pub/sub* (SPS). Given a system that implements SPS, reproducing this system for each publisher yields a correct pub/sub system. Clearly, an implementation can merge several SPS instances, sharing common structures.

We also require that an implementation of SPS delivers publications in FIFO order and that every node issues an initial subscription s_0, which is known to all correct nodes. This allows us to reuse publication dissemination to implement subscription updates, as we explain in Section 3.4. Since sequence numbers are already needed in SPS to filter duplicates and detect gaps, implementing FIFO poses little additional cost. Also providing initial subscriptions is clearly a feasible requirement.

2.3 Pub/Sub as a Consensus Problem

We now relate the pub/sub specification to a classical consensus problem, by showing that SPS can be implemented by a series of consensus instances using a single proposer.

Given a single proposer and a set of acceptors and learners, the following properties specify a *weak form* of consensus. They must hold despite f faulty acceptors.

SC1 No two correct learners learn different values.

SC2 If a correct learner learns a value, it was proposed by the proposer.

SC3 A correct learner learns at most once.

LC1 If the proposer is correct and proposes p, all correct learners eventually learn p.

An additional liveness property is necessary to obtain full consensus [18]:

LC2 If one correct learner learns p, all correct learners eventually learn p.

We note that even with a single proposer, $3f + 1$ acceptors are needed to solve consensus in presence of arbitrary failures [21]. If the proposer is always correct, while acceptors can be Byzantine, only $2f + 1$ acceptors are needed. Note that by **LC1**, progress is not required when the proposer fails. Therefore we do not need the view change protocol of classical consensus algorithms, since it mainly serves to change the proposer.

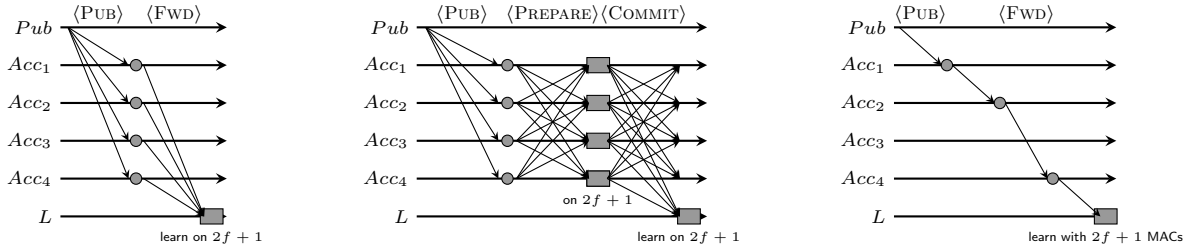
SPS can be easily implemented using full consensus. For every publication, the publisher is used as proposer in a new consensus instance, with all subscribers as learners. After learning, a subscriber delivers only publications matching its subscription. The weak form of consensus can be used for finitely many publications and SPS will still be fulfilled. However, using weak consensus for all publications does not fulfill the pub/sub specification, since fairness might be violated. This scheme can be highly inefficient, since publications are broadcasted, even if there is no interested subscriber. We will however leverage it in our design.

Note that consensus with a single proposer is equivalent to reliable broadcast [3]. We chose to use consensus above, since the roles of proposers, acceptors, and learners in consensus closely match that of publishers, brokers, and subscribers in pub/sub.

2.4 Decoupling in a Fault Tolerant Overlay

The principle of decoupling between different agents is essential to facilitate scalability in modern pub/sub systems [11]. Thus publishers and subscribers only communicate with the system through a single connection. In the

[1]In [20], **Liveness** allows missed publications during a finite time interval, rather than a finite number of publications. However, without explicit timing constrains, this formulations is equivalent.



(a) Authentication. (b) Reliable Broadcast. (c) Chain.

Figure 1: BFT-Publisher.

classical tree overlay, this decoupling also extends to brokers, requiring that brokers only know their neighbors. Clearly we cannot adopt this condition, since the failure of one broker in the tree would cause the overlay to partition. Instead we restrict the set of brokers that one agent knows of to its neighborhood brokers [14] within a bounded number of hops. Also, to implement this decoupling in our system, publishers and subscribers cannot share cryptographic keys.

