Anonymous Publish-Subscribe Systems

Binh Vo(✉) and Steven Bellovin

Columbia University, 116th St. and Broadway, New York, NY 10025, USA
{binh,smb}@columbia.edu

Abstract. Publish-subscribe protocols offer a unique means of data distribution, that has many applications for distributed systems. These protocols enable message delivery based on subscription rather than specific addressing; meaning a message is addressed by a subject string rather than to a specific recipient. Recipients may then subscribe to subjects they are interested in receiving using a variety of parameters, and receive these messages immediately without having to poll for them. This format is a natural match for anonymous delivery systems: systems that enable users to send messages without revealing their identity. These systems are an area of great interest, ranging from messaging relays like Tor, to publication systems like FreeHaven. However, existing systems do not allow delivery based on topics, a mechanism which is a natural match for anonymous communication since it is not addressed based on identity. We concretely describe the properties of and propose a system that allows publish-subscribe based delivery, while protecting the identities of both the publishers and subscribers from each other, from outside parties, and from entities that handle the implementation of the system.

Keywords: Anonymous · Publish subscribe · Push · Multicast

1 Introduction

In the publish-subscribe model, messages can be published to topics, rather than addressed to recipients. These are then multicast to the entire set of recipients that have previously subscribed to those topics. These topics can be anything from a set of specific match strings to ranged attributes on a multi-dimensional array. These types of messaging systems are typically implemented using a third party who manages subscriptions and acts as a relay between publishers and subscribers, though distributed systems have been implemented which allow for greater scalability.

This paradigm adds a different kind of flexibility in that senders and recipients are decoupled and can operate without even knowing of each other's existence. This can be a more suitable mode of operation for many kinds of systems. For example, a chat or newsgroup application is more cleanly implemented where the speaker does not have to obtain and maintain an enumerated list of all people who are interested in what he has to say. In a normal addressing system, he would have to be aware of all the people he means to send messages to. In a

J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 195–211, 2015.
DOI: 10.1007/978-3-319-23829-6_15

publish-subscribe system, he only needs to publish his message on a topic, and all interested readers can subscribe to those topics without either party knowing of the other. Another possible application is a search protocol in a document publishing and distribution system, where data providers subscribe to topics they provide and queriers publish messages indicating their interest. Data providers would then become aware of all the people searching for their content and could initiate a transfer. Another system that would benefit from publish-subscribe would be a notification system to mobile users that are interested in events on a geographical basis. This could be implemented using ranged attribute subscriptions filled in via GPS coordinates. Any system where addressing is preferably based upon the nature of the content rather than knowledge of the recipient can benefit from a publish-subscribe architecture.

Another area of interest is anonymous communication systems, wherein users can contact each other and exchange data while protecting their identities from each other and from outside parties. These include messaging systems and relays such as mixnets [4,11] and onion routing networks [13], and publishing systems like FreeHaven [9] and FreeNet [6]. Anonymous relays allow users to send addressed messages while protecting their identities from the recipients and against all third parties. They may also allow users to create pseudonymous "addresses" that they can announce, whereby others may contact them without knowing who their true identities. Anonymous publishing systems allow users to store and advertise documents online that can then be freely accessed by the public without revealing the identity of the authors. They may also protect the identities of readers who are either accessing these documents or using search protocols that allow them to find documents of interest to them.

We introduce a new system that achieves anonymous publish-subscription. We do so by creating a network of multi-cast nodes using an existing point-to-point anonymous communication network (such as an onion-routing network like TOR [13]). This supports a push publish-subscribe architecture: messages will be delivered to recipients without needing to be polled or requested on an individual basis. It also supports publication topics as string matches, integer ranges, and multi-attribute ranges mapping integer values to multiple labels.

1.1 Why Merge Publish-Subscribe with Anonymous Communication?

Anonymous communication systems are of clear value for protecting sensitive data or interests. However, to date, we are unaware of systems that work on a publish-subscribe basis while providing any sort of clearly defined anonymity guarantees for sender or receiver. Although there are some systems that claim to provide anonymous publish/subscribe, neither the difficulty of identifying publishers nor the efficiency of the system is thoroughly analyzed. This is unfortunate, since the publish-subscribe paradigm is a natural match for anonymous communication. In many scenarios which require anonymous communication, there are two separate problems: how to establish relationships between sender

and receiver when neither knows the other's identity, and then how to anonymously deliver their messages. Many anonymous communication systems do not address the first problem of how to establish anonymous relationships where meaningful communication should occur; this is left as outside the scope of the system. But by its very nature, publish-subscribe aims to support communication based on content rather than by identity, and users need not concern themselves with the details of *finding* the entities they aim to communicate with anonymously. It thus naturally solves this issue.

