PUB-2-SUB: A Content-based Publish/Subscribe Framework for Cooperative P2P Networks

Duc A. Tran and Cuong Pham

Department of Computer Science
University of Massachusetts, Boston
100 Morrissey Blvd, Boston, MA 02125, USA
{duc, cpham}@cs.umb.edu

Abstract. This paper is focused on the content-based publish/subscribe service and our problem is to devise an efficient mechanism that enables this service in any given P2P network of cooperative nodes. Most techniques require some overlay structuralization added on top of the network. We propose a solution called PUB-2-SUB which works with any unstructured network topology. In addition, multiple independent publish/subscribe applications can run simultaneously on a single instance of PUB-2-SUB. We show that this mechanism is efficient in terms of both costs and time. Our theoretical findings are complemented by a simulation-based evaluation.

Key words: P2P, publish/subscribe, prefix tree, search

1 Introduction

Unlike traditional search, a query in the publish/subscribe model is submitted and stored in advance, for which the results may not yet exist but the query subscriber expects to be notified when they later become available. This model is thus suitable for search applications where queries await future information, as opposed to traditional applications where the information to be searched must pre-exist. Focusing on the publish/subscribe model, our goal is to devise a mechanism that can be integrated into a given P2P network to enable applications of this model. In particular, we are interested in distributed networks where the participating nodes are cooperative, reliable, and rather static. In these networks, such as grid computing networks and institutional communication networks, P2P can be adopted as an effective way to share resources, minimize server costs, and promote boundary-crossing collaborations [1–4]. A publish/subscribe functionality should be useful to these networks.

To enable publish/subscribe applications, a simple way is to broadcast a query to all the nodes in the network or to employ a centralized index of all the queries subscribed and information published [5–7]. This mechanism is neither efficient nor scalable if applied to a large-scale network. Consequently, a number of distributed publish/subscribe mechanisms have been proposed. They follow two main approaches: structuralization-based or gossip-based. The first approach [8–13] requires the nodes to be organized into some overlay structure (e.g., DHTs [14–17] and Skip Lists [18]) and develops publish/subscribe methods on top of it. The other approach [19–21] is for unstructured networks, in which the subscriber nodes and publisher nodes find each other via exchanges of information using the existing peer links, typically based on some form of randomization.

The structuralization-based approach is favored for its efficiency over the the gossip-based approach, but when applied to a given network, the former introduces an additional overhead to construct and maintain the required overlay structure. There may also be practical cases where the new links, that are part of the new structure, are not allowed due to the policy or technicality restrictions of the given network. On the other hand, although the gossip-based approach does not require an additional structuralization, due to the nature of gossiping, a query or a publication of new information must populate a sufficiently large portion of the network to be able to find each other at some rendezvous node with a high probability. The costs can be expensive as a result, including the communication cost to disseminate the query or publish the new information, the storage cost to replicate the query in the network, and the computation cost to evaluate the query matching condition.

We propose a publish/subscribe mechanism called PUB-2-SUB which, like the gossip-based approach, does not change the connectivity of the given network, but is aimed at a much better performance. PUB-2-SUB allows any number of independent publish/subscribe applications to run simultaneously. It is based on two key design components: the virtualization component and the indexing component. The virtualization component assigns to each node a unique binary string called a virtual address so that the virtual addresses of all the nodes form a prefix tree. Based on this tree, each node is assigned a unique zone partitioned from the universe of binary strings. The indexing component hashes queries and publications to binary strings and, based on their overlapping with the node zones, chooses subscription and notification paths appropriately and deterministically.

Because PUB-2-SUB is based on directed routing, it has the potential to be more efficient than the gossip-based approach. Our evaluation study shows that PUB-2-SUB results in lower storage and communication costs than BubbleStorm [19] – a recent gossip-based search technique. In terms of computation cost, PUB-2-SUB requires only a node to evaluate its local queries to find those matching a given information publication. The proposed technique also incurs small notification delay and is robust under network failures.