3. DESIGN

In this section we describe the design of our state machine based, Byzantine fault tolerant, single source pub/sub system. We do this in several steps. First we use the relationship between pub/sub and consensus, established in Section 2.3, to design a small scale pub/sub system, called the BFT-publisher. We then design an extension, the BFT-broker, that can be used to match publications from the BFT-publisher, against subscriptions and forward them to the appropriate subscribers. This allows the system to scale up to a larger number of subscribers by taking advantage of diverse subscriptions and only forward matching publications. We then elaborate on how to combine a BFT-publisher and several BFT-brokers in a tree overlay. We first consider a system where subscribers only hold an initial subscription s_0 before startup. We then show how this can be used to implement the full specification.

3.1 The BFT-publisher

We first identify an existing solution for both the weak and full, single proposer consensus. We then combine them to a small scale SPS system, with minor adjustments to the protocol.

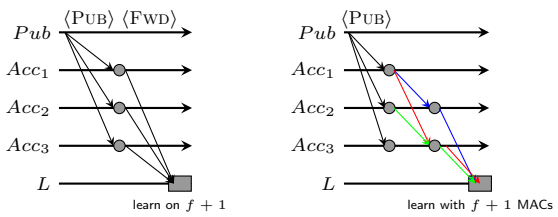
An existing solution, implementing weak consensus is the authentication stage of the UpRight library [6, 8], depicted in Figure 1(a). In this protocol, using $3f + 1$ acceptors, the proposer send a publication to all acceptors, each of which forwards it to the learners. Learners accept the publication after receiving it from $2f + 1$ acceptors, with correct message authentication codes (MACs). To see that this protocol does not implement **LC2**, consider the following example: A faulty publisher can send to only $f + 1$ correct acceptors. The faulty acceptors can selectively forward to only some learners. Thus one learner might receive $2f + 1$ forwarded publications, while others only receive $f + 1$.

Full consensus with a single proposer can be solved by Bracha's Reliable Broadcast (RelBC) algorithm [3], depicted in Figure 1(b). In this protocol, acceptors send a prepare message to all other acceptors on receiving the publication from the publisher. After receiving $2f + 1$ prepare messages or $f + 1$ commit messages, acceptors send a commit message to all other acceptors and the learners. Learners learn on $2f + 1$ commit messages. RelBC allows an acceptor to send a commit message, even if it received a different or no publication from the publisher. Therefore if some correct learner learns, all $2f + 1$ correct acceptors will send a commit message. Thus **LC2** holds. However this solution is costly, since every acceptor has to send at least $4f$ messages to authenticate a single publication. We therefore propose a hybrid version, using weak consensus in the critical path of publications, and using the full RelBC message pattern only once every n publications. This guarantees that two correct learners cannot diverge by more than $2n$ publications. Thus the publisher can use the authentication stage for every publication, but after n publications, the publisher sends a special publication containing the history of the last n weak instances through the RelBC protocol. Acceptors accept the special publication only if the history matches the preceding publications. Furthermore, acceptors only participate in a weak consensus instance, if it is no more than $2n$ instances ahead of the last RelBC instance.

While the broadcast message pattern used above works, it differs significantly from the usual forwarding in a tree overlay, and therefore conflicts with our goals. This is particularly relevant for the weak consensus protocol, which is run often and in the critical path. Taking ideas from [13] we modify the weak consensus protocol, to forward publications along a chain, as shown in Figure 1(c). Thus the publisher chooses $2f + 1$ acceptors ordered in a chain, and sends the publication (with MACs) to the first. Each acceptor forwards the publication to the next acceptor in the chain. The last acceptor forwards to the learners, which need to validate MACs from the last $f + 1$ acceptors.