Such a system could for example allow for newsgroups and real-time chat applications that discuss sensitive topics like medical conditions or radical political movements such as discussions between members of Falun Gong or Arab Spring. Since in a group discussion a user is already sending messages without requiring awareness of the recipients, it is a natural step to provide a guarantee that this identity remain anonymous.

These applications could not be efficiently met by existing anonymous communication systems, which do not support any form of multi-cast and work based on known-recipient addressing. Nor could they be met by anonymous publishing systems, which work on a pull-basis rather than a push-basis. This makes them unsuitable for real-time applications. Publish-subscribe systems naturally provide flexibility that is likely to be useful for any type of anonymous communication need, since they do not require assumptions about participant identity by other participants.

1.2 Paper Organization

In Sect. 2 we overview related work on the subject. Section 3 concretely states the framework that our systems aim to fulfill. We present the system design itself and compare it to naive approaches in Sect. 4. Implementation details and performance results are given in Sect. 5. We summarize our results in Sect. 6.

2 Related Work

Non-anonymous publish subscribe systems were developed a great deal by TIBCO, who developed the Rendezvous system [14], which introduced wildcard topic matching, used a de-centralized architecture which supported topic priority in routing. The most currently used publish-subscribe system is PubSubHubbub [10]. PubSubHubbub is an extension of the RSS web feed protocol, but improves upon it by implementing the delivery of messages using a push mechanism. In other words, feed updates are pushed from the sender immediately to the receivers rather than waiting for them to poll the feed, making it a publish-subscribe protocol. Similarly, there are cloud-based content distribution mechanisms designed to ensure secure, but not identity-hidden, delivery of messages [3,7,15]. None of these systems, however, provide any anonymity protection. They are intended to be used between openly known clients.

There are numerous anonymous data distribution systems besides those that work on a publish-subscribe basis. Tor provides a simple anonymous routing network that relays messages through a number of nodes with layered encryption, such that unless an attacker can either compromise all nodes on the path or monitor both the beginning and end, the sender cannot be identified [13]. Since messages are addressed, this is not a publish-subscribe system of delivery. Also, the recipient's identity is not protected. One thing a publish-subscribe system handles well that an addressed system does not is broadcast delivery of messages to a wide audience. There are anonymous systems that do this, but not using a publish-subscribe model.

One well known system for widescale document distribution is the Free Haven project [9], a peer-to-peer file-sharing system. In this system, users can publish documents, making them freely available without revealing their identities. It is based on a community of servers that distribute storage of split shares of published documents. Recipients broadcast requests throughout the storage space, and those with pieces of interest return them encrypted. The documents are associated with private key encryption pairs to maintain ownership through updates and deletions.

Another similar system is FreeNet [6], also a peer-to-peer file-sharing system, but one with routed document requests rather than universally broadcast ones. Documents are associated with hashes of descriptive keyword strings, and migrated over time so that similar documents tend to migrate to geographically close servers on the network topology. Queries are then sent on a hill-climbing search over these lexical hashes. It is more scalable, and has more flexible document retrieval (keyword search rather than simple unique-name lookup) than FreeHaven, however it does not protect recipient identity, only that of the document owners.

Finally, another approach to anonymous distribution is TOR hidden services [13]. These allow a user to create a pseudonymous address through which they may be reached anonymously through the TOR network. Other users can then initiate connections through this address without revealing their own identities or knowing who they are contacting behind the pseudonymous address. These are not inherently multi-cast systems; each recipient must establish an individual connection, which creates additional load on the server when many clients are involved.

In all of these anonymous distribution systems, since users must expressly request messages, they are not push systems and furthermore do not allow for continuous messages based upon a topic. They also are not generally intended for low-latency delivery of messages; a distributor stores a message to the network whereupon they will be fetched by an interested party at a later time. They are thus not publish-subscribe systems. Document publishing systems such as these are suited to applications where a sender wishes to send a limited amount of data in a short time to be made available over a much longer time period. Publish-subscribe systems, however, are better suited to applications where there will be an ongoing stream of data relating to a specific subject, and where messages have a shorter lifespan.

The only existing publish-subscribe system we are aware of that aims to provide anonymity is by Datta et al. [1,8]. They propose a routing system based on maintaining multiple layers of weakly connected directed acyclic graphs. In this system, one or more sink nodes, which may change over time, become dissemination points receiving all publications and forwarding them to subscribers. However, anonymity is provided only by stating that the node a receiver gets a message from may not be the original publisher. However, an adversary would still know that node could possibly be an original publisher. Without probabilistic analysis of this possibility, it is difficult to say how well protected the publishers actually are. Also, no mention is made as to how difficult it is to identify subscribers in the system. Further, the system is neither analyzed for efficiency and scalability, nor implemented, so it is unclear at what cost this protection comes. There is no guarantee that the shape of the directed graphs that forms over very large networks scales in an efficient manner.