The remainder of the paper is organized as follows. We introduce the concept of PUB-2-SUB in more detail in Section 2, followed by a discussion on the evaluation results in Section 3. The paper is concluded in Section 4 with pointers to our future work.

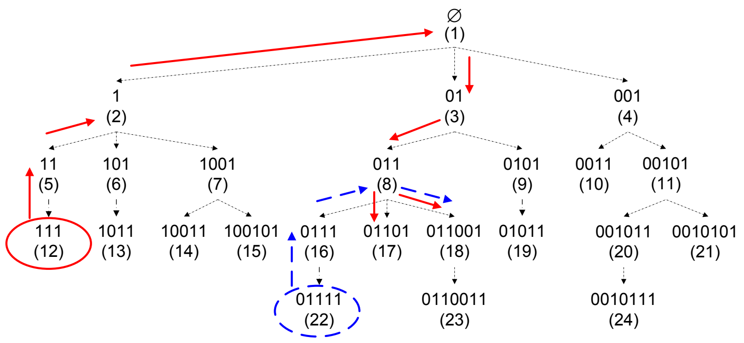


Fig. 1. Virtual address instance initiated by node 1 (the VAs of all the nodes form a prefix tree): solid-bold path represents the subscription path of query [‘0110001’, ‘0110101’]; dashed-bold path represents the notification path of event <‘0110010’>

2 PUB-2-SUB: The Proposed Solution

Consider a cooperative P2P network $\{S_1, S_2, ..., S_n\}$ that is constructed and maintained according to its built-in underlying protocols. The nodes should remain functional as much as possible although there may be failures that cannot be avoided, and whenever a new node joins or a failure occurs we assume that this network can re-organize itself. We are required not to modify the existing connectivity of the network; all communication must be via the provided links. PUB-2-SUB is based on two key design components, the virtualization component and the indexing component, which are described below.

2.1 Virtualization

A virtualization procedure can be initiated by any node to result in a “virtual address instance” (VA-instance), where each node is assigned a virtual address (VA) being a binary string chosen from $\{0,1\}^*$. Suppose that the initiating node is S^*. In the corresponding VA-instance, denoted by INSTANCE(S^*), we denote the VA of each node S_i by $VA(S_i : S^*)$. To start the virtualization, node S^* assigns itself $VA(S^* : S^*) = \varnothing$ and sends a message inviting its neighbor nodes to join INSTANCE(S^*). A neighbor S_i ignores this invitation if already part of the instance. Else, by joining, S_i is called a “child” of S^* and receives from S^* a VA that is the shortest string of the form $VA(S^* : S^*)$ + ‘0*1’ unused by any other child node of S^*. Once assigned a VA, node S_i forwards the invitation to its neighbor nodes and the same VA assignment procedure repeats. In generalization, the rule to compute the VA for a node S_j that accepts an invitation from node S_i is: $VA(S_j : S^*)$ is the shortest string of the form $VA(S_i : S^*)$ + ‘0*1’ unused by any current child node of S_i.

Eventually, every node is assigned a VA and the VAs altogether form a prefix-tree rooted at node S^*. We call this tree a VA-tree and denote it by $TREE(S^*)$. For example, Figure 1 shows the VA-tree with VAs assigned to the nodes as a result of the virtualization procedure initiated by node 1. It is noted that the links of this spanning tree already exist in the original network (we do not create any new links). In this figure, the nodes’ labels (1, 2, ..., 24) represent the order they join the VA-tree. Each time a node joins, its VA is assigned by its parent according to the VA assignment rule above. Thus, node 2 is the first child of node 1 and given $VA(2:1) = VA(1:1)$ + '1' = '1', node 3 is the next child and given $VA(3:1) = VA(1:1)$ + '01' = '01', and node 4 last and given $VA(4:1) = VA(1:1)$ + '001' = '001'. Other nodes' VAs are assigned similarly. For example, consider node 18 which is the third child of node 8 (VA '011'). The VA of node 18 is the shortest binary string that is unused by any other child node of node 8 and of the form $VA(8:1)$ + '0*1'. Because the other children 16 and 17 already occupy '0111' and '01101', node 18's VA is '011001'.

