SCIENCE CHINA
Information Sciences

Feverfew: a scalable coverage-based hybrid overlay for Internet-scale pub/sub networks

XingKong MA, YiJie WANG and WeiDong SUN

Citation: SCIENCE CHINA Information Sciences **57**, 052103 (2014); doi: 10.1007/s11432-013-4923-5

View online: http://engine.scichina.com/doi/10.1007/s11432-013-4923-5

View Table of Contents: http://engine.scichina.com/publisher/scp/journal/SCIS/57/5

Published by the Science China Press

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2014, Vol. 57 052103:1–052103:14
doi: 10.1007/s11432-013-4923-5

Feverfew: a scalable coverage-based hybrid overlay for Internet-scale pub/sub networks

MA XingKong, WANG YiJie* & SUN WeiDong

National Key Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073*, China*

Received May 29, 2013; accepted June 28, 2013; published online January 24, 2014

Abstract The publish/subscribe (pub/sub) paradigm is a popular communication model for data dissemination in large-scale distributed networks. However, scalability comes with a contradiction between the delivery latency and the memory cost. On one hand, constructing a separate overly per topic guarantees real-time dissemination, while the number of node degrees rapidly increases with the number of subscriptions. On the other hand, maintaining a bounded number of connections per node guarantees small memory cost, while each message has to traverse a large number of uninterested nodes before reaching the subscribers. In this paper, we propose Feverfew, a coverage-based hybrid overlay that disseminates messages to all subscribers without uninterested nodes involved in, and increases the average number of node connections slowly with an increase in the number of subscribers and nodes. The major novelty of Feverfew lies in its heuristic coverage mechanism implemented by combining a gossip-based sampling protocol with a probabilistic searching protocol. Based on the practical workload, our experimental results show that Feverfew significantly outperforms existing coverage-based overlay and DHT-based overlay in various dynamic network environments.

Keywords publish/subscribe, coverage-based, data dissemination, topic-based, gossip

Citation Ma X K, Wang Y J, Sun W D. Feverfew: a scalable coverage-based hybrid overlay for Internet-scale pub/sub networks. Sci China Inf Sci, 2014, 57: 052103(14), doi: 10.1007/s11432-013-4923-5

1 Introduction

Publish/subscribe (pub/sub) systems are widely used to disseminate data in a decoupled fashion. Senders (publishers) produce and publish their messages (events) through logical channels; receivers (subscribers) register their interests (subscriptions) and receive messages from the channels that they are interested in; the pub/sub systems maintain the logical channels and dispatch messages to all interested subscribers. Because of the decoupled fashion, publishers and subscribers exchange information without directly knowing each other, thus enabling the design of a simple interface and allowing the pub/sub systems to reach Internet-scale. Pub/sub systems are classified as either content-based or topic-based. In the content-based model, each message has multiple dimensions and subscribers express their interests by specifying conditions for the content of messages they want to receive. In the topic-based model, messages are grouped in topics and subscribers declare their interests by specifying the unique topic id related to the

*Corresponding author (email: wangyijie@nudt.edu.cn)

messages. Because of its simplicity, the topic-based model is widely applied to many large-scale and loosely coupled applications including news syndication, social networks and IPTV. Topic-based pub/sub systems are the subject of this paper.

The rapid growth of subscriptions and nodes requires the pub/sub system to be a scalable service. Each message should be delivered to its corresponding subscribers with low latency and traffic cost when the topics and nodes expand to Internet-scale. Recently, cloud computing environments and distributed computing environments with high performance [1] have provided great opportunities to meet the requirements of complex computing and high speed communication [2–4]. Amazon SNS[1)], as an example, provides a real-time topic-based pub/sub service that finds users interested in different events. Events are broadcast to their corresponding subscribers by the servers of data centers. Increasing the number of servers allows the server/client model to be expanded to large-scale. However, this model results in a high cost of providing servers. Moreover, the broadcasting routing scheme leads to high routing latency as the number of events and subscribers increase. Therefore, many works focus on P2P-based overlays to design scalable pub/sub systems.

Existing P2P-based systems seek to provide a scalable pub/sub service employing two different solutions. One solution is to maintain a connected overlay for each topic, where a node only receives the events that it has subscribed to, such as Rappel [5], Tera [6]. Although uninterested nodes do not participate in the event routing, the node degrees may grow rapidly with the number of topics subscribed to, which results in a large memory cost.

The other solution is to manage all subscriptions by exploiting a single overlay, where each node keeps a bounded number of connections, such as Vitis [7], Quasar [8], Scribe [9], and PeerChatter [10]. These systems use a gossip-based overlay or structured overlay to efficiently maintain all node connections. Nevertheless, a single overlay leads to extra delivery latency and traffic overhead, because each message has to frequently traverse a large number of nodes that are not interested in it. With the rapidly increasing number of nodes, high delivery latency and traffic overhead may render the system poorly scalable.

In this paper, we introduce Feverfew, a hybrid overlay for the topic-based pub/sub network, which constructs a separate overlay per topic. Although each node maintains connections to each topic subscribed to, we show that Feverfew possesses both advantages of the aforementioned solutions: low delivery latency without the involvement of uninterested nodes, a small average node degree increasing slowly with the number of subscriptions, and low traffic cost of maintaining multiple topic overlays.

Our proposal is inspired by a random graph argument [11] and a gossip-based delivery argument [12]. The first declares that the disappearance of isolated vertices has a sharp threshold of $p_n = \ln n/n$ in a random graph, where n is the network size and p_n is the probability of a link between any pair of nodes. The second claims that a node with $O(\ln N)$ random connections employing a simple *push&pull* scheme informs all other nodes in time $\log_3 n + O(\ln \ln n)$ by sending $O(n \ln \ln n)$ messages with high probability.

The innovation of Feverfew is the combination of both aforementioned arguments in a topic-based pub/sub setting. In general, the rationale criterion of the local neighbor selection for each node is to guarantee that each topic subscribed to is *ln-covered*, i.e., each subscription has at least $\ln n$ neighbors. Thus, these $\ln n$ connections per topic guarantee strong connectivity of the corresponding topic overlay and real-time dissemination according to the aforementioned arguments. The resulting structure of the connections to subscribed topics per node resembles the crown of a composite plant, which has multiple layers with different sizes, thus inspiring the name Feverfew.

To enable each subscription to be covered $\ln n$ times with a small number of connections, Feverfew nodes run a gossip-based hybrid sampling service to explore the subscription similarities and select the nodes with which they share the most subscriptions as neighbors. As shown in Figure 1, more than 70% of topics have size not greater than 16 for 10^5 users on the social networks of Facebook[2)] and Douban[3)]. To reach *ln-covered*, the average node degrees of these small topics are fewer than four. Although the strongly correlated subscriptions reduce the total number of connections greatly, there is no guarantee

1) http://aws.amazoncom/sns/.