3 Anonymous Publish-Subscribe

Our system will aim to provide publish-subscribe functionality while protecting sender and receiver identities. This means that in terms of functionality, it will allow users to subscribe and unsubscribe to topics, and publish to topics, ensuring that published messages on a topic are delivered to all of its subscribers. More concretely, the system provides the following functions to its users:

- `Subscribe(u, t)`*:* User u specifies interest in topic t. The system maintains an internal, protected subscription of the tuple (u, t). u listens for messages sent with the topic t.
- `Unsubscribe(u, t)`*:* The system removes any subscription of (u, t) if present, and u ceases to listen for relevant messages.
- `Publish(m, t)`*:* A message m is sent into the system under topic t. For every subscription tuple (u_i, t_i) s.t. t matches t_i, m will be sent to u_i.

In the above functions, the nature of a topic and what constitutes a match between publication and subscription topics are left undefined. A variety of different matching types can be supported by a publish-subscribe system depending on what it is trying to accomplish. The most basic of these is exact string matching; in other words users subscribe specifically to a unique topic, and receive messages that are published exactly to that topic string. This is useful for establishing communication between defined clusters of users, such as newsgroups.

We will also deal with less concrete groupings, and allow users to instead define communication on one or more dimensions of ranges so that we can define geometric shapes of users. In such a case, a topic would consist of one or more labels, each being associated with an integer value within some pre-defined and limited range. A subscription would then be a list of label-range pairs indicating what range of values to accept along each axis. This can be useful for applications such as geographically based communication, or alert systems that are notified of values in certain ranges generated from physical sensor networks. Thus we have matching types on strings, integer ranges, and ranges across multiple dimensions:

- *Labels:* Each topic is a human-readable string. A publication and subscription are deemed to match if their topics are identical.
- *Ranges:* A publication topic is a numerical value v (either integer or float). A subscription consists of a tuple (l, h) s.t. $l \leq h$. A publication and subscription are deemed to match if $l \leq v \leq h$.
- *Multi-attribute ranges:* A publication topic is a list of tuples (t, v). A subscription topic is a list of tuples (t, l, h). A publication and subscription are deemed to match if for every tuple in the subscription (t_s, l_s, v_s), there exists at least one tuple in the publication (t_p, v_p) s.t. $t_s = t_p$ and $l_s \leq v_p \leq h_s$.

Our system will be able to make guarantees that messages will be successfully delivered. It should also make guarantees in regards to the amount of excess delivery that occurs. Delivery of messages that were not subscribed to is acceptable to an extent, since the receiver can simply ignore them himself. By default, these systems are not designed to prevent users from subscribing to any topic of their choosing, so it is not considered a leakage for them to receive extra messages. If it *were* desirable to prevent such leakages, that could be achieved independently using encryption systems for each topic. Hence, for the underlying message delivery system, excessive message receipt is an efficiency issue, not a security one. To be called anonymous, the system should ensure that using the basic publish and subscribe functionality does not compromise one's identity. In other words, we aim to prevent interactors and third parties from identifying two parties: the publishers and the subscribers. The specifics of this protection, which parties are prevented from identifying the participants and under what circumstances, are dependent on the implementing system.

We begin with correctness definitions:

- *Completeness:* For every publication (m_p, t_p) and subscription (u_s, t_s), if t_p and t_s match, then m_p will be delivered to u_s.
- *Non-excessiveness:* For every publication (m_p, t_p) and subscription (u_s, t_s), if t_p and t_s do not match, then m_p will be delivered to u_s with probability $Pr \leq \epsilon$.

These capture the requirement that messages be delivered to those who are subscribed to them, and that the system not produce undue load by delivering them to a large amount of uninterested parties. More complicated are the security definitions. First is publisher anonymity: we will guarantee that no adversary can learn the identity of the publisher of any message.

Definition 1. Publisher anonymity: *Let p be the publisher of message m, and let H be the set of h honest users in the system. Let A be any collaboration of entities not in H, including subscribers, entities related to the operation of the system, outside observers, and other publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. A cannot then identify p given m with probability greater than $\frac{1}{h}$.*

We allow an adversary to collude with or compromise any number of other users (both publishers and subscribers) in the system. They may then for any

length of time take any of the actions those entities might take: publishing messages, subscribing to topics, and observing the messages received as a result of those subscriptions or through the normal routing of other messages in the system. They may do so in an adaptive fashion, choosing what types of publications or subscriptions to issue based on observations from previous messages, including the one they are attempting to de-anonymize. They may also attempt to subvert the system by refusing to forward messages the protocol would otherwise require them to, and observe the results of such actions in terms of additional traffic sent. We claim that our system will prevent such an adversary from identifying the publisher of any given message with probability any better than random guessing from amongst the pool of non-compromised users.