A VA-tree resembles the shortest-delay spanning tree rooted at the initiating node; i.e., the path from the root to a node should be the quickest path among those paths connecting them. It can be built quickly because only a single broadcast of the VA invitation is needed to assign VAs to all the nodes. To help with indexing, in $INSTANCE(S^*)$, each node S_i is associated with a unique "zone", denoted by $ZONE(S_i : S^*)$, consisting of all the binary strings *str* such that: (i) $VA(S_i : S^*)$ is a prefix of *str*, and (ii) no child of S_i has VA a prefix of *str*. In other words, among all the nodes in the network, node S_i is the one whose VA is the maximal prefix of *str*. We call S_i the "designated node" of *str* and use $NODE(str : S^*)$ to denote this node. For example, using the virtual instance $TREE(1)$ in Figure 1, the zone of node 11 (VA '00101') is the set of binary strings '00101', '001010', and all the strings of the form '0010100...', for which node 11 is the designated node. The following properties can be proved, which are important to designing our indexing component:

1. $ZONE(S_i : S^*) \cap ZONE(S_j : S^*) \neq \emptyset$, for every $i \neq j$
2. $\bigcup_{i=1}^{n} ZONE(S_i : S^*) = \{0,1\}^*$
3. $\bigcup \{ZONE(S' : S^*) \mid S'$ is S_i or a descendant of $S_i\} = \{str \in \{0,1\}^* \mid VA(S_i : S^*)$ is a prefix of $str\}$, for every i

2.2 Indexing

For each publish/subscribe application under deployment, the information of interest is assumed to have a fixed number of attributes called the dimension of this application. PUB-2-SUB supports any dimension and allows multiple applications to run on the network simultaneously, whose dimension can be different from one another's. We use the term "event" to refer to some new information that a node wants to publish. The queries of interest are those that specify a lower-bound and an upper-bound on each event attribute. For ease of presentation, we assume that events are unidimensional. The idea can easily be extended for the case of multidimensionality (see our extended work [22]).

Without loss of generality, we represent an event x as a k-bit binary string (the parameter k should be chosen to be larger than the longest VA length in the network). A query Q is represented as an interval $Q = [q_l, q_h]$, where q_l, $q_h \in \{0,1\}^k$, subscribing to all events x belonging to this interval (events are "ordered" lexicographically). As an example, if $k = 3$, the events matching a query ['001', '101'] are {'001', '010', '011','100', '101'}.

Supposing that every node has been assigned a VA as a result of a virtualization procedure initiated by a node S^*, we propose that

Query subscription: Each query Q is stored at every node S_i such that the zone of this node $ZONE(S_i : S^*)$ intersects with Q.

Event notification: Each event x is sent to $NODE(x : S^*)$ – the designated node of string x. It is guaranteed that if x satisfies Q then Q can always be found at node $NODE(x : S^*)$ (because this node's zone must intersect Q).

Figure 1 shows an example with $k = 7$. Suppose that node 12 wants to subscribe a query $Q =$ [‘0110001’, ‘0110101’], thus looking to be notified upon any of the following events {‘0110001’, ‘0110010’, ‘0110011’, ‘0110100’, ‘0110101’}. Therefore, this query will be stored at nodes {8, 17, 18}, whose zone intersects with Q. For example, node 8's zone intersects Q because they both contain ‘0110001’. The path to disseminate this query is $12 \to 5 \to 2 \to 1 \to 3 \to 8 \to \{17, 18\}$ (represented by the solid arrow lines in Figure 1). Now, suppose that node 22 wants to publish an event $x =$ <‘0110010’>. Firstly, this event will be routed upstream to node 8 – the *first* node that is a prefix with ‘0110010’ (path $22 \to 16 \to 8$). Afterwards, it is routed downstream to the designated node $NODE$(‘0110010’:1), which is node 18 (path $8 \to 18$). Node 18 searches its local queries to find the matching queries. Because query $Q =$ [‘01011111’, ‘01100011’] is stored at node 18, this query will also be found.