2) http://www.facebook.com.
3) http://www.douban.com.

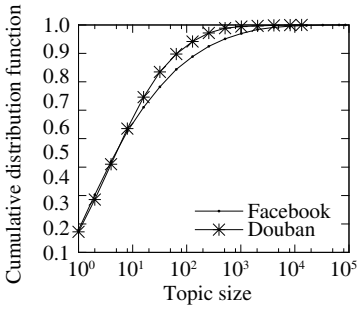

Figure 1 Cumulative distribution function of topic size for 10^5 users on Facebook and Douban.

that all nodes per topic are connected together in the limited time, especially in the case of the topics with fewer subscribers. Therefore, the challenge of Feverfew is how to quickly cover each subscription $\ln n$ times using small node degrees.

To this end, we propose a heuristic coverage mechanism implemented by combining a gossip-based sampling protocol with a probabilistic searching protocol. To reduce the average node degree, each node uses the gossip-based sampling protocol to explore subscription similarity among nodes. The probabilistic searching protocol enables each node to locate at least one matching node in the limited time, which greatly reduces the convergence time of maintaining the topic overlays.

We evaluate the performance of Feverfew via large-scale simulations with real world subscriptions from the social network Douban. We compare Feverfew with two typical solutions: 1) constant coverage-based systems that connect each topic overlay with a constant node degree in unstructured overlays and 2) structured overlays that efficiently route the subscription to the rendezvous node using a Distributed Hash Table (DHT). Compared with these solutions, our results show that Feverfew maintains smaller topic overlay diameters by having fewer connections, and locates nodes with low latency and traffic cost. All these results are achieved in a very large setting involving up to 10^4 nodes, 31289 topics, and 48 subscriptions per node.

The rest of this paper is organized as follows: In Section 2, we introduce the related works and the position of Feverfew. In Section 3, we present the details and the key ideas of our proposal. We evaluate the solution through extensive simulations in Section 4, and conclude the work in Section 5.

2 Related work

The broker network, where subscriptions and events are matched by brokers, is one of the main choices available to actual topic-based pub/sub applications, e.g., Gryphon [13], SIENA [14]. According to the topology of the brokers formed, there are two solutions, tree-based and mesh-based solutions. In a tree-based topology, publishers diffuse the events from the root broker of the tree, while subscribers receive the interested events from the leaf brokers of the tree. In a mesh-based topology, a broker can connect with any other broker and there is a more effective load balance with respect to the brokers in a tree-based topology. The broker network provides simple interfaces with which to design pub/sub systems. However, its scalability relies on the performance of the brokers.

In scaling the pub/sub system to Internet-scale, P2P networks construct structured or unstructured overlays to obtain an effective load balance at each node. Scribe [9] and CAN [15] are typical examples of structured overlay networks using DHTs to match subscriptions and events efficiently. In Scribe, each topic is mapped to a rendezvous node that manages subscriptions and diffuses events to the target subscribers. Leveraging the lookup procedure of Pastry, each rendezvous node constructs the corresponding topic overlay as a multicast tree to ensure real-time dissemination. However, the multicast tree may contain uninterested nodes that forward the events, and the rendezvous node has to handle all the events and subscriptions of the specific topic even if it is uninterested in that topic. To avoid messages being forwarded to unsubscribed nodes, Feverfew constructs a strongly connected overlay for each topic with a

small number of connections.

Rappel [5] and Tera [6] fall into the category of unstructured overlays, which follow gossip-based approaches. Rappel maintains a connected overlay among subscribers by leveraging both interest locality and network locality, where a spanning tree for dissemination of each topic is constructed taking a bottom-up approach. Tera maintains a random graph for each topic employing Cyclon [16]. The major novelty of Tera lies in the outer-cluster routing mechanism for subscribing and diffusing events using multiple random walks. Although a node in Rappel and Tera only relays the events it has subscribed to, a large number of connections for each topic leads to a high memory cost and makes the systems poorly scalable. In contrast, Feverfew reduces the average node degree by exploiting subscription similarities.

In the category of hybrid overlays, Quasar [8] and Vitis [7] take the advantages of both structured and unstructured overlays, and manage all the subscriptions using a bounded number of connections. Each node in Quasar has an attenuated bloom filter to collect all the neighbors up to a few hops away. Leveraging parallel random walks, each node relays messages to a directed node or random node according to the attenuated bloom filter. In spite of the low bounded number of connections, Quasar does not guarantee that the subscribers can receive each event even in a static environment. In addition, the parallel random walks of Quasar bring high traffic overhead, especially when the size of the topic is small. To achieve both bounded the node degree and low traffic cost, Vitis nodes run T-Man [17] to select nodes with similar subscriptions and construct the small world overlay. In Vitis, all subscribers of a topic may fall into multiple clusters, each of which elects a gateway node. These gateway nodes connect with each other via the rendezvous node. Using the navigable small world overlay, any rendezvous node can be found in $O(\log^2 n)$ hops, where n is the network size. Nevertheless, this leads to large delivery latency to small topics in the practical running of pub/sub applications. To implement real-time and reliable data dissemination, Feverfew employs a gossip-based *push&pull* scheme to deliver messages in each topic overlay.

Regardless of how the overlay is constructed, coverage-based overlays are proposed to manage all topics using a small number of connections. Feverfew belongs to this category. To guarantee all nodes interested in topic t are connected, given a collection of nodes and their subscriptions, Min-TCO [18] connects the nodes using the minimum possible number of edges. In contrast, MinMax-TCO [19] connects the nodes into a graph which has least possible maximum degree. Both cases have small average node degrees, however, the topic overlays can be divided into pieces with high probability in the continually fluctuating network environment. Araneola [20] claimed that the overlay network exhibits robust connectivity, logarithmic diameter in the number of nodes, and churn resistance if each node has three random links to other nodes. According to this design principle, each subscription in Spidercast [21] has four nodes as neighbors that also have the subscription. Although Spidercast offers an average node degree growing sub-linearly as the number of topics increases, each node needs to know the identities and interests of at least 5% of other nodes, which may result in long sampling latency for the large network size. Compared with Spidercast, Feverfew locates nodes in a bounded number of hops employing a probabilistic sampling service.