If the publisher chooses only correct acceptors, all learners are guaranteed to learn. As shown in [13] this technique reduces the number of cryptographic operations, since only MACs from the last $f+1$ acceptors have to be verified. However it increases latency, especially when f is large, since $2f + 1$ communication steps are needed for learning a publication. Note that when more then f nodes are suspected to be Byzantine, the publisher can choose to run several different chains simultaneously with the same publication. BChain [10] develops tactics for choosing $2f+1$ correct nodes out of $3f + 1$.

Full BFT protocols, such as Shuttle [22] or BChain [10] can be used to also implement full consensus with a chain based protocol. However, other than RelBC, these protocols rely on view change and an eventually correct leader to guarantee **LC2**, even with a single proposer. Adjusting these protocols to guarantee **LC2** without view change has high priority for our future work.



(a) Broker. (b) Optional Paths.

Figure 2: BFT-Broker.

3.2 The BFT-broker

Clearly it is not feasible to connect an arbitrarily large number of learners directly to the BFT-publisher, especially since it does not filter any publications, simply broadcasting to all learners. We therefore introduce another replicated agent, called the BFT-broker, used to implement a fault tolerant broker. A BFT-broker forwards publications to interested subscribers and other brokers in presence of up to f Byzantine faults. We assume that the BFT-broker receives publications from a single, correct publisher. It is therefore sufficient to use $2f+1$ acceptors. We later explain how this publisher is implemented by another BFT-broker or the BFT-publisher.

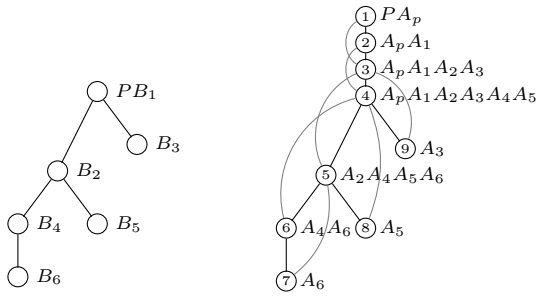
The correct publisher sends its publications to all acceptors in correct order. The acceptors match publications against subscriptions and forward the publication to all interested learners (i.e. subscribers and brokers). Acceptors also keep a counter for every learner, to add a learner-specific sequence number to the publication. This is necessary since the publisher's sequence numbers cannot be used to distinguish between filtered and lost publications. A correct learner delivers a publication only if it was received from $f+1$ acceptors carrying the next sequence number. This is depicted in Figure 2(a).

As before, having all acceptors broadcast to all learners does not comply with our goal of keeping the tree structure. We therefore change this pattern, assuming again that acceptors are ordered in a chain. Thus every acceptor sends to the next $f+1$ acceptors, or if they are fewer than $f+1$, to the remaining acceptors in the chain. The acceptors include MACs for the following acceptors and the learners. After collecting $f+1$ MACs, including its own, an acceptor can forward these to the interested learners. This is depicted in Figure 2(b). Note that if the involved acceptors are correct, it is sufficient to send only the blue, green, or red messages. This allows us to use only one of these paths during normal operation, and only use the other paths for every nth publication carrying the recent history, or if an acceptor is suspected.

Further, if some learner is placed on the same node as an acceptor, this learner can learn directly from its local acceptor, and need not receive messages from other acceptors. This is because we can assume that this acceptor and learner only fail together.

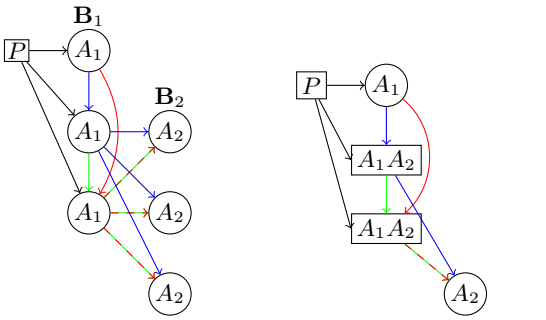
3.3 Building the Tree

We now explain how a BFT-publisher and BFT-brokers are combined to form a tree-based pub/sub system, and how to place acceptors on nodes. We assume a tree T of brokers, denoted B_i, as in Figure 3(a), with publisher P. We replace broker B_i in the tree with a BFT-broker $\mathbf{B}_i$ and the



(a) Broker Overlay T. (b) BFT Overlay.