Next is subscriber anonymity, which encapsulates the protection of the identities of users who are subscribed to a topic. There are two types of anonymity we wish to protect:

Definition 2. Topic subscriber anonymity: *Let t be a topic for which there are s subscribers out of a group of S total participants in the system. Let A be any collaboration of entities having A_s subscribers, and possibly including entities related to the operation of the system, outside observers, and publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. A cannot identify determine if user u is subscribed to t with probability greater than $\frac{s-A_s}{S-A_s}$.*

This captures subscriber anonymity in the first direction, an adversary should not be able to identify the subscribers of a given topic with probability greater than random guessing. Again, we assume an adversary may compromise any number of users in the system, and learn whatever information it can by taking all actions normally available to those compromised users. It may again also learn adaptively, using observations from previous messages to form new publications and subscriptions to enter into the system. It may do so indefinitely over the lifetime of a subscription. We claim our system will prevent such an adversary from identifying any subscriber of a given topic with probability better than random guessing from amongst the pool of non-compromised users.

Definition 3. Subscription anonymity: *Let t be a topic and s be a user subscribed to t. Let A be any collaboration of entities including those related to the operation of the system, outside observers, other subscribers, and publishers. These entities may enter any number and type of published messages as publishers,and observe the outputs as subscribers. Given s, A cannot identify t with probability greater than $\frac{1}{T}$ where T is the total number of possible topics.*

This captures the opposite direction: an adversary should not be able to, given a user, determine what topics he is subscribed to. This will assume the same types of powers for the adversary as with subscriber anonymity, and the adversary will attempt to defeat our system by guessing from amongst the pool of possible subscriptions.

4 Our Systems

We introduce two systems for providing anonymous publish-subscribe. The first provides a stronger anonymity guarantee, but uses a central point of dissemination. As such, it does not scale as well as the second system which provides better scalability at a cost of weaker anonymity. Both systems assume an honest-but-curious model and do not aim to protect against a global passive adversary.

4.1 Central Server Routing

Our first solution will be based off of a central server that handles the logic of matching publications to subscriptions and routing. To guarantee anonymity from this server, and from other participants, both publishers and subscribers will connect to it through an obfuscating proxy, which is trusted not to collaborate with the server. There are thus four types of entities:

- *Server:* Stores subscriptions, matches publications, and routes messages. It should not be able to read message content, subject content, or identify senders or recipients.
- *Publisher:* Sends messages into the system.
- *Subscriber:* Sends subscriptions and receives matching messages from the system.
- *Proxy:* Entry point for communication between the server and publishers or subscribers. It is responsible for all contact with these entities, and for obscuring their identities from the server.

Trust is separated between the proxy and the server. The proxy will be able to see the identities of senders and recipients of messages, but will not be able to see the content of the messages being sent or the subjects they are being sent upon. The server will be able to see a deterministic encryption of this information, but will not know who is sending or receiving the messages. Although these deterministic encryptions can be matched to each other, since all origin points look identical to the server, he cannot link publishers. This separation of trust ensures to the user that no single entity can monitor his behavior.

To achieve this separation of information, we make use of a protocol called re-routable encryption [12]. This protocol allows for a sender and a receiver, each with unique symmetric encryption keys, and a third router entity. It provides a fast multi-party computation between the three parties, resulting in the router receiving a transformation key which then allows him to transform messages encrypted by the sender into messages encrypted by the receiver's key without being able to see or compute the cleartext on his own. This protocol allows us to efficiently realize the separation of trust between the server and proxy. These transformation keys will be generated between the server, proxy, and client (publisher or subscriber) once to introduce each participant to the system. Owning transformation keys allows the proxy to relay messages to and from the server without seeing their content or revealing the other communicating party without the expensive overhead of an obfuscating mixnet.

We also make use of Bloom filters [2] to manage and match large quantities of publications and subscriptions on the server end while obscuring topic content from the server. Bloom Filters allow matching against sets that can store any number of elements with a boundable false positive rate that can be reduced by increasing Bloom Filter size relative to the number of terms stored.

The server will store an index of all subscriptions on a per-subscriber basis as a Bloom Filter index. Each subscriber is represented as one Bloom Filter storing all of his subscriptions. The exact nature of the subscriptions can be anything supported by Bloom Filters (exact topic keywords, ranges using our range-query protocol, multi-dimensional ranges, etc.) In our system, the subscribers will be pseudonymous from the server's point of view. If subscribers wish to prevent linkage between their subscriptions, they can do so by creating a pseudonym per subscription.