3 Feverfew

Recall that our objective is to make each subscription *ln-covered*. That is, for each topic t in which node p is interested, p tries to maintain connections to $\ln n$ other nodes that also have subscribed to topic t. To this end, each node in Feverfew maintains a coverage-based hybrid overlay by periodically running two protocols. One is a gossip-based sampling protocol GossipSample (Subsection 3.1), which exploits nodes' semantic similarity to reduce the total number of connections. The other is a probabilistic sampling protocol BubbleSample (Subsection 3.2), which samples nodes for the topics of small size to reduce the convergence time.

Each node p in Feverfew maintains a bounded-size routing table, called the *ln-covered view*, which covers at least $\ln n$ connections for each topic t that p is interested in, where n is the size of t. The entries of the *ln-covered view* are selected from either similar connections or agent connections. The similar

connections provide highly correlated nodes via GossipSample, and we refer to this type of connections as a *similar view*. The agent connections provide correlated nodes subscribed to small size topics via BubbleSample, and we refer to this type of connections as an *agent view*.

According to the random graph argument [11] and the gossip-based delivery argument [12], *ln-covered* connections guarantee each topic overlay is connected strongly with low delivery latency. Moreover, according to the power law distribution of the topic size and the correlated subscriptions in practical applications, the average node degree of Feverfew is decreased greatly by the *ln-covered* criterion.

3.1 GossipSample

Algorithm 1: GossipSample-Active Thread

```
1: q ← getRandomNode(similar view)
2: buffer ← GetRandomView()                                   // provided by Cyclon
3: buffer.merge(similar view)
4: Send buffer to q
5: Receive newBuffer from q
6: buffer.merge(newBuffer)
7: similar view ← SelectSimilarNeighbors(buffer,restSub(myNode))   // (2)
8: restSub(myNode) ← UpdateUnFilledSubscriptions(ln-covered view)
9: if currentTime%Δ==0 then
10:    ln-covered view ← SelectlnCoveredNeighbors(similar view)    // Algorithm 3
11:    clear(similar view)
12: end if
```

Inspired by the idea of T-Man [17], Feverfew uses a gossip-based sampling protocol GossipSample to reduce the average node degree by exploring subscription similarities. All nodes are organized into two layers in GossipSample. In the lower layer, node p uses Cyclon [16] to sample unbiased nodes, which are cached into a *random view*. To get fresh nodes, pair nodes in Cyclon periodically exchange some items of their *random views*, and the number of exchanged items is called the shuffle length. Meanwhile in the upper layer, the basic idea is to find the similar nodes quickly according to the subscription correlation. The similarity is defined as a pair-wise utility function:

$$utility(p,q) = \frac{\sum_t (t \in (sub(p) \cap sub(q)))}{\sum_t (t \in (sub(p) \cup sub(q)))}, \tag{1}$$

where $sub(p)$ indicates the set of topics that node p is interested in. Owing to the bounded number of items in *similar view*, many subscriptions of node p cannot be *ln-covered*. Additionally, the same items will be obtained from the *similar view* if we periodically sample similar connections using (1). That is, the *ln-covered view* cannot sample more fresh nodes from the *similar view*. A simple solution is to extract p's subscriptions that have not been *ln-covered* by the *similar view* as the criterion of the similarity. The new utility function is

$$utility(p,q)_2 = \frac{\sum_t (t \in (restSub(p) \cap sub(q)))}{\sum_t (t \in (restSub(p) \cup sub(q)))}, \tag{2}$$

where $restSub(p)$ indicates the set of p's subscriptions that have not been *ln-covered*. In the upper layer, each node p periodically exchanges its *similar view* with node q, uniformly chosen from p's *similar view*. Node p, then, merges its *similar view* with both q's *similar view* and p's *random view*. Next, from this resulting list, node p selects node q with maximum value of $utility(p,q)_2$ until the *similar view* is filled up (Algorithm 1, lines 3–7). Afterward, node p updates $restSub(p)$ immediately to get more fresh nodes (Algorithm 1, line 8). Similar processes are executed at node q (Algorithm 2).

In addition, the *ln-covered view* is updated at intervals Δ (Algorithm 1, lines 9–12). To minimize the items of the *ln-covered view*, each node p updates the *ln-covered view* employing a greedy method (Algorithm 3). Specifically, the *ln-covered view* and the *similar view* are merged into a temporary list. The *ln-covered view* is then cleared up. Periodically, from this temporary list, the node q with maximum

Algorithm 2: GossipSample - Passive Thread

1: Receive buffer from the node p
2: newBuffer ← getSampledNodes()
3: newBuffer.merge(*similar view*)
4: Send newBuffer to p
5: newBuffer.merge(buffer)
6: *similar view* ← SelectSimilarNeighbors(buffer, $restSub(myNode)$)
7: $restSub(myNode)$ ← UpdateUnFilledSubscriptions(*ln-covered view*)

Algorithm 3: SelectlnCoveredNeighbors

1: buffer = null
2: buffer.merge(*similar view*)
3: buffer.merge(*ln-covered view*)
4: clear(*ln-covered view*)
5: **repeat**
6: q ← GetMaximumSimilarNode(buffer,$restSub(myNode)$)
7: **if** $utility(myNode, q)_2 == 0$ || buffer.isEmpty() **then**
8: **return**
9: **end if**
10: buffer.delete(q)
11: (*ln-covered view*).merge(q)
12: $restSub(myNode)$ ← UpdateUnFilledSubscriptions(*ln-covered view*)
13: **until** all the subscriptions of *myNode* is *ln-covered*

value of $utility(p, q)_2$ is inserted into p's *ln-covered view*. This process takes place until all p's subscriptions are *ln-covered* or $utility(p, q)_2 = 0$ or the temporary list is empty (Algorithm 3, lines 5–13).

For the subscription of a large topic, it is easy to reach *ln-coverage* using GossipSample. We give the following theorem.

Theorem 1. All the subscriptions in topic t can be ln-covered at most $\frac{N}{(c1+c2)} \ln \frac{n}{n-\ln n}$ cycles by GossipSample, if each node s samples $c1$ random nodes and $c2$ similar nodes per cycle, where n is the size of topic t and N is the network size.

Proof. As a similar node has a higher probability to be inserted into the *similar view* than a random node, the number of nodes inserted into *similar view* from $c1$ random nodes and $c2$ similar nodes is higher than that from $c1 + c2$ random nodes. Over n boxes, with m balls distributed uniformly at random, the expectation of the number of empty boxes is $ne^{-m/n}$. According to this conclusion, and supposing node s samples $c1 + c2$ random nodes per cycle, the total number of sampled nodes is $N(1 - e^{-(c1+c2)T/N})$ in T cycles. Because the matching probability is n/N for each random node, the expected number of matching nodes is $n(1-e^{-(c1+c2)T/N})$. Based on the definition of *ln-coverage*, $n(1-e^{-(c1+c2)T/N})$ should be not less than $\ln n$. Thus, we have $T \geqslant \frac{N}{(c1+c2)} \ln \frac{n}{n-\ln n}$.