The storage and communication costs for a query's subscription depend on its range; the wider the range, the larger costs. For an event, the communication cost measured as the number of hops traveled to publish an event is $O(h)$ where h is the tree height ($h = O(\sqrt{n})$ in most cases). The delay to notify a matching subscriber is the time to travel this path; hence, also $O(h)$. The computation cost should be small because only one node – the designated node $NODE(x : S^*)$ – needs to search its stored queries to find those matching x. Our evaluation study in Section 3 indeed finds these costs reasonably small.

2.3 Update Methods

There may be changes in the network such as when a new node is added or an existing node fails. Supposing that the network is virtualized according to the VA-instance $INSTANCE(S^*)$, PUB-2-SUB addresses these changes as follows.

Node Addition: Consider a new node S_{new} that has just joined the network according to the network's underlying join protocol. As a result, it is connected to a number of neighbors. We need to add this node to the VA-instance. First, this node communicates with its neighbors and asks the neighbor $S_{neighbor}$ with the minimum tree depth to be its parent; tie is broken by choosing the one with fewest children. This strategy helps keep the tree as balanced as possible so its height can be short and workload fairly distributed among the nodes. The neighbor will then assign S_{new} a VA that is the shortest unused binary string of the form $VA(S_{neighbor} : S^*)$ + ‘0*1’. Because $ZONE(S_{neighbor} : S^*)$ is changed, the next task is for the parent node

$S_{neighbor}$ to delete those queries that do not intersect $ZONE(S_{neighbor} : S^*)$ anymore. Also, this parent node needs to forward to S_{new} the queries that intersect $ZONE(S_{new} : S^*)$.

Node Removal: When a node fails to function, it is removed from the network according to the underlying maintenance protocol. This removal however affects the connectedness of the VA-instance in place. Because the VAs of the child nodes are computed based on that of the parent node, the child nodes of the departing node need to find a new parent so the VA-instance remains valid. Consider such a child node S_{child}. This node selects a new parent among its neighbors. The new parent, say node S_{parent}, computes a new VA for S_{child} (similar to node addition). Then, S_{child} re-computes the VAs for its children and informs them of the changes. Each child node follows the same procedure recursively to inform all its descendant nodes downstream. The query transfer/forwarding from S_{parent} to S_{child} and, if necessary, from S_{child} to the descendant nodes of S_{child} is similar to the case of adding a new node.

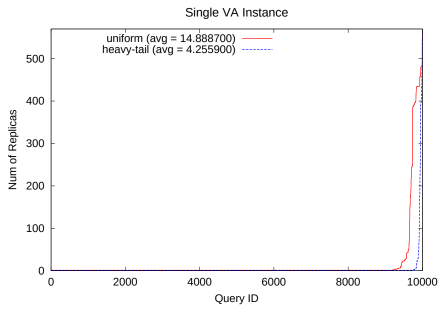
In addition, because each descendant node S_i of the removed node is now assigned a new VA and thus a new zone, the queries that are stored at S_i before the VA adjustment may no longer intersect its new zone. These queries can be either deleted or re-subscribed to the network depending on the priority we can set at the first time they are subscribed to the network. If a query is marked as "high-priority", it is stored in the network permanently until the subscriber determines to unsubscribe it (the unsubscription procedure is similar to the subscription procedure). On the other hand, if a query is marked as "low-priority", it is associated with a lease time after which the query will expire and be deleted. How to implement these two types of priority is determined by the application developer.

The worst case with node removal is when the root node fails in which we have no way to recover other than rebuilding the entire VA instance. To avoid this unfortunate case, we propose that the root of a VA-instance be a dedicated node deployed by the network administrator and thus we can assume that this node never fails. This requirement can easily be realized in practice.