The proxies will use re-routable encryption to deliver messages from source to destination: deterministic for communication of subjects from publisher to server, and non-deterministic for communication of messages from publisher to server to subscriber. The proxy contains a transformation key from the server key to and from the key of each subscriber. This key must be computed between the user, proxy, and server once to join each new user into the system. The proxy maintains this mapping using the same pseudonyms used by the Bloom Filter index held on the message server. The server maintains its own encryption key k_r. To subscribe to a subject, a user generates his own key k_u, engages in secure multiparty computation with the proxy and server that results in the server learning transformation key $k_{\frac{r}{u}}$. He then deterministically encrypts his subject subscription under k_u and sends it to the proxy, who transforms it to encryption under k_r and forwards it to the server where it is stored, along with a pseudonym that the proxy would understand to correspond to the user. We are now ready for publishers to send messages to subscribers. The system and message path is thus laid out as in Fig. 1.

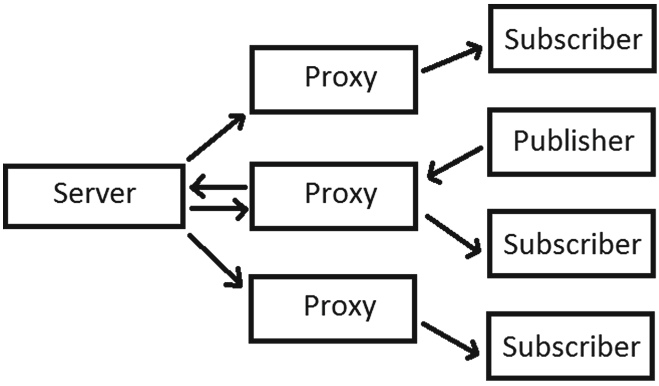

Fig. 1. Centralized anonymous publish-subscribe system

A publisher generates a message m, and encrypts it non-deterministically under k_u. He determines a subject s, and encrypts it deterministically under k_u. These are both transformed by the proxy to be encrypted under k_r before being forwarded to the server. The server checks s under deterministic encryption by k_r against his BF index. He then re-randomizes the random component of m encrypted by k_r and sends it to the proxy, along with all the BF match identities. For each corresponding recipient u', the proxy transforms m to encryption under $k_{u'}$, and forwards the message to the corresponding users.

Security Analysis. That our system achieves *completeness* and *non-excessiveness* is easy to see: since Bloom Filters promise a zero false negative rate all messages will be properly routed to their appropriate destinations. By adjusting the Bloom Filter parameters, the amount of excess publications created by false positives can be made arbitrarily small.

We aim to protect the identity of the participants from the server, and the content of the messages and subjects from the proxy. This is under the assumption that the proxy and server do not collaborate with each other, and that the proxy does not collaborate with other publishers or subscribers. The server and proxy are trusted to be honest-but-curious; that is they will obey the protocols, but may attempt to learn more than they should from the results.

Claim. Our system achieves trusted-party publisher anonymity.

Let us assume that there were a full collaboration of all other meaningful entities except for the proxy, who we will treat as a trusted party. The receiver sees only a delivered message which is entirely agnostic to its origin. Similarly, the server sees only the message and topic delivered by the proxy after transforming them to encryption under his own key. These would look identical regardless of which user originated the publication. Thus even if the adversary consisted of the server, the receiver, and any number of dishonest publishers, their combined view of any given message looks identical regardless of which honest user sent it. Therefore, they cannot gain an advantage in identifying the user.

Claim. Our system achieves both trusted-party topic subscriber anonymity and trusted-party subscription anonymity.

Again, we can assume collaboration between the server and any number of publishers publishing on a given topic. The subscription is delivered to the server by the proxy after transforming the encryption to his own key. This would look identical regardless of which user is subscribing to the topic. Thus for a given topic, he will see only a set of subscriptions which do not give any information that distinguishes between subscribers. In the reverse, given a subscriber, the subscriptions seen by the server do not look different whether or not he is the origin. Thus the server cannot gain any information that would help distinguish between subscribers given a topic, or identify topics given a subscriber.

The proxy is privy to both of these identities, and is thus treated as the trusted third party. However, he cannot see what content is being delivered or

what topic it is being published to. This is given under the same assumptions as the underlying re-routable encryption scheme.

Owing to the use of Bloom Filters to match publications and subscriptions, there is an existing false positive rate. However, since the system is used for open subscription, this is a non-issue from a security standpoint. The recipient can simply ignore any messages he is not interested in. As we mentioned before, this system is only secure if the proxy cannot act as a user. If he is able to, then he can use transformation keys to transform anything encrypted by the server's key into his own key, and thus read messages that are routed through him, breaking the security of the system.

4.2 Spanning Tree Routing

Our second solution will route messages to all subscribers using per-subscriber spanning tree structures. This will be accomplished by providing an overlay network of the nodes, and then representing each spanning tree within the routing tables of the nodes in the overlay.