According to the result of Theorem 1, T decreases as the topic size n grows; however, the deficiency of GossipSample is the high latency of sampling neighbors from small topics in a large-scale network. This is because the subscriber to a small topic has a small probability of finding other subscribers. An extreme case is that a topic t is subscribed to by only two nodes: p and q. That is, the size of topic t is 2. Suppose that p and q are both only interested in t. This means that p and q can only collide with each other using uniform sampling in the lower layer of GossipSample. If each node samples c uniform nodes periodically, for network size n, the expected number of cycles of colliding q and p is n/c which is huge for large n and small c.

3.2 BubbleSample

To maintain each topic overlay with small latency, we propose a novel sampling service BubbleSample for the topics with few subscribers. For the request node p's subscription s that has not been *ln-covered* in a long time, the core idea of BubbleSample is to buffer s to a group of helpers, called *agent nodes*, which help sample matching nodes. Once a matching node is found by any *agent node*, both the request node p and the matching node will be notified immediately.

More specifically, the request node p's subscriptions that have not been *ln-covered* are buffered into *restSub(p)* of (2). At the beginning, *restSub(p)* contains all node p's subscriptions and the corresponding traffic overhead is very large if we run BubbleSample. Fortunately, the subscriptions with a large topic size can quickly be *ln-covered* via GossipSample (Subsection 3.1). To reduce traffic overhead, BubbleSample is started after $\frac{N}{(c1+c2)} \ln \frac{n}{n-1-\ln n}$ cycles (Theorem 1) to ensure all subscriptions with topic size greater

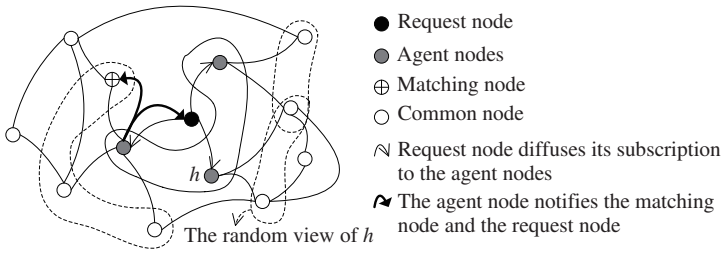

Figure 2 The request node diffuses its subscription that has not been *ln-covered* to a group of *agent nodes*, which helps the request node find a matching node from their *random views*. Once a matching node is found, both the request node and matching node will be notified immediately.

than n have been *ln-covered*. Then, as shown in Figure 2, each subscription in node p's *restSub(p)* is diffused to a group of *agent nodes* (Algorithm 4). The *agent node* h caches the received subscription into its *agent view*. Periodically, each subscription of h's *agent view* is compared with the subscriptions of each sampled node in h's *random view*. Then, for each node q in the *random view*, once p's subscription s in h's *restSub(h)* is also subscribed to by a matching node q, node h will notify both p and q. The pseudo code for obtaining matching nodes is shown in Algorithm 5.

Note that each message in BubbleSample can be disseminated as a piggyback on the shuffle message of Cyclon [16], which samples random nodes as the lower layer service of GossipSample. Therefore, the process of diffusing *agent nodes* in BubbleSample will not bring extra traffic overhead.

Algorithm 4: ForwardRestSub

```
msg ← getMessageFromNeighbor()
if msg.TTL ≠ 0 then
    for i = 0 to msg.TTL do
        p ← getRandomNode(random view)
        newMsg.copy(msg)
        newMsg.TTL = msg.TTL-i − 1
        send newMsg to p
    end for
end if
```

Algorithm 5: GetMatchingNode

```
for all t in the agent view do
    for all r in random view do
        if node r subscribes the subscription t then
            send r to t.myNode
            send t.myNode to r
        end if
    end for
end for
```

A critical problem is to determine the number of agent nodes needed for each subscription, or the initialization value of time to live (TTL) in Algorithm 4. The results are presented as follows.

Theorem 2. For node p's subscription s to topic t, BubbleSample guarantees that there is no one matching node of subscription s with probability smaller than $1/N$ in $\log_2(-N\ln(1-M/N))$ cycles if the number of agent nodes M is not less than $-\frac{N}{c}\ln(1-\ln N/[N(1-\mathrm{e}^{-cn/N})])$, where N is the network size, n is the size of topic t, and c is the shuffle length per cycle in Cyclon.

Proof. Over n boxes, distributing m balls uniformly at random, the expectation of the number of empty boxes is $n\mathrm{e}^{-m/n}$. According to this result, in one cycle, the total expected number of sampled nodes is $N(1-\mathrm{e}^{-cM/N})$ for all the *agent nodes*. Similarly, all nodes interested in topic t sample $N(1-\mathrm{e}^{-cn/N})$ nodes per cycle.

Over n boxes, distributing r red balls and g green balls uniformly at random, the chance that no box has both a green and red ball is less than $\mathrm{e}^{-rg/n}$. According to this conclusion, the probability that no one matching node of subscription s is found must satisfy: $f(s) < \mathrm{e}^{-N^2(1-\mathrm{e}^{-cM/N})(1-\mathrm{e}^{-cn/N})/N}$. Therefore, $f(s)$ will be smaller than $1/N$, if $\mathrm{e}^{-N(1-\mathrm{e}^{-cM/N})(1-\mathrm{e}^{-cn/N})} < 1/N$. Consequently, we have $M > -\frac{N}{c}\ln(1-\ln N/[N(1-\mathrm{e}^{-cn/N})])$.

As shown in Algorithm 4, the subscription s is diffused at an exponential rate. In TLL cycles, BubbleSample will diffuse 2^{TTL} messages, and the expected number of *agent nodes* is $N(1-\mathrm{e}^{-2^{\mathrm{TTL}}/N})$. As we need $N(1-\mathrm{e}^{-2^{\mathrm{TTL}}/N}) \geqslant M$, we have $\mathrm{TTL} \geqslant \log_2(-N\ln(1-M/N))$.