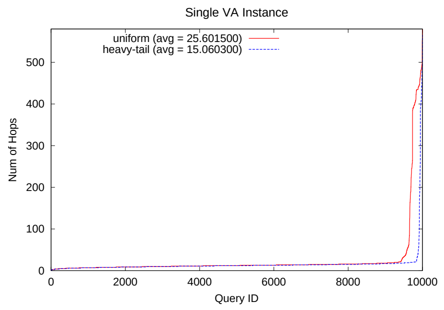
3 Evaluation Study

We conducted a preliminary study to evaluate the performance of PUB-2-SUB. This study was based on a simulation on a 1000-node network with 2766 links, whose topology was a Waxman uniform random graph generated with the BRITE generator [23]. A random node was chosen to be the root for the VA instance. Ten random choices for this root node were used with the simulation and the results averaged over these ten choices are discussed in this section.

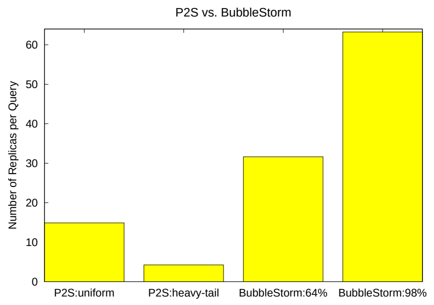
An event was represented as a k-bit string and a query an arbitrary interval of k-bit strings. A query or event was initiated by a random node chosen uniformly. To cover a large domain of possible events, we set k to 50 bits, thus able to specify



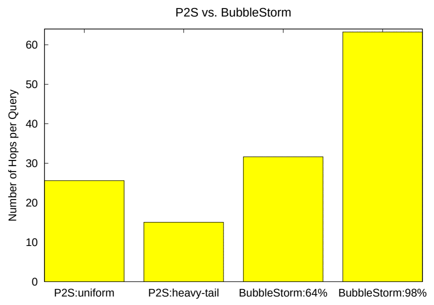
(a) Storage



(b) Communication



(c) Storage: comparison



(d) Communication: comparison

Fig. 2. Query subscription costs

up to 2^{50} different events. From this domain, 10,000 events were chosen uniformly in random. The subscription load consisted of 10,000 queries, each having a range chosen according to the following Zipf's law: in the set of possible ranges $\{2^0, 2^1, ..., 2^{49}\}$, range 2^i is picked with probability $\frac{1/i^\alpha}{\sum_{j=1}^{50}(1/j^\alpha)}$. We considered two models: $\alpha = 0$ (uniform distribution) and $\alpha = 0.8$ (heavy-tail distribution where a vast majority of the queries are specific).

We evaluated PUB-2-SUB in the following aspects: subscription efficiency, notification efficiency, notification delay, and failure effect. We also compared PUB-2-SUB to two versions of BubbleStorm [19] – a recent search technique designed for unstructured P2P networks: (1) BubbleStorm-64%: each query or event is sent to $\sqrt{n}$ nodes, resulting in a 64% query/event matching success rate (when there is no failure), and (2) BubbleStorm-98%: each query or event is sent to $2\sqrt{n}$ nodes, resulting in a 98% success rate (when there is no failure).

3.1 Subscription Efficiency

This efficiency is measured in terms of the storage cost and the communication cost. The storage cost is computed as the number of nodes that store a given query, and the communication cost as the number of hops (nodes) that have to forward this query during its subscription procedure. Figure 2(a) and Figure 2(b) show these costs respectively for every query, which are sorted in the non-decreasing order. It is observed for either cost that all queries result in a small cost except for a very few with a high cost. These high-cost queries are those with long ranges. As such they intersect with the zones of many nodes and thus have to travel more to be stored at these nodes. Despite so, on average, a query is replicated at only 15 nodes (uniform case) and 4.3 nodes (heavy-tail case), resulting in a communication cost of 25.6 hops (uniform case) and 15 hops (heavy-tail case).