Figure 3: Building the tree.



(a) Connected Brokers. (b) Combined Brokers.

Figure 4: Combining Brokers.

publisher with a BFT-publisher, $\mathbf{P}$. An acceptor A_j in $\mathbf{B}_j$ receives a publication from the fault-free publisher by learning it from $\mathbf{B}_i$, where B_j is a child of B_i in T. Acceptors in $\mathbf{B}_1$ learn from the BFT-publisher. Figure 4(a) shows the connections between two BFT-brokers, including the messages from Figure 2(b).

We map the acceptors A_i in $\mathbf{B}_i$ to the initial tree by placing them on the node associated with B_i and its parents, as shown in Figure 3(b). We also introduce $3f$ new nodes above B_1, which are used to place the acceptors A_p in $\mathbf{P}$. This placement lets us omit several messages from Figure 4(a), as shown in Figure 4(b). Thus all nodes except those in $\mathbf{P}$, need links only to successors at most $f+1$ hops away, and shared symmetric keys with nodes $2f+1$ hops away. Nodes in $\mathbf{P}$ need keys and links to nodes at most $3f$ hops away. Thus we achieve decoupling of the different agents, since an agent's view does not depend on the total size of the system.

The additional agents necessary for the BFT-publisher can also be located on the same nodes as other agents. However, in the example from Figure 3(b), agent 3 should not be placed with agent 5 or 9, since they share acceptors from the same broker.

Finally, in case of no failures, we can circumvent some agents in the original forwarding path. This is easy to see from Figure 4(b) where it suffices to send messages of one color, if no failures occur. This technique can positively effect both throughput and latencies of our system, as evaluated in [15].

3.4 Managing Subscriptions

In most pub/sub systems [19], subscriptions are simply propagated to the brokers, and each broker can update its subscription table as soon as it receives the subscription. However with replicated brokers, inconsistencies can occur

if two acceptors of the same broker insert a new subscription at different points in the publication stream. That is, they may disagree on which publications to match against the new subscription. Thus the learner-specific publication counter for this subscriber is no longer consistent among the correct replicas. To avoid this scenario, a new subscription s_i from subscriber S is first routed to the publisher. This can be done by simply sending upwards on the existing paths. The publisher then issues a special publication, that matches the last subscription s_{i-1} from S, and contains s_i. Brokers only apply s_i to publications issued after this special publication. FIFO ordering ensures that all correct acceptors filter the same set of publications against the new subscription. Since we consider every publisher separately, we can ignore, if different publishers or subtrees receive diverging subscriptions from a faulty subscriber. We therefore don't need to filter a subscription through a BFT-publisher, before it is forwarded in the tree.

To ensure that subscriptions, issued by a correct subscriber are eventually inserted, we must force a faulty publisher to insert the special publication. To do this we also send the subscription to the acceptors in the BFT-publisher. These stop forwarding, if the special publication is not inserted within $2n$ publications. Note that to use this scheme, without blocking a correct publisher, we must ensure that eventually, no correct acceptor in the BFT-publisher lacks more than $2n$ recent publications.

4. CONCLUSIONS AND FUTURE WORK

We have presented a design for a BFT pub/sub system, based on state machine replication. Our design is derived from the initial structure of a tree overlay, deviates only where necessary, and uses only a single forwarding path where no failures are detected. Our design incorporates the decoupling principle, since an agent's view only depends on the trees fanout and the desired level of fault tolerance f, but not on the total size of the system.

Note that, since our broker maintains a counter for every connected subscriber, it is a stateful service. Therefore our design can also be applied to more sophisticated stream processing applications, than simple subscription matching.