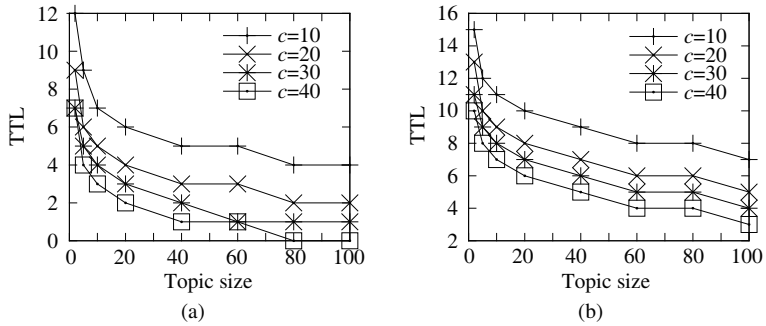

Figure 3 Expected TTLs of BubbleSample. (a) Expected TTL for different shuffle lengths when the network size is 10^4; (b) expected TTL for different shuffle lengths when the network size is 10^5.

According to Theorem 2, the number of *agent nodes* M increases with n and c, and decreases with increasing N. Figure 3 shows the expected TTLs of BubbleSample for different topic sizes, where c is the shuffle length of Cyclon. It is seen that the topic of smaller size has higher TTL because it needs to involve more agent nodes. Figure 3 also shows that the expected TTL increases slowly with the growth of network size owing to the diffusion at an exponential rate. Moreover, note that diffusing the messages does not bring extra traffic overhead because they are packed into the shuffle message of Cyclon as mentioned before.

We use anti-entropy aggregation protocol (AAP) [22] to estimate each topic size and the network size. In AAP, the nodes exchange size estimation values in pairs and reach high convergence in a small number of cycles. Nevertheless, once the topic is divided into subcomponents, each node only obtains the size of the corresponding subcomponent, and not the topic size. In particular, when the node is isolated, the subcomponent size is 1. In this situation, we suppose there is at least one neighbor for each subscriber, i.e., the estimated topic size is 2. In this case, we suppose the number of *agent nodes* M is $N-1$, and the maximum TTL is bounded by $\log_2(N \ln N)$ according to Theorem 2.

4 Experiments

We compare Feverfew with two baseline solutions. The first is a coverage-based solution that maintains each topic with a small number of connections. We choose Spidercast [21] as the candidate because of the low diameter of each topic and small average node degree. The second is a structured solution that maintains each topic employing the DHT-based routing method with a bounded number of connections per node. Both Feverfew and Spidercast are implemented on Peersim[4)], a Java open source simulation framework for the P2P protocol, while the performance of the structured solution has been proven and evaluated on Chord [23]. Since GossipSample can efficiently maintain various overlays in dynamic networks, these two baseline solutions are also evaluated using GossipSample for a fair comparison. Unless otherwise mentioned, the shuffle length of Cyclon per node is 10, and the numbers of items in *random view*, *similar view*, and *agent view* are 20, 40, and 40, respectively.

To gather the real subscription correlation, we crawled 10^5 users and their subscriptions on the popular Chinese social network Douban using five concurrent threads from April 9, 2011 to April 24, 2011. To sample unbiased nodes from the whole network, we implemented USDSG [24], which employs a random walk to sample unbiased nodes evenly in a dynamic directed graph. Because of the high memory cost of event-driven simulation on Peersim, we select 10^4 users for the experimental dataset, which contains $31,289$ topics and $486,842$ subscriptions. As show in Figure 4, the distributions of the subscription size and topic size have power law trends. The maximum topic size is 1304, and the minimum topic size is 2.

In our experiment, we measure the following metrics:

• **Convergence time.** It denotes the maximum number of hops when all subscriptions are *ln-covered*.

4) Jelasity M, Montresor A, Jesi G, et al. PeerSim: a peer-to-peer simulator. http://peersim.sourceforge.net.

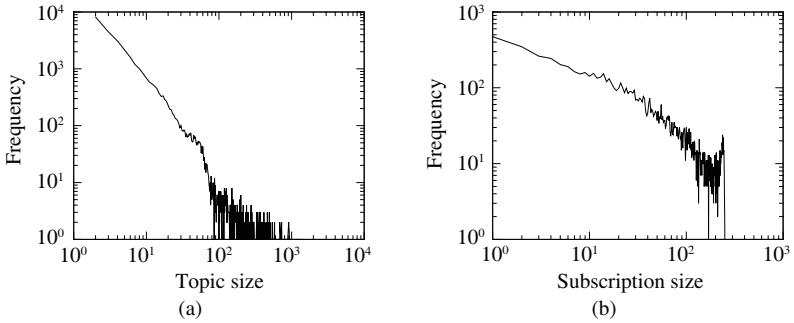

Figure 4 Dataset. (a) Topic size distribution; (b) subscription size distribution.

• **Average node degree.** It denotes the average number of items in the *ln-covered view* when all subscriptions are *ln-covered*.

• **Topic diameter.** It denotes the largest distance between any pair of nodes in one topic overlay.

• **Clustering coefficient.** It is a measure of the degree to which nodes in a graph tend to cluster together.

• **Robustness.** It denotes the completion rate and latency when delivering messages with nodes leaving and joining frequently.

4.1 Convergence time

4.1.1 *Convergence time of GossipSample*

We run GossipSample and update *ln-covered* subscriptions per cycle. In Figure 5, the x-axis denotes the interval of the topic size and the y-axis denotes the percentage of *ln-covered* subscriptions in different topic intervals. After 10 cycles, no more than 10% of the subscriptions with topic size smaller than 20 can be *ln-covered*, while such subscriptions account for more than 70% of the network. In contrast, after 30 cycles, more than 70% of such subscriptions are *ln-covered*. Additionally, Figure 5 shows that almost all subscriptions with topic size greater than 40 can be *ln-covered* after 30 cycles.

In the experiment, we choose the end time of GossipSample to be the 30th cycle which is also the starting time of BubbleSample. The maximum number of sampled nodes per cycle is 50, containing both the size of the *similar view* and the shuffle length of Cyclon. To validate the convergence, Figures 6 and 7 show the minimum cycles when all subscriptions to a topic are *ln-covered* according to Theorem 1 with network sizes of 10^4 and 10^5 respectively, where $c1$ and $c2$ respectively denote the numbers of sampled random nodes and similar nodes per cycle. According to Figure 6, all subscriptions with topic size larger than 50 should be *ln-covered* after 21 cycles; however, the experimental result shows that 1%–2% of subscriptions are still not *ln-covered*. This is because the bounded size of the *similar view* reduces the number of sampled nodes. Once node p's *similar view* is filled up, a new sampled node q will be abandoned if the similarity between p and q is not greater than any item of p's *similar view*. However, the abandoned node q may become an item of p's *ln-covered view* in the foreseeable future.