These costs are much lower than that incurred by BubbleStorm. Figure 2(c) shows that BubbleStorm-64% stores an average query at 33 nodes, more than twice the storage cost of PUB-2-SUB (uniform) and eight times the cost of PUB-2-SUB (heavy-tail). The storage cost of BubbleStorm-98% is even higher. In comparison on the communication cost, as seen in Figure 2(d), a query in BubbleStorm-64% and BubbleStorm-98% has to travel 33 hops and 66 hops, respectively, which are also higher than the communication cost of PUB-2-SUB.

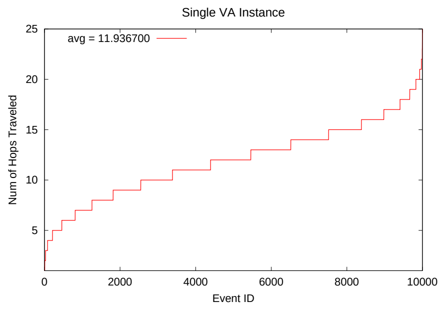
3.2 Notification Efficiency

This efficiency is measured in terms of the communication cost and the computation cost. The communication cost is computed as the number of hops (nodes) that have to forward a given event during its publication procedure, and the computation cost as the number of queries evaluated to match this event.

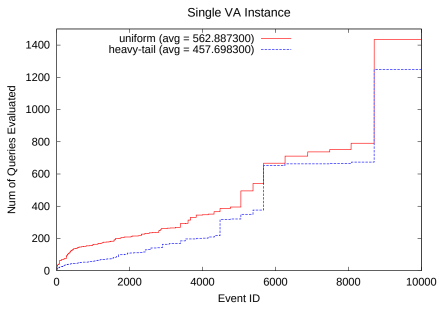
Because an event is routed based on the nodes' VAs, its communication cost is independent of the query model used, uniform or heavy-tail. Figure 3(a) shows that this cost is distributed normally from zero hop (best-case) to 25 hops (worst-case), having an average of 12 hops. The communication cost is also much lower (by approximately three times at least) when compared to BubbleStorm (Figure 3(c)). Together with the study on the subscription efficiency it is evident that PUB-2-SUB clearly outperforms BubbleStorm in both storage cost and communication cost. In terms of computation cost, Figure 3(b) shows that in the worst case about 1400 queries are evaluated to find all those matching a given event; i.e., only 14% of the entire query population. On average, the computation cost is only 563 query evaluations (uniform case) and 458 query evaluations (heavy-tail case), corresponding to 5.63% and 4.58% of the query population, respectively.

3.3 Notification Delay

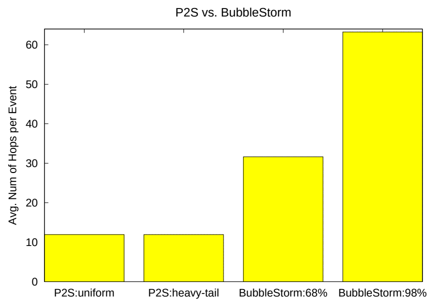
When an event is published, there may be more than one queries subscribing to this event. To represent the notification delay for each (event, query) matching pair, we compute the ratio a/b where a is the hopcount-based distance the event has to travel from the publisher node to the subscriber node and b is the hopcount-based distance directly between these two nodes. This ratio is at least 1.0 because even if the publisher knows the subscriber, it must already take b hops to send the event to the subscriber. In practice, because the publisher