The list of nodes in the network will be registered in a global directory, which can be either a single server, or a DHT for greater scalability. Nodes will then use Bloom Filter indexes as routing tables to forward messages by checking their subjects against the indexes. A destination will be represented as a single Bloom Filter, storing subjects as elements in the filter. Subscriptions live as elements in these filters. We can thus support routing based on topic for any type of topic that can be represented in a Bloom Filter (i.e. exact strings, ranges, etc.). In order to prevent cycles, each message will carry a header with a Bloom Filter storing unique labels that nodes can check to see if they have already forwarded the same message. These will be randomly generated and updated on regular intervals.

All that remains now is to set the routing Bloom Filters such that all published messages will be received by all interested subscribers. To do this, each subscriber will construct a unique spanning tree of the network, rooted on himself. We assume the existence of an underlying anonymous communication network that allows both sending a message while protecting the identity of the sender, and providing a pseudonymous address by which other users can route messages to a recipient who wishes to protect their true identity. In our implementation, we use Tor [13] to provide these functionalities. Although Tor has many known limitations, it is efficient and used often in the real world.

A subscriber will then anonymously instruct all nodes in the network to add a routing entry for that subject to their parent in his uniquely constructed tree. Thus, any message anywhere in the network, when routed on this subject, will find its way to every subscriber that is interested. Although expensive for subscription, this is fast for publication, and so is well suited for systems where publications dominate subscriptions in terms of network load. Unsubscription is a little trickier, and is not handled by our current implementation. We could handle this in the future either by allowing Bloom Filters to expire and requiring

subscriptions to be updated on a regular basis, or by using counting Bloom Filters which will allow deletion of entries.

- `Subscribe(u, t)`*:* User u looks up the node from directory D. As a constant parameter of the system, he assumes a routing chain length of r. He then constructs a random, balanced spanning tree of depth r using all nodes in the network with himself as the root. For each node in the tree, he anonymously contacts that node, and instructs it to route all messages with subject t to their parent in the tree. This is done more efficiently by forwarding instructions for each node through their parents with layered encryption in the same manner as an onion routing network. Thus, we multicast the subscription along the same structure as the tree itself.
- `Publish(m, t)`*:* The sender picks a random origin point in the network, and uses the underlying anonymous communication network to send his message to that node. That node routes his message to the nodes indicated by looking up the subject on its own Bloom Filter index. All other nodes will forward the message in similar fashion, except first checking their loop-detection label, then inserting it into the header of the message.

The system and message path is thus laid out as in Fig. 2.

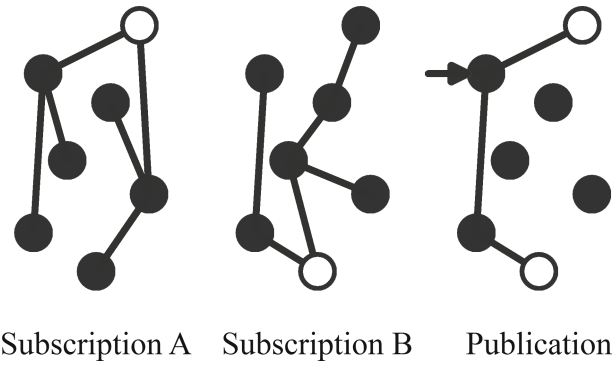

Fig. 2. Spanning tree anonymous publish-subscribe system

In the absence of false positives within the routing Bloom Filters, the common case is that for each publication, there will be a randomly selected path of length r from the initial node the publisher selects to each subscriber. If the number of nodes is significantly larger than the number of subscribers for a single topic, then in all likelihood, the initial node will have to multiply the message for each end subscriber. However, this is still a benefit over the central server solution, since the central server solution uses a single node to multiply all traffic within

the system, whereas with the spanning tree solution, each publication will use a different randomly selected initial point. This will better distribute load when there are a large number of publications going out simultaneously.

Efficiency Comparison. We now compare the efficiency of our protocols for n receivers, and a tree depth of k, and a load involving p simultaneous publishers.

The primary tradeoffs are between the longest path, which affects the latency of message delivery, and the bottleneck branching point, which determines how scalable the protocol is. The central server solution will have a fixed longest path of four hops (from publisher, to proxy, to server, to proxy, to subscriber). So for low-load situations, we can expect the latency to be $O(1)$. Conversely, the spanning tree solution's path will scale with the depth of the spanning trees selected, and so has a worse $O(k)$ behavior. However, spanning tree depth is simply chosen to add layers of indirection and does not need to scale with the size of the system. So while the spanning tree has worse latency, it is boundable.

Both solutions involve a single point of multicast for each publication in the expected case. For the central server solution, this is the server. For the spanning tree solution, if the subscribers are not a significant portion of the total userbase, then likely each has a unique path from the publisher, making the publisher the point of multicast. However, in the central server solution, all published messages share the same point, whereas in the spanning tree solution each has a unique one. Thus the central server solution faces a load of $\theta(kp)$ on the server, whereas the spanning tree solution faces a load no greater than $\theta(k)$ on any one node. Clearly, the spanning tree solution can handle multiple publisher load scaling better.