In addition, Figures 6 and 7 indicate that the subscriptions to small topics need much time to be *ln-covered*. In particular, for the large network size shown in Figure 7, the convergence time is unacceptably long. Similarly, using a weighted mix of greedy and random coverage, Spidercast manages constant neighbors for each topic to which a node has subscribed to. Consequently, each node in Spidercast will need a very long time to reach coverage of 3 or 4.

In brief, GossipSample ensures that the percentage of *ln-covered* subscriptions improves as the corresponding topic size grows. It ensures almost all subscriptions with topic size greater than 40 are *ln-covered* after 30 cycles, while there are still many subscriptions to small topics that are not *ln-covered* completely.

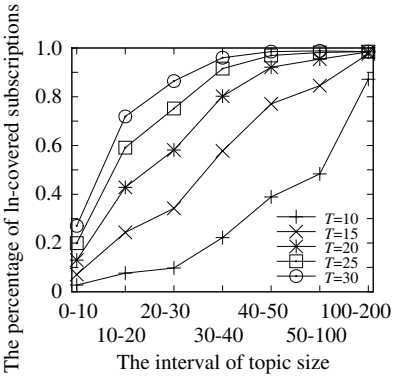

Figure 5 *Ln-covered* percentage in GossipSample for different topic sizes.

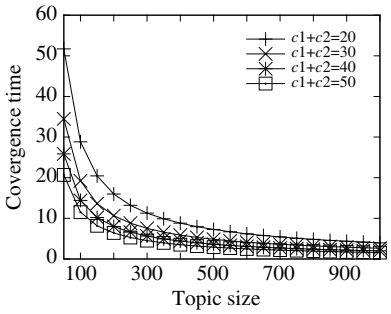

Figure 6 Convergence time according to Theorem 1 with 10^4 nodes.

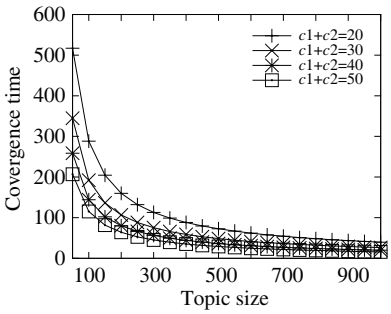

Figure 7 Convergence time according to Theorem 1 with 10^5 nodes.

Table 1 The matching time of BubbleSample and DHT

	Topic size (n)	2	10	100	200	400	600	825	1007	1110	1208	1304
$N = 10^4$	DHT	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3
	Theory	11.5	6.77	3.30	2.36	1.49	1.03	0.72	0.54	0.46	0.40	0.34
	Practical	11	8	5	4	2	2	2	1	1	1	1
$N = 10^5$	DHT	16.6	16.6	16.6	16.6	16.6	16.6	16.6	16.6	16.6	16.6	16.6
	Theory	14.7	10.4	6.88	5.87	4.88	4.31	3.86	3.59	3.46	3.34	3.24

Table 2 The average traffic overhead of BubbleSample and DHT

Topic size (n)	2	10	100	200	400	600	825	1007	1110	1208	1304
DHT	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3
Practical	12.26	4.18	6.74	6.2	5.28	8.2	4.7	6.1	6.88	7.66	7.12

4.1.2 *Matching time of BubbleSample*

As previously mentioned, the initial number of hops is the key parameter affecting the probability and latency of searching the matching node. In this experiment, we set the initial number of hops for subscriptions that have not been *ln-covered* according to Theorem 2. Each node only samples 10 random nodes in each cycle, and the simulation ends when the TTL of the diffusing message in Algorithm 4 reduces to zero. Table 1 gives the first matching times for different topic sizes and network sizes. "DHT" refers to the bound of the hops in structured overlays with N nodes; i.e., $\log_2 N$. "Theoretical" refers to the value expected according to Theorem 2. "Practical" refers to our experimental results. As shown in Table 1, BubbleSample can support more efficient sampling than the DHT-based approaches in spite of the topic size n and network size N. Because the success probability of Theorem 2 does not reach 1, the theoretical results are not larger than the practical results in most cases. In addition, the theoretical results indicate that the initial number of hops increases slowly with the network size.

Table 2 gives the average traffic costs for different topic sizes. As mentioned above, diffusing subscriptions by packing them into the shuffle process of Cyclon will not bring extra traffic cost. Consequently, the main traffic cost comes from the messages that are forwarded to the matching node and request node by the *agent nodes*. We obtain the average traffic cost by running the experiments 100 times. Table 2 shows that the average traffic overhead of BubbleSample is smaller than that for the DHT-based method. Because multiple *agent nodes* in BubbleSample may find different matching nodes, the matching process of BubbleSample is further accelerated.

In conclusion, BubbleSample guarantees all subscriptions find their matching nodes with smaller latency and traffic cost than in the case of the DHT-based methods. Meanwhile, the latency and traffic cost of searching for the matching node in BubbleSample increase slowly with increasing network size.

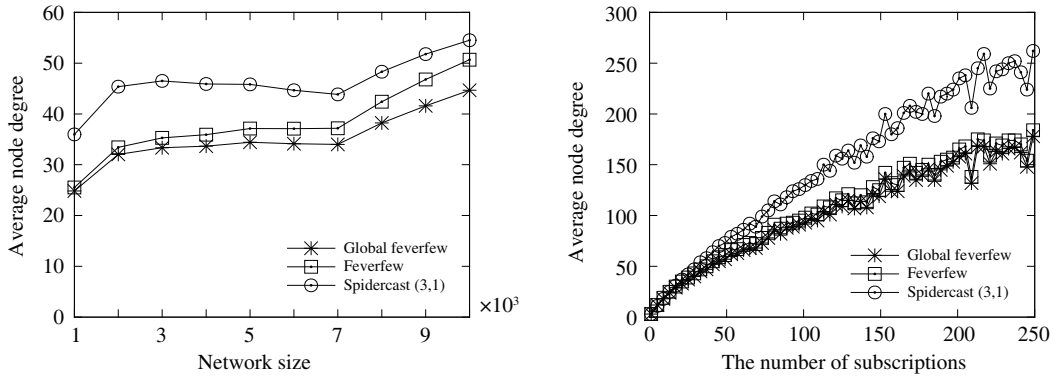

Figure 8 Average node degree. (a) Average node degree with increasing topics and network sizes; (b) average node degree distributions for different number of subscriptions with 10^4 nodes.

4.2 Average node degree

We evaluate the average node degree of Feverfew by comparing with Spidercast [21] and Global Feverfew. "Global" means each node knows all the other nodes' subscriptions. In Spidercast, each node aims for greedy 3-coverage and random 1-coverage to guarantee topic connectivity.