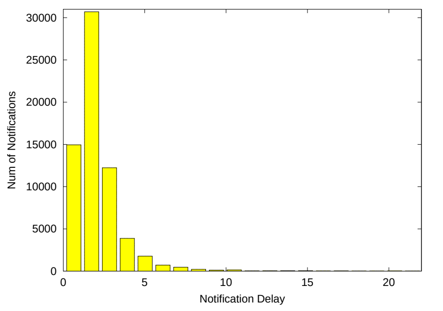
(a) Communication



(b) Computation



(c) Communication: Comparison



(d) Histogram of notification delay

Fig. 3. Event notification costs

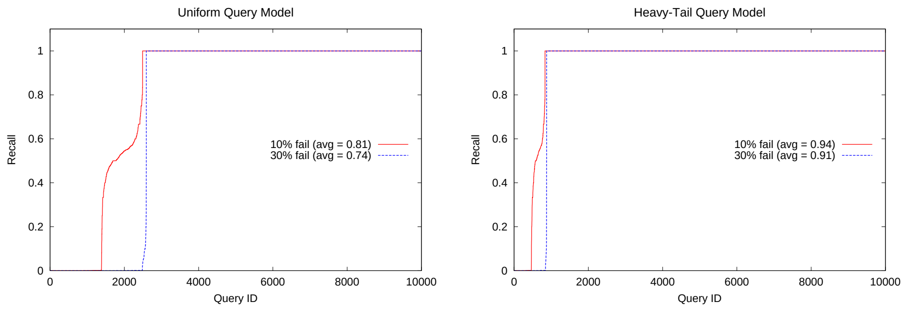
and the subscriber initially do not know each other, it is impossible to obtain a perfect 1.0 ratio.

Figure 3(d) plots the histogram of notification delay incurred by PUB-2-SUB. Approximately, 70% of the notifications have a delay not exceeding 2.0 (i.e., twice the perfect delay) and 90% have a delay not exceeding 3.0 (i.e., three times the perfect delay). Thus, despite a few (event, query) pairs with high notification delay, a vast majority of events can notify their matching queries reasonably quickly.

3.4 Failure Effect

When a node stops functioning, an event may fail to notify its subscribers. To evaluate the failure effect, we compute "recall" – the percentage of the returned events that match a given query out of all the matching events. We consider the case where 10% of the nodes fail simultaneously and the case where 30% fail.

Figure 4(a) shows the results for the uniform-query-model case, where it is observed that 75% of the queries are successfully notified by all the matching events (i.e., recall = 100%) even when 30% of the nodes fail. The difference between the 10%-fail case and the 30%-fail case is that in the latter case most



(a) Uniform query model (b) Heavy-tail query model

Fig. 4. Effect of Failures: 10% of nodes fail and 30% of nodes fail

of the remaining queries (the remaining 25%) fail to receive any matching event while in the former case about half of the queries do not receive any matching event and the other half receiving at least some portion of the matching events. On average, the recall for the 10%-fail case is 81%, and for the 30%-fail case, 74%. Higher recall is obtained when the query model is heavy-tail (see Figure 4(b)). The average recall is 91% when 30% of the nodes fail and 94% when 10% fail. The results are encouraging because in practice the query range should follow the heavy-tail model more often than the uniform model. This study is demonstrative of PUB-2-SUB's sustainable effectiveness when a large portion of the network fails to operate.

4 Conclusions

We have proposed a publish/subscribe mechanism, called PUB-2-SUB, which can be integrated into any unstructured P2P network. Using PUB-2-SUB, any number of content-based publish/subscribe applications can be deployed simultaneously. Unlike the gossip-based approach previously recommended for unstructured networks, the proposed technique is based on directed routing and incurs less storage and communication costs. This is evident in an evaluation study in which PUB-2-SUB is compared to a representative technique of the other approach. It is also found that our technique results in low computation cost and low notification delay and remains highly effective in cases when many nodes in the network stop to function.

We do not recommend PUB-2-SUB for use in highly dynamic networks with the nodes being on and off frequently. Instead, PUB-2-SUB works best for P2P-based cooperative networks in which the nodes are supposed to be functional most of the time and failures should not happen too often. Thus, data grid networks and institutional collaborative networks can take full advantage of the proposed technique. The work described in this paper remains preliminary.

More evaluation is needed for our future work in which we will also include investigation into better methods addressing failures and load balancing.

Acknowledgment

This work was funded in part by the US National Science Foundation under Grants CNS-0615055 and CNS-0753066 and by the University of Massachusetts under the Proposal Development Grant. The authors are thankful for these supports.