Security Analysis. Our system provides *Completeness* and *Non-excessiveness* under the honest-but-curious model, that is when all parties perform the protocol correctly but may try to learn more than they should. If all nodes are forwarding correctly, messages are guaranteed to be routed to all interested subscribers, with a boundable false positive rate on loop detection.

Claim. Assuming the security of the underlying anonymous communication system, our system achieves complete publisher anonymity.

The message itself does not contain any information unique to the publisher. From the perspective of the initial receiving node, any message it receives is only visible as an output of the underlying anonymous communication system. Thus, if it can identify the origin, then that implies a failure of that system. From then on, clearly no other node in the network can do better in terms of identifying the publisher.

Claim. Assuming our underlying TOR system is secure against identification attacks, our system achieves topic subscriber anonymity.

For any given topic in the system, every node will have one or more nodes which it is expected to route matching messages towards. And if TOR protects the identities of its senders, then the subscription process itself will not reveal the subscriber identity. Thus, no node can distinguish between a neighbor who is an interested party, and a neighbor who is merely forwarding towards one. In order to identify a node as an endpoint, an adversary would need to identify a publication originating from one of the leaves of its subscription tree, and then compromise all of the nodes on the path from publisher to subscriber, a requirement as stringent as for an adversary of TOR.

A more complicated issue is denial of service attacks. An attacker who was himself a subscriber could refuse delivery of messages to nodes he is intended to forward towards. However, since it is the subscribers who choose the routing tree that leads to them, an attacker would not be able to fully block a particular subscriber from receiving messages, nor fully block a publisher from disseminating them. Nor would he have any control over which particular publisher-subscriber relationships he could interfere with. Furthermore, if subscriptions are updated on a regular basis, his sphere of influence would be steadily changing. A system of checks wherein a subscriber occasionally publishes test messages and begins them at different points in the network could be implemented to specifically identify malicious nodes, however this remains to be further developed.

5 Performance

We implemented and tested both our central server and splay-tree based systems to observe scaling issues both in subscription and publication. Unfortunately, to our knowledge, there exist no other anonymous publish-subscribe systems to compare to, so we show only to demonstrate usability in comparison to normal network transactions, and to demonstrate efficiency differences between the two. To obtain a large number of nodes for scalability testing, we used the PlanetLab network [5]. Each participating node has at minimum 4x 2.4 Ghz Intel cores, 4 GByte ram, and 500 GB disk space. Nodes are distributed around the globe to provide a simulation of internet traffic. For our experiments, we used nodes with varying geographic locations contained within the US.

Figure 3 shows time to add subscriptions for a varying number of subscribers for the central server and spanning tree solutions. This was done for a system with a total of 500 participating nodes. Measurements for the central server solution were taken using a one server and two proxy arrangement (one proxy for publishers and one for subscribers). This was measured from a start time when the subscription requests are queued into the systems, to the time when the last subscriber completes their request. Time scales roughly linearly for the central server system, because it is bottlenecked by the single point of connection and later subscriptions must wait for earlier ones to complete. The spanning tree solution performs worse at lower numbers of subscribers, due to its more involved protocol. However, it scales much better for larger numbers of subscribers as they can be handled concurrently. The growth is not entirely smooth, as the

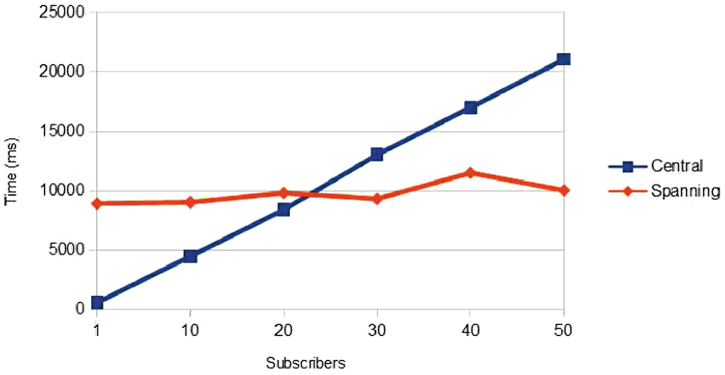

Fig. 3. Subscription cost

tree generation for each of the subscribers is random, and can cause more or less requests to be bottlenecked by various nodes depending on what kinds of overlap results.