To evaluate the scalability, we investigate the average node degree with increasing numbers of topics and subscriptions. In the first experiment, the network size increases linearly from 1000 to 10 000 nodes in steps of 1000 nodes, and the number of topics increases from 6 109 to 31 289. As shown in Figure 8(a), the average node degree of Feverfew is close to that of Global Feverfew and smaller than that of Spidercast, because there are very few large topics and a node degree of 4 is excessive. For the second experiment, Figure 8(b) shows the average node degree distribution for different numbers of subscriptions when the network size is fixed to 10 000. The average node degree of Feverfew in Figure 8(b) is close to that of Global Feverfew and smaller than that of Spidercast. Furthermore, when all topics are *ln-covered*, the average node degree per subscription at each node in Feverfew is smaller than 0.7, which reduces memory overhead by 22% compared with the case for Spidercast.

Accordingly, Feverfew's average node degree grows sub-linearly with the increasing number of topics and subscriptions; this node degree is smaller than that of Spidercast. Compared with Global Feverfew, Feverfew samples almost the minimum number of connections to reduce the memory overhead and make the overlay more scalable.

4.3 Topic diameter and clustering coefficient

From the dataset, we sample 10 topics, whose sizes range from 10 to 1304. As shown in Table 3, all nodes in each topic are connected. Although the small topics in Spidercast [21] have a lower clustering coefficient than those in Feverfew, the topic overlays in Feverfew have the same diameter as those in Spidercast. This means that the *ln-covered view* does not increase the dissemination latency in Feverfew. Note that more than 70% of topics are smaller than 16. Thus, compared with Spidercast, Feverfew nodes have fewer connections to these topics. As the topic size increases, Feverfew's diameter is not larger than Spidercast's, and the clustering coefficient is higher than that of Spidercast. This is because each node in Feverfew has a node degree larger than the node degree of Spidercast of 4 when the topic size is larger than 55 ($\ln 55 \approx 4$).

In conclusion, each topic overlay in Feverfew has a low diameter, a large clustering coefficient and strong connectivity while requiring each node to maintain a small number of connections. These properties mean that Feverfew can implement real-time and reliable data dissemination under high churn networks.

4.4 Robustness

To study the robustness of Feverfew, we disseminate messages in a highly dynamic network by adopting a gossip-based *push&pull* method [12], in which each informed subscriber pushes and pulls unless the

Table 3 The diameter and clustering coefficient (CC) of different topics

Topic size	10	100	200	400	600	825	1007	1110	1208	1304
Feverfew diameter	2	3	3	3	3	3	3	3	3	3
Spidercast diameter	2	3	3	3	3	4	3	3	3	4
Feverfew CC	0.359	0.196	0.182	0.196	0.194	0.187	0.195	0.179	0.175	0.167
Spidercast CC	0.533	0.142	0.124	0.120	0.117	0.111	0.122	0.101	0.104	0.100

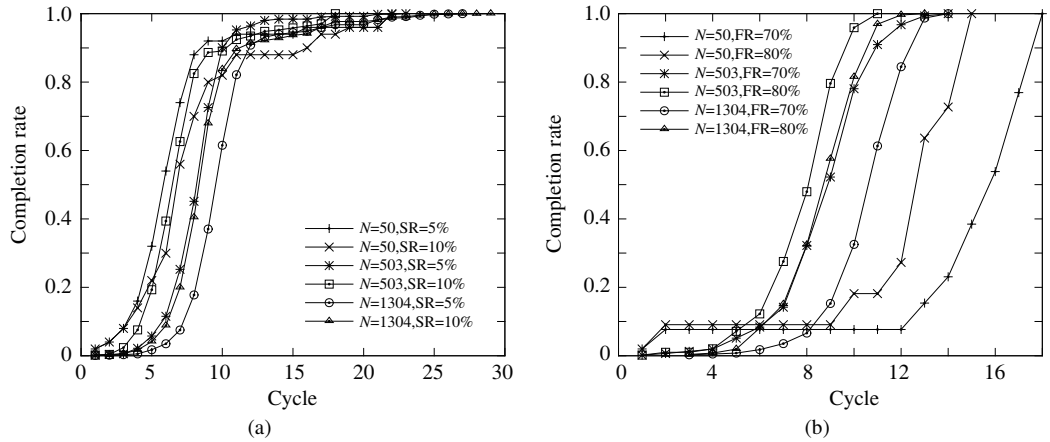

Figure 9 Completion rate of data dissemination. (a) Completion rate for different node substitution rates (SRs); (b) completion rate for different node failure rates (FRs).

age of the message is higher than $\log_3 n + O(\ln \ln n)$. We define the *completion rate* as the proportion of subscribers having received a message to the topic size. Three topics with overlay sizes of 50, 503, and 1 304 are chosen for the dissemination of messages in two different dynamic situations: 1) a fixed number of nodes are substituted in intervals and 2) large scale nodes fail instantaneously.

In the situation of node substitution, the failed nodes are selected randomly from current alive nodes and the new nodes are those that failed in the previous cycle. Figure 9(a) shows the change in the *completion rate* with time when 0.5%, 1% of nodes are substituted for different sizes of topics. As the figure shows, the early rise of the *complete rate* is more rapid than the late rise, owing to the latency of getting new neighbors for the *ln-covered view*. In addition, all nodes in each topic receive their messages in 31 cycles, which indicates that GossipSample and BubbleSample allow each topic to be strongly connected even in a continually fluctuating network.

In the situation of node failure, each subscription has *ln-covered* connections at cycle 0. Figure 9(b) shows the change in the *completion rate* for the failure of 70% and 80% of nodes in cycle 1. As the figure shows, the message can reach all subscriptions to various topics, and all the alive nodes receive the message in 18 cycles when 80% of the 1304 nodes fail. Note that the earlier stage of completion is slower than the back stage because small topics are probably divided into multiple subcomponents once the instantaneous failure happens and some subcomponents will only have one subscriber. For the topic with 50 subscribers, the *completion rate* increases quickly after 11 cycles as shown in the figure, which validates the small convergence time of maintaining the *ln-covered* view.

In conclusion, Feverfew shows high robustness and efficiency against node churn and instantaneous failures. Although small topics may be split into multiple subcomponents when a large number of nodes fail, Feverfew has a strong self-healing ability and connects these subcomponents together quickly.