References

1. Sun, X., Liu, J., Yao, E., Chen, X.: A scalable p2p platform for the knowledge grid. IEEE Trans. on Knowl. and Data Eng. **17**(12) (2005) 1721–1736
2. Teranishi, Y., Tanaka, H., Ishi, Y., Yoshida, M.: A geographical observation system based on p2p agents. In: PERCOM '08: Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications, Washington, DC, USA, IEEE Computer Society (2008) 615–620
3. Shalaby, N., Zinky, J.: Towards an architecture for extreme p2p applications. In: Parallel and Distributed Computing and Systems Conference (PDCS), Cambridge, MA (November 2007)
4. Cai, M., Frank, M., Chen, J., Szekely, P.: Maan: A multi-attribute addressable network for grid information services. In: GRID '03: Proceedings of the 4th International Workshop on Grid Computing, Washington, DC, USA, IEEE Computer Society (2003) 184
5. Hanson, E.N., Carnes, C., Huang, L., Konyala, M., Noronha, L., Parthasarathy, S., Park, J.B., Vernon, A.: Scalable trigger processing. In: Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia, IEEE Computer Society (1999) 266–275
6. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: Niagaracq: a scalable continuous query system for internet databases. SIGMOD Rec. **29**(2) (2000) 379–390
7. Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., Shasha, D.: Filtering algorithms and implementation for very fast publish/subscribe systems. In: SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (2001) 115–126
8. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in communications (JSAC) **20**(8) (2002) 1489–1499
9. Gupta, A., Sahin, O.D., Agrawal, D., Abbadi, A.E.: Meghdoot: content-based publish/subscribe over p2p networks. In: Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2004) 254–273
10. Terpstra, W.W., Behnel, S., Fiege, L., Zeidler, A., Buchmann, A.P.: A peer-to-peer approach to content-based publish/subscribe. In: DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems, New York, NY, USA, ACM Press (2003) 1–8
11. Aekaterinidis, I., Triantafillou, P.: Internet scale string attribute publish/subscribe data networks. In: CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management, ACM Press (2005) 44–51

12. Tran, D.A., Nguyen, T.: Publish/subscribe service in can-based p2p networks: Dimension mismatch and the random projection approach. In: IEEE Conference on Computer Communications and Networks (ICCCN '08), Virgin Island, USA, IEEE Press (August 2008)
13. Bianchi, S., Felber, P., Gradinariu, M.: Content-based publish/subscribe using distributed r-trees. In: Euro-Par. (2007) 537–548
14. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: ACM SIGCOMM, San Diego, CA (August 2001) 161–172
15. Stoica, I., Morris, R., Karger, D., Kaashock, M., Balakrishman, H.: Chord: A scalable peer-to-peer lookup protocol for internet applications. In: ACM SIGCOMM, San Diego, CA (August 2001) 149–160
16. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany (November 2001) 329–350
17. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., , Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **22**(1) (January 2004)
18. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Communications of the ACM **33** (1990) 668–676
19. Terpstra, W.W., Kangasharju, J., Leng, C., Buchmann, A.P.: Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In: SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2007) 49–60
20. Wong, B., Guha, S.: Quasar: A Probabilistic Publish-Subscribe System for Social Networks. In: Proceedings of The 7th International Workshop on Peer-to-Peer Systems (IPTPS '08), Tampa Bay, FL (February 2008)
21. Gkantsidis, C., Mihail, M., Saberi, A.: Random walks in peer-to-peer networks: algorithms and evaluation. Perform. Eval. **63**(3) (2006) 241–263
22. Tran, D.A., Pham, C.: Enabling publish/subscribe services in cooperative p2p networks. Technical Report, University of Massachusetts Boston (February 2009)
23. Medina, A., Lakhina, A., Matta, I., Byers, J.: Brite: An approach to universal topology generation. In: MASCOTS '01: Proceedings of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Washington, DC, USA, IEEE Computer Society (2001) 346