Several problems remain unsolved, such as how a learner should request retransmission when it detects a lost publication, and how long a publication must be stored to enable retransmission. Note that retransmission is only guaranteed to succeed, after a full consensus instance. Also, if Byzantine learners can request retransmission, they can cause additional load. Therefore, the retransmission and garbage collection frequencies need to be matched to available resources. In addition we aim to identify simplifications and unifications for combining multiple instances of SPS. We plan to address these problems in our implementation.

Although the reduction to SPS cannot be extended to more restrictive specifications, e.g. to achieve total order of publications, we believe that our state machine based approach, and the division into BFT-publishers and BFT-brokers, can be used to achieve these guarantees.

5. ACKNOWLEDGEMENT

We want to thank Allen Clement for fruitful discussions on early stages of this work.

6. REFERENCES

[1] Adya, A., Cooper, G., Myers, D., and Piatek, M. Thialfi: a client notification service for internet-scale applications. In *SOSP* (2011).

[2] Bharambe, A. R., Rao, S., and Seshan, S. Mercury: a scalable publish-subscribe system for internet games. In *NetGames* (2002).

[3] Bracha, G. Asynchronous byzantine agreement protocols. *Inf. Comput. 75*, 2 (Nov. 1987).

[4] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. Design and evaluation of a wide-area event notification service. *ACM TOCS 19*, 3 (Aug. 2001).

[5] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM TOCS 20*, 4 (Nov. 2002).

[6] Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., and Riche, T. Upright cluster services. In *SOSP* (2009).

[7] Clement, A., Wong, E., Alvisi, L., Dahlin, M., and Marchetti, M. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI* (2009).

[8] Clement, A. G. *UpRight Fault Tolerance*. PhD thesis, University of Texas at Austin, 2010.

[9] Di Nitto, E., Dubois, D. J., and Margara, A. Reconfiguration Primitives for Self-adapting Overlays in Distributed Publish-Subscribe Systems. In *SASO* (2012).

[10] Duan, S., Meling, H., Peisert, S., and Zhang, H. Bchain: Byzantine replication with high throughput and embedded reconfiguration. *Unpublished* (2013).

[11] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. The many faces of publish/subscribe. *ACM CSUR 35*, 2 (June 2003).

[12] García, J., Autrel, F., Borrell, J., Castillo, S., Cuppens, F., and Navarro, G. Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation. In *ICICS* (2004).

[13] Guerraoui, R., Knežević, N., Quéma, V., and Vukolić, M. The next 700 bft protocols. In *EuroSys* (2010).

[14] Kazemzadeh, R. S., and Jacobsen, H.-A. Reliable and highly available distributed publish/subscribe service. In *SRDS* (2009).

[15] Kazemzadeh, R. S., and Jacobsen, H.-A. Opportunistic multipath forwarding in content-based publish/subscribe overlays. In *Middleware* (2012).

[16] Kazemzadeh, R. S., and Jacobsen, H.-A. Publiyprime: Exploiting overlay neighborhoods to defeat byzantine publish/subscribe brokers.

[17] Lamport, L. Paxos made simple. *ACM SIGACT News 32*, 4 (December 2001).

[18] Martin, J.-P., and Alvisi, L. Fast byzantine consensus. *IEEE TDSC 3*, 3 (July 2006).

[19] Mühl, G., Fiege, L., and Pietzuch, P. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[20] Mühl, G., Jaeger, M. A., Herrmann, K., Weis, T., Ulbrich, A., and Fiege, L. Self-stabilizing publish/subscribe systems: algorithms and evaluation. In *Euro-Par* (2005).

[21] Pease, M., Shostak, R., and Lamport, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980).

[22] van Renesse, R., Ho, C., and Schiper, N. Byzantine chain replication. In *OPODIS* (2012).

[23] Zhang, K., Muthusamy, V., and Jacobsen, H.-A. Total order in content-based publish/subscribe systems. In *ICDCS* (2012).

[24] Zhang, K., Sadoghi, M., Muthusamy, V., and Jacobsen, H.-A. Distributed ranked data dissemination in social networks. In *ICDCS* (2013).