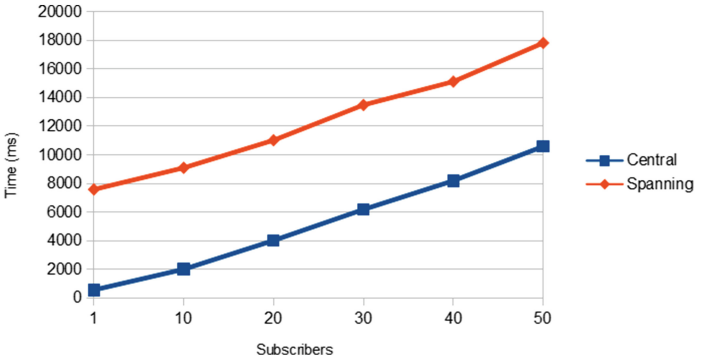

Fig. 4. Publication cost

Figure 4 shows time to deliver a publication. This was measured by observing a single node which subscribes to the same topic it is publishing on, and recording the time taken to receive its own message. Messages chosen were small text strings. This was measured from start time when the publisher initiated the publication to time when the last subscriber reported reception of the message. Again, time taken scales linearly with the number of subscribing nodes, again bottle-necking on the server which must duplicate the message once for each subscriber. In this test, only one publication is issued into the system at a time. Because of this, the spanning tree solution scales with similar behavior to the central server solution, but with a large constant overhead for the multiple hop message transmissions.

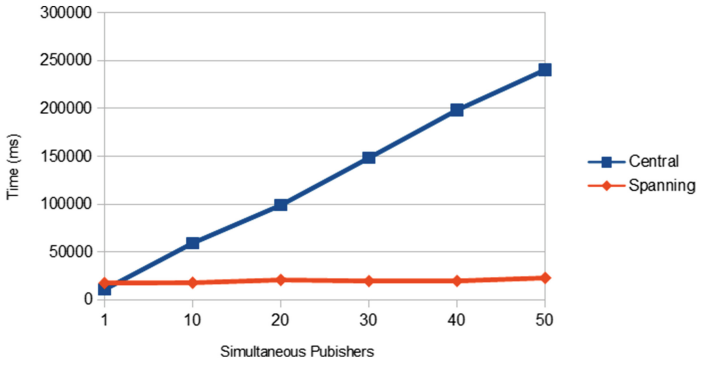

Fig. 5. Publication cost in active system

Figure 5 shows the same measurements taken with increasing numbers of simultaneous publications issued into the system. The number of subscribers is kept constant at 50. The X axis shows the number of participating publishers, with each sending a publication at the same time. The Y axis shows the time taken for a single node which we are monitoring to receive a single publication it has itself sent into the system. In this case, we see that the spanning tree solution steadily outperforms the central server solution, demonstrating a greater ability to distribute the load in an active system with multiple publishers and subscribers.

6 Conclusion

We have proposed combining anonymous communication with publish-subscribe routing, a pairing which is not explored by existing research. We have further proposed two systems for accomplishing this, with tradeoffs in performance and security. Although the latency overhead to handle publish-subscribe anonymously is significant, it is reasonable for systems with large numbers of users with small communities formed around particular topics. This fills a void in existing anonymous communication: the question of how anonymous entities should decide who to communicate with.

References

1. Anceaume, E., Datta, A.K., Gradinariu, M., Simon, G.: Publish/subscribe scheme for mobile networks. In: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (2002)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
3. Broberg, J., Buyya, R., Tari, Z.: Metacdn: harnessing storage clouds for high performance content delivery. In: Proceedings of ACM Conference on Data and Application Security and Privacy (2012)

4. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudo-nyms. In: Communications of the ACM (1982)
5. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. **33**, 3–12 (2003)
6. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability, pp. 46–66. Springer, New York (2001)
7. Koglin, Y., Yao, D., Bertino, E., Tamassia, R.: Decentralized authorization and data security in web content delivery. In: Proceedings of the 22nd ACM Symposium on Applied Computing (2007)
8. Datta, A.K., Gradinariu, M., Raynal, M., Simon, G.: Anonymous publish/subscribe in p2p networks. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing (2003)
9. Dingledine, R., Freedman, M.J., Molnar, D.: The Free Haven Project: distributed anonymous storage service. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 67–95. Springer, Heidelberg (2001)
10. Fitzpatrick, B., Slatkin, B.: Pubsubhubbub
11. Mathewson, N., Danezis, G., Dingledine, R.: Mixminion: design of a type iii anonymous remailer protocol. In: Security and Privacy (2003)
12. Raykova, M., Vo, B., Bellovin, S., Malkin, T.: Secure anonymous database search. In: CCSW 2009 (2009)
13. Syverson, P., Dingledine, R., Mathewson, N.: Tor: the second-generation onion router. In: Usenix Security (2004)
14. TIBCO: Tib rendezvous. In: White paper. TIBCO (1999)
15. Xiong, H., Zhang, X., Yao, D., Wu, X., Wen, Y.: Towards end-to-end secure content storage and delivery with public cloud. In: Proceedings of ACM Conference on Data and Application Security and Privacy (2012)