5 Conclusion

We presented Feverfew, a topic-based publish/subscribe system, which scales well with the number of nodes, topics, and subscriptions. The main contribution of this paper is maintaining a *ln-covered view* for

each subscription to guarantee real-time and reliable dissemination in a continually fluctuating network environment. To maintain a *ln-covered view* with a small node degree, Feverfew adopts a gossip-based sampling protocol GossipSample to exploit semantic similarity among nodes. Furthermore, a probabilistic searching protocol BubbleSample is used to accelerate the process of constructing strongly connected overlays for small topics. Compared with the baseline model of constant coverage-based overlay, our experimental results show that Feverfew allows each topic overlay to be strongly connected with low latency and traffic overhead.

On the basis of the encouraging results of this work, we plan to evaluate several aspects of the performance of Feverfew. Firstly, we note that the number of *agent nodes* for each subscription in BubbleSample is based on the corresponding topic size. Currently, our topic size estimation method is borrowed from AAP [22], which brings extra traffic overhead for each node. Estimating the subgraph size using less traffic overhead would be significant to the scalability of the overlay. One possible method is that each node is responsible for estimating the percentages of a set of random topics using the *random view* of Cyclon [16]. That is, the percentage of each topic is estimated by a large number of nodes. Through aggregating and averaging these percentages for the same topic, we can obtain an accurate percentage for each topic. Secondly, we are planning to evaluate the effects of bandwidth, memory, and computation resources on the performance of Feverfew. These factors will possibly require strategies that are more adaptive in sampling nodes and balancing the workload of each node. Take the bandwidth as an example, the aggregate upload capacity of all nodes is the main bottleneck of data dissemination in a P2P overlay. One approach is to dispatch extra servers to improve the upload bandwidth and accelerate message routing. Thirdly, although Feverfew guarantees reliable data dissemination among online nodes, we still wish to provide messages to offline nodes when they are online again. For the dynamic networks, it is necessary to provide an efficient data replication scheme [25] to guarantee reliable access to these offline nodes.

Acknowledgements

This work was supported by National Grand Fundamental Research 973 Program of China (Grant No. 2011CB302601), National Natural Science Foundation of China (Grant No. 61379052), National High-tech R&D Program of China (863) (Grant No.2013AA01A213), Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No. S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20124307110015).

References

1 Lu X, Wang H, Wang J, et al. Internet-based virtual computing environment: beyond the data center as a computer. Future Gener Comput Syst, 2011, 29: 309–322
2 Li M, Ye F, Kim M, et al. A scalable and elastic publish/subscribe service. In: IEEE International Symposium on Parallel & Distributed Processing, Anchorage, 2011. 1254–1265
3 Rao W, Chen L, Hui P, et al. Move: a large scale keyword-based content filtering and dissemination system. In: Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems. Washington DC: IEEE, 2012. 445–454
4 Wang Y, Li X, Li X, et al. A survey of queries over uncertain data. Knowl Inf Syst, 2013, 37: 485–530
5 Patel J A, Riviere E, Gupta I, et al. Rappel: exploiting interest and network locality to improve fairness in publish-subscribe systems. Comput Netw, 2009, 53: 2304–2320
6 Baldoni R, Beraldi R, Quéma V, et al. Tera: topic-based event routing for peer-to-peer architectures. In: Proceedings of the Inaugural International Conference on Distributed Event-Based Systems. New York: ACM, 2007. 2–13
7 Rahimian F, Girdzijauskas S, Payberah A, et al. Vitis: a gossip-based hybrid overlay for Internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In: IEEE International Symposium on Parallel & Distributed Processing, Anchorage, 2011. 746–757
8 Wong B, Guha S. Quasar: a probabilistic publish-subscribe system for social networks. In: Proceedings of the 7th International Conference on Peer-to-peer systems. Berkeley: USENIX Association, 2008. 2
9 Castro M, Druschel P, Kermarrec A, et al. Scribe: a large-scale and decentralized application-level multicast infrastructure. IEEE J Sel Areas Commun, 2002, 20: 1489–1499

10 Zheng Z, Wang Y, Ma X. PeerChatter: a Peer-to-Peer architecture for data distribution over social networks. Inf-An Int Interdisc J, 2011, 15: 259–266
11 Bollobás B. Random Graphs. Cambridge: Cambridge University Press, 2001
12 Karp R M, Schindelhauer C, Shenker S, et al. Randomized rumor spreading. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, 2000. 565–574
13 Strom R, Banavar G, Chandra T, et al. Gryphon: an information flow based approach to message brokering. In: International Symposium on Software Reliability Engineering, Paderborn, 1998
14 Carzaniga A, Rosenblum D S, Wolf A L. Achieving scalability and expressiveness in an Internet-scale event notification service. In: Proceedings of the 19th Annual ACM symposium on Principles of Distributed Computing. New York: ACM, 2000. 219–227
15 Ratnasamy S, Handley M, Karp R, et al. Application-level multicast using content-addressable networks. In: Proceedings of the 3rd International COST264 Workshop on Networked Group Communication. London: Springer-Verlag, 2001. 14–29
16 Voulgaris S, Gavidia D, Steen M. Cyclon: inexpensive membership management for unstructured P2P overlays. J Netw Syst Manage, 2005, 13: 197–217
17 Jelasity M, Montresor A, Babaoglu Ö. T-man: gossip-based fast overlay topology construction. Comput Netw, 2009, 53: 2321–2339
18 Chockler G, Melamed R, Tock Y, et al. Constructing scalable overlays for pub-sub with many topics. In: Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing, Portland, 2007. 109–118
19 Onus M, Richa A W. Minimum maximum degree publish-subscribe overlay network design. IEEE/ACM Trans Netw, 2011, 19: 1331–1343
20 Melamed R, Keidar I. Araneola: a scalable reliable multicast system for dynamic environments. In: Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications, Cambridge, 2004. 5–14
21 Chockler G, Melamed R, Tock Y, et al. Spidercast: a scalable Interest-aware overlay for topic-based pub/sub communication. In: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems. New York: ACM, 2007. 14–25
22 Jelasity M, Montresor A, Babaoglu Ö. Gossip-based aggregation in large dynamic networks. ACM Trans Comput Syst, 2005, 23: 219–252
23 Stoica I, Morris R, Karger D, et al. Chord: a scalable peer-to-peer lookup service for Internet applications. ACM SIGCOMM Comput Commun Rev, 2001, 31: 149–160
24 Wang T, Chen Y, Zhang Z, et al. Unbiased sampling in directed social graph. In: SIGCOMM, New Delhi, 2010. 401–402
25 Wang Y, Li S. Research and performance evaluation of data replication technology in distributed storage systems. Int J Comput Math Appl, 2006, 51: 1625